A collection of recreational problems

Riddler Solutions

Mostly computational solutions in Python

Ishmav Jayladhan

Jayladhan & Pollagudi · London

Contents

Chapter 1

Where do the Rancher's children roam?

Consider four square-shaped ranches arranged in a 2-by-2 pattern, as if part of a larger checkerboard. One family lives on each ranch, and each family builds a small house independently at a random place within the property. Later, as the families in adjacent quadrants become acquainted, they construct four straight-line paths between the houses that go across the boundaries between the ranches. These paths form a quadrilateral circuit path connecting all four houses. This circuit path is also the boundary of the area where the families' children are allowed to roam. What is the probability that the children are able to travel in a straight line from any allowed place to any other allowed place without leaving the boundaries? (In other words, what is the probability that the quadrilateral is convex?)

Solution

The idea is to use Monte Carlo simulation to generate thousands of quadrilaterals and count the percentage of quadrilaterals that are convex. For convex polygons, all internal angles are less than 180 degrees. We will check each internal angle by determining the sign of the angle created by the two vectors intersecting at each vertex. This can be done by taking the sign of determinant of the two vectors. If all angles are the same direction(sign), the polygon is convex else it is concave. Using the Python code for the simulation given below

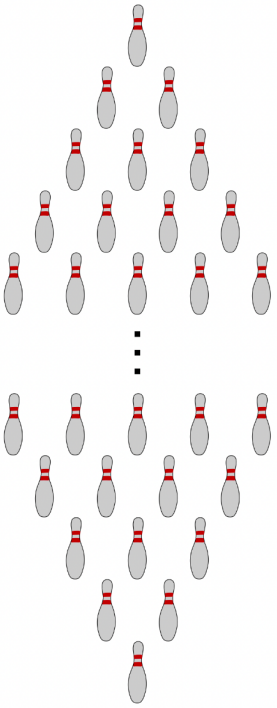
we see that the probability of the quadrilateral being convex is 0.908288.

```
from numpy.linalg import det
from numpy import sign, array

def isConvex(poly):
  n = len(poly)
  prev = sign(det(array([poly[0]-poly[1],poly[2]-poly[1]])))
  for i in range(1,n):
    curr = sign(det(array([poly[i]-poly[(i+1)%n],poly[(i+2)%n]-poly[(i+1)%n]]
    if prev != curr:
      return False
    prev = curr
return True
^^I
from numpy.random import uniform
runs = 1000000
cnt_convex = 0
for _ in range(runs):
  q1=array([uniform(0, 1), uniform(0, 1)])
  q2=array([uniform(-1, 0), uniform(0, 1)])
  q3=array([uniform(-1, 0), uniform(-1, 0)])
  q4=array([uniform(0, 1), uniform(-1, 0)])
  if isConvex(array([q1,q2,q3,q4])):
    cnt_convex += 1
print(cnt_convex/runs)
```

Chapter 2

Can You Knock Down The Last Bowling Pin?

Magritte the bowler is back! This time, he is competing head-to-head against fellow bowler Fosse. However, rather than knocking down 10 pins arranged in a triangular formation, they are trying to knock down N^2 pins (where N is some very, very large number) arranged in a rhombus, as shown below:



When Magritte rolls, he always knocks down the topmost pin. Then, if any pin gets knocked down, it has a 50 percent chance of knocking down either of the two pins directly behind it, independently of each other. (If there is only one pin directly behind it, then it too has a 50 percent chance of being knocked over.)

Fosse is a stronger bowler than Magritte. Like Magritte, she always knocks down the topmost pin. But each pin that's knocked down then has a 70 percent chance (rather than Magritte's 50 percent) of knocking down any of the pins directly behind it.

What are Magritte's and Fosse's respective probabilities of knocking down the bottommost pin in the rhombus formation?

Solution

This happens to be an *unsolved problem* from **percolation theory**. Instead of trying to find a closed form analytical solution, we will rely on numerical simulation. For the simulation, we first create a directed graph corresponding to the rhombic layout of the pins. Each pin corresponds to a node in the graph and it connected by a directed edge to one or two pins below that can be knocked down by it. In each run of the simulation we iteratively (maximum number of iterations per run is 2n-2) identify the pins that are knocked down by looking at the neighbours of the pins that were knocked down in the previous level and the probability of each pin being knocked down. From the Monte Carlo simulation we see that the probability of Magritte's knocking down the bottommost pin in the rhombus formation is **0** and that of Fosse's is **0.583**.

```
def create_rhombic_pin_graph(n):
    pin_graph = nx.DiGraph()
    for i in range(1,n^2+1):
        pin_graph.add_node(i)
    prev =  [1]
    for i in range(2,n+1):
        curr, l = [], prev[-1]
        for _ in range(i):
            l += 1
            curr.append(l)
```

```
        for j,k in enumerate(prev):
            pin_graph.add_edge(k,curr[j])
            pin_graph.add_edge(k,curr[j+1])
        prev = curr
    for i in range(n-1,0,-1):
        curr, l = [], prev[-1]
        for _ in range(i):
            l += 1
            curr.append(l)
        for j,k in enumerate(curr):
            pin_graph.add_edge(prev[j],k)
            pin_graph.add_edge(prev[j+1],k)
        prev = curr
    return pin_graph

from random import random

def isKnocked(prob):
    if random()<prob:
        return True
    else:
        return False

def prob_last_pin_knocked(knock_prob, n_pins=500, runs = 10000):
    pin_graph = create_rhombic_pin_graph(n_pins)
    cnt_success = 0
    for _ in range(runs):
        p_knckd = set([1])
        for _ in range(2*n_pins-2):
            c_knckd = set()
            for p in p_knckd:
                for nbr in pin_graph[p].keys():
                    if isKnocked(knock_prob):
                        c_knckd.add(nbr)
            p_knckd = c_knckd.copy()
        if n_pins**2 in p_knckd:
            cnt_success += 1
    return cnt_success/runs

print(prob_last_pin_knocked(0.7))
```

Chapter 3

Can You Find Your Pills?

I've been prescribed to take 1.5 pills of a certain medication every day for 10 days, so I have a bottle with 15 pills. Each morning, I take two pills out of the bottle at random.

On the first morning, these are guaranteed to be two full pills. I consume one of them, split the other in half using a precision blade, consume half of that second pill, and place the remaining half back into the bottle.

On subsequent mornings when I take out two pills, there are three possibilities:

I get two full pills. As on the first morning, I split one and place the unused half back into the bottle. I get one full pill and one half-pill, both of which I consume. I get two half-pills. In this case, I take out another pill at random. If it's a half-pill, then I consume all three halves. But if it's a full pill, I split it and place the unused half back in the bottle. Assume that each pill — whether it is a full pill or a half-pill — is equally likely to be taken out of the bottle.

On the $10th$ day, I again take out two pills and consume them. In a rush, I immediately throw the bottle in the trash before bothering to check whether I had just consumed full pills or half-pills. What's the probability that I took the full dosage, meaning I don't have to dig through the trash for a remaining half-pill?

Solution

The Python code for the Monte Carlo simulation below gives us a probability of **0.79** for taking the full dosage.

```
from random import sample
runs, n_pills, n_days = 100000, 15, 10
cnt_full_dose = 0
for _ in range(runs):
  pills = [('f',i) for i in range(n_pills)]
  for _ in range(n_days-1):
    consumed_pills = sample(pills,2)
    for p in consumed_pills:
      pills.remove(p)
    (s1,i), (s2,_) = consumed_pills
    if s1=='f' and s2=='f':
      pills.append(('h',i))
    if s1=='h' and s2=='h':
      pill = sample(pills,1)[0]
      pills.remove(pill)
      if pill[0]=='f':
        pills.append(('h',pill[1]))
  if len(pills)==2:
    cnt_full_dose += 1
print(cnt_full_dose/runs)
```

Chapter 4

Can You Make Room For Goats?

A goat tower has 10 floors, each of which can accommodate a single goat. Ten goats approach the tower, and each goat has its own (random) preference of floor. Multiple goats can prefer the same floor.

One by one, each goat walks up the tower to its preferred room. If the floor is empty, the goat will make itself at home. But if the floor is already occupied by another goat, then it will keep going up until it finds the next empty floor, which it will occupy. But if it does not find any empty floors, the goat will be stuck on the roof of the tower.

What is the probability that all 10 goats will have their own floor, meaning no goat is left stranded on the roof of the tower?

Solution

Here is the Python code for a Monte Carlo simulation. The probability that no goat is left stranded on the roof of the tower is **0.235745**.

```
from random import choices
runs, n_goats = 1000000, 10
cnt_success = 0
for _ in range(runs):
    tower = [-1 for _ in range(n_goats)]
    for i,j in enumerate(choices(range(n_goats), k=n_goats)):
        if tower[j]==-1:
            tower[j]=i
        else:
            k = j+1
            while k < n_goats-1 and tower[k]!=-1:
                k += 1
            if k <= n_goats-1:
```

```
                tower[k] = i
    if -1 not in tower:
        cnt_success += 1
print(cnt_success/runs)
```

Chapter 5

How Long Will Your Smartphone Distract You From Family Dinner?

You've just finished unwrapping your holiday presents. You and your sister got brand-new smartphones, opening them at the same moment. You immediately both start doing important tasks on the Internet, and each task you do takes one to five minutes. (All tasks take exactly one, two, three, four or five minutes, with an equal probability of each). After each task, you have a brief moment of clarity. During these, you remember that you and your sister are supposed to join the rest of the family for dinner and that you promised each other you'd arrive together. You ask if your sister is ready to eat, but if she is still in the middle of a task, she asks for time to finish it. In that case, you now have time to kill, so you start a new task (again, it will take one, two, three, four or five minutes, exactly, with an equal probability of each). If she asks you if it's time for dinner while you're still busy, you ask for time to finish up and she starts a new task and so on. From the moment you first open your gifts, how long on average does it take for both of you to be between tasks at the same time so you can finally eat? (You can assume the "moments of clarity" are so brief as to take no measurable time at all.)

Solution

From the Python code below using Monte-Carlo simulation, we see that on average it takes

```
from random import randrange
from itertools import accumulate
```

```
def break_points():
    return set(accumulate([randrange(1, 6) for _ in range(100)]))

runs = 10000
s_runs, total_t = 0, 0
for _ in range(runs):
    brother, sister = break_points(), break_points()
    common_points = brother.intersection(sister)
    if common_points:
        total_t += min(common_points)
        s_runs += 1
print(total_t/s_runs)
```

Chapter 6

Will The Neurotic Basketball Player Make His Next Free Throw?

A basketball player is in the gym practicing free throws. He makes his first shot, then misses his second. This player tends to get inside his own head a little bit, so this isn't good news. Specifically, the probability he hits any subsequent shot is equal to the overall percentage of shots that he's made thus far. (His neuroses are very exacting.) His coach, who knows his psychological tendency and saw the first two shots, leaves the gym and doesn't see the next 96 shots. The coach returns, and sees the player make shot No. 99. What is the probability, from the coach's point of view, that he makes shot No. 100?

Solution

The probability from the coach's point of view that the player will make shot No.100 is **.666**.

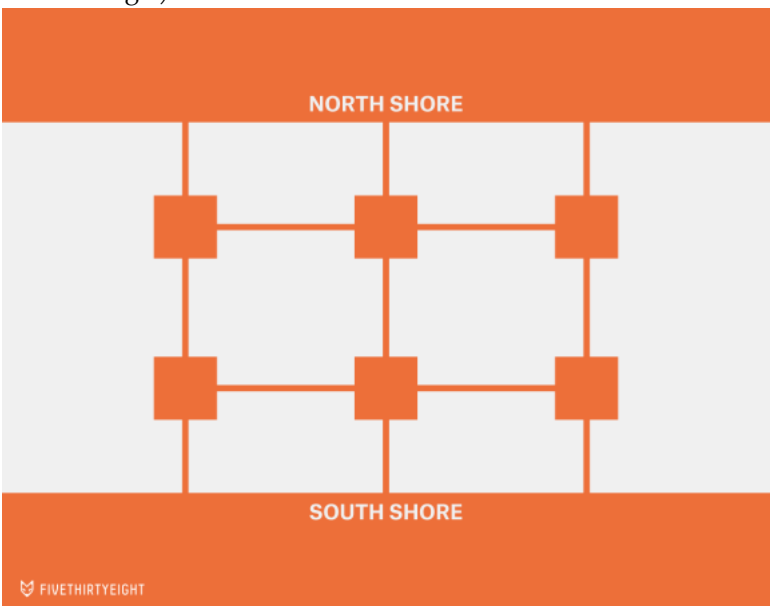
```
from random import random
runs = 100000
num_99s, num_99_100s = 0, 0
for _ in range(runs):
    throws = 2
    hits = 1
    for _ in range(3, 99):
        prob_next = hits/throws
        throws += 1
        if random() <= prob_next:
            hits += 1
    if random() <= (hits/throws):
        throws += 1
        hits += 1
        num_99s += 1
```

```
        if random() <= (hits/throws):
            throws += 1
            hits += 1
            num_99_100s += 1
print(num_99_100s/num_99s)
```

Chapter 7

Night Falls. A Storm Rolls In. Can You Cross The River?

You're on the north shore of a river, and want to cross to the south, via a series of 13 bridges and six islands, which you can see in the diagram below. But, as you approach the water, night falls, a bad storm rolls in, and you're forced to wait until morning to try to cross. Overnight, the storm could render some of the bridges unusable — it has a 50 percent chance of knocking out each of the bridges. (The chance is independent for each bridge.)



What's the probability you will be able to cross the river in the morning? (You have no boat, can't swim, can't fix the bridges, etc. No tricks.)

Solution

The probability that you will be able to cross the river in the morning is **0.5**.

```
from collections import defaultdict
from random import random

def find_path(graph, start, end, path=[]):
        path = path + [start]
        if start == end:
            return path
        if start not in graph.keys():
            return None
        for node in graph[start]:
            if node not in path:
                newpath = find_path(graph, node, end, path)
                if newpath: return newpath
        return None

def create_graph(bridges):
    graph = defaultdict(list)
    for (n1, n2) in bridges:
        graph[n1].append(n2)
        graph[n2].append(n1)
    return graph

runs = 100000
bridges = [(0,1), (0,2), (0,3), (1,2),
           (1,4), (2,3), (2,5), (3,6),
           (4,5), (4,7), (5,6), (6,7), (7,5)]
knock_out_prob = 0.5
cnt_success = 0
for _ in range(runs):
    remaining_bridges = []
    for bridge in bridges:
        if random() <= knock_out_prob:
            remaining_bridges.append(bridge)
    if find_path(create_graph(remaining_bridges), 0, 7):
        cnt_success += 1
print(cnt_success/runs)
```

Chapter 8

How Many Cars Will Get Stuck In Traffic?

There is a very long, straight highway with some number of cars (N) placed somewhere along it, randomly. The highway is only one lane, so the cars can't pass each other. Each car is going in the same direction, and each driver has a distinct positive speed at which she prefers to travel. Each preferred speed is chosen at random. Each driver travels at her preferred speed unless she gets stuck behind a slower car, in which case she remains stuck behind the slower car. On average, how many groups of cars will eventually form? (A group is one or more cars travelling at the same speed.)

For example, if the car in the very front happens to be slowest, there will be exactly one group — everybody will eventually pile up behind the slowpoke. If the cars happen to end up in order, fastest to slowest, there will be N groups — no car ever gets stuck behind a slower car.

Solution

```
from itertools import permutations

def groups(perm):
    groups = 1
    min_speed = perm[0]
    for i in range(1, len(perm)):
        if perm[i] < min_speed:
            groups += 1
            min_speed = perm[i]
    return groups
```

```
runs = 11
for n in range(2, runs):
    cnt = 0
    total_groups = 0
    for p in permutations(range(n)):
        total_groups += groups(p)
        cnt += 1
    print(total_groups/cnt)
```

Chapter 9

Will Someone Be Sitting In Your Seat On The Plane?

There's an airplane with 100 seats, and there are 100 ticketed passengers each with an assigned seat. They line up to board in some random order. However, the first person to board is the worst person alive, and just sits in a random seat, without even looking at his boarding pass. Each subsequent passenger sits in his or her own assigned seat if it's empty, but sits in a random open seat if the assigned seat is occupied. What is the probability that you, the hundredth passenger to board, finds your seat unoccupied?

Solution

The probability of the hundredth passenger finding his or her seat unoccupied is 0.5.

This probability is independent of the number of seats/passengers.

```
from random import sample

runs = 10000
num_seats = 100
cnt_success = 0
for _ in range(runs):
    seats = set(range(num_seats))
    first_seat = sample(seats, 1)[0]
    seats.remove(first_seat)
    for i in range(1, num_seats-1):
        if i in seats:
            seats.remove(i)
        else:
```

```
            seat = sample(seats, 1)[0]
            seats.remove(seat)
    if num_seats-1 in seats:
        cnt_success += 1
print(cnt_success/runs)
```

Chapter 10

Should You Pay $250 To Play This Casino Game?

Suppose a casino invents a new game that you must pay $250 to play. The game works like this: The casino draws random numbers between 0 and 1, from a uniform distribution. It adds them together until their sum is greater than 1, at which time it stops drawing new numbers. You get a payout of $100 each time a new number is drawn.

For example, suppose the casino draws 0.4 and then 0.7. Since the sum is greater than 1, it will stop after these two draws, and you receive $200. If instead it draws 0.2, 0.3, 0.3, and then 0.6, it will stop after the fourth draw and you will receive $400. Given the $250 entrance fee, should you play the game?

Specifically, what is the expected value of your winnings?

Solution

The average winnings is ≈ 272.

```
from random import random

runs = 10000

def average_winnings(payout, runs = runs):
    total = 0
    for _ in range(runs):
        total_draw = random()
        total += payout
        while(total_draw < 1):
            total_draw += random()
            total += payout
```

```
    return total/runs

print("Average winnings: ", average_winnings(100))
```

Chapter 11

Should You Shoot Free Throws Underhand?

Hark! The NCAA Tournament starts next week, and the "granny shot" has reappeared as a free throw technique. Its proponents claim that it improves accuracy because there are fewer moving parts — the elbows and wrists are held more stable, for example, and the move is symmetric because one's arms are, more or less, equal length. Let's find out how effective the granny shot really is.

Consider the following simplified model of free throws. Imagine the rim to be a circle (which we'll call C) that has a radius of 1, and is centered at the origin (the point $(0, 0)$). Let V be a random point in the plane, with coordinates X and Y, and where X and Y are independent normal random variables, with means equal to zero and each having equal variance — think of this as the point where your free throw winds up, in the rim's plane. If V is in the circle, your shot goes in. Finally, suppose that the variance is chosen such that the probability that V is in C is exactly 75 percent (roughly the NBA free-throw average).

But suppose you switch it up, and go granny-style, which in this universe eliminates any possible left-right error in your free throws. What's the probability you make your shot now? (Put another way, calculate the probability that $|Y| < 1$.)

Solution

$R \sim \text{Rayleigh}(\sigma)$ is Rayleigh distributed if $R = \sqrt{X^2 + Y^2}$, where $X \sim N(0, \sigma^2)$ and $Y \sim N(0, \sigma^2)$ are independent normal random variables.

From the above, it is clear that V follows the Rayleigh distribution.

The CDF of V is given by $1 - e^{-x^2/\sigma^2}$.

The fact that the shot goes in 75% of the time is equivalent to

$$\mathbb{P}[V \leq 1] = 1 - e^{-1/2\sigma^2} = 0.75$$

Therefore, $\sigma^2 = 0.36$ and $Y \sim N(0, 0.36)$.

The probability that $|Y| < 1$ is given by $\mathbb{P}[-1 \leq Y \leq 1] = 0.904419$.

Chapter 12

Can You Slay The Puzzle Of The Monsters' Gems?

A video game requires you to slay monsters to collect gems. Every time you slay a monster, it drops one of three types of gems: a common gem, an uncommon gem or a rare gem. The probabilities of these gems being dropped are in the ratio of 3:2:1 — three common gems for every two uncommon gems for every one rare gem, on average. If you slay monsters until you have at least one of each of the three types of gems, how many of the most common gems will you end up with, on average?

Solution

```
from random import random

runs = 100000
total_c = 0
for _ in range(runs):
    c, u, r = 0, 0, 0
    while(True):
        p = random()
        if p <= 1/6:
            r += 1
        elif  p >= 1/6 and p <= 1/2:
            u += 1
        else:
            c += 1
        if c > 0 and r > 0 and u > 0:
            break
    total_c += c
print("Average number of common gems:", total_c/runs)
```

The average number of common gems collected before we have one of each gem is 3.65.

Can You Solve This Elevator Button Puzzle?

In a building's lobby, some number (N) of people get on an elevator that goes to some number (M) of floors. There may be more people than floors, or more floors than people. Each person is equally likely to choose any floor, independently of one another. When a floor button is pushed, it will light up. What is the expected number of lit buttons when the elevator begins its ascent?

Solution

Let $L = F_1 + F_2 + F_3 + \ldots F_M$ be the random variable indicating the number of lit buttons where F_i is the indicator random variable which is 1 when at least 1 person chooses floor i and 0 otherwise.

$\mathbb{P}[F_i = 1] = 1 - (\frac{M-1}{M})^N$ where $(\frac{M-1}{M})^N$ is the probability of no one selecting floor i.

Therefore $\mathbb{E}[L] = \mathbb{E}[\sum_{i=1}^{M} F_i] = M(1 - (\frac{M-1}{M})^N)$.

Computational solution

```
from random import random, choice
from collections import Counter

n, m = 30, 20
runs = 10000
total_lit = 0
for _ in range(runs):
    choices = [choice(range(m)) for _ in range(n)]
```

```
    num_lit = sum([1 if c > 0 else 0 for l, c in Counter(choices).items()])
    total_lit += num_lit
print(total_lit/runs)
```

Chapter 14

The Puzzle Of Misanthropic Neighbors?

The misanthropes are coming. Suppose there is a row of some number, N, of houses in a new, initially empty development. Misanthropes are moving into the development one at a time and selecting a house at random from those that have nobody in them and nobody living next door. They keep on coming until no acceptable houses remain. At most, one out of two houses will be occupied; at least one out of three houses will be. But what's the expected fraction of occupied houses as the development gets larger, that is, as N goes to infinity?

Solution

The fraction of the occupied houses for large n is approximately is 0.4325773.

```
from random import sample

runs = 10000
n = 1000
total_occupancy = 0
for _ in range(runs):
    remaining_houses = set(range(n))
    chosen_houses = []
    while remaining_houses:
        chosen_house = sample(remaining_houses, 1)[0]
        chosen_houses.append(chosen_house)
        remaining_houses.discard(chosen_house)
        for adj in [-1, 1]:
            remaining_houses.discard(chosen_house + adj)
    total_occupancy += len(chosen_houses)
print(total_occupancy/(n*runs))
```

Chapter 15

You Have $1 Billion To Win A Space Race. Go.

You are the CEO of a space transport company in the year 2080, and your chief scientist comes in to tell you that one of your space probes has detected an alien artifact at the Jupiter Solar Lagrangian (L2) point.

You want to be the first to get to it! But you know that the story will leak soon and you only have a short time to make critical decisions. With standard technology available to anyone with a few billion dollars, a manned rocket can be quickly assembled and arrive at the artifact in 1,600 days. But with some nonstandard items you can reduce that time and beat the competition. Your accountants tell you that they can get you an immediate line of credit of $1 billion.

You can buy:

Big Russian engines. There are only three in the world and the Russians want $400 million for each of them. Buying one will reduce the trip time by 200 days. Buying two will allow you to split your payload and will save another 100 days. NASA ion engines. There are only eight of these $140 million large-scale engines in the world. Each will consume 5,000 kilograms of xenon during the trip. There are 30,000 kg of xenon available worldwide at a price of $2,000/kg, so 5,000 kg costs $10 million. Bottom line: For each $150 million fully fueled xenon engine you buy, you can take 50 days off of the trip. Light payloads. For $50 million each, you can send one of four return flight fuel tanks out ahead of the mission, using existing technology. Each time you do this, you lighten the

main mission and reduce the arrival time by 25 days. What's your best strategy to get there first?

Solution

```
from itertools import product

#Number available
num_re = 3
num_ie = 8
num_ft = 4
num_xe = 6 #30000/5000

#Costs in millions
re_cost = 400
ie_cost = 140
ft_cost = 50
xe_cost = 10

#Savings in days
ie_saving = 50
ft_saving = 25
re_saving = {0:0, 1:200, 2:300, 3:500}

#Available loan
loan = 1000

optimal_sols =[]
for r, i, f, x in product(range(num_re+1), range(num_ie+1),
                          range(num_ft+1),  range(num_xe+1)):
    days_saved = f*ft_saving + min(x, i)*ie_saving + re_saving[r]
    total_cost = f*ft_cost + i*ie_cost + r*re_cost + x*xe_cost
    days_savings_avlbl = (num_ft-f)*ft_saving + \
        min(num_ie-i, num_xe-x)*ie_saving + re_saving[num_re-r]
    if total_cost <= loan and days_saved > days_savings_avlbl:
        optimal_sols.append((days_savings_avlbl, days_saved, r, i, f, x))

for dsa, ds, r, i, f , x in optimal_sols:
    print("Number of Russian Engines: ", r)
    print("Number of Ion Engines: ", i)
    print("Number of Fuel Tanks: ", f)
    print("Number of Xenon kgs: ", x*5000)
    print("Total number of days saved: ", ds)
    print("Total day savings available: ", dsa)
    print("\n")
```

Valid combinations

Number of Russian Engines: 1
Number of Ion Engines: 1
Number of Fuel Tanks: 4
Number of Xenon kgs: 30000
Total number of days saved: 350
Total day savings available: 300

Number of Russian Engines: 1
Number of Ion Engines: 2
Number of Fuel Tanks: 3
Number of Xenon kgs: 30000
Total number of days saved: 375
Total day savings available: 325

Number of Russian Engines: 1
Number of Ion Engines: 2
Number of Fuel Tanks: 4
Number of Xenon kgs: 25000
Total number of days saved: 400
Total day savings available: 350

Number of Russian Engines: 1
Number of Ion Engines: 2
Number of Fuel Tanks: 4
Number of Xenon kgs: 30000
Total number of days saved: 400
Total day savings available: 300

Number of Russian Engines: 1
Number of Ion Engines: 3
Number of Fuel Tanks: 2
Number of Xenon kgs: 30000
Total number of days saved: 400
Total day savings available: 350

Number of Russian Engines: 2
Number of Ion Engines: 0
Number of Fuel Tanks: 1

Number of Xenon kgs: 30000
Total number of days saved: 325
Total day savings available: 275

Number of Russian Engines: 2
Number of Ion Engines: 0
Number of Fuel Tanks: 2
Number of Xenon kgs: 25000
Total number of days saved: 350
Total day savings available: 300

Number of Russian Engines: 2
Number of Ion Engines: 0
Number of Fuel Tanks: 2
Number of Xenon kgs: 30000
Total number of days saved: 350
Total day savings available: 250

Number of Russian Engines: 2
Number of Ion Engines: 0
Number of Fuel Tanks: 3
Number of Xenon kgs: 20000
Total number of days saved: 375
Total day savings available: 325

Number of Russian Engines: 2
Number of Ion Engines: 0
Number of Fuel Tanks: 3
Number of Xenon kgs: 25000
Total number of days saved: 375
Total day savings available: 275

Number of Russian Engines: 2
Number of Ion Engines: 1
Number of Fuel Tanks: 0
Number of Xenon kgs: 30000
Total number of days saved: 350
Total day savings available: 300

Chapter 16

Should The Grizzly Bear Eat The Salmon?

A grizzly bear stands in the shallows of a river during salmon spawning season. Precisely once every hour, a fish swims within its reach. The bear can either catch the fish and eat it, or let it swim past to safety. This grizzly is, as many grizzlies are, persnickety. It'll only eat fish that are at least as big as every fish it ate before.

Each fish weighs some amount, randomly and uniformly distributed between 0 and 1 kilogram. (Each fish's weight is independent of the others, and the skilled bear can tell how much each weighs just by looking at it.) The bear wants to maximize its intake of salmon, as measured in kilograms. Suppose the bear's fishing expedition is two hours long. Under what circumstances should it eat the first fish within its reach? What if the expedition is three hours long?

Solution

Let X_i be the random variable denoting the weight of fish seen each hour where $i = 1, 2, \ldots, n$.

We have $X_i \sim \mathcal{U}(0, 1)$ for $i = 1, 2, \ldots, n$.

Greedy Strategy

By following the **Greedy Strategy**, the bear eats as much salmon as it can starting with the first salmon in the first hour.

Let Y_i be the weight of the fish consumed in the i^{th} hour using the greedy strategy.

We have the following equations for the greedy strategy:

$$Z_i = \sum_i^n Y_i \tag{16.1}$$

$$Y_1 = X_1 \tag{16.2}$$

$$Y_i = X_i, \text{ if } X_i \geq \max(X_1, X_2, \ldots, X_{i-1}) \text{ and } i > 1 \tag{16.3}$$

The distribution function and the density function of a random variable $M_{i-1} = \max(X_1, X_2, \ldots, X_{i-1})$ are given by

$$F_{M_{i-1}}(x) = x^{i-1} \tag{16.4}$$

$$f_{M_{i-1}}(x) = (i-1)x^{i-2} \tag{16.5}$$

where $X_i \sim \mathcal{U}(0,1)$ and $i = 2, 3, \ldots, n$.
From 16.3, we have

$$\mathbb{E}[Y_i|M_{i-1}] = \int_{M_{i-1}}^1 x_i dx_i = \frac{1}{2} - \frac{M_{i-1}^2}{2} \tag{16.6}$$

The expectation of M_{i-1}^2 can be calculated as follows

$$\mathbb{E}[M_{i-1}^2] = \int_0^1 m^2 f_{M_{i-1}}(m)dm = \int_0^1 m^2(i-1)m^{i-2}dm = \frac{i-1}{i+1} \tag{16.7}$$

From 16.6, we have

$$\mathbb{E}[Y_i] = \mathbb{E}[\mathbb{E}[Y_i|M_{i-1}]] = \frac{1}{2} - \frac{1}{2}\mathbb{E}[M_{i-1}^2] = \frac{1}{2} - \frac{1}{2}\frac{i-1}{i+1} = \frac{1}{i+1} \tag{16.8}$$

Two hour expedition

To get the average total weight of salmon consumed by bear in 2 hours under the greedy strategy we need to calculate $\mathbb{E}[Z_2]$.

$$\mathbb{E}[Z_2] = \mathbb{E}[Y_1] + \mathbb{E}[Y_2] = \frac{1}{2} + \frac{1}{3} = \frac{5}{6} \tag{16.9}$$

Three hour expedition

To get the average total weight of salmon consumed by bear in 3 hours under the greedy strategy we need to calculate $\mathbb{E}[Z_3]$.

$$\mathbb{E}[Z_3] = \sum_{i=1}^{3} \mathbb{E}[Y_i] = \frac{1}{2} + \frac{1}{3} + \frac{1}{4} = \frac{13}{12} \tag{16.10}$$

```
from random import random

runs = 100000

total_weight = 0
for _ in range(runs):
    r1, r2 = random(), random()
    total_weight += r1
    if r2 > r1:
        total_weight += r2

print("Avg weight of fish consumed in 2 hrs :", total_weight/runs)

total_weight = 0
for _ in range(runs):
    r1, r2, r3 = random(), random(), random()
    total_weight += r1
    if r2 > r1:
        total_weight += r2
        if r3 > r2:
            total_weight += r3
    else:
        if r3 > r1:
            total_weight += r3

print("Avg weight of fish consumed in 3 hrs :", total_weight/runs)
```

The bear should **always eat the first fish**.

Chapter 17

How Long Will It Take To Blow Out The Birthday Candles?

It's your 30th birthday (congrats, by the way), and your friends bought you a cake with 30 candles on it. You make a wish and try to blow them out. Every time you blow, you blow out a random number of candles between one and the number that remain, including one and that other number. How many times do you blow before all the candles are extinguished, on average?

Solution

```
from random import randint

num_candles = 30
runs = 10000

total_num = 0
for _ in range(runs):
    candles = num_candles
    n = 0
    while(candles):
        blown = randint(1, candles)
        candles -= blown
        n += 1
    total_num += n
print("Average number of blows: ", total_num/runs)
```

The average number of blows based on the simulation above is ≈ 4.

Chapter 18

Who Steals The Most In A Town Full Of Thieves?

Riddler Express

You're driving a car down a two-mile track. For the first mile, you drive 30 miles per hour. How fast do you have to go for the second mile in order to average 60 miles per hour for the whole track?

Solution

We have the following equation:

$$60 = \frac{2}{\frac{1}{30} + \frac{1}{s}}$$

From the above it is obvious that achieving an average speed of 60 miles per hour is not possible.

Riddler Classic

A town of $1,000$ households has a strange law intended to prevent wealth-hoarding. On January 1 of every year, each household robs one other household, selected at random, moving all of that house's money into their own house. The order in which the robberies take place is also random and is determined by a lottery. (Note that if House A robs House B first, and then C robs A, the houses of A and B would each be empty and C would have acquired the resources of both A and B.)

Two questions about this fateful day:

What is the probability that a house is not robbed over the course of the day?

Suppose that every house has the same amount of cash to begin with — say $100. Which position in the lottery has the most expected cash at the end of the day, and what is that amount?

Solution

```
from random import sample

runs = 10000
num_houses = 1000
final_amts = [0] * num_houses
total_num_safe = 0

rob_choices = {i:(set(range(num_houses))-set([i]))
               for i in range(num_houses)}
lottery = list(range(num_houses))
rob = [None] * num_houses
amts = [None] * num_houses
for _ in range(runs):
    for i in range(num_houses):
        amts[i] = 100
        rob[i] = sample(rob_choices[i], 1)[0]
    total_num_safe += (num_houses - len(set(rob)))
    shuffle(lottery)
    for h in lottery:
        amts[h] += amts[rob[h]]
        amts[rob[h]] = 0
    for i,j in enumerate(lottery):
        final_amts[i] += amts[j]

print(total_num_safe/(runs*num_houses))
amt, pos = max(zip(final_amts, range(num_houses)))
print(amt/runs, pos)
```

The percent of houses that are not robbed on average is 37. The position in the lottery with the most expected cash is the **last** position and the maximum expected cash amount is $137.

Chapter 19

Can You Pick Up Sticks? Can You Help A Frogger Out?

A frog needs to jump across 20 lily pads. He starts on the shore (Number 0) and ends precisely on the last lily pad (Number 20). He can jump one or two lily pads at a time. How many different ways can he reach his destination?

What if he can jump one, two or three at a time? Or four? Five? Six? Etc.

Solution

Let J_n be the number of jumps required by the frog to reach the n^{th} lily pad. It is easy to see that $J_n = J_{n-1} + J_{n-2}$ and $J_1 = 1$ and $J_2 = 2$.

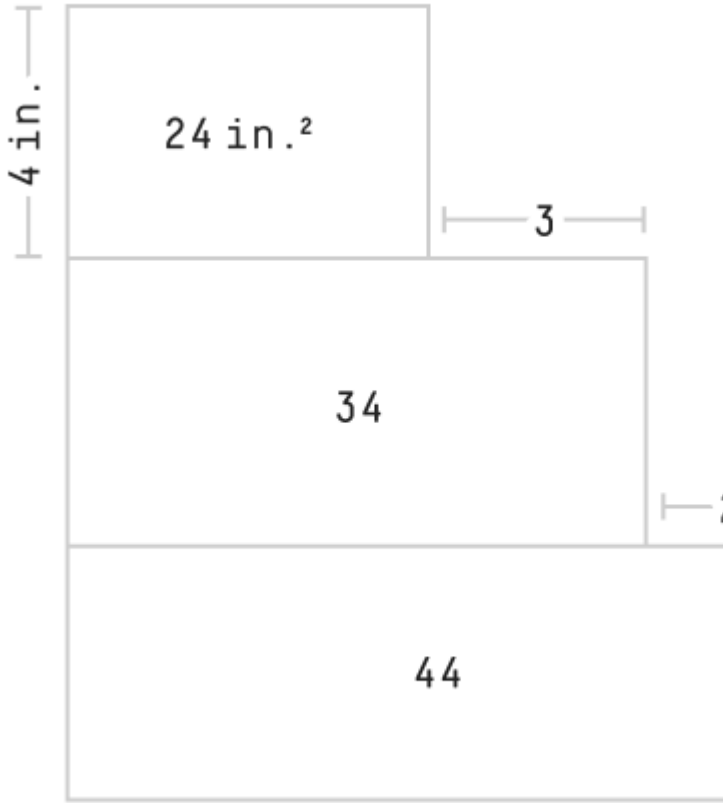
Therefore J_n is the **Fibonacci Sequence**.

Chapter 20

How Do You Like Them Rectangles?

✠

Riddler Express



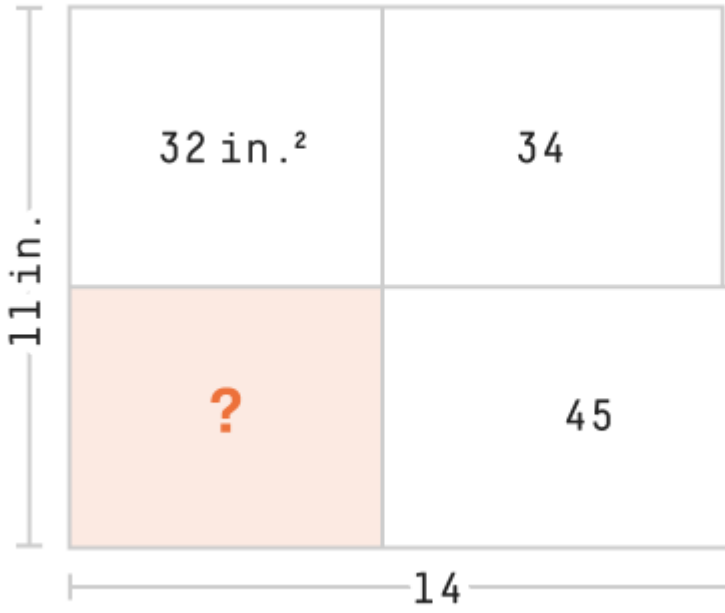
Solution

The length of the top rectangle = $24/4 = 6\ in$.

The length of the bottom rectangle = $6 + 3 + 2 = 11\ in$.

The height of the bottom rectangle = $44/11 = 4\ in$

Riddler Classic



Solution

Let xin and yin be the length and height of the shaded rectangle.

We have the following constraints:

$$(14 - x)y = 45 \tag{20.1}$$

$$(11 - y)x = 32 \tag{20.2}$$

$$\frac{66}{11 - y} < 14 \implies y < \frac{44}{7} \tag{20.3}$$

Subtracting (2) from (1) we have $14y - 11x = 13$.
Substituting $y = \frac{13+11x}{14}$ in (1), we have

$$\begin{aligned} 13 + 11x - 13x/14 - 11x^2/14 &= 45 \\ \implies 11x^2 - 141x + 448 &= 0 \\ \implies x &= \frac{141 \pm \sqrt{141^2 - 4 \cdot 11 \cdot 448}}{2 \cdot 11} \end{aligned}$$

$$\Longrightarrow x = 7 \; and \; y = \frac{45}{7} \quad or \quad x = \frac{64}{11} \; and \; y = \frac{11}{2}$$

The first solution has to be discarded because of constraint (3).

The area of the shaded region is $xy = \frac{64}{11}\frac{11}{2} = 32 \; in.^2$

Chapter 21

Will You Be A Ghostbuster Or A World Destroyer?

Here are four questions of increasing difficulty about finding sticks in the woods, breaking them and making shapes:

Problem 1

If you break a stick in two places at random, forming three pieces, what is the probability of being able to form a triangle with the pieces?

Solution

```
from random import random

runs, s = 1000000, 0
for _ in range(runs):
    r1,r2 = random(), random()
    if r2 > r1:
        x1, x2, x3 = r1, r2 - r1, 1 - r2
    else:
        x1, x2, x3 = r2, r1 - r2, 1 - r1
    if (x1 + x2 > x3) and (x2 + x3 > x1) and (x1 + x3 > x2): s += 1
print(s/runs)
```

The probability is 0.25.

Problem 2

If you select three sticks, each of random length (between 0 and 1), what is the probability of being able to form a triangle with them?

Solution

```
from random import random

runs, s = 1000000, 0
for _ in range(runs):
    x1,x2,x3 = random(), random(), random()
    if (x1 + x2 > x3) and (x2 + x3 > x1) and (x1 + x3 > x2): s += 1
print(s/runs)
```

The probability is 0.079256.

Problem 3

If you break a stick in two places at random, what is the probability of being able to form an acute triangle — where each angle is less than 90 degrees — with the pieces?

Solution

```
from random import random
runs, s = 1000000, 0

for _ in range(runs):
    r1,r2 = random(), random()
    if r2 > r1:
        x1, x2, x3 = r1, r2 - r1, 1 - r2
    else:
        x1, x2, x3 = r2, r1 - r2, 1 - r1
    if (x1 + x2 > x3) and (x2 + x3 > x1) and (x1 + x3 > x2):
        cos_a = (x1**2 + x2**2 - x3**2)/2*x1*x2
        cos_b = (x1**2 + x3**2 - x2**2)/2*x1*x3
        cos_c = (x2**2 + x3**2 - x1**2)/2*x2*x3
        if  cos_a >= 0 and cos_b >= 0 and cos_c >=0:
            s += 1
print(s/runs)
```

The probability is 0.214148.

Problem 4

If you select three sticks, each of random length (between 0 and 1), what is the probability of being able to form an acute triangle with the sticks?

Solution

```
from random import random
runs, s = 1000000, 0

for _ in range(runs):
    x1, x2, x3 = random(), random(), random()
    if (x1 + x2 > x3) and (x2 + x3 > x1) and (x1 + x3 > x2):
        cos_a = (x1**2 + x2**2 - x3**2)/2*x1*x2
        cos_b = (x1**2 + x3**2 - x2**2)/2*x1*x3
        cos_c = (x2**2 + x3**2 - x1**2)/2*x2*x3
        if  cos_a >= 0 and cos_b >= 0 and cos_c >=0:
            s += 1
print(s/runs)
```

The probability is 0.214148.

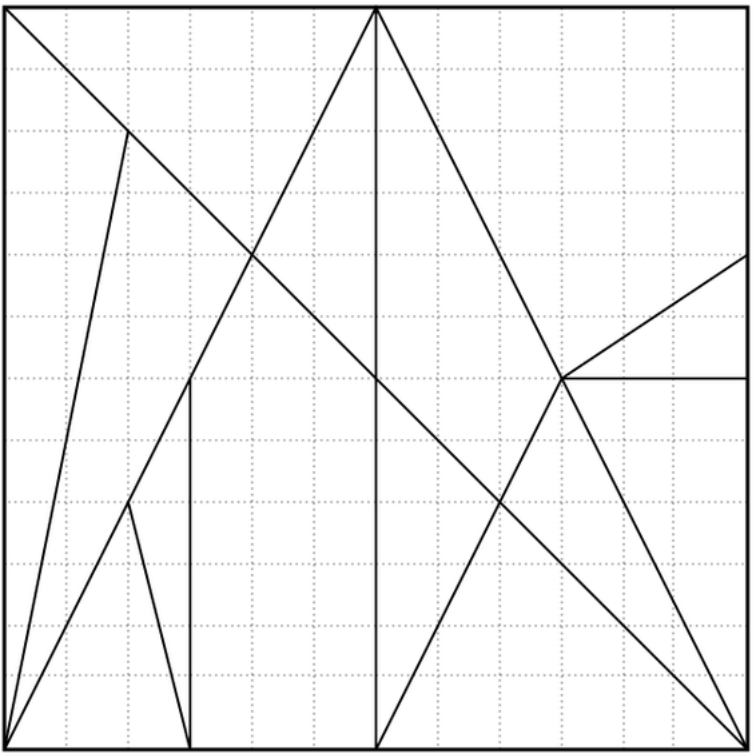
Chapter 22

How Often Does The Senate Vote In Palindromes?

The famous four-color theorem states, essentially, that you can color in the regions of any map using at most four colors in such a way that no neighboring regions share a color. A computer-based proof of the theorem was offered in 1976.

Some 2,200 years earlier, the legendary Greek mathematician Archimedes described something called an Ostomachion. It's a group of pieces, similar to tangrams, that divides a 12-by-12 square into 14 regions. The object is to rearrange the pieces into interesting shapes, such as a Tyrannosaurus rex. It's often called the oldest known mathematical puzzle.

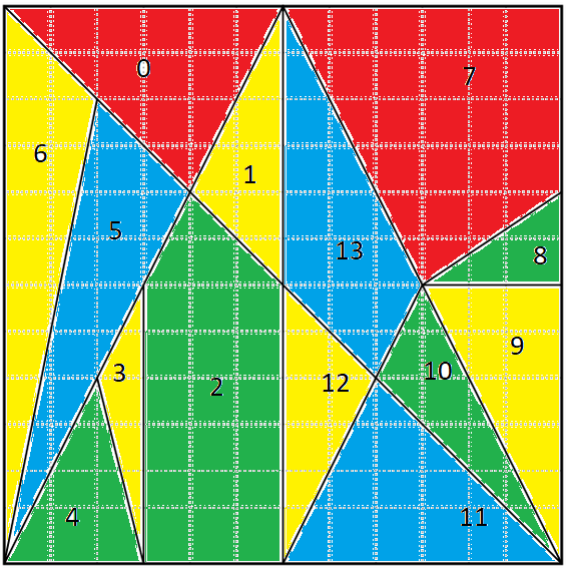
Your challenge today: Color in the regions of the Ostomachion square with four colors such that each color shades an equal area. (That is, each color needs to shade 36 square units.)



Extra credit: How many solutions to this challenge are there?

Solution

Let the areas be labelled as follows:




```
from z3 import *
from math import factorial

connections = {
    0:[1,5,6], 1:[0,2,13], 2:[1,12,3,5], 3:[2,4,5],
    4:[3,5], 5:[0,2,3,4,6], 6:[0,5], 7:[8,13], 8:[7,9],
    9:[8,10], 10:[9,11,13], 11:[10,12], 12:[11,13,2], 13:[1,7,10,12]
}

areas = {
    0:12, 1:6, 2:21, 3:3, 4:6, 5:12, 6:12,
    7:24, 8:3, 9:9, 10:6, 11:12, 12:6, 13:12
}

col_map = {0:"Red", 1:"Blue", 2:"Green", 3:"Yellow", 4:"Orange", 5:"Pink"}

num_colours = 4
total_area = 144
def OstomachionSolver():
    X = [Int("x_%s" % i) for i in range(14)]
    s = Solver()
    s.add([And(0 <= X[i], X[i]<= num_colours-1) for i in range(14)])
    for c in range(num_colours):
        s.add(sum([If(X[i] == c, areas[i], 0) for i in range(14)])
                == (total_area//num_colours))
    for i in range(14):
        for j in connections[i]:
            s.add(X[i] != X[j])

    cnt_sol = 0
```

```
    while s.check() == sat:
        cnt_sol += 1
        m = s.model()
        s.add(Or([X[i] != m.eval(X[i]) for i in range(14)]))
    print("Unique solutions :", cnt_sol / factorial(num_colours))
    for i in range(14):
        print("Area %d - %s " % (i , col_map[int(str(m.evaluate(X[i])))]))

OstomachionSolver()
```

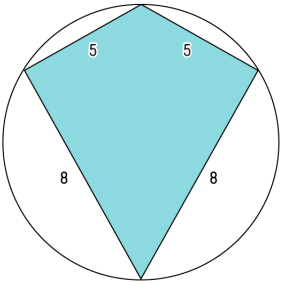
A colouring which satisfies the constraints is as follows:
Area 0 - Red
Area 1 - Yellow
Area 2 - Green
Area 3 - Yellow
Area 4 - Green
Area 5 - Blue
Area 6 - Yellow
Area 7 - Red
Area 8 - Green
Area 9 - Yellow
Area 10 - Green
Area 11 - Blue
Area 12 - Yellow
Area 13 - Blue

Chapter 23

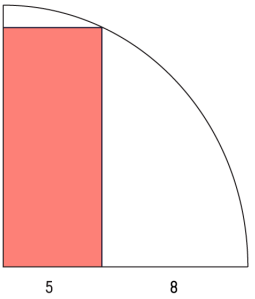
Where Will The Seven Dwarfs Sleep Tonight?

Riddler Express

A kite shape is inscribed in a circle, as shown below.



What is the kite's area?
A rectangle is drawn inside a quarter circle, as shown below.



What is the rectangle's area?

Solution

The longer diagonal of the kite is the diameter of the circle because of symmetry. Therefore, one half of a kite is a right angled triangle. The area of the kite is 40.

The diagonal of the rectangle is equal to the radius of the quadrant $= 13$. The length of the rectangle is 12 by Pythagoras theorem. Therefor the area of the rectangle is 60.

Riddler Classic

Each of the seven dwarfs sleeps in his own bed in a shared dormitory. Every night, they retire to bed one at a time, always in the same sequential order, with the youngest dwarf retiring first and the oldest retiring last. On a particular evening, the youngest dwarf is in a jolly mood. He decides not to go to his own bed but rather to choose one at random from among the other six beds. As each of the other dwarfs retires, he chooses his own bed if it is not occupied, and otherwise chooses another unoccupied bed at random.

What is the probability that the oldest dwarf sleeps in his own bed? What is the expected number of dwarfs who do not sleep in their own beds?

Solution

Expected number of wrong beds: 2.86. Chance of oldest in own bed: 0.4168.

```
from random import sample

runs = 100000
num_dwarfs = 7
cnt_last_wrong_bed = 0
cnt_wrong_beds = 0
for _ in range(runs):
    num_wrong_beds = 1
    remaining_beds = set(range(num_dwarfs))
    b = sample(range(1, num_dwarfs), 1)[0]
    remaining_beds.discard(b)
    for i in range(1, num_dwarfs-1):
        if i in remaining_beds:
            remaining_beds.discard(i)
```

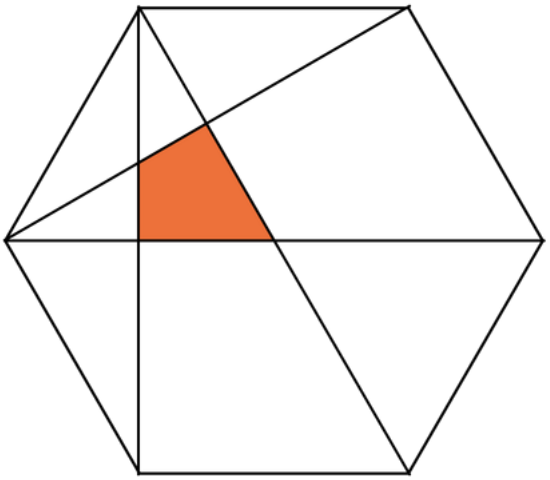
```
        else:
            num_wrong_beds += 1
            b = sample(remaining_beds, 1)[0]
            remaining_beds.discard(b)
    if num_dwarfs-1 not in remaining_beds:
        cnt_last_wrong_bed += 1
        num_wrong_beds += 1
    cnt_wrong_beds += num_wrong_beds
print("Expected number of wrong beds: ", cnt_wrong_beds/runs)
print("Chance of oldest in own bed: ", 1 - cnt_last_wrong_bed/runs)
```

Chapter 24

Help Us Find These Missing Pieces

Riddler Express

The regular hexagon below has an area of 1. What is the area of the shaded region?

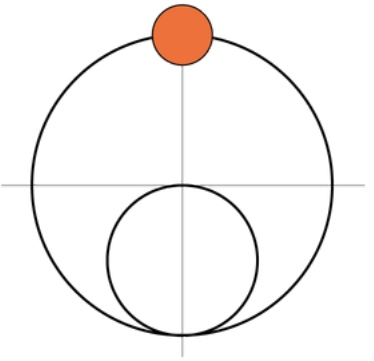


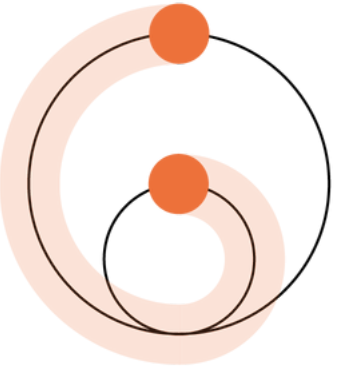
Solution

From the symmetry of the figure, we can see that the quadrilateral is $\frac{1}{3}^{th}$ the area of the equilateral triangle in which it is embedded. The area of the equilateral triangle is $\frac{1}{6}^{th}$ of the area of the hexagon. Therefore the area of the shaded quadrilateral is $\frac{1}{6} \cdot \frac{1}{3} \cdot 1 = \frac{1}{18}$.

Riddler Riddler Classic

The largest circle below has a radius of 10, the medium circle has a radius of 5 and the small, orange circle has a radius of 2. The orange circle crawls counterclockwise along the edge of the largest circle until it meets the medium circle, at which point it crawls up along the edge of the medium circle until it reaches the crest. What is the area of the shaded orange region in the right image?





Solution

The area of the shaded orange region has 3 parts.

The area of the small circle given by 4π.

The area of the semi annulus with internal radius 9 and outer radius 11 given by $\frac{\pi}{2}(12^2 - 8^2) = 40\pi$.

The area of the semi annulus with internal radius 4 and outer radius 6 given by $\frac{\pi}{2}(7^2 - 3^2) = 20\pi$.

Therefore the total area of the shaded region is 64π.

Chapter 25

When Will The Arithmetic Anarchists Attack?

Riddler Express

The year is 2000, and an arithmetical anarchist group has an idea. For the next 100 years, it will vandalize a famous landmark whenever the year (in two-digit form, for example this year is "18") is the product of the month and date (i.e. month × date = year, in the MM/DD/YY format).

A few questions about the lawless ensuing century: How many attacks will happen between the beginning of 2001 and the end of 2099? What year will see the most vandalism? The least? What will be the longest gap between attacks?

Solution

The year with the maximum number of attacks is **2024** with **7** attacks in total. There are a number of years with no attacks at all. The total number of attacks in all years is **212**. The maximum gap in days between two attacks is **1096**.

```
from datetime import date, timedelta
days_in_month = {
    1:31,
    2:28,
    3:31,
    4:30,
    5:31,
    6:30,
    7:31,
    8:31,
    9:30,
```

```
    10:31,
    11:30,
    12:31
}

attacks_by_year = {}
date_last_attack = date(2001,1,1)
max_gap = 0
for year in range(1, 100):
    attacks_by_year[year] = 0
    for month in days_in_month.keys():
        days = days_in_month[month]
        if year % 4 == 0 and month == 2:
            days += 1
        for day in range(1, days+1):
            if month*day == year:
                attacks_by_year[year] += 1
                date_latest_attack = date(2000+year, month, day)
                gap = date_latest_attack - date_last_attack
                if gap.days >= max_gap:
                    max_gap = gap.days
                date_last_attack = date_latest_attack


print("Max gap in days between attacks:", max_gap-1)
print("Attacks by year: ", attacks_by_year)
print("Total number of attacks: ", sum(attacks_by_year.values()))
print("Year of max attacks: ", max(attacks_by_year.items(),
                                   key = lambda x: x[1]))
```

Chapter 26

Can You Shuffle Numbers? Can You Find All The World Cup Results?

Riddler Classic

Imagine taking a number and moving its last digit to the front. For example, 1,234 would become 4,123. What is the smallest positive integer such that when you do this, the result is exactly double the original number? (For bonus points, solve this one without a computer.)

Solution

Let the last digit of our mystery number be a, and the rest of it be b. So the number we're searching for equals $10b + a$. Let b be an n-digit number. Then, moving the last digit to the front gives a new number equal to $10n \cdot a + b = 2(10b + a)$.

Simplifying that gives $(10^n - 2)a = 19b$. For a and b to be integers, we need the left-hand side to be divisible by 19, like the right-hand side is. In other words, we need it to be the case that $10^n = 2 \bmod 19$.

From **Little Fermat's Theorem** we have

$$\begin{aligned} 10^{18} &\equiv 1 \pmod{19} \\ \implies 2 \cdot 10^{18} &\equiv 2 \pmod{19} \\ \implies 20 \cdot 10^{17} &\equiv 2 \pmod{19} \\ \implies 10^{17} &\equiv 2 \pmod{19} \end{aligned}$$

Therefore the smallest n for which $10^n \equiv 2(\mod 19)$ is true is $n = 17$. So we have $[(10^{17}-2)/19]\cdot a = 5,263,157,894,736,842\cdot a = b$.

Can You Rescue The Paratroopers?

Riddler Express

You have one token, and I have two tokens. Naturally, we both crave more tokens, so we play a game of skill that unfolds over a number of rounds in which the winner of each round gets to steal one token from the loser. The game itself ends when one of us is out of tokens — that person loses. Suppose that you're better than me at this game and that you win each round two-thirds of the time and lose one-third of the time.

What is your probability of winning the game?

Solution

The probability of me winning the game is 57%.

```
from random import random

runs = 100000
cnt = 0
for _ in range(runs):
    me, you = 1, 2
    while(you != 0 and me != 0):
        if random() <= 2/3:
            you -= 1
            me += 1
        else:
            you += 1
            me -= 1
        if you == 0:
            cnt += 1
print(cnt/runs)
```

Chapter 28

Will The Grasshopper Land In Your Yard?

Riddler Express

You and your spouse each take two gummy vitamins every day. You share a single bottle of 60 vitamins, which come in two flavors. You each prefer a different flavor, but it seems childish to fish out two of each type (but not to take gummy vitamins). So you just take the first four that fall out and then divide them up according to your preferences. For example, if there are two of each flavor, you and your spouse get the vitamins you prefer, but if three of your preferred flavor come out, you get two of the ones you like and your spouse will get one of each.

The question is, on average, what percentage of the vitamins you take are the flavor you prefer? (Assume that the bottle starts out with a 50-50 split between flavors, and that the four selected each day are selected uniformly at random.)

Solution

The percentage of vitamins taken with the preferred flavour is 82%.

```
from random import sample

runs =  100000
total_pref_pct = 0
for _ in range(runs):
    pills = set(range(60))
    num_pref_pills = 0
```

```
    for _ in range(15):
        fish4 = sample(pills, 4)
        pills -= set(fish4)
        num_pref_pills += min(sum([1 if p % 2 == 0 else 0 for p in fish4]), 2
    total_pref_pct += num_pref_pills/30
print(total_pref_pct/runs)
```

Chapter 29

The Eternal Question: How Much Do These Apricots Weigh?

Riddler Express

You loaded a drying shed containing 1,000 kilograms of apricots. They were 99 percent water. After a day in the shed, they are now 98 percent water. How much do the apricots weigh now?

Solution

One day 1, total weight of water = 990 kgs and non-water stuff = 10 kgs. One day 2, after the evaporation of some water, let us say we are left with x kgs of water. We have,

$$\frac{x}{x+10} = .98$$
$$\implies x = 490kgs$$

Therefore total weight of apricots on day 2 = $x + 10 = 500$ kgs

Riddler Classic

I flip a coin. If it's heads, I've won the game. If it's tails, then I have to flip again, now needing to get two heads in a row to win. If, on my second toss, I get another tails instead of a heads, then I now need three heads in a row to win. If, instead, I get a heads on my second toss (having flipped a tails on the first toss) then I still need to get a second heads to have two heads in a row and win, but if my next toss is a tails (having

thus tossed tails-heads-tails), I now need to flip three heads in a row to win, and so on. The more tails you've tossed, the more heads in a row you'll need to win this game.

I may flip a potentially infinite number of times, always needing to flip a series of N heads in a row to win, where N is T + 1 and T is the number of cumulative tails tossed. I win when I flip the required number of heads in a row.

What are my chances of winning this game? (A computer program could calculate the probability to any degree of precision, but is there a more elegant mathematical expression for the probability of winning?)

Chapter 30

What Are The Odds You'd Already Have My Number?

My daughter recently noticed that the 10 digits of our old landline phone number are the same digits as those in my wife's cell phone. The first three digits match exactly because we are in the same area code as before. And the last seven digits of my wife's cell phone are an exact scrambled version of the last seven digits of the old landline. By "exact scramble," I mean a string of numbers that is different than the original string of numbers but contains all of the same digits with the exact same number of repetitions (so, for example, if "4" appears exactly three times in the landline number, it also appears exactly three times in my wife's cell number).

My daughter asked, "What are the odds of that?"

To make this concrete, assume that landlines and cell numbers in my area code are assigned randomly, such that a person is equally likely to get any of the 10,000,000 numbers between and including 000-0000 to 999-9999. Given that assumption, what is the probability that the last seven digits of the cell number are an exact scramble of the last seven digits of our landline?

Solution

The probability that the last seven digits of the cell number are an exact scramble of the last seven digits of our landline is 0.017 percent.

```
from itertools import product
from collections import defaultdict
```

```
phone_num_classes = defaultdict(int)
for p in product(*([range(10)]*7)):
    phone_num_classes["".join(sorted(map(str, p)))] += 1

prob = 0
for _, cnt in phone_num_classes.items():
    prob += (cnt/10000000)*((cnt-1)/9999999)
print(prob)
```

Chapter 31

What Comes After 840? The Answer May Surprise You

If N points are generated at random places on the perimeter of a circle, what is the probability that you can pick a diameter such that all of those points are on only one side of the newly halved circle?

Solution

Each of the N points determines a diameter of the circle. The probability of all the other $N-1$ points falling on one side of diameter determined by the first point is given by $\frac{1}{2^{N-1}}$. Therefore the probability of picking a diameter such that all of those points are on one side of the newly halved circle is $\sum_{i=1}^{N} \frac{1}{2^{N-1}} = \frac{N}{2^{N-1}}$.

Computational solution

```
from math import pi
from random import uniform
from collections import defaultdict

def total_angle(pts, n):
    rl = pts[n:] + pts[:n]
    nl = [d - pts[n] if d - pts[n] >= 0 else 2*pi + d - pts[n] for d in rl]
    angles = [x - nl[i - 1] for i, x in enumerate(nl)][1:]
    return sum(angles)

runs = 100000
N = 10
cnt_suc = defaultdict(int)
```

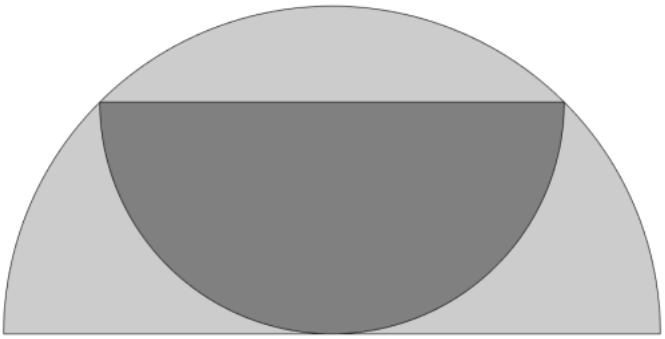
```
for n in range(3, N):
    for _ in range(runs):
        pts = [uniform(0, 2*pi) for _ in range(n)]
        pts.sort()
        min_angle = min([total_angle(pts, r) for r in range(n)])
        if min_angle <= pi:
            cnt_suc[n] += 1
    print("%d random points, Estimated probability %f, \
        Theoretical probability %f" % (n, cnt_suc[n]/runs, n/2**(n-1)))
```

Chapter 32

How Low Can You Roll?

Riddler Express

Inspired by Catriona Shearer (if you don't know who she is, seriously, check out her puzzles), this week's Riddler Express is a geometric conundrum:



Upside-down semicircle inscribed inside a right side-up semicircle. The smaller semicircle is dark gray, while the larger one is light gray. The picture above shows two semicircles. The lighter region (inside the larger semicircle but outside the smaller one) has an area of 7. What's the area of the darker region?

Solution

It is easy to see that the radius of the larger semicircle is $\sqrt{2}$ times the radius of the smaller semi circle. We have

$$\frac{\pi(\sqrt{2}r)^2}{2} - \frac{\pi r^2}{2} = 7 \implies \frac{\pi r^2}{2} = 7$$

where r is the radius of the smaller semi circle.

Therefore, the area of the darker region which is $\frac{\pi r^2}{2} = \mathbf{7}$.

Riddler Classic

You are given a fair, unweighted 10-sided die with sides labeled 0 to 9 and a sheet of paper to record your score. (If the very notion of a fair 10-sided die bothers you, and you need to know what sort of three-dimensional solid it is, then forget it — you have a random number generator that gives you an integer value from 0 to 9 with equal probability. Your loss — the die was a collector's item.)

To start the game, you roll the die. Your current "score" is the number shown, divided by 10. For example, if you were to roll a 7, then your score would be 0.7. Then, you keep rolling the die over and over again. Each time you roll, if the digit shown by the die is less than or equal to the last digit of your score, then that roll becomes the new last digit of your score. Otherwise you just go ahead and roll again. The game ends when you roll a zero.

For example, suppose you roll the following: 6, 2, 5, 1, 8, 1, 0. After your first roll, your score would be 0.6, After the second, it's 0.62. You ignore the third roll, since 5 is greater than the current last digit, 2. After the fourth roll, your score is 0.621. You ignore the fifth roll, since 8 is greater than the current last digit, 1. After the sixth roll, your score is 0.6211. And after the seventh roll, the game is over — 0.6211 is your final score.

What will be your average final score in this game?

Solution

```
from random import choice
```

```
def roll():
    return choice(range(10))

runs = 1000000
total_score = 0
for _ in range(runs):
    rolls = []
    while(True):
        r = roll()
        if r == 0:
            break
        elif not rolls:
            rolls.append(r)
        else:
            if r <= rolls[-1]:
                rolls.append(r)
    score = sum([r*10**(-i-1) for i, r in enumerate(rolls)])
    total_score += score
print(total_score/runs)
```

The average final score in the game is **0.47360**.

Chapter 33

How Many Ways Can You Raid The Candy Shop?

Riddler Express

Suppose I asked you to generate the biggest number you could using exactly three nines. Specifically, you can add, subtract, multiply, divide, exponentiate or write them side-by-side. Given this challenge, 9×9×9 is a pretty good start — it equals 729. Better yet is just writing the nines side-by-side, giving you 999. The biggest number is 9^{9^9}, which equals 9387420489. If you were to write one digit of that number every second, it would take you more than a decade to write the whole thing.

Now let's up the challenge: What's the biggest number you can generate using exactly four threes?

Solution

The largest number that can be created by four threes is $3^{3^{33}}$.

Riddler Classic

Now that Halloween has come and gone, your chances of getting free candy have similarly disappeared. Desperate for sugar, you wander into the candy store, where three kinds of candy are being sold: Almond Soys (yummy, sounds vegan!), Butterflingers and Candy Kernels.

You'd like to buy at least one candy and at most 100, but you don't care precisely how many you get of each or how many you get overall. So you might buy one of each, or you might buy 30 Almond Soys, six Butterflingers and 64 Candy Kernels. As long as you have somewhere between one and 100 candies, you'll leave the store happy.

But as a member of Riddler Nation, you can't help but wonder: How many distinct ways are there for you to make your candy purchase?

Solution

```
from itertools import product

cnt = 0
for a,b,c in product(range(101), range(101), range(101)):
    if a + b + c >= 1 and a+b+c <= 100:
        cnt += 1
print(cnt)
```

The number of distinct ways to make the candy purchase is **176850**.

Chapter 34

Who Wants To Be A Riddler Millionaire?

Riddler Express

You've made it to the $1 million question, but it's a tough one. Out of the four choices, A, B, C and D, you're 70 percent sure the answer is B, and none of the remaining choices looks more plausible than another. You decide to use your final lifeline, the $50:50$, which leaves you with two possible answers, one of them correct. Lo and behold, B remains an option! How confident are you now that B is the correct answer?

Solution

Let B be the event that B is the correct answer. Let L be the event that the two options (one of which is the correct answer) from lifeline contains B.

We have $\mathbb{P}[B] = .7$ and we are interested in $\mathbb{P}[B|L]$.

From **Bayes Theorem**, we have

$$\mathbb{P}[B|L] = \frac{\mathbb{P}[L|B]\mathbb{P}[B]}{\mathbb{P}[L|B]\mathbb{P}[B] + \mathbb{P}[L|B']\mathbb{P}[B']} = \frac{1 \cdot 0.7}{1 \cdot 0.7 + \frac{1}{3} \cdot 0.3} = \frac{7}{8} = 0.875$$

Therefore the probability that B is the correct answer after the lifeline is 0.875.

Riddler Classic

The classic birthday problem asks about how many people need to be in a room together before you have better-than-even

odds that at least two of them have the same birthday. Ignoring leap years, the answer is, paradoxically, only 23 people — fewer than you might intuitively think.

But Joel noticed something interesting about a well-known group of 100 people: In the U.S. Senate, three senators happen to share the same birthday of October 20: Kamala Harris, Brian Schatz and Sheldon Whitehouse.

And so Joel has thrown a new wrinkle into the classic birthday problem. How many people do you need to have better-than-even odds that at least three of them have the same birthday? (Again, ignore leap years.)

Chapter 35

Can You Track The Delirious Ducks?

After a long night of frivolous quackery, two delirious ducks are having a difficult time finding each other in their pond. The pond happens to contain a 3×3 grid of rocks.

Every minute, each duck randomly swims, independently of the other duck, from one rock to a neighboring rock in the 3×3 grid — up, down, left or right, but not diagonally. So if a duck is at the middle rock, it will next swim to one of the four side rocks with probability 1/4. From a side rock, it will swim to one of the two adjacent corner rocks or back to the middle rock, each with probability 1/3. And from a corner rock, it will swim to one of the two adjacent side rocks with probability 1/2.

If the ducks both start at the middle rock, then on average, how long will it take until they're at the same rock again? (Of course, there's a 1/4 chance that they'll swim in the same direction after the first minute, in which case it would only take one minute for them to be at the same rock again. But it could take much longer, if they happen to keep missing each other.)

Extra credit: What if there are three or more ducks? If they all start in the middle rock, on average, how long will it take until they are all at the same rock again?

Solution

```
from random import randrange
from itertools import combinations


grid_size = 3
```

```
def nbrs(pos, grid_size=3):
    i, j = pos
    nbrs = []
    for d in [-1, 1]:
        if (i+d) >= 0 and (i+d) < grid_size:
            nbrs.append((i+d, j))
        if (j+d) >= 0 and (j+d) < grid_size:
            nbrs.append((i, j+d))
    return nbrs


neighbours = {(i, j): nbrs((i, j)) for i in range(grid_size)
              for j in range(grid_size)}


def avgTimeToMeet(ducks_start, runs=100000):
    total_t = 0
    for _ in range(runs):
        ducks = ducks_start
        t = 0
        while (True):
            ducks_t = []
            for duck in ducks:
                duck_nbrs = neighbours[duck]
                duck = duck_nbrs[randrange(len(duck_nbrs))]
                ducks_t.append(duck)
            ducks = ducks_t
            t += 1
            if all([d1 == d2 for d1, d2 in combinations(ducks, 2)]):
                break
        total_t += t
    return total_t/runs


for ducks in [[(1, 1)]*2, [(1, 1)]*3]:
    print("Average time for %d ducks to meet: %f" % (len(ducks),  avgTimeToMeet(ducks)))
```

Chapter 36

Can You Find A Number Worth Its Weight In Letters?

In Jewish study, "Gematria" is an alphanumeric code where words are assigned numerical values based on their letters. We can do the same in English, assigning 1 to the letter A, 2 to the letter B, and so on, up to 26 for the letter Z. The value of a word is then the sum of the values of its letters. For example, RIDDLER has an alphanumeric value of 70, since R + I + D + D + L + E + R becomes 18 + 9 + 4 + 4 + 12 + 5 + 18 = 70.

But what about the values of different numbers themselves, spelled out as words? The number 1 (ONE) has an alphanumeric value of 15 + 14 + 5 = 34, and 2 (TWO) has an alphanumeric value of 20 + 23 + 15 = 58. Both of these values are bigger than the numbers themselves.

Meanwhile, if we look at larger numbers, 1,417 (ONE THOUSAND FOUR HUNDRED SEVENTEEN) has an alphanumeric value of 379, while 3,140,275 (THREE MILLION ONE HUNDRED FORTY THOUSAND TWO HUNDRED SEVENTY FIVE) has an alphanumeric value of 718. These values are much smaller than the numbers themselves.

If we consider all the whole numbers that are less than their alphanumeric value, what is the largest of these numbers?

Solution

```
NUM_TO_NAME = {
0:"ZERO",
1:"ONE",
2:"TWO",
3:"THREE",
```

```
4:"FOUR",
5:"FIVE",
6:"SIX",
7:"SEVEN",
8:"EIGHT",
9:"NINE",
10:"TEN",
11:"ELEVEN",
12:"TWELVE",
13:"THIRTEEN",
14:"FOURTEEN",
15:"FIFTEEN",
16:"SIXTEEN",
17:"SEVENTEEN",
18:"EIGHTEEN",
19:"NINETEEN",
20:"TWENTY",
30:"THIRTY",
40:"FORTY",
50:"FIFTY",
60:"SIXTY",
70:"SEVENTY",
80:"EIGHTY",
90:"NINTY",
100:"HUNDRED"
}

LETTER_TO_NUM = {c:(ord(c) - ord('A')+1) for c in ("ABCDEFGHIJKLMNOPRQSTUVWXYZ")}
LETTER_TO_NUM[' '] = 0

def value(name):
    print(name)
    return sum([LETTER_TO_NUM[c] for c in name])

def num_to_name(num):
    name = None
    if num >=1 and num <=10:
       name = NUM_TO_NAME[num]
    elif num <= 100 and num % 10 == 0:
       name = NUM_TO_NAME[num]
    elif num < 100:
        name = NUM_TO_NAME[num - (num % 10)] + " " + NUM_TO_NAME[num % 10]
    elif num < 1000:
        name = NUM_TO_NAME[num // 100 ] + " HUNDRED " + num_to_name(num % 100)
    return name


print(max(n for n in range(1,500) if value(num_to_name(n))>n))
```

Chapter 37

Can You Solve The Vexing Vexillology?

The New York Times recently launched some new word puzzles, one of which is Spelling Bee. In this game, seven letters are arranged in a honeycomb lattice, with one letter in the center. Here's the lattice from December 24, 2019:

Spelling Bee screenshot, with the required letter G, and the additional letters L, A, P, X, M and E. The goal is to identify as many words that meet the following criteria:

The word must be at least four letters long. The word must include the central letter. The word cannot include any letter beyond the seven given letters. Note that letters can be repeated. For example, the words GAME and AMALGAM are both acceptable words. Four-letter words are worth 1 point each, while five-letter words are worth 5 points, six-letter words are worth 6 points, seven-letter words are worth 7 points, etc. Words that use all of the seven letters in the honeycomb are known as "pangrams" and earn 7 bonus points (in addition to the points for the length of the word). So in the above example, MEGAPLEX is worth 15 points.

Which seven-letter honeycomb results in the highest possible game score? To be a valid choice of seven letters, no letter can be repeated, it must not contain the letter S (that would be too easy) and there must be at least one pangram.

For consistency, please use this word list to check your game score.2

Solution

```
import datetime
from collections import defaultdict
from itertools import combinations
import urllib.request

# Load all dictionary words from file
DICT_URL = "https://norvig.com/ngrams/enable1.txt"


def load_dictionary(url):
    response = urllib.request.urlopen(url)
    return [l.decode('utf-8').strip() for l in response.readlines()]


ALL_WORDS = load_dictionary(DICT_URL)

print(ALL_WORDS[0:10])

"""
Calculate word score based on the rules below
Four-letter words are worth 1 point each,
while five-letter words are worth 5 points,
six-letter words are worth 6 points,
seven-letter words are worth 7 points, etc.
Words that use all of the seven letters in the honeycomb are known as "pangrams" and earn 7
(in addition to the points for the length of the word)
"""


def word_score(word):
    bonus = 7 if len(set(word)) == 7 else 0
    return 1 if len(word) == 4 else len(word) + bonus


"""
Valid HoneyCombPuzzle words as per the rules below
The word must be at least four letters long.
The word must include the central letter.
The word cannot include any letter beyond the seven given letters.
"""

HCP_WORDS = [w for w in ALL_WORDS if len(w) >= 4 and ("s" not in w)
             and len(set(w)) <= 7]

# Word scores for all HCP WORD
HCP_WORD_SCORES = {w: word_score(w) for w in HCP_WORDS}

# The set of HPC Words for a set of letters
HCP_WORDSETS = defaultdict(list)
```

```
for w in HCP_WORDS:
    HCP_WORDSETS[frozenset(w)].append(w)


"""
Algorithm for calculating the score obtainable in a HoneyCombPuzzle
1. Find all subsets using the 6 letters surrounding the central letter (this r
2. Add the central letter to each of the subsets (as central character should
3. For each subset, identify the list of HPC words that can be formed using le
4. Taking the union of all the words that can be formed by each subset, gives
5. Sum the scores of all the HPC words for all the subsets to get the puzzle s
"""


def letter_subsets(hc):
    center, surrounding = hc
    subsets = []
    for r in range(1, 7):
        for comb_r in combinations(surrounding, r):
            subset = frozenset(list(comb_r)+[center])
            subsets.append(subset)
    return subsets


def score(hc):
    return sum([HCP_WORD_SCORES[w] for subset in letter_subsets(hc)
                for w in HCP_WORDSETS[subset]])


def solution(hc):
    return [w for subset in letter_subsets(hc)
            for w in HCP_WORDSETS[subset]]


def print_solution(hc):
    print("HoneyCombPuzzle:", hc)
    words = solution(hc)
    print("Number of words:", len(words))
    for w in words:
        print(w, HCP_WORD_SCORES[w])
    sc = score(hc)
    print("Total score:", sc)

    """
    Which seven-letter honeycomb results in the highest possible game score?
    To be a valid choice of seven letters, no letter can be repeated, it must
    and there must be at least one pangram.

    Algorithm for identifying all valid sets of seven letter honeycombs
    Out of all valid HPC words, identify words which are formed with seven di
    From each word identify the set of 7 distinct letters
```

```
    Each set of 7 distinct letters gives rise to 7 distinct honeycombs
    Identify the honeycomb with the largest score
    """

# Words that use all of the seven letters in the honeycomb are known as "pangrams"


def best():
    def valid_honeycombs():
        pangram_sets = [set(w) for w in HCP_WORDS if len(set(w)) == 7]
        return [(c, ls.difference(c)) for ls in pangram_sets for c in ls]
    return max([(score(hc), hc) for hc in valid_honeycombs()])

if __name__ == "__main__":
    print_solution(('g', 'ameplx'))
    print("Start:", datetime.datetime.now())
    print("The Best HoneyCombPuzzle Score: %d, Puzzle: %s" % best())
    print("End:", datetime.datetime.now())
```

Chapter 38

Can You Find A Matching Pair Of Socks?

I have 10 pairs of socks in a drawer. Each pair is distinct from another and consists of two matching socks. Alas, I'm negligent when it comes to folding my laundry, and so the socks are not folded into pairs. This morning, fumbling around in the dark, I pull the socks out of the drawer, randomly and one at a time, until I have a matching pair of socks among the ones I've removed from the drawer.

On average, how many socks will I pull out of the drawer in order to get my first matching pair?

(Note: This is different from asking how many socks I must pull out of the drawer to guarantee that I have a matching pair. The answer to that question, by the pigeonhole principle, is 11 socks. This question is instead asking about the average.)

Extra credit: Instead of 10 pairs of socks, what if I have some large number N pairs of socks?

Solution

```
from random import sample


num_sock_pairs = 30
runs = 10000
socks = list(range(num_sock_pairs))*2

total_num_socks = 0
for _ in range(runs):
    samp = sample(socks, num_sock_pairs+1)
    start_pos, end_pos = {}, {}
```

```
    for i, sock in enumerate(samp):
        if sock not in start_pos.keys():
            start_pos[sock] = i
        else:
            end_pos[sock] = i
        if sock in end_pos.keys():
            break
    total_num_socks += end_pos[sock]+1
print("Average number of socks ", total_num_socks/runs)
```

Chapter 39

Can You Get A Haircut Already?

At your local barbershop, there are always four barbers working simultaneously. Each haircut takes exactly 15 minutes, and there's almost always one or more customers waiting their turn on a first-come, first-served basis.

Being a regular, you prefer to get your hair cut by the owner, Tiffany. If one of the other three chairs opens up, and it's your turn, you'll say, "No thanks, I'm waiting for Tiffany." The person behind you in line will then be offered the open chair, and you'll remain at the front of the line until Tiffany is available.

Unfortunately, you're not alone in requesting Tiffany — a quarter of the other customers will hold out for Tiffany, while no one will hold out for any of the other barbers.

One Friday morning, you arrive at the barber shop to see that all four barbers are cutting hair, and there is one customer waiting. You have no idea how far along any of the barbers is in their haircuts, and you don't know whether or not the customer in line will hold out for Tiffany.

What is the expected wait time for getting a haircut from Tiffany?

Solution

Let T be the random variable of the remaining time for haircut being done by Tiffany.

Let B_i be the random variable of the remaining time for haircut being done by Barber i, for $i = 1, 2, 3$.

Let C_T be the indicator random variable which takes the value 1 if the waiting customer holds out for Tiffany and 0 otherwise.

We have,

$$\begin{aligned} T &\sim \mathcal{U}(0,15) \\ B_i &\sim \mathcal{U}(0,15), i = 1,2,3 \\ \mathbb{P}[C_T = 1] = \frac{1}{4} \quad &, \quad \mathbb{P}[C_T = 0] = \frac{3}{4} \end{aligned}$$

Let Y be the random variable of the waiting time for Tiffany when there is one customer waiting.

We have,

$$Y = \begin{cases} 15 + T & C_T = 1 \\ 15 + T & C_T = 0, T < \min(B_1, B_2, B_3) \\ T & C_T = 0, T \geq \min(B_1, B_2, B_3) \end{cases} \tag{39.1}$$

The expected wait time for getting a haircut from Tiffany is given by

$$\mathbb{E}[Y] = \mathbb{P}[C_T = 1]\cdot\mathbb{E}[Y|C_T = 1]+\mathbb{P}[C_T = 0]\cdot\mathbb{E}[Y|C_T = 0] = \frac{1}{4}(15+\mathbb{E}[T])+\frac{3}{4}\mathbb{E}[Y|C_T = 0] \tag{39.2}$$

As $T \sim \mathcal{U}(0,15)$,

$$\mathbb{E}[T] = \frac{0+15}{2} = 7.5 \tag{39.3}$$

To calculate $\mathbb{E}[Y|C_T = 0]$, we need the distribution function of $B = \min(B_1, B_2, B_3)$.

By definition,

$$F_B(x) = \mathbb{P}[B \leq x] = 1 - \mathbb{P}[B > x] = 1 - \mathbb{P}[\min(B_1, B_2, B_3) > x] \tag{39.4}$$

From 39.1 and 39.4 we get,

$$\mathbb{E}[Y|T, C_T = 0] = (15+T)(1-F_B(T))+T\cdot F_B(T) = 15(1-F_B(T))+T \tag{39.5}$$

Using Symmetry to calculate $\mathbb{E}[Y|C_T = 0]$

An elegant way of calculating $\mathbb{E}[Y|C_T = 0]$ from 39.5 without deriving F_B explicitly is as follows,

$$\mathbb{E}[Y|C_T = 0] = \mathbb{E}[\mathbb{E}[Y|T, C_T = 0]] = \mathbb{E}[15(1-F_B(T))]+\mathbb{E}[T] = 15\cdot\frac{1}{4}+7.5 = 11.25 \tag{39.6}$$

The quantity $\mathbb{E}[1 - F_B(T)]$ can be interpreted as the average probability of T being the minimum of T, B_1, B_2 and B_3, which by **symmetry** is $\frac{1}{4}$.

Using the distribution function F_B to calculate $\mathbb{E}[Y|C_T = 0]$

Derivation of F_B

To derive F_B we need to first calculate $\mathbb{P}[\min(B_1, B_2, B_3) > x]$.

Of course, $min(B_1, B_2, B_3) > x$ exactly when $B_i > x$ for $i = 1, 2$ and 3.

Since these variables are i.i.d., we have $\mathbb{P}[\min(B_1, B_2, B_3) > x] = \mathbb{P}[B_1 > x] \cdot \mathbb{P}[B_2 > x] \cdot \mathbb{P}[B_3 > x] = \mathbb{P}(B_1 > x)^3$.

As the B_i are uniformly distributed on $(0, 15)$, this yields

$$F_B(x) = \begin{cases} 1 - \left(\frac{15-x}{15}\right)^3 & x \in (0, 15) \\ 0 & x < 0 \\ 1 & y > 15 \end{cases} \tag{39.7}$$

From 39.7 and 39.5 we get,

$$\mathbb{E}[Y|T, C_T = 0] = 15\left(\frac{15 - T}{15}\right)^3 + T \tag{39.8}$$

From 39.8 we get,

$$\mathbb{E}[Y|C_T = 0] = \int_0^{15} f_T(t)\left(15\left(\frac{15 - t}{15}\right)^3 + t\right)dt = \int_0^{15} \frac{1}{15}\left(15\left(\frac{15 - t}{15}\right)^3 + t\right) \tag{39.9}$$

where $f_T(t) = \frac{1}{15}$ is the density function of T.

Calculating $\mathbb{E}[Y]$

From 39.2, 39.3 and 39.6 (or 39.9) and we get,

$$\mathbb{E}[Y] = \frac{1}{4}(15+\mathbb{E}[T])+\frac{3}{4}\mathbb{E}[Y|C_T = 0] = \frac{1}{4}\cdot 22.5+\frac{3}{4}\cdot 11.25 = 14.0625$$

Therefore the expected wait time for getting a haircut from Tiffany is **14.0625** mins.

Computational Solution

```
from random import random, uniform
```

```
runs = 1000000
total_wait_time = 0
for _ in range(runs):
    t, b1, b2, b3 = [uniform(0, 15) for _ in range(4)]
    if random() <= 0.25:
        wait_time = 15 + t
    else:
        if t < min(b1, b2, b3):
            wait_time = 15 + t
        else:
            wait_time = t
    total_wait_time += wait_time
print("Average wait time: ", total_wait_time/runs)
```

How Good Are You At Guess Who?

Riddler Express

A local cafe has board games on a shelf, designed to keep kids (and some adults) entertained while they wait on their food. One of the games is a tic-tac-toe board, which comes with nine pieces that you and your opponent can place: five Xs and four Os.

When I took my two-year-old with me, he wasn't particularly interested in the game itself, but rather in the placement of the pieces.

If he randomly places all nine pieces in the nine slots on the tic-tac-toe board (with one piece in each slot), what's the probability that X wins? That is, what's the probability that there will be at least one occurrence of three Xs in a row at the same time there are no occurrences of three Os in a row?

Solution

```
import altair as alt
from itertools import combinations
alt.renderers.enable('default')

def win(p, piece):
    rows_cols_diags, diag1, diag2 = [], [], []
    for i in range(3):
        rows_cols_diags.append([e[1] for e in filter(lambda x: x[0][0]==i, p)
        rows_cols_diags.append([e[1] for e in filter(lambda x: x[0][1]==i, p)
        diag1.append(next(filter(lambda x: x[0][0]==i and x[0][1]==i, p))[1])
        diag2.append(next(filter(lambda x: x[0][0]==i and x[0][1]==2-i, p))[1
    rows_cols_diags.extend([diag1, diag2])
    return any(map(lambda x: x == [piece]*3, rows_cols_diags))
```

```
num_x = 5
piece_x, piece_o = "X", "O"
squares = set([(i,j) for i in range(3) for j in range(3)])
positions = []
for i, comb in enumerate(combinations(squares, num_x)):
    p = [(c, piece_x) for c in comb] + [(c, piece_o) for c in squares - set(comb)]
    if win(p, piece_x) and not win(p, piece_o):
        positions.append((p, "win"))
    else:
        positions.append((p, "loss"))
print("Number of winning positions:", len(list(filter(lambda x: x[1]=="win", positions))))

def pos_viz(pos):
    position, win_loss = pos
    if win_loss == "win":
        pos = alt.Data(values=[{'x':x, 'y':y, 'p':p, 'col':"green" if p==piece_x else "red"
                for (x,y),p in position])
    else:
        pos = alt.Data(values=[{'x':x, 'y':y, 'p':p, 'col':"grey"} for (x,y),p in position])
    grid = alt.Chart(pos).mark_rect(stroke="white", strokeWidth=1, opacity=0.6).encode(
        x = alt.X('x:O',axis=None),
        y = alt.Y('y:O',axis=None),
        color = alt.Color('col:N', scale=None),
    )
    pieces = grid.mark_text().encode(
        text =  'p:N',
        color = alt.value('black')
    )
    return grid + pieces

group = lambda flat, size: [flat[i:i+size] for i in range(0, len(flat), size)]

chart = alt.vconcat()
for row in group(sorted(positions, key = lambda k:k[1], reverse=True), 9):
    chart_row = alt.hconcat()
    for pos in row:
        chart_row |= pos_viz(pos)
    chart &= chart_row

chart
```

The probability that 'X' wins is 0.4924.
Here are all the winning positions:

Riddler Classic

In the game of "Guess Who," each player first randomly (and independently of their opponent) selects one of N character

tiles. While it's unlikely, both players can choose the same character. Each of the N characters is distinct in appearance — for example, characters have different skin tones, hair color, hair length and accessories like hats or glasses.

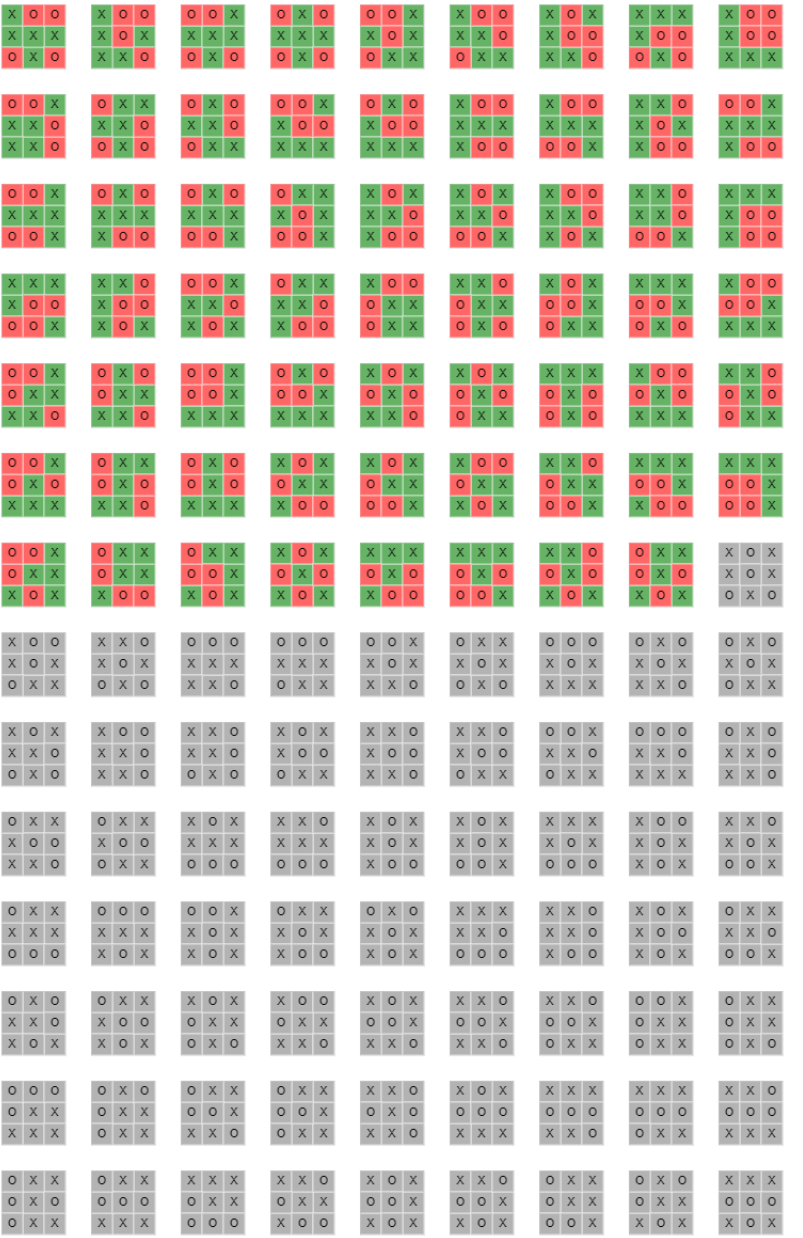
Each player also has access to a board with images of all N characters. The players alternate taking turns, and during each turn a player has two options:

Make a specific guess as to their opponent's selected character. If correct, the player who made the guess immediately wins. Otherwise, that player immediately loses. Ask a yes-or-no question about their opponent's chosen character, in order to eliminate some of the candidates. Importantly, if only one possible character is left after the question, the player must still wait until their next turn to officially guess that character. Assume both players are highly skilled at choosing yes-or-no questions, so that they can always craft a question to rule out any desired number of candidates. Also, both are playing to maximize their own probability of winning.

Let's keep things (relatively) simple, and suppose that $N = 4$. How likely is it that the player who goes first will win?

Extra credit: If N is instead 24 (the number of characters in the original "Guess Who" game), now how likely is it that the player who goes first will win?

Extra extra credit: If N is instead 14, now how likely is it that the player who goes first will win?



Chapter 41

Can You Time The Stoplight Just Right?

You and I find ourselves indoors one rainy afternoon, with nothing but some loose change in the couch cushions to entertain us. We decide that we'll take turns flipping a coin, and that the winner will be whoever flips 10 heads first. The winner gets to keep all the change in the couch! Predictably, an enormous argument erupts: We both want to be the one to go first.

What is the first flipper's advantage? In other words, what percentage of the time does the first flipper win this game?

Solution

```
from random import random

def flip():
    return 1 if random() > 0.5 else 0

runs = 100000

nfw = 0
for _ in range(runs):
    nah, nbh = 0, 0
    while(True):
        if flip():
            nah += 1
        if nah == 10:
            nfw += 1
            break
        if flip():
            nbh += 1
        if nbh == 10:
```

```
            break
print("Percentage of first person wins: ", nfw/runs)
```

Chapter 42

Same number on all the dies

You start with a fair 6-sided die and roll it six times, recording the results of each roll. You then write these numbers on the six faces of another, unlabeled fair die. For example, if your six rolls were 3, 5, 3, 6, 1 and 2, then your second die wouldn't have a 4 on it; instead, it would have two 3s.

Next, you roll this second die six times. You take those six numbers and write them on the faces of yet another fair die, and you continue this process of generating a new die from the previous one.

Eventually, you'll have a die with the same number on all six faces. What is the average number of rolls it will take to reach this state?

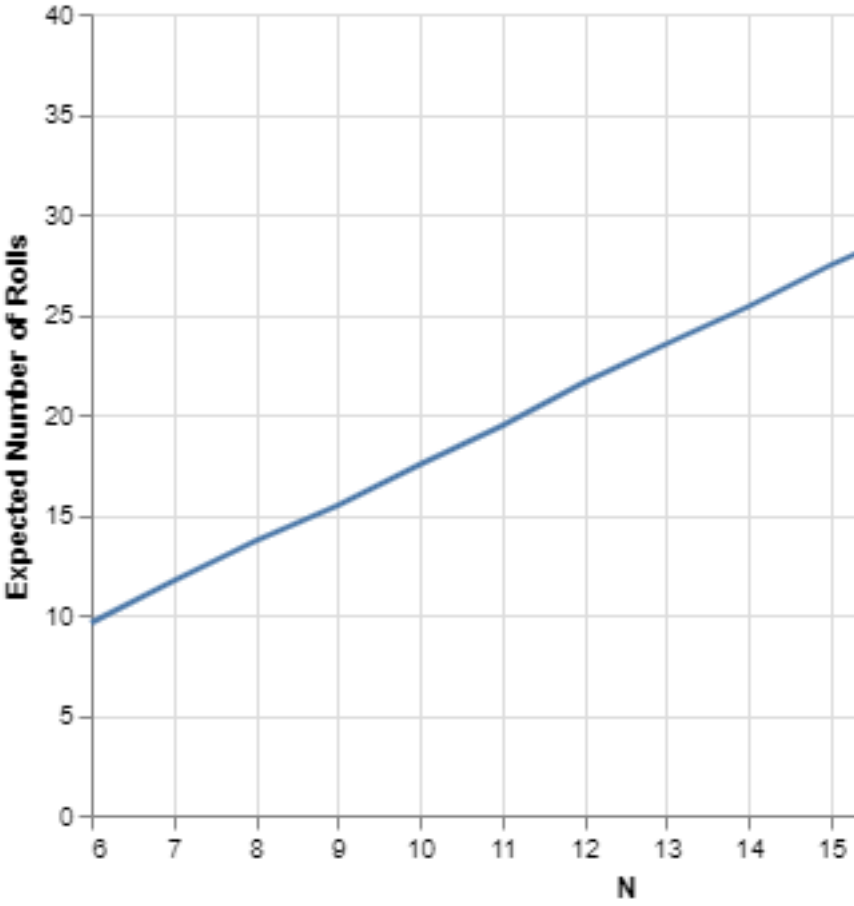
Extra credit: Instead of a standard 6-sided die, suppose you have an N-sided die, whose sides are numbered from 1 to N. What is the average number of rolls it would take until all N sides show the same number?

Solution

```
from collections import Counter
from random import choices
import altair as alt
import pandas as pd

alt.renderers.enable('default')

N = 20
runs = 10000
exp_number = []
for n in range(6, N):
    total_succ_cnt = 0
```

```
    for _ in range(runs):
        num_count = Counter(range(n))
        succ_cnt = 0
        while(True):
            num_count = Counter(choices(list(num_count.keys()),
                                        weights=list(num_count.values()),
                                        k=n))
            succ_cnt += 1
            if n in num_count.values():
                break
        total_succ_cnt += succ_cnt
    exp_number.append({'N': n, 'Expected Number of Rolls': total_succ_cnt/runs})


alt.Chart(pd.DataFrame(exp_number)).mark_line().encode(
                    x='N:Q',
                    y='Expected Number of Rolls:Q'
                )
```

Here is the graph of number of faces on the dice Vs Expected number of rolls to get same number on all dies dies



Chapter 43

Can You Tell When The Snow Started?

One morning, it starts snowing. The snow falls at a constant rate, and it continues the rest of the day.

At noon, a snowplow begins to clear the road. The more snow there is on the ground, the slower the plow moves. In fact, the plow's speed is inversely proportional to the depth of the snow — if you were to double the amount of snow on the ground, the plow would move half as fast.

In its first hour on the road, the plow travels 2 miles. In the second hour, the plow travels only 1 mile.

When did it start snowing?

Solution

We have

$$\frac{dx}{dt} = \frac{k}{rt} \tag{43.1}$$

where k is a constant and r is the constant rate of snow fall.

Integrating the above equation we have the following equations for the distances covered in the first and second hours by the snowplow

$$2 = \int_{t_s}^{t_s+1} \frac{k}{rt} dt = \frac{k}{r} ln\left(\frac{t_s+1}{t_s}\right) \tag{43.2}$$

$$1 = \int_{t_s+1}^{t_s+2} \frac{k}{rt} dt = \frac{k}{r} ln\left(\frac{t_s+2}{t_s+1}\right) \tag{43.3}$$

where t_s is the time in hrs the snowplow starts after the snow fall begins.

From the above we have,

$$\left(\frac{t_1+2}{t_s+1}\right)^2 = \frac{t_s+1}{t_s} \implies t_s = 0 \vee t_s^2 + t_s - 1 = 0 \qquad (43.4)$$

Ignoring the trivial and the implausible solutions, we have,

$$t_s = \frac{-1+\sqrt{5}}{2} = \frac{1}{\phi} \qquad (43.5)$$

As the snowplow starts at noon, the snow fall must have started at $12:00 - t_s$hrs i.e. at 11:23 am.

Chapter 44

Can you flip the magic coin?

I have a coin with a sun on the front and the mooon on the back. I claim that on most days it's a fair coin, with a 50 percent chance of landing on either the sun or the moon. But once a year, on the summer solstice, the coin absorbs the sun's rays and exhibits a strange power:it always comes up the opposite side as the previous flip. Of course, you are skeptical of my claim. You figure that there's a 1 percent chance that the coin is magical and a 99 percent chance that it's just an ordinary fair coin. You then ask me to 'prove' that the coin is magical by flipping it some number of times. How many successfully alternating coin flips it will take for you to think there's a 99 percent chance the coin is magical (or, more likely, that I've rigged it in some ways so it always alternates)?

Solution

This is a classic application of Bayes Theorem. Let M be the even that the coin is magical. The probability that the coin is magical is given by $\mathbb{P}[M] = 0.01$. The probability that the coin is not magical is given by $\mathbb{P}[M^c] = 0.99$. Let n be the number of alternating flips required for you to think that the coin is magical with a probability of 99 percent. Let A_n be the event of getting n alternating flips on tossing the coin n times. This means that the conditional probability of the coin being considered by you as magical given n alternating flips is given $\mathbb{P}[M|A_n] \geq 0.99$. For a magical coin the probability of alternate flips is always 1, so we have $\mathbb{P}[A_n|M] = 1$. For a normal coin, the probability of alternate flips is given by $\mathbb{P}[A_n|M^c] = \frac{2}{2^n}$

because the n alternate flips can start with a head or a tail. Therefore, from Bayes theorem, we have

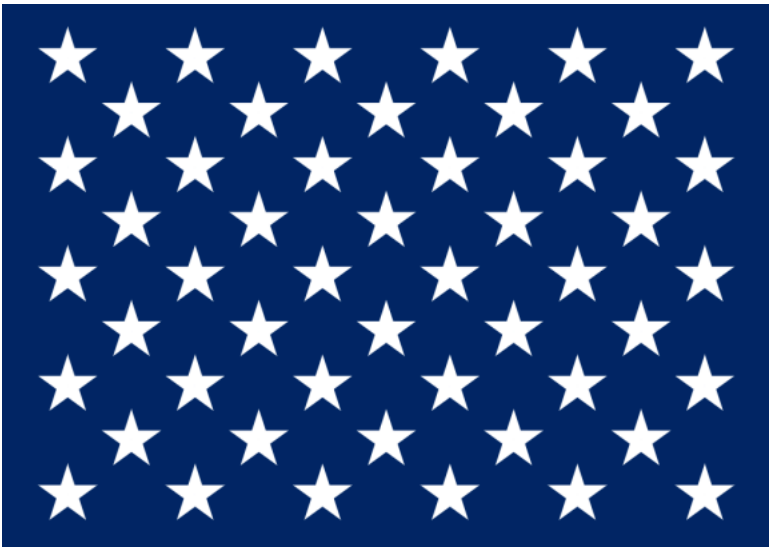
$$\begin{aligned} 0.99 \leq \mathbb{P}[M|A_n] &= \frac{\mathbb{P}[A_n|M] \cdot \mathbb{P}[M]}{\mathbb{P}[A_n|M] \cdot \mathbb{P}[M] + \mathbb{P}[A_n|M^c] \cdot \mathbb{P}[M^c]} \\ &= \frac{1 \cdot 0.01}{1 \cdot 0.01 + \frac{2}{2^n} \cdot 0.99} \end{aligned}$$

We see that we need $n \geq 15$ alternating flips so that we can be 99 percent confident that the coin is magical.

Chapter 45

Centered Pentagonal Number

The 50 stars on the American flag are arranged in such a way that they form two rectangles. The larger rectangle is 5 stars wide, 6 stars long; the smaller rectangle is embedded inside the larger and is 4 stars wide, 5 stars long. This square-like pattern of stars is possible because the number of states (50) is twice a square number (25).



Now that the House of Representatives has passed legislation that would make the District of Columbia the fifty-first US state — and renamed Washington, Douglass Commonwealth, in honor of Frederick Douglass — a natural question is how to aesthetically arrange 51 stars on the flag.

One pleasing design has a star in the middle, surrounded by concentric pentagons of increasing side length, as shown below. The innermost pentagon has five stars, and subsequent pentagons are made up of 10, 15 and 20 stars. All told, that's 51 stars.

It just so happens that when N equals 50, N is twice a square and $N + 1$ is a centered pentagonal number. After 50, what is the next integer N with these properties?

Solution

We are looking for an N that satisifies the following:

$$N + 1 = \frac{5p^2 + 5p + 2}{2}$$
$$N = 2q^2$$

for some integers p and q.

From the above we have $5(2p+1)^2 - 5 = 16q^2$ or $x^2 - 80q^2 = 1$ where $x = 2p + 1$ and $q = 5y$.

The equation $x^2 - 80y^2 = 1$ is a **Pell's equation**.

If $(x, y) = (u, v)$ is a solution of the above equation, a whole family of solutions can be found by taking each side to the n^{th} power ,

$$x^2 - 80y^2 = (u^2 - 80v^2)^n = 1.$$

Factoring gives,

$$x + \sqrt{80}y = (u + \sqrt{80}v)^n \, x - \sqrt{80}y = (u - \sqrt{80}v)^n$$

which gives the whole family of solutions

$$x = \frac{(u + v\sqrt{80})^n + (u - v\sqrt{80})^n}{2} \, y = \frac{(u + v\sqrt{80})^n - (u - v\sqrt{80})^n}{2\sqrt{80}}$$

It is easy to see that $(u, v) = (9, 1)$ is a solution to $x^2 - 80y^2 = 1$.

For $n = 1$, we have $x = 9$, therefore $p = 4$ and $N = 50$. For $n = 2$, we have $x = 161$, therefore $p = 80$ and $N = 16200$.

Brute force computational solution

```
from math import sqrt

for p in range(1, 1000):
    t = (5*p**2 + 5*p)/4
    if sqrt(t).is_integer():
        print("N = ", int(2*t))
```

Chapter 46

Breaking a slide rule into four pieces

The Riddler Manufacturing Company makes all sorts of mathematical tools: compasses, protractors, slide rules — you name it!

Recently, there was an issue with the production of foot-long rulers. It seems that each ruler was accidentally sliced at three random points along the ruler, resulting in four pieces. Looking on the bright side, that means there are now four times as many rulers — they just happen to have different lengths.

On average, how long are the pieces that contain the 6-inch mark?

Solution

Order Statistics

Let $X_1, X_2, \cdots, X_n$ be a random sample of size n from a continuous distribution with distribution function $F(x)$. We order the random sample in increasing order and obtain $Y_1, Y_2, \cdots, Y_n$. In other words, we have:

Y_1= the smallest of $X_1, X_2, \cdots, X_n$ Y_2= the second smallest of $X_1, X_2, \cdots, X_n$

.

.

.

Y_n = the largest of $X_1, X_2, \cdots, X_n$

We set $Y_{min} = Y_1$ and $Y_{max} = Y_n$. The order statistic Y_i is called the i^{th} order statistic. Since we are working with a continuous distribution, we assume that the probability of two sample items being equal is zero. Thus we can assume that $Y_1 < Y_2 < \cdots < Y_n$. That is, the probability of a tie is zero among the order statistics.

The Distribution Functions of the Order Statistics

The distribution function of Y_i is an upper tail of a binomial distribution. If the event $Y_i \leq y$ occurs, then there are at least i many X_j in the sample that are less than or equal to y. Consider the event that $X \leq y$ as a success and $F(y) = P[X \leq y]$ as the probability of success. Then the drawing of each sample item becomes a Bernoulli trial (a success or a failure). We are interested in the probability of having at least i many successes. Thus the following is the distribution function of Y_i:

$$F_{Y_i}(y) = P[Y_i \leq y] = \sum_{k=i}^{n} \binom{n}{k} F(y)^k [1 - F(y)]^{n-k}$$

The Probability Density Functions of the Order Statistics

The probability density function of Y_i is given by:

$$f_{Y_i}(y) = \frac{n!}{(i-1)!(n-i)!} F(y)^{i-1} [1 - F(y)]^{n-i} f_X(y)$$

The details of the derivation can be found here - [Order Statistics](https://probabilityandstats.wordpress.com/2010/02/20/the-distributions-of-the-order-statistics/).

The Order Statistics of the Uniform Distribution

When $X_i \sim \mathcal{U}[0, 1]$, we have $F(y) = y$ and $f_X(y) = 1$, therefore

$$F_{Y_i}(y) = P[Y_i \leq y] = \sum_{k=i}^{n} \binom{n}{k} y^k (1 - y)^{n-k}$$

$$f_{Y_i}(y) = \frac{n!}{(i-1)!(n-i)!} y^{i-1} (1 - y)^{n-i}$$

$$\begin{aligned}\mathbb{E}[Y_i] &= \int_0^1 f_{Y_i}(y) y dy \\ &= \int_0^1 i\binom{n}{i} y^i (1-y)^{n-i} dy = \frac{i}{n+1}\end{aligned}$$

Putting it all together

Let LY_1, LY_2 and LY_n be the n places where the ruler has been sliced where L is the length of the ruler in inches, $X_i \sim \mathcal{U}[0,1]$ and Y_i are the order statistics of the Uniform distribution.

When the mark m lies between LY_k and LY_{k+1}, we have

$$\begin{aligned}\mathbb{P}[Y_k \leq a \leq Y_{k+1}] &= 1 - (\mathbb{P}[Y_{k+1} \leq a] + \mathbb{P}[Y_k \geq a]) \\ &= 1 - \left(F_{Y_{k+1}}(a) + 1 - F_{Y_k}(a)\right) \\ &= F_{Y_k}(a) - F_{Y_{k+1}}(a) \\ &= \binom{n}{k} a^k (1-a)^{n-k}\end{aligned}$$

$$\mathbb{E}[LY_{k+1} - LY_k | Y_k \leq a \leq Y_{k+1}] = \frac{m}{k+1} + \frac{L-m}{n-k+1}$$

where $a = \frac{m}{L}$.

The average length of the piece which has the m inch mark is given by

$$\begin{aligned}&(1 - F_{Y_1}(a))(m + \frac{L-m}{n+1}) + (F_{Y_1}(a) - F_{Y_2}(a))(\frac{m}{2} + \frac{L-m}{n}) + \\ &\qquad \cdots + F_{Y_n}(a)(\frac{m}{n+1} + L - m) \\ &= (1-a)^n (m + \frac{L-m}{n+1}) + \binom{n}{1} a(1-a)^{n-1}(\frac{m}{2} + \frac{L-m}{n}) + \\ &\qquad \cdots + a^n (\frac{m}{n+1} + L - m) \\ &= m \int_0^1 (ax + 1 - a)^n dx + (L-m) \int_0^1 (a + (1-a)x)^n dx \\ &= \frac{m}{a(n+1)} - \frac{m(1-a)^{n+1}}{a(n+1)} + \frac{m}{a(n+1)} - \frac{(L-m)a^{n+1}}{(1-a)(n+1)}\end{aligned}$$

$$= \frac{L}{n+1}\left(2 - (1-a)^{n+1} - a^{n+1}\right)$$

In the given problem, $L = 12$, $m = 6$, $n = 3$ and $a = \frac{6}{12} = \frac{1}{2}$, therefore the average length of the piece we are interested in is given by

$$\frac{12}{3+1}\left(2 - \frac{1}{2^{3+1}} - \frac{1}{2^{3+1}}\right) = 6 - \frac{3}{8} = 5.625$$

Computational solution

```
from random import uniform

l = 0
u = 12
m = 6
n = 4
runs = 500000

tl = 0
for _ in range(runs):
    endpoints = [l] + sorted([uniform(l, u) for _ in range(n-1)]) + [u]
    for i, ep in enumerate(endpoints[:-1]):
        if ep < m < endpoints[i+1]:
            tl += (endpoints[i+1] - ep)

print("Avg length of the piece that contains the mark:", tl/runs)
```

Chapter 47

Can You Reach The Summit First?

Riddler Express

Once a week, folks from Blacksburg, Greensboro, and Silver Spring get together for a game of pickup basketball. Every week, anywhere from one to five individuals will show up from each town, with each outcome equally likely.

Using all the players that show up, they want to create exactly two teams of equal size. Being a prideful bunch, everyone wears a jersey that matches the color mentioned in the name of their city. However, since it might create confusion to have one jersey playing for both sides, they agree that the residents of two towns will combine forces to play against the third town's residents.

What is the probability that, on any given week, it's possible to form two equal teams with everyone playing, where two towns are pitted against the third?

Extra credit: Suppose that, instead of anywhere from one to five individuals per town, anywhere from one to N individuals show up per town. Now what's the probability that there will be two equal teams?

Solution

Let b, g and s be the number of people from Blacksburg, Greensboro and Silver spring on a given week.

The total number of $3-tuples$ of the form (b, g, s) where $1 \leq b \leq N$,$1 \leq g \leq N$ and $1 \leq s \leq N$ is N^3.

When equal teams can be formed from b, g and s, we have $b + g = s$ or $b + s = g$ or $s + g = b$.

The equation $b + g = n$ has the following solutions:

$$1 + (n-1)2 + (n-2)...(n-1) + 1$$

Therefore, the number of solutions of the equation is $n - 1$.

The total number of solutions to the set of equations $b+g = s$ or $b + s = g$ or $s + g = b$ where $1 \leq b \leq N$,$1 \leq g \leq N$ and $1 \leq s \leq N$ is given by $3(1 + 2 + \cdots + N - 1) = 3N(N - 1)/2$.

The probability of forming two equal teams satisfying the given criteria is $\frac{3N(N-1)/2}{N^3}$.

Computational solution in Python

```
from itertools import product
n = 5

m = 0
for b,g,s in product(*[range(1, n+1)]*3):
    if b+g==s or b+s==g or s+g==b:
        m += 1
print(m, m/n**3)
```

Riddler Classic

For every mountain in the Tour de FiveThirtyEight, the first few riders to reach the summit are awarded points. The rider with the most such points at the end of the Tour is named "King of the Mountains" and gets to wear a special polka dot jersey.

At the moment, you are racing against three other riders up one of the mountains. The first rider over the top gets 5 points, the second rider gets 3, the third rider gets 2, and the fourth rider gets 1.

All four of you are of equal ability — that is, under normal circumstances, you all have an equal chance of reaching the summit first. But there's a catch — two of your competitors are on the same team. Teammates are able to work together, drafting and setting a tempo up the mountain. Whichever teammate happens to be slower on the climb will get a boost from

their faster teammate, and the two of them will both reach the summit at the faster teammate's time.

As a lone rider, the odds may be stacked against you. In your quest for the polka dot jersey, how many points can you expect to win on this mountain, on average?

Solution

```
from more_itertools import locate

cnt,tp = 0, 0
for p in distinct_permutations('ABCC'):
    cnt += 1
    pos_A = list(locate(p, lambda x: x == 'A'))[0]
    pos1_C, pos2_C = list(locate(p, lambda x: x == 'C'))
    if pos_A == 0:
        tp += 5
    if pos_A == 3:
        tp += 1
    if pos1_C < pos_A < pos2_C:
        if pos_A == 1:
            tp += 2
        if pos_A == 2:
            tp += 1
    else:
        if pos_A == 1:
            tp += 3
        if pos_A == 2:
            tp += 2

print(tp/cnt)
```

Chapter 48

Can You Cut The Square ... Into More Squares?

Robin of Foxley has entered the FiveThirtyEight archery tournament. Her aim is excellent (relatively speaking), as she is guaranteed to hit the circular target, which has no subdivisions — it's just one big circle. However, her arrows are equally likely to hit each location within the target.

Her true love, Marian, has issued a challenge. Robin must fire as many arrows as she can, such that each arrow is closer to the center of the target than the previous arrow. For example, if Robin fires three arrows, each closer to the center than the previous, but the fourth arrow is farther than the third, then she is done with the challenge and her score is four.

On average, what score can Robin expect to achieve in this archery challenge?

Extra credit: Marian now uses a target with 10 concentric circles, whose radii are $1, 2, 3$, etc., all the way up to 10 — the radius of the entire target. This time, Robin must fire as many arrows as she can, such that each arrow falls within a smaller concentric circle than the previous arrow. On average, what score can Robin expect to achieve in this version of the archery challenge?

Solution

```
from random import random

runs = 1000000

def avg_score_challenge1():
```

```
    total_cnt = 0
    for _ in range(runs):
        prev_pos = random()
        cnt = 1
        while(True):
            cur_pos=random()
            cnt += 1
            if cur_pos < prev_pos:
                prev_pos = cur_pos
            else:
                break
        total_cnt += cnt
    return total_cnt/runs

def avg_score_challenge2():
    n = 10
    interval_to_circle = [(i,i**2/n**2, (i+1)**2/n**2) for i in range(n)]
    def number_to_circle(p):
        for i,s,e in interval_to_circle:
            if  s < p and p < e:
                return i

    total_cnt = 0
    for _ in range(runs):
        prev_pos = number_to_circle(random())
        cnt = 1
        while(True):
            cur_pos = number_to_circle(random())
            cnt += 1
            if  cur_pos < prev_pos:
                prev_pos = cur_pos
            else:
                break
        total_cnt += cnt
    return total_cnt/runs

print(avg_score_challenge1(), exp(1))
print(avg_score_challenge2())
```

Chapter 49

Can You Hunt For The Mysterious Numbers?

Riddler Express

You're a contestant on the hit new game show, "You Bet Your Fife." On the show, a random real number (i.e., decimals are allowed) is chosen between 0 and 100. Your job is to guess a value that is less than this randomly chosen number. Your reward for winning is a novelty fife that is valued precisely at your guess. For example, if the number is 75 and you guess 5, you'd win a \$5 fife, but if you'd guessed 60, you'd win a \$60 fife. Meanwhile, a guess of 80 would win you nothing. What number should you guess to maximize the average value of your fifing winnings?

Solution

Let $X \sim \mathcal{U}[0, 100]$ be the random variable representing the random real number that is chosen. Let Y be the random variable representing the fifing winnings.

We have

$$Y = \begin{cases} g, & g < X \\ 0, & g \geq X \end{cases}$$

where g is the number you should guess.
Also, $\mathbb{P}[Y = g] = \mathbb{P}[X > g] = \frac{100-g}{100}$.
Therefore, $\mathbb{E}[Y] = g \cdot \mathbb{P}[Y = g] = \frac{g(100-g)}{100}$.

The expectation takes the maximum value when $g = 100 - g = 50$ and maximum value of the expectation is 25.

Riddler Classic

There are eight three-digit numbers — each belongs in a row of the table below, with one digit per cell. The products of the three digits of each number are shown in the rightmost column. Meanwhile, the products of the digits in the hundreds, tens, and ones places, respectively, are shown in the bottom row.

Table for eight three-digit numbers. The products of the three digits in each number are, respectively, 294, 216, 135, 98, 112, 84, 245 and 40. The product of the hundreds digits is $8,890,560$. The product of the tens digit is $156,800$. The product of the ones digit is $55,5664$.

			294
			216
			135
			98
			112
			84
			245
			40
8,890,560	156,800	55,566	

Can you find all eight three-digit numbers and complete the table?

Solution

```
from z3 import *

class MysteriousNumbersSolver:
    def __init__(self):
        self.X = [[Int("self.X_%s_%s" % (i, j)) for j in range(3)]
                        for i in range(8)]
        self.s = Solver()
        self.s.add([And(0 <= self.X[i][j], self.X[i][j]<= 9) for i in range(8) for j in ran
        self.s.add([And(0 <= self.X[i][0], self.X[i][0]<= 9) for i in range(8)])

    def set_constraints(self):
        self.s.add(self.X[0][0]*self.X[0][1]*self.X[0][2]==294)
        self.s.add(self.X[1][0]*self.X[1][1]*self.X[1][2]==216)
        self.s.add(self.X[2][0]*self.X[2][1]*self.X[2][2]==135)
        self.s.add(self.X[3][0]*self.X[3][1]*self.X[3][2]==98)
        self.s.add(self.X[4][0]*self.X[4][1]*self.X[4][2]==112)
        self.s.add(self.X[5][0]*self.X[5][1]*self.X[5][2]==84)
        self.s.add(self.X[6][0]*self.X[6][1]*self.X[6][2]==245)
        self.s.add(self.X[7][0]*self.X[7][1]*self.X[7][2]==40)
        self.s.add(self.X[0][0]*self.X[1][0]*self.X[2][0]*self.X[3][0]*self.X[4][0]*self.X[
        self.s.add(self.X[0][1]*self.X[1][1]*self.X[2][1]*self.X[3][1]*self.X[4][1]*self.X[
        self.s.add(self.X[0][2]*self.X[1][2]*self.X[2][2]*self.X[3][2]*self.X[4][2]*self.X[

    def output_solution(self):
        m = self.s.model()
        for i in range(8):
            print(" ".join([str(m.evaluate(self.X[i][j])) for j in range(3)]))

    def solve(self):
        self.set_constraints()
        if self.s.check() == sat:
            self.output_solution()
        else:
            print(self.s)
            print("Failed to solve.")

s = MysteriousNumbersSolver()
s.solve()
```

Here is the solution

```
7 7 6
9 8 3
9 5 3
7 2 7
8 2 7
7 4 3
```

```
5 7 7
8 5 1
```

Chapter 50

Can You Skillfully Ski The Slopes?

Congratulations, you've made it to the finals of the Riddler Ski Federation's winter championship! There's just one opponent left to beat, and then the gold medal will be yours.

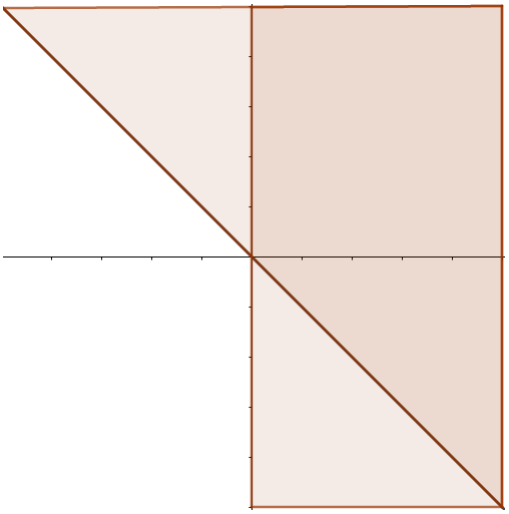
Each of you will complete two runs down the mountain, and the times of your runs will be added together. Whoever skis in the least overall time is the winner. Also, this being the Riddler Ski Federation, you have been presented detailed data on both you and your opponent. You are evenly matched, and both have the same normal probability distribution of finishing times for each run. And for both of you, your time on the first run is completely independent of your time on the second run.

For the first runs, your opponent goes first. Then, it's your turn. As you cross the finish line, your coach excitedly signals to you that you were faster than your opponent. Without knowing either exact time, what's the probability that you will still be ahead after the second run and earn your gold medal?

Extra credit: Over in the snowboarding championship, there are 30 finalists, including you (apparently, you're a dual-sport threat!). Again, you are the last one to complete the first run, and your coach signals that you are in the lead. What is the probability that you'll win gold in snowboarding?

Solution

Let the finishing times for the two rounds of the two players be denoted by A_1, A_2, B_1 and B_2 which are independent and identically distributed normal random variables. We are



given that $B_1 < A_1$. For player B to win the gold medal, $A_1 + A_2 > B_1 + B_2$. If we define two new random variables $X = A_1 - B_1$ and $Y = A_2 - B_2$, the required probability is given by $\mathbb{P}[X + Y > 0 | X > 0]$. As A_1, A_2, B_1 and B_2 are independent identically distributed normal random variables, X and Y are also independent normal random variables with mean 0. The joint probability density function of X and Y, is symmetric in x and y. The probability $\mathbb{P}[X > 0 \wedge X + Y > 0]$ can be obtained by integrating the joint density function over the darker region in the figure below:

Using the symmetry of the joint density function, it is easy to see that the probability is $\frac{3}{8}$.

Therefore, the required probability is given by

$$\begin{aligned}\mathbb{P}[X + Y > 0 | X > 0] &= \frac{\mathbb{P}[X > 0 \wedge X + Y > 0]}{\mathbb{P}[X > 0]} \\ &= \frac{\frac{3}{8}}{\frac{1}{2}} = \frac{3}{4}.\end{aligned}$$

The above value matches with that obtained by simulation using the code below. When you have 30 players the probability of you winning the gold given that you finished with the lowest time in the first run is $.31$.

Computational Solution

```
from numpy.random import normal

runs = 1000000
def prob_gold(n):
    num, den = 0, 0
    for _ in range(runs):
        firstrun_times = normal(0,1,n)
        total_times = firstrun_times + normal(0,1,n)
        if min(firstrun_times) == firstrun_times[n-1]:
            den += 1
            if min(total_times) == total_times[n-1]:
                num += 1
    return num/den

print("Probability of gold when number of players=%i is %f" % (2, prob_gold(2)))
print("Probability of gold when number of players=%i is %f" % (30, prob_gold(30)))
```

Chapter 51

Little Cubes

Riddler Express

I recently found four cubic blocks in a peculiar arrangement. Three of them were flat on the ground, with their corners touching and enclosing an equilateral triangle. Meanwhile, the fourth cube was above the other three, filling in the gap between them i Here's a photo I took of this ar

Now, if each of the four cubes has side length 1, then how far above the ground is the bottommost corner of the cube on top?

Solution

The part of the fourth cube on the top enclosed by the three cubes on the floor is an inverted tetrahedron whose base is an equilateral triangle. Using symmetry it is easy to see that

the bottommost corner of the tetrahedron is directly above the center of the base. The triangle formed by the following line segments is a right angled triangle:

1. The line segment from the bottommost corner to the center of the base. Let this be h units.

2. The line segment from the center of the base to the midpoint of a side of the base. The length of this line segment is $1/2\sqrt{3}$ units.

3. The line segment from the midpoint of the base to the bottommost corner along the face of the tetrahedron. This is the hypotenuse of the triangle. The length of this line segment is $1/2$ units. This follows from the fact that the face of the tetrahedron is an right angled isoceles triangle with base 1 unit.

Therefore, we have $h^2 + (\frac{1}{2\sqrt{3}})^2 = (\frac{1}{2})^2 \implies h = \frac{1}{\sqrt{6}}$.

The center of the base of the inverted tetrahedron is at a height of 1 unit (the length of the side of a cube) from the ground. The bottommost corner is therefore at a height of $1 - \frac{1}{\sqrt{6}}$ from the ground.

Can You Randomly Move The Tower?

Riddler Express

For your first weekly CrossProduct, there are five three-digit numbers - each belongs in a row of the table below, with one digit per cell. The products of the three digits of each number are shown in the rightmost column. Meanwhile, the products of the digits in the hundreds, tens and ones places, respectively, are shown in the bottom row.

3,000	3,969	640

Can you find all five three-digit numbers and complete the table?

Solution

```
from z3 import *

class MysteriousNumbersSolver:
    def __init__(self):
        self.X = [[Int("self.X_%s_%s" % (i, j)) for j in range(3)]
                      for i in range(5)]
```

```
        self.s = Solver()
        self.s.add([And(0 <= self.X[i][j], self.X[i][j]<= 9) for i in range(5) for j in ran
        self.s.add([And(0 <= self.X[i][0], self.X[i][0]<= 9) for i in range(5)])

    def set_constraints(self):
        self.s.add(self.X[0][0]*self.X[0][1]*self.X[0][2]==135)
        self.s.add(self.X[1][0]*self.X[1][1]*self.X[1][2]==45)
        self.s.add(self.X[2][0]*self.X[2][1]*self.X[2][2]==64)
        self.s.add(self.X[3][0]*self.X[3][1]*self.X[3][2]==280)
        self.s.add(self.X[4][0]*self.X[4][1]*self.X[4][2]==70)
        self.s.add(self.X[0][0]*self.X[1][0]*self.X[2][0]*self.X[3][0]*self.X[4][0]==3000)
        self.s.add(self.X[0][1]*self.X[1][1]*self.X[2][1]*self.X[3][1]*self.X[4][1]==3969)
        self.s.add(self.X[0][2]*self.X[1][2]*self.X[2][2]*self.X[3][2]*self.X[4][2]==640)

    def output_solution(self):
        m = self.s.model()
        for i in range(5):
            print(" ".join([str(m.evaluate(self.X[i][j])) for j in range(3)]))

    def solve(self):
        self.set_constraints()
        if self.s.check() == sat:
            self.output_solution()
        else:
            print(self.s)
            print("Failed to solve.")

s = MysteriousNumbersSolver()
s.solve()
```

Here is the solution

```
3 9 5
5 9 1
8 1 8
5 7 8
5 7 2
```

Riddler Classic

Cassius the ape (a friend of Caesar's) has gotten his hands on a Lucas' Tower puzzle (also commonly referred to as the "Tower of Hanoi"). This particular puzzle consists of three poles and three disks, all of which start on the same pole. The three disks have different diameters — the biggest disk is at the bottom and the smallest disk is at the top. The goal is to move all three disks from one pole to any other pole, one at

a time, but there's a catch. At no point can a larger disk ever sit atop a smaller disk. It turns out that Cassius couldn't care less about solving the puzzle, but he is very good at following directions and understands a larger disk can never sit atop a smaller disk. With each move, he randomly chooses one among the set of valid moves.

On average, how many moves will it take for Cassius to solve this puzzle with three disks?

Solution

The analytical solution for the case of n disks has been provided in this paper.

On average it will take Cassius 141.45 moves to solve the puzzle with three disks.

```
from random import choice

runs, n, total_cnt = 100000, 3, 0
other_towers = [[1,2],[0,2],[0,1]]
for _ in range(runs):
    cnt, towers = 0, [list(range(n,0,-1)),[],[]]
    while towers[2] != list(range(n,0,-1)):
        moves = []
        for i in [0,1,2]:
            for j in other_towers[i]:
                if towers[i] and (not towers[j] or towers[j][-1]>towers[i][-1
                    moves.append((i,j))
        f, t = choice(moves)
        towers[t].append(towers[f].pop())
        cnt += 1
    total_cnt += cnt
print(total_cnt/runs)
```

Chapter 53

Can You Cross Like A Boss?

Riddler Express

This time around, there are six three-digit numbers — each belongs in a row of the table below, with one digit per cell. The products of the three digits of each number are shown in the rightmost column. Meanwhile, the products of the digits in the hundreds, tens and ones places, respectively, are shown in the bottom row.

6,615	15,552	420	

Can you find all six three-digit numbers and complete the table?

Solution

```
from z3 import *

class MysteriousNumbersSolver:
```

```
    def __init__(self):
        self.X = [[Int("self.X_%s_%s" % (i, j)) for j in range(3)]
                      for i in range(6)]
        self.s = Solver()
        self.s.add([And(0 <= self.X[i][j], self.X[i][j]<= 9) for i in range(6
        self.s.add([And(0 <= self.X[i][0], self.X[i][0]<= 9) for i in range(6

    def set_constraints(self):
        self.s.add(self.X[0][0]*self.X[0][1]*self.X[0][2]==210)
        self.s.add(self.X[1][0]*self.X[1][1]*self.X[1][2]==144)
        self.s.add(self.X[2][0]*self.X[2][1]*self.X[2][2]==54)
        self.s.add(self.X[3][0]*self.X[3][1]*self.X[3][2]==135)
        self.s.add(self.X[4][0]*self.X[4][1]*self.X[4][2]==4)
        self.s.add(self.X[5][0]*self.X[5][1]*self.X[5][2]==49)
        self.s.add(self.X[0][0]*self.X[1][0]*self.X[2][0]*self.X[3][0]*self.X
        self.s.add(self.X[0][1]*self.X[1][1]*self.X[2][1]*self.X[3][1]*self.X
        self.s.add(self.X[0][2]*self.X[1][2]*self.X[2][2]*self.X[3][2]*self.X

    def output_solution(self):
        m = self.s.model()
        for i in range(6):
            print(" ".join([str(m.evaluate(self.X[i][j])) for j in range(3)]))

    def solve(self):
        self.set_constraints()
        if self.s.check() == sat:
            self.output_solution()
        else:
            print(self.s)
            print("Failed to solve.")

s = MysteriousNumbersSolver()
s.solve()
```

Here is the solution

```
7 6 5
9 8 2
3 9 2
5 9 3
1 4 1
7 1 7
```

Chapter 54

En Garde! Can You Win The Fencing Relay?

Riddler Express

My condo complex has a single elevator that serves four stories: the garage (G), the first floor (1), the second floor (2) and the third floor (3). Unfortunately, the elevator is malfunctioning and stopping at every single floor, no matter what. The elevator always goes $G, 1, 2, 3, 2, 1, G, 1, 2$, etc.

I want to board the elevator on a random floor (with all four floors being equally likely). As I round the corner to approach the elevator, I hear that its doors have closed, but I have no further information about which floor it's on or whether the elevator is going up or down. The doors might have just closed on my floor, for all I know.

On average, how many stops will the elevator make until it opens on my floor (including the stop on your floor)? For example, if I am waiting on the second floor, and I heard the doors closing on the garage level, then the elevator would open on my floor in two stops.

Extra credit: Instead of four floors, suppose my condo had N floors. On average, how many stops will the elevator make until it opens on my floor?

Solution

From the simulation below, we see that the average number of stops when there are **four** floors is **2.83**.

```
from itertools import cycle
from random import choice, randint

def avg_stops(n = 4, runs = 100000):
    rotate = lambda l, n: l[-n:] + l[:-n]
    floors, elevator_cycle = list(range(n)), list(range(n)) + list(range(n-2,
    total_stops = 0
    for _ in range(runs):
        my_floor, start = choice(floors), randint(0, len(elevator_cycle)-1)
        for f in cycle(rotate(elevator_cycle, start)):
            total_stops += 1
            if f == my_floor:
                break
    return total_stops/runs

print(avg_stops())
```

Riddler Classic

You are the coach at Riddler Fencing Academy, where your three students are squaring off against a neighboring squad. Each of your students has a different probability of winning any given point in a match. The strongest fencer has a 75 percent chance of winning each point. The weakest has only a 25 percent chance of winning each point. The remaining fencer has a 50 percent probability of winning each point.

The match will be a relay. First, one of your students will face off against an opponent. As soon as one of them reaches a score of 15, they are both swapped out. Then, a different student of yours faces a different opponent, continuing from wherever the score left off. When one team reaches 30 (not necessarily from the same team that first reached 15), both fencers are swapped out. The remaining two fencers continue the relay until one team reaches 45 points.

As the coach, you can choose the order in which your three students occupy the three positions in the relay: going first, second or third. How will you order them? And then what will be your team's chances of winning the relay?

Solution

From the simulation below, we see that the probability of the permutation **weakest, medium, strongest** winning is the highest at $\approx$ **93%**.

```
from itertools import permutations
from random import random

def winning_probabilities(runs = 100000):
    win_prob = {'w':0.25,'s':0.75, 'm':0.5}
    probs = []
    for p1,p2,p3 in permutations(win_prob.keys()):
        total_wins = 0
        for _ in range(runs):
            s1, s2 = 0, 0
            while (s1 < 15 and s2 < 15):
                if (random() < win_prob[p1]):
                    s1 += 1
                else:
                    s2 += 1
            while (s1 < 30 and s2 < 30):
                if (random() < win_prob[p2]):
                    s1 += 1
                else:
                    s2 += 1
            while (s1 < 45 and s2 < 45):
                if (random() < win_prob[p3]):
                    s1 += 1
                else:
                    s2 += 1
            if s1 == 45:
                total_wins += 1
        probs.append((p1,p2,p3,total_wins/runs))
    return probs

for p1,p2,p3,prob in winning_probabilities():
    print(f"Probability of the permutation {(p1,p2,p3)} winning is {prob}")
```

Chapter 55

Can You Slice The Ice?

Riddler Express

I have a most peculiar menorah. Like most menorahs, it has nine total candles — a central candle, called the shamash, four to the left of the shamash and another four to the right. But unlike most menorahs, the eight candles on either side of the shamash are numbered. The two candles adjacent to the shamash are both 1, the next two candles out from the shamash are 2, the next pair are 3, and the outermost pair are 4.

The shamash is always lit. How many ways are there to light the remaining eight candles so that sums on either side of the menorah are "balanced"? (For example, one such way is to light candles 1 and 4 on one side and candles 2 and 3 on the other side. In this case, the sums on both sides are 5, so the menorah is balanced.)

Solution

The number of ways of lighting the candles satisfying the conditions is **25**. The different ways of lighting the candles is given below:

(('l', 1), ('r', 1)) (('l', 2), ('r', 2)) (('l', 3), ('r', 3)) (('l', 4), ('r', 4)) (('l', 1), ('l', 2), ('r', 3)) (('l', 1), ('l', 3), ('r', 4)) (('l', 3), ('r', 1), ('r', 2)) (('l', 4), ('r', 1), ('r', 3)) (('l', 1), ('l', 2), ('r', 1), ('r', 2)) (('l', 1), ('l', 3), ('r', 1), ('r', 3)) (('l', 1), ('l', 4), ('r', 1), ('r', 4)) (('l', 1), ('l', 4), ('r', 2), ('r', 3)) (('l', 2), ('l', 3), ('r', 1), ('r', 4)) (('l', 2), ('l', 3), ('r', 2), ('r', 3)) (('l', 2), ('l', 4), ('r', 2),

('r', 4)) (('l', 3), ('l', 4), ('r', 3), ('r', 4)) (('l', 1), ('l', 2), ('l', 3), ('r', 2), ('r', 4)) (('l', 1), ('l', 2), ('l', 4), ('r', 3), ('r', 4)) (('l', 2), ('l', 4), ('r', 1), ('r', 2), ('r', 3)) (('l', 3), ('l', 4), ('r', 1), ('r', 2), ('r', 4)) (('l', 1), ('l', 2), ('l', 3), ('r', 1), ('r', 2), ('r', 3)) (('l', 1), ('l', 2), ('l', 4), ('r', 1), ('r', 2), ('r', 4)) (('l', 1), ('l', 3), ('l', 4), ('r', 1), ('r', 3), ('r', 4)) (('l', 2), ('l', 3), ('l', 4), ('r', 2), ('r', 3), ('r', 4)) (('l', 1), ('l', 2), ('l', 3), ('l', 4), ('r', 1), ('r', 2), ('r', 3), ('r', 4))

```
from itertools import product, combinations

def menorah_lighting(n=4):
    side_sum = lambda comb, side: sum([i for s, i in comb if s == side])
    candles = list(product(["l","r"], range(1, n+1)))
    cnt, lightings = 0, []
    for k in range(2, 2*n+1):
        for comb in combinations(candles, k):
            if side_sum(comb, "l") == side_sum(comb, "r"):
                lightings.append(comb)
                cnt += 1
    return cnt, lightings

cnt, lightings = menorah_lighting()
print(cnt)
for l in lightings:
    print(l)
```

Chapter 56

Can you stick it to the genie?

Riddler Express

I have three dice ($d4, d6, d8$) on my desk that I fiddle with while working, much to the chagrin of my co-workers. For the uninitiated, the $d4$ is a tetrahedron that is equally likely to land on any of its four faces (numbered 1 through 4), the $d6$ is a cube that is equally likely to land on any of its six faces (numbered 1 through 6), and the $d8$ is an octahedron that is equally likely to land on any of its eight faces (numbered 1 through 8).

I like to play a game in which I roll all three dice in "numerical" order: $d4$, then $d6$ and then $d8$. I win this game when the three rolls form a strictly increasing sequence (such as $2-4-7$, but not $2-4-4$). What is my probability of winning?

Extra credit: Instead of three dice, I now have six dice: $d4, d6, d8, d10, d12$ and $d20$. If I roll all six dice in "numerical" order, what is the probability I'll get a strictly increasing sequence?

Solution

From the simulation below, we see that the probability of the winning with $d4, d6$ and $d8$ is **0.25** and the probability of winning with $d4, d6, d8, d10, d12$ and $d20$ is **0.0118**.

```
from random import choice
def prob(dice_num_faces, runs=10000000):
    dice = {n:list(range(1, n+1)) for n in dice_num_faces}
    cnt_succ = 0
    for _ in range(runs):
```

```
        roll = [choice(dice[d]) for d in sorted(dice.keys())]
        cnt_succ += all(i < j for i, j in zip(roll, roll[1:]))
    return cnt_succ/runs

print(prob([4,6,8]))
print(prob([4,6,8,10,12,20]))
```

Chapter 57

How Many Friends Are On The Riddler Social Network?

Riddler Express

A group of 101 people join $\mu\epsilon\theta\alpha$, and each person has a random, 50 percent chance of being friends with each of the other 100 people. Friendship is a symmetric relationship on $\mu\epsilon\theta\alpha$, so if you're friends with me, then I am also friends with you.I pick a random person among the 101 — let's suppose her name is Marcia. On average, how many friends would you expect each of Marcia's friends to have?

Solution

From the simulation below, we see that the expected number of friends of each of Marcia's friends is **50.5**. It is interesting to note that Marcia on average would have only 50 friends.

```
import networkx as nx
from random import random, randint, choice

def exp_num_friends(n, runs = 10000):
    total_deg = 0
    for _ in range(runs):
        G = nx.Graph()
        for i in range(n):
            G.add_node(i)
        for i in range(n-1):
            for j in range(i+1, n):
                if random() < 0.5:
                    G.add_edge(i,j)
        marcia = randint(0,n-1)
```

```
        marcia_friends =  list(G.adj[marcia].keys())
        if marcia_friends:
            total_deg += G.degree[choice(marcia_friends)]
    return total_deg/runs

print(exp_num_friends(101))
```

Riddler Classic

The sum of the factors of 36 — including 36 itself — is 91. Coincidentally, 36 inches rounded to the nearest centimeter is ... 91 centimeters!

Can you find another whole number like 36, where you can "compute" the sum of its factors by converting from inches to centimeters?

Extra credit: Can you find a third whole number with this property? How many more whole numbers can you find?

Solution

From the code below we see that 378 and 49600 are two numbers below 1000000 that satisify the given property.

```
from functools import reduce

def sum_of_factors(n):
    return sum(set(reduce(list.__add__, ([i, n//i] for i in range(1, int(n**0.5) + 1) if n

def inches_to_cm_same_as_sum_of_divisors():
    nums= []
    for i in range(37,1000000):
        if round(i*2.54) == sum_of_factors(i):
            nums.append(i)
    return nums

print(inches_to_cm_same_as_sum_of_divisors())
```

Chapter 58

Can You Survive Squid Game?

Riddler Express

I have a spherical pumpkin. I carefully calculate its volume in cubic inches, as well as its surface area in square inches.

But when I came back to my calculations, I saw that my units — the square inches and the cubic inches — had mysteriously disappeared from my calculations. But it didn't matter, because both numerical values were the same!

What is the radius of my spherical pumpkin?

Extra credit: Let's dispense with 3D thinking. Instead, suppose I have an n-hyper spherical pumpkin. Once again, I calculate its volume (with units in^n) and surface area (with units in^{n-1}). Miraculously, the numerical values are once again the same! What is the radius of my n-hyper spherical pumpkin?

Solution

Let r be the radius of the spherical pumpkin. We have

$$\frac{4}{3}\pi r^3 = 4\pi r^2 \implies r = 3in$$

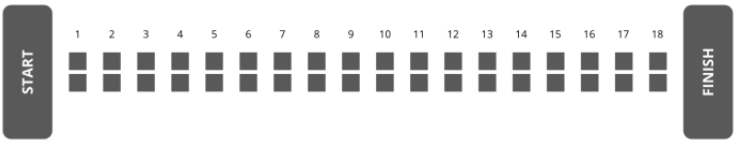
The recurrence relation for the surface area of an n-ball $V_n(R)$ is given by

$$A_n(R) = \frac{d}{dR}V_n(R) = \frac{n}{R}V_n(R)$$

If $A_n(R) = V_n(R)$, we have $R = n$.

Riddler Classic

Congratulations, you've made it to the fifth round of The Squiddler - a competition that takes place on a remote island. In this round, you are one of the 16 remaining competitors who must cross a bridge made up of 18 pairs of separated glass squares. Here is what the bridge looks like from above:



To cross the bridge, you must jump from one pair of squares to the next. However, you must choose one of the two squares in a pair to land on. Within each pair, one square is made of tempered glass, while the other is made of normal glass. If you jump onto tempered glass, all is well, and you can continue on to the next pair of squares. But if you jump onto normal glass, it will break, and you will be eliminated from the competition.

You and your competitors have no knowledge of which square within each pair is made of tempered glass. The only way to figure it out is to take a leap of faith and jump onto a square. Once a pair is revealed — either when someone lands on a tempered square or a normal square — all remaining competitors take notice and will choose the tempered glass when they arrive at that pair.

On average, how many of the 16 competitors will survive and make it to the next round of the competition?

Solution

Let $S(n, m)$ be the expected number of survivors when there are n competitors and m pairs of glasses. We have the recurrence relation

$$S(n,m) = \frac{1}{2}S(n,m-1) + \frac{1}{2}S(n-1,m-1) S(n,0) = nS(0,m) = 0$$

The Python code to compute $S(n, m)$ is given below:

```
def S(n , m):
    if n == 0:
        return 0
    if m == 0:
        return n
    return 0.5*S(n, m-1) + 0.5*S(n-1, m-1)
```

On average, if there are 16 competitors and 18 pairs of glasses, we will have **7** survivors.

Chapter 59

Who Betrayed Dune's Duke Leto?

Riddler Express

Duke Leto Atreides knows for a fact that there are not one, but two traitors within his royal household. The suspects are Lady Jessica, Dr. Wellington Yueh, Gurney Halleck and Duncan Idaho. Leto's advisor, Thufir Hawat, will assist him in questioning the four suspects. Anyone who is a traitor will tell a lie, while anyone who is not a traitor will tell the truth.

Upon interrogation, Jessica says that she is not the traitor, while Wellington similarly says that he is not the traitor. Gurney says that Jessica or Wellington is a traitor. Finally, Duncan says that Jessica or Gurney is a traitor. (Thufir, being the logician that he is, notes that when someone says thing A is true or thing B is true, both A and B can technically be true.)

After playing back the interrogations in his mind, Thufir is ready to report the name of one of the traitors to the duke. Whose name does he report?

Solution

Let j, w, g, d be the boolean variables indicating whether or not Jessica, Wellington, Gurney and Duncan are traitors or not. The statements made by them and the fact that there are **two** traitors reduce to the following set of logical propositions:

1. If $g \implies \neg(j \vee w)$
2. If $\neg g \implies (j \vee w)$
3. If $d \implies \neg(j \vee g)$

4. If $\neg d \implies (j \vee g)$

5. $(j \wedge w) \vee (j \wedge g) \vee (j \wedge d) \vee (w \wedge g) \vee (w \wedge d) \vee (g \wedge d)$

Here is the Python code in Z3 to check if the propositions above can be satisfied:

```
from z3 import *
j = Bool('j')
w = Bool('w')
g = Bool('g')
d = Bool('d')
s = Solver()

s.add(Implies(g,Not(Or(j,w))))
s.add(Implies(Not(g),Or(j,w)))
s.add(Implies(d,Not(Or(j,g))))
s.add(Implies(Not(d),Or(j,g)))
s.add(Or([And(j,w), And(j,g), And(j,d), And(w,g), And(w,d), And(g,d)]))

while s.check() == sat:
    m = s.model()
    print(m)
    s.add(Or(j != m[j], g != m[g], w != m[w], d != m[d]))
```

We see that there are two cases where the propositions above are satisfied:

[g = False, j = True, w = True, d = False] [w = True, g = False, j = False, d = True]

In both the cases, Wellington is one of the traitors, which means that Thufir reported **Wellington** as one of the traitors to the Duke.

Riddler Classic

Suppose you have an equilateral triangle. You pick three random points, one along each of its three edges, uniformly along the length of each edge — that is, each point along each edge has the same probability of being selected.

With those three randomly selected points, you can form a new triangle inside the original one. What is the probability that the center of the larger triangle also lies inside the smaller one?

Solution

The logic for determining if a point is inside or outside a given triangle is described here.

From the simulation, we see that the probability the center of the equilateral triangle lies inside the smaller random triangle is **0.46**.

The Python code for implementing the above logic is given below:

```
import numpy as np
from math import sqrt
from random import random

triangle = [np.array([-.5,-sqrt(3)/6]),
            np.array([.5,-sqrt(3)/6]),
            np.array([0, sqrt(3)/3])]
center = np.array([0,0])

def exp_percent_center_inside(triangle, center, runs = 100000):
    det = np.cross
    c=0
    for _ in range(runs):
        [A, B, C], v = triangle, center
        D, E, F = A + random()*(B - A), B + random()*(C - B), C + random()*(A - C)
        v0, v1, v2 = D,  E - D, F - D
        a = (det(v, v2) - det(v0, v2))/det(v1, v2)
        b = -((det(v, v1) - det(v0, v1))/det(v1, v2))
        if a > 0 and b > 0 and a + b < 1:
            c += 1
    return c/runs

print(exp_percent_center_inside(triangle, center))
```

Chapter 60

Can You Climb Your Way To Victory?

Riddler Express

While watching batter spread out on my waffle iron, and thinking back to a recent conversation I had with Friend-of-The-Riddler™ Benjamin Dickman, I noticed the following sequence of numbers:

$$1, 4, 4, 0, 4, 8, 0, 0, 4, 4, 8,$$

Before you ask — yes, you can find this sequence on the On-Line Encyclopedia of Integer Sequences. However, for the full Riddler experience, I urge you to not look it up. See if you can find the next few terms in the sequence, as well as the pattern.

Now, for the actual riddle: Once you've determined the pattern, can you figure out the average value of the entire sequence?

Solution

Let the sequence be denoted by $s(n)$. You can find more about the sequence here . Then $s(n)$ represents the number of ways an integer n can be expressed as a sum of two squares (positive, negative, or zero). That is, $s(n)$ denotes the number of solutions in integers $(x, y$ to the equation $x^2 + y^2 = n$. For example, $s(5) = 8$ since the solutions to $x^2 + y^2 = 5$ are $(1, 2)$, $(2, 1)$, $(-1, 2)$, $(2, -1)$, $(1, -2)$, $(-2, 1)$, $(-1, -2)$, and $(-2, -1)$. Because $s(n) = 0$ whenever n has the form $4k - 1$, $s(n)$ is a very erratic function.Thankfully, the problem is about the average value of $s(n)$ as $n \to \infty$. If we define $t(n)$ to be the number of

solutions in integers to $x^2 + y^2 \le n$, then the average of $s(k)$ for $0 \le k \le n$ is

$$\frac{s(0) + s(1) + \cdots + s(n)}{n+1} = \frac{t(n)}{n+1}$$

The code (using brute force) for calculating $t(n)$ and $t(n)/(n+1)$ is given below:

```
from math import sqrt, ceil

def t(n):
    t, l = 0, ceil(sqrt(n))+1
    for i in range(0, n+1):
        for j in range(-l, l):
            for k in range(-l, l):
                if j**2 + k**2 == i:
                    t += 1
    return t, t/(n+1)

print(list(map(t, [1,2,3,4,5,10,20,50,100])))
```

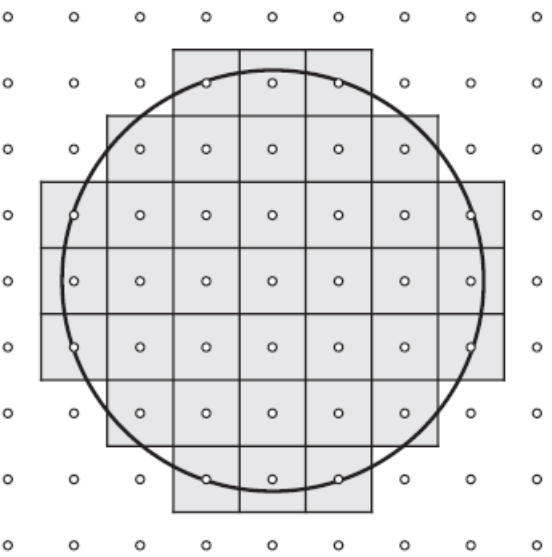
Here is a table of $t(n)$ and $t(n)/(n+1)$ for a few values of n:

n	1	2	3	4	5	10	20	50	100
$t(n)$	5	9	9	13	21	37	69	161	317
$\frac{t(n)}{n+1}$	2.5	3	2.25	2.6	3.5	3.36	3.29	3.16	3.15

Theorem . $\lim_{n\to\infty} \frac{t(n)}{n+1} = \pi$.

Proof. The proof from the reference below is based on a geometric interpretation of $t(n)$, and is due to Carl Friedrich Gauss (17771855): $t(n)$ is the number of points in or on a circle $x^2 + y^2 = n$ with integer coordinates. For example, $t(10) = 37$ since the circle centered at the origin of radius $\sqrt{10}$ contains 37 lattice points as illustrated in the figure below:

If we draw a square with area 1 centered at each of the 37 points, then the total area (in grey) of the squares is also $t(n)$. Thus we would expect the area of the squares to be approximately the area of the circle, or in general, $t(n)$ to be approximately $\pi(\sqrt{n})^2 = \pi n$. If we expand the circle of radius $\sqrt{n}$ by half the length of the diagonal $\sqrt{2}/2$ of a square of area 1, then the expanded circle contains all the squares. If we contract the circle by the same amount, then the contracted circle would be



contained in the union of all the squares as seen in the figure below:

Thus,

$$\pi(\sqrt{n} - \sqrt{2}/2)^2 \leq t(n) \leq \pi(\sqrt{n} + \sqrt{2}/2)^2$$

Dividing each term by $n + 1$ and applying the squeeze theorem for limits yields the desired result. ■

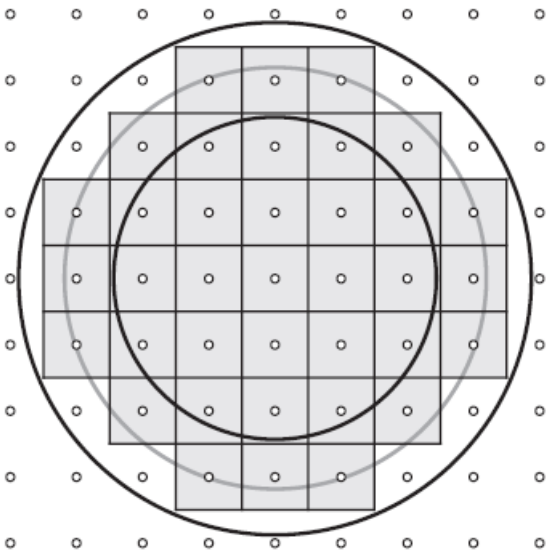
References

Charming Proofs: A Journey into Elegant Mathematics

Riddler Classic

The finals of the sport climbing competition has eight climbers, each of whom compete in three different events: speed climbing, bouldering and lead climbing. Based on their time and performance, each of the eight climbers is given a ranking (first through eighth, with no ties allowed) in each event, as well as a corresponding score (1 through 8, respectively).

The three scores each climber earns are then multiplied together to give a final score. For example, a climber who placed



second in speed climbing, fifth in bouldering and sixth in lead climbing would receive a score of 256, or 60 points. The gold medalist is whoever achieves the lowest final score among the eight finalists.

What is the highest (i.e., worst) score one could achieve in this event and still have a chance of winning (or at least tying for first place overall)?

Solution

From the **Monte-Carlo simulation** below, we see that the worst score one could achieve and still have a chance of winning is **48**.

```
from random import shuffle
from operator import mul
from functools import reduce

def max_min_score(np, ne, runs = 1000000):
    max_min_score = 0
    for _ in range(runs):
        ranks = [list(range(1,np+1)) for i in range(ne)]
        for i in range(ne):
            shuffle(ranks[i])
        scores = [reduce(mul, [ranks[j][i] for j in range(ne)]) for i in range(np)]
        min_score = min(scores)
```

```
            if min_score > max_min_score:
                max_min_score = min_score
        return max_min_score

print(max_min_score(8,3))
```

Chapter 61

Can You Bake The Radish Pie?

Riddler Express

I recently came across a rather peculiar recipe for something called Babylonian radish pie. Intrigued, I began to follow the directions, which said I could start with any number of cups of flour.

Any number? I mean, I had to start with some flour, so zero cups wasn't an option. But according to the recipe, any positive value was fair game. Next, I needed a second amount of flour that was 3 divided by my original number. For example, if I had started with two cups of flour, then the recipe told me I now needed 3 divided by 2, or 1.5, cups at this point.

I was then instructed to combine these amounts of flour and discard half. Apparently, this was my new starting amount of flour. I was to repeat the process, combining this amount with 3 divided by it and then discarding half.

The recipe told me to keep doing this, over and over. Eventually, I'd have the proper number of cups of flour for my radish pie.

How many cups of flour does the recipe ultimately call for?

Solution 1

In the limit after convergence, let f be number of cups of flour required as per the recipe. We have

$$f = \frac{1}{2}\left(f + \frac{3}{f}\right)$$

$$\implies f^2 = 3$$
$$\implies f = \sqrt{3}$$

Solution 2

This involves recognizing the recipe is infact a classic algorithm that is "Newton's" method to compute square roots $x = \sqrt{a}$ for $a > 0$, i.e. to solve $x^2 = a$. The algorithm involves the following steps:

1. Starts with some guess $x_1 > 0$.
2. Compute the sequence of improved guesses:

$$x_{n+1} = \frac{1}{2}\left(x_n + \frac{a}{x_n}\right)$$

In light of the above, it is easy to see that the recipe ultimately calls for $\sqrt{3}$ cups of flour.

Computational solution

From the code below, we see that the recipe ultimately calls for $\sqrt{3} \approx 1.732$ cups.

```
def num_cups():
    s, c = 2, 0
    while True:
        c = 0.5*(s + 3/s)
        if abs(s-c) < 0.000001:
            break
        s = c
    return c

print(num_cups())
```

Riddler Classic

Lately, Rushabh has been thinking about very large regular polygons — that is, a polygon all of whose sides and angles are congruent. His latest construction is a particular regular $1,000$-gon, which has $1,000$ sides of length 2. Rushabh picks

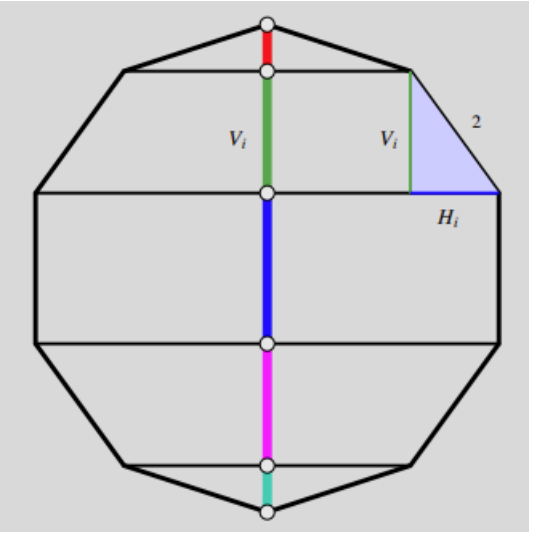
one of its longest diagonals, which connects two opposite vertices.

Now, this $1,000$-gon has many diagonals, but only some are perpendicular to that first diagonal Rushabh picked. If you were to slice the polygon along all these perpendicular diagonals, you'd break the first diagonal into 500 distinct pieces. Rushabh is curious — what is the product of the lengths of all these pieces?

Extra credit: Now suppose you have a regular $1,001$-gon, each of whose sides has length 2. You pick a vertex and draw an altitude to the opposite side of the polygon. Again, you slice the polygon along all the perpendicular diagonals, breaking the altitude into 500 distinct pieces. What's the product of the lengths of all these pieces this time?

Solution

The segments of the problem are the vertical components of the polygon's sides on its right half as shown in the diagram below; these lengths are $V_k = 2\sin\left((2k-1)\frac{\pi}{n}\right)$, with $k = 1, 2, \ldots, n/2$ where n is even (here we make use of the fact that the polygon's side rotates through $2\pi/n$ as as it moves to the next side).



The product of the lengths of all these segments is given by

$$\begin{aligned}
P &= \prod_{k=1}^{n/2} V_k \\
&= \prod_{k=1}^{n/2} 2\sin\left((2k-1)\frac{\pi}{n}\right) \\
&= \frac{1}{i^{n/2}} \prod_{k=1}^{n/2} e^{i(2k-1)t} - e^{-i(2k-1)t}, \text{ where } t = \pi/n \\
&= \frac{1}{i^{n/2}} \prod_{k=1}^{n/2} e^{i(2k-1)t} \prod_{k=1}^{n/2} 1 - e^{-i(4k-2)t} \\
&= \frac{1}{i^{n/2}} e^{in\pi/4} \prod_{k=1}^{n/2} 1 - e^{-i(4k-2)t} \\
&= \frac{1}{i^{n/2}} e^{in\pi/4} \prod_{k=1}^{n-1} 1 - e^{-i2kt} \Big/ \prod_{k=1}^{\frac{n}{2}-1} 1 - e^{-i4kt} \\
&= \frac{1}{i^{n/2}} e^{in\pi/4} \prod_{k=1}^{n-1} 1 - e^{-i2k\pi/n} \Big/ \prod_{k=1}^{\frac{n}{2}-1} 1 - e^{-i2\pi k/\frac{n}{2}}
\end{aligned}$$

Let $f(x) = \prod_{k=1}^{n-1}(x - e^{-i2\pi k/n})$. The roots of this polynomial are the non-trivial n-th roots of unity, so

$$f(x) = \frac{x^n - 1}{x - 1} = 1 + x + x^2 + \ldots + x^{n-1}$$

Plugging in 1 for x yields $f(1) = \prod_{k=1}^{n-1}(1 - e^{-i2\pi k/n}) = n$. Using the above result, we have

$$\begin{aligned}
\prod_{k=1}^{n-1} 1 - e^{-i2k\pi/n} &= n \\
\prod_{k=1}^{\frac{n}{2}-1} 1 - e^{-i2\pi k/\frac{n}{2}} &= n/2
\end{aligned}$$

Therefore, when $n = 1000$, the required product $P = 2e^{in\pi}/i^{n/2}$ has the value **2**.

Extra Credit

For the case when n is odd, we need to calculate the product

$$P = \prod_{k=1}^{\frac{n-1}{2}} V_k = \prod_{k=1}^{\frac{n-1}{2}} 2\sin\left((2k-1)\frac{\pi}{n}\right)$$

We make use of the result below. You can find the proof here.

$$\prod_{k=1}^{n-1} 2\sin\frac{k\pi}{n} = n$$

We have,

$$\begin{aligned}
\prod_{k=1}^{n-1} 2\sin\frac{k\pi}{n} &= n \\
\implies \prod_{k=1}^{\frac{n-1}{2}} 2\sin\frac{(2k-1)\pi}{n} \cdot \prod_{k=1}^{\frac{n-1}{2}} 2\sin\frac{2k\pi}{n} &= n \\
\implies P \cdot \prod_{k=1}^{\frac{n-1}{2}} 2\sin\left(\pi - \frac{2k\pi}{n}\right) &= n \\
\implies P \cdot P &= n \\
\implies P &= \sqrt{n}
\end{aligned}$$

When $n = 1001$, the required product has the value $\sqrt{1001} \approx$ **31.63854**.

Chapter 62

Can You Get The Paper Cut?

Riddler Express

Earlier this year, Dakota Jones used a crystal key to gain access to a hidden temple, deep in the Riddlerian Jungle. According to an ancient text, the crystal had exactly six edges, five of which were 1 inch long. Also, the key was the largest such polyhedron (by volume) with these edge lengths.

However, after consulting an expert, Jones realized she had the wrong translation. Instead of definitively having five edges that were 1 inch long, the crystal only needed to have four edges that were 1 inch long. In other words, five edges could have been 1 inch (or all six for that matter), but the crystal definitely had at least four edges that were 1 inch long.

The translator confirmed that the key was indeed the largest such polyhedron (by volume) with these edge lengths.

Once again, Jones needs your help. Now what is the volume of the crystal key?

Solution 1

Given the distances between the vertices of a tetrahedron the volume can be computed using the **Cayley–Menger determinant**:

$$288 \cdot V^2 = \begin{vmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & d_{12}^2 & d_{13}^2 & d_{14}^2 \\ 1 & d_{12}^2 & 0 & d_{23}^2 & d_{24}^2 \\ 1 & d_{13}^2 & d_{23}^2 & 0 & d_{34}^2 \\ 1 & d_{14}^2 & d_{24}^2 & d_{34}^2 & 0 \end{vmatrix}$$

where the subscripts $i, j \in \{1, 2, 3, 4\}$ represent the vertices and d_{ij} is the pairwise distance between them – i.e., the length of the edge connecting the two vertices.

If a, b, c be the three edges that meet at a point, and x, y, z the opposite edges. The volume V of the tetrahedron is given by

$$V = \frac{\sqrt{4a^2b^2c^2 - a^2X^2 - b^2Y^2 - c^2Z^2 + XYZ}}{12}$$

where

$$\begin{aligned} X &= b^2 + c^2 - x^2 \\ Y &= a^2 + c^2 - y^2 \\ Z &= a^2 + b^2 - z^2 \end{aligned}$$

In our case, let $a = b = c = y = 1$, we have

$$\begin{aligned} X &= 2 - x^2 \\ Y &= 1 \\ Z &= 2 - z^2 \end{aligned}$$

We need to find the value of x and z that maximizes

$$V = \frac{\sqrt{4 - (2 - x^2)^2 - 1 - (2 - z^2)^2 + (2 - x^2)(2 - z^2)}}{12}$$

Setting the partial derivates $\frac{\partial V}{\partial x}$ and $\frac{\partial V}{\partial z}$ to zero, we get the equations

$$\begin{aligned} 2 - 2x^2 + z^2 &= 0 \\ 2 - 2z^2 + x^2 &= 0 \end{aligned}$$

Therefore, V attains the maximum value of $\frac{1}{4\sqrt{3}}$ when $x = z = \sqrt{2}$.

References

Wikipedia

Solution 2

A polyhedron with 6 edges has to be a tetrahedron. In this particular case, we have a tetrahedron where one face is an equilateral triangle of side length 1. The volume of tetrahedron (which is a triangular pyramid where the base is an equilateral triangle) is given by

$$\begin{aligned} Vol &= \frac{1}{3} \times base \times height \\ &= \frac{1}{3} \times \frac{\sqrt{3}}{4} \times height \end{aligned}$$

The volume is maximized when the height is maximized i.e. when another edge of length 1 is perpendicular to the base. Therefore the volume of the crystal key is

$$Vol_{max} = \frac{1}{3} \times \frac{\sqrt{3}}{4} \times 1 = \frac{1}{4\sqrt{3}}.$$

Riddler Classic

One morning, Phil was playing with my daughter, who loves to cut paper with her safety scissors. She especially likes cutting paper into "strips," which are rectangular pieces of paper whose shorter sides are at most 1 inch long.

Whenever Phil gives her a piece of standard printer paper (8.5 inches by 11 inches), she picks one of the four sides at random and then cuts a 1-inch wide strip parallel to that side. Next, she discards the strip and repeats the process, picking another side at random and cutting the strip. Eventually, she is left with nothing but strips.

On average, how many cuts will she make before she is left only with strips?

Extra credit: Instead of 8.5 by 11-inch paper, what if the paper measures m by n inches? (And for a special case of this, what if the paper is square?)

Solution 1

From the **Monte-Carlo** simulation below, we see that the expected number of cuts before we are left only with strips is **14.29**.

```
from random import random

def avg_num_cuts_mc(m, n, runs= 1000000):
    sum_num_cuts = 0
    for _ in range(runs):
        num_cuts, cm, cn = 0, m, n
        while cm > 1 and cn > 1:
            r = random()
            if r < 0.5:
                cm -= 1
            else:
                cn -= 1
            num_cuts += 1
        sum_num_cuts += num_cuts
    return sum_num_cuts/runs

print(f"Expected number of cuts is {avg_num_cuts_mc(11, 9)}")
```

Solution 2

We have the following recurrence relation for the expected number of cuts

$$C(m,n) = 1 + \frac{1}{2}C(m-1,n) + \frac{1}{2}C(n-1,m)$$

with initial conditions $C(m,0) = C(0,n) = 0$.

Solving the recurrence relation using **memoization** we see that the expected number of cuts is indeed **14.29**.

```
def avg_num_cuts_rr(m, n):
    C = [[0 for _ in range(n)] for _ in range(m)]
    for i in range(1, m):
        for j in range(1, n):
            C[i][j] = 1 + 0.5*(C[i][j-1] + C[i-1][j])
    return C[m-1][n-1]

print(f"Expected number of cuts is {avg_num_cuts_rr(11, 9)}")
```

Chapter 63

Can You Draft A Riddler Fantasy Football Dream Team?

Hames Jarrison has just intercepted a pass at one end zone of a football field, and begins running — at a constant speed of 15 miles per hour — to the other end zone, 100 yards away.

At the moment he catches the ball, you are on the very same goal line, but on the other end of the field, 50 yards away from Jarrison. Caught up in the moment, you decide you will always run directly toward Jarrison's current position, rather than plan ahead to meet him downfield along a more strategic course.

Assuming you run at a constant speed (i.e., don't worry about any transient acceleration), how fast must you be in order to catch Jarrison before he scores a touchdown?

Solution

Let the chaser be at $(0, 0)$ and the runner be $(x_0, 0)$ at the time $t = 0$ respectively, the instant the pursuit begins, with the runner running at constant speed V_r along the line $x = x_0$. The chaser runs at a constant speed V_c along a curved path such that he is always moving directly toward the runner, that is, the velocity vector of the chaser points directly at the runner at every instant of time.

To find the curve of pursuit of the chaser, we assume that the chaser is at the location (x, y) at time $t \geq 0$. At time t, the runner is at the point $(x_0, V_r t)$ and so, the slope of the tangent line to the pursuit curve (the value of dy/dx at (x, y)) is given by

$$\frac{dy}{dx} = \frac{V_r t - y}{x_0 - x}$$

We also know that the chaser would have ran a distance $V_c t$ along it by the time t. This arc-length is also given by the expression on the right below:

$$V_c t = \int_0^x \sqrt{1 + \left(\frac{dy}{dz}\right)^2}\, dz$$

Eliminating t from the above two equations, we get

$$\begin{aligned}
&\frac{1}{V_c}\int_0^x \sqrt{1 + \left(\frac{dy}{dz}\right)^2}\, dz = \frac{y}{V_r} - \frac{(x - x_0)}{V_r} \cdot \frac{dy}{dx} \\
\implies &\frac{1}{V_c}\int_0^x \sqrt{1 + p^2(z)} dz = \frac{y}{V_r} - \frac{(x - x_0)}{V_r} \cdot p(x)\ \text{, where } dy/dx = p(x).
\end{aligned}$$

Differentiating under the integral sign with respect to x, we arrive at

$$\begin{aligned}
&\frac{1}{V_c}\sqrt{1 + p^2(x)} = \frac{p(x)}{V_r} - \frac{(x - x_0)}{V_r} \cdot \frac{dp(x)}{dx} - \frac{1}{V_r} p(x) \\
\implies &(x - x_0)\frac{dp}{dx} = -\frac{V_r}{V_c}\sqrt{1 + p^2(x)} \\
\implies &\frac{dp}{\sqrt{1 + p^2(x)}} = \frac{n dx}{(x_0 - x)}, \text{ where } n = V_r/V_c
\end{aligned}$$

Integrating the above equation, we get

$$\ln(p + \sqrt{1 + p^2}) + c = -n \ln(x_0 - x).$$

We see at $t = 0$, that $p = dy/dx = 0$ when $x = 0$ because at that instant, the runner as well as the chaser are on the $x-axis$. It follows that $c = -n \ln(x_0)$ and so

$$\begin{aligned}
&\ln\left[\left(p + \sqrt{1 + p^2}\right)\left(1 - \frac{x}{x_0}\right)^n\right] = 0 \\
\implies &\left(p + \sqrt{1 + p^2}\right)\left(1 - \frac{x}{x_0}\right)^n = 1.
\end{aligned}$$

From the above, we get

$$p(x) = \frac{dy}{dx} = \frac{1}{2}\left[\left(1 - \frac{x}{x_0}\right)^{-n} - \left(1 - \frac{x}{x_0}\right)^{n}\right]$$

Integrating the above equation, we get

$$y(x) + c = \frac{1}{2}(x_0 - x)\left[\frac{(1 - x/x_0)^n}{1+n} - \frac{(1 - x/x_0)^{-n}}{1-n}\right].$$

Since $y = 0$ when $x = 0$, pursuit curve equation is given by

$$y(x) = \frac{n}{1-n^2}x_0 + \frac{1}{2}(x_0 - x)\left[\frac{(1 - x/x_0)^n}{1+n} - \frac{(1 - x/x_0)^{-n}}{1-n}\right]$$

In the given problem, we have $V_r = 15$ miles/hr, $x_0 = 50$ and $y = 100$ when $x = 50$. Substituting these values in the above equation, we get

$$\begin{aligned} 100 &= \frac{n}{1-n^2} \cdot 50 \\ \implies 2n^2 + n - 2 &= 0 \end{aligned}$$

Solving the above quadratic and taking the positive value for n, we see that the speed of the chaser V_c should be

$$\frac{\sqrt{17}+1}{4} \cdot 15 \text{ miles/hr}$$

Chapter 64

Can you catch the cricket?

Riddler Express

Help, there's a cricket on my floor! I want to trap it with a cup so that I can safely move it outside. But every time I get close, it hops exactly 1 foot in a random direction.

I take note of its starting position and come closer. Boom — it hops in a random direction. I get close again. Boom — it takes another hop in a random direction, independent of the direction of the first hop.

What is the most probable distance between the cricket's current position after two random jumps and its starting position? (Note: This puzzle is not asking for the expected distance, but rather the most probable distance. In other words, if you consider the probability distribution over all possible distances, where is the peak of this distribution?)

Solution

Let θ be the angle between the cricket's jumps. Using the cosine law, we see that the distance from the center to the end point of the second hop is $\sqrt{2 - 2\cos\theta}$. The least distance from the starting point is 0 and the maximum distance is 2. We can asssume that $\theta \sim \mathcal{U}[0, \pi]$.

For any real $x \in [0, 2]$, we have the CDF of $X = \sqrt{2 - 2\cos\theta}$,

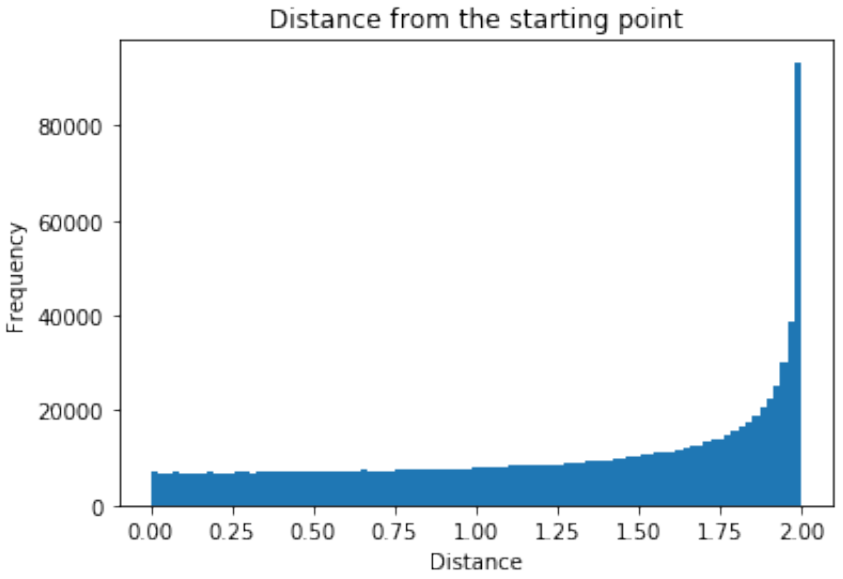
$$
\begin{aligned}
F(x) &= \mathbb{P}[X \le x] \\
&= \mathbb{P}[\sqrt{2 - 2\cos\theta} \le x] \\
&= \mathbb{P}[\frac{2 - x^2}{2} \le \cos\theta] \\
&= \mathbb{P}[\arccos\left(\frac{2 - x^2}{2}\right) \ge \theta] \\
&= \frac{1}{\pi}\arccos\left(\frac{2 - x^2}{2}\right)
\end{aligned}
$$

Taking the derivative of the CDF, the PDF of X is $f(x) = \frac{2}{\pi\sqrt{4-x^2}}$ on $[0, 2]$.

The PDF approaches infinity as $x \to 2$, therefore the most probable distance is **2** feet.

Computational solution

From the histogram below, we see that the most probable distance is indeed **2** feet.




```
import numpy as np
from math import pi, sqrt, sin, cos
import matplotlib.pyplot as plt
```

```
def distances(num_samples = 1000000):
    def dist(row):
        return sqrt((cos(row[0])-cos(row[1]))**2 + (sin(row[0])-sin(row[1]))**2)
    angles = np.random.uniform(0, 2*pi, (num_samples,2))
    return np.apply_along_axis(dist, 1, angles)

def plot_dist_hist(distances):
    plt.hist(distances, bins='auto')
    plt.xlabel('Distance')
    plt.ylabel('Frequency')
    plt.title('Distance from the starting point')

plot_dist_hist(distances())
```

Riddler Classic

When you roll a pair of fair dice, the most likely outcome is 7 (which occurs $1/6$ of the time) and the least likely outcomes are 2 and 12 (which each occur $1/36$ of the time).

Annoyed by the variance of these probabilities, I set out to create a pair of "uniform dice." These dice still have sides that are uniquely numbered from 1 to 6, and they are identical to each other. However, they are weighted so that their sum is more uniformly distributed between 2 and 12 than that of fair dice.

Unfortunately, it is impossible to create a pair of such dice so that the probabilities of all 11 sums from 2 to 12 are identical (i.e., they are all $1/11$). But I bet we can get pretty close.

The variance of the 11 probabilities is the average value of the squared difference between each probability and the average probability (which is, again, $1/11$). One way to make my dice as uniform as possible is to minimize this variance.

So how should I make my dice as uniform as possible? In other words, which specific weighting of the dice minimizes the variance among the 11 probabilities? That is, what should the probabilities be for rolling $1, 2, 3, 4, 5$ or 6 with one of the dice?

Solution

Let $p_1, p_2, p_3, p_4, p_5, p_6$ be the probabilities for rolling $1, 2, 3, 4, 5$ or 6 with one of the dice. We have $\sum_{i=1}^{6} p_i = 1$ and $0 < p_i < 1$ for $i \in \{1, \ldots, 6\}$. To calculate the variance among the 11 probabilities, we first need to calculate $\mathbb{P}[sum = i]$ for $i \in \{2, \ldots, 12\}$. The probabilities are fairly straightforward to calculate e.g.

$$\mathbb{P}[sum = 2] = p_1^2$$
$$\mathbb{P}[sum = 3] = 2p_1p_2$$
$$\mathbb{P}[sum = 4] = p_2^2 + 2p_1p_3$$
$$\mathbb{P}[sum = 5] = 2p_1p_4 + 2p_2p_3$$

The variance is then given by

$$\frac{1}{11}\sum_{i=2}^{12}(\mathbb{P}[sum = i] - \frac{1}{11})^2$$

From the code below we see that the minimum value of the variance is **0.001217** and the probabilities are $p_1 =$ **0.244**, $p_2 =$ **0.137**, $p_3 =$ **0.118**, $p_4 =$ **0.118**, $p_5 =$ **0.137** and $p_6 =$ **0.244**.

```
from scipy.optimize import minimize

def variance(probs):
    prob_sums={}
    for d1,d2 in [(i,j) for i in range(6) for j in range(6)]:
        prob_sums[d1+d2] = prob_sums.get(d1+d2, 0) + probs[d1]*probs[6+d2]
    return sum([(p - 1/11)**2 for p in prob_sums.values()])/11

minimize(variance, [0.5]*12, constraints=[{'type':'eq', 'fun':lambda probs:su
                            {'type':'eq', 'fun':lambda probs:sum(probs[6:])-1
```

If we allow the two dice to have different weightings, we can do much better.From the code below we see that the minimum value of the variance is **0.000258** and the probabilities are $p_1 =$ **0.5**, $p_2 =$ **0**, $p_3 =$ **0**, $p_4 =$ **0**, $p_5 =$ **0** and $p_6 =$ **0.5** for the first die and $p_1 =$ **0.125**, $p_2 =$ **0.186**, $p_3 =$ **0.189**, $p_4 =$ **0.189**, $p_5 =$ **0.186** and $p_6 =$ **0.125** for the second.

```
minimize(variance, [0.5]*12, bounds=[(0,1)]*12,
                            constraints=[{'type':'eq', 'fun':lambda probs:sum
                            {'type':'eq', 'fun':lambda probs:sum(probs[6:])-1
                            {'type':'eq', 'fun':lambda probs:probs[0]-0.5}])
```

Chapter 65

Are You Clever Enough?

Riddler Express

You are very clever when it comes to solving Riddler Express puzzles. You are so clever, in fact, that you are in the top 10 percent of solvers in Riddler Nation (which, as you know, has a very large population). You don't know where in the top 10 percent you are — in fact, you realize that you are equally likely to be anywhere in the topmost decile. Also, no two people in Riddler Nation are equally clever.

One Friday morning, you walk into a room with *nine* members randomly selected from Riddler Nation. What is the probability that you are the cleverest solver in the room?

Solution

Without loss of generality we can assume that the normalized IQ of the population follows the Uniform distribution $\mathcal{U}[0,1]$.Let $X_1, X_2, \ldots, X_n$ be the random variables representing the IQs of the other individuals in the room apart from you where $X_i \sim \mathcal{U}[0,1]$ for $i \in \{1,\ldots,n\}$. Let $Y \sim \mathcal{U}[a,b]$ be the random variable representing your IQ where $0 \leq a < b \leq 1$. We need the probability $\mathbb{P}[X_{(n)} < Y]$ where $X_{(n)} = \max(X_1,\ldots,X_n)$. From Order Statistics of the Uniform distribution, we know that $\mathbb{P}[X_{(n)} < y] = y^n$. Therefore the required probability is given by

$$\begin{aligned}\mathbb{P}[X_{(n)} < Y] &= \int_a^b y^n \frac{1}{b-a} dy \\ &= \frac{b^{n+1} - a^{n+1}}{(n+1)(b-a)}\end{aligned}$$

When $n = 9$, $a = 0.9$ and $b = 1.0$ the required probability is **0.65132**.

Computational solution

From the simulation below, we can see that the required probability is indeed $\approx$ **0.651**.

```
import numpy as np

def prob_you_cleverest(n, start_iq = 0.9, end_iq = 1.0, runs = 5000000):
    others_iqs = np.random.rand(runs, n)
    your_iqs = np.random.uniform(start_iq, end_iq, runs)
    return np.mean(np.where(np.max(others_iqs, axis=1) < your_iqs, [1], [0]))

print(prob_you_cleverest(9))
```

Riddler Classic

You have four standard dice, and your goal is simple: Maximize the sum of your rolls. So you roll all four dice at once, hoping to achieve a high score.

But wait, there's more! If you're not happy with your roll, you can choose to reroll zero, one, two or three of the dice. In other words, you must "freeze" one or more dice and set them aside, never to be rerolled.

You repeat this process with the remaining dice — you roll them all and then freeze at least one. You repeat this process until all the dice are frozen.

If you play strategically, what score can you expect to achieve on average?

Extra credit: Instead of four dice, what if you start with five dice? What if you start with six dice? What if you start with N dice?

Computational solution

Let the number of dice be N, a roll can be represented as an N-tuple $(d_1, d_2, \ldots, d_N)$ where $d_i \in \{1, \ldots, 6\}$ for $i \in \{1, \ldots, 6\}$. There are a total of 6^N possible rolls. Let $MES(roll)$ be the function which gives you the maximum expected score that can be achieved starting from a given roll by choosing to reroll zero, one, two or $N-1$ dice. The expected maximum score is then given by

$$EMS(N) = \sum \frac{1}{6^N} \cdot MES(roll)$$

where $roll \in \{(d_1, d_2, \ldots, d_N) | d_i \in \{1, \ldots, 6\}, i \in \{1, \ldots, N\}\}$.

For a given roll $(d_1, d_2, \ldots, d_N)$, let $(d_{s(1)}, d_{s(2)}, \ldots, d_{s(N)})$ be the corresponding tuple sorted in **descending** order.

The maximimum expected score that can be obtained by following the optimal strategy starting from a given initial roll can be calculated as follows:

1. We first freeze the maximum value of a roll $d_{s(1)}$.
2. We then calculate the maximum expected score that can result by rerolling $0, \ldots, N-1$ remaining dice. If we are rolling j dice, the maximum expected score is the sum of the expected maximum score for those dice $EMS(j)$ and the top $N-1-j$ of the remaining values of the roll after removing the maximum value which was already frozen.

This leads to the following definition of the function MES:

$$\begin{aligned} MES((d_1, d_2, \ldots, d_N)) &= d_{s(1)} + \\ \max \left\{ EMS(j) + \sum_{i=2}^{N-j} d_{s(j)} \middle| j \in \{0, \ldots, N-1\} \right\} \\ = \max \left\{ EMS(j) + \sum_{i=1}^{N-j} d_{s(j)} \middle| j \in \{0, \ldots, N-1\} \right\} \end{aligned}$$

where j stands for the number of dice that are rerolled.

From the above it is clear that $EMS(N)$ is ultimately defined in terms of $EMS(0) = 0, EMS(1), \ldots, EMS(N-1)$ and it can be calculated iteratively.

Using the code below, we see that the maximum score that can be achieved on average is **989065/52488=18.843**.

```
from itertools import product
from fractions import Fraction

def MESs(num_dice):
    exp_max_scores = {}
    exp_max_scores[0] = 0
    for n in range(1, num_dice+1):
        total_mes = 0
        for roll in product(range(1, 6 + 1), repeat = n):
            sorted_roll = sorted(list(roll), reverse=True)
            total_mes += max(sum(sorted_roll[:n-j]) + exp_max_scores[j] for j
        exp_max_scores[n]=Fraction(total_mes, 6**n)
    return exp_max_scores

mess = MESs(5)
for i in range(1,5):
    print(f"The average max score for {i} dice is {mess[i]}")
```

```
The average max score for 1 dice is 7/2
The average max score for 2 dice is 593/72
The average max score for 3 dice is 13049/972
The average max score for 4 dice is 989065/52488
```

Chapter 66

Will Riddler Nation Win Gold In Archery?

Riddler Express

Riddler Nation is competing against Conundrum Country at an Olympic archery event. Each team fires three arrows toward a circular target 70 meters away. Hitting the bull's-eye earns a team 10 points, while regions successively farther away from the bull's-eye are worth fewer and fewer points.

Whichever team has more points after three rounds wins. However, if the teams are tied after each team has taken three shots, both sides will fire another three arrows. (If they remain tied, they will continue firing three arrows each until the tie is broken.)

For every shot, each archer of Riddler Nation has a one-third chance of hitting the bull's-eye (i.e., earning 10 points), a one-third chance of earning 9 points and a one-third chance of earning 5 points.

Meanwhile, each archer of Conundrum Country earns 8 points with every arrow.

Which team is favored to win?

Extra credit: What is the probability that the team you identified as the favorite will win?

Solution

Let p_w be the probability of Riddler Nation winning. Riddler Nation either wins the first time with a probability of p_{wf} or

Riddler Nation draws with a probability of p_d and then wins with a probability p_w. Therefore we have,

$$p_w = p_{wf} + p_d \cdot p_w$$
$$\implies p_w = \frac{p_{wf}}{1 - p_d}$$

Calculating probabilities

There are a couple of ways by which p_{wf} and p_d can be calculated.

One approach is to use brute-force enumeration of all the 3^3 3-tuples of the points earned in each of the 3 shots and count the number of tuples whose sum is greater or equal to 24. The Python code for counting the tuples is given below:

```
from itertools import product
sum([1 for p in product(*[[10,9,5]]*3) if sum(p) > 24])
```

The other approach is to use **generating functions** i.e the coefficient of x^{24} in $(x^{10} + x^9 + x^5)^3$ gives the count of the number of cases where the sum of the scores of the three shots is 24.

Computational Solution

From the simulation below, we see that the **Riddler Nation** is favoured to win and the probability of win is $\approx$ **0.524**.

```
from random import random

def shot_pts_riddler():
    p = random()
    if p < 0.33333333:
        return 10
    elif p > 0.33333333 and p < 0.66666666:
        return 9
    else:
        return 5

def shot_pts_conundrum():
    return 8

def prob_win_riddler(runs=1000000):
    total_wins = 0
    for _ in range(runs):
```

```
        pts_riddler, pts_conundrum  = 0, 0
        while pts_riddler == pts_conundrum:
            pts_riddler = sum([shot_pts_riddler() for _ in range(3)])
            pts_conundrum = sum([shot_pts_conundrum() for _ in range(3)])
        if pts_riddler > pts_conundrum:
                total_wins += 1
    return total_wins/runs

print(prob_win_riddler())
```

Riddler Classic

Suppose you have a chain with infinitely many flat (i.e., one-dimensional) links. The first link has length 1, and the length of each successive link is a fraction f of the previous link's length. As you might expect, f is less than 1. You place the chain flat on a table and some ink at the very end of the chain (i.e., the end with the infinitesimal links).

Initially, the chain forms a straight line segment, and the longest link is fixed in place. From there, the links are constrained to move in a very specific way: The angle between each chain and the next, smaller link is always the same throughout the chain. For example, if the Nth link and the $N + 1st$ link form a 40 degree clockwise angle, then so do the $N + 1st$ link and the $N + 2nd$ link.

After you move the chain around as much as you can, what shape is drawn by the ink that was at the tail end of the chain?

Solution

The lengths of the links in the chain for a geometric progression $1, f, f^2, \ldots$.

Using the properties of **complex numbers**, the position p_i for $i \geq 2$ of the end point of each link in the chain can be represented as a complex function of θ (the angle between each chain link measured anticlockwise). We have,

$$
\begin{aligned}
p_1 &= 1 \\
p_2(\theta) &= 1 + fe^{i\theta} \\
&\vdots \\
p_n(\theta) &= 1 + fe^{i\theta} + \cdots + f^{n-1}e^{(n-1)i\theta} = \sum_{k=0}^{n-1} f^k e^{ik\theta} \\
p_\infty(\theta) &= \frac{1}{1 - fe^{i\theta}} = \frac{1}{1 - f\cos\theta - if\sin\theta}
\end{aligned}
$$

If x and y are the coordinates of the end point of the last link in the chain, we have

$$
\begin{aligned}
x &= \frac{1 - f\cos\theta}{1 + f^2 - 2f\cos\theta} \\
y &= \frac{f\sin\theta}{1 + f^2 - 2f\cos\theta}
\end{aligned}
$$

Eliminating θ from the above using brute force, we get

$$
\left(x - \frac{1}{1 - f^2}\right)^2 + y^2 = \left(\frac{f}{1 - f^2}\right)^2
$$

which is the equation of a circle centered at $\left(\frac{1}{1-f^2}, 0\right)$ and radius $\frac{f}{1-f^2}$.

Using Mobius transformation

The mapping $\frac{1}{1-fe^{i\theta}}$ is a **Mobius Transformation** of $e^{i\theta}$ a complex number on the unit circle centered at the origin.

Here are a couple of fundamental results related to Mobius Transformations that we will use:

- Mobius transformations map circles to circles.
- If two points symmetric with respect to a circle, then their images under a Mobius transformation are symmetric to the image circle. This is called the 'Symmetry Principle'.

Two end points of a diameter $(-1, 0)$ and $(0, 1)$ of the unit circle get mapped to the two end points $\frac{1}{1+f}$ and $\frac{1}{1-f}$ of a dia-

meter of the mapped circle by the symmetry principle. Therefore the centre of the mapped circle is at

$$\frac{1}{2}\left(\frac{1}{1-f}+\frac{1}{1+f}\right)=\frac{1}{1-f^2}$$

The radius of the circle is given by

$$\frac{1}{2}\left|\left(\frac{1}{1-f}-\frac{1}{1+f}\right)\right|=\frac{f}{1-f^2}$$

Chapter 67

Can You Hop Across The Chess Board?

Riddler Express

The following 8-by-8 grid is covered with a total of 64 chess pieces, with one piece on each square. You should begin this puzzle at the white bishop on the green square. You can then move from white piece to white piece via the following rules:

If you are on a pawn, move up one space diagonally (left or right).

If you are on a knight, move in an "L" shape — two spaces up, down, left or right, and then one space in a perpendicular direction.

If you are on a bishop, move one space in any diagonal direction.

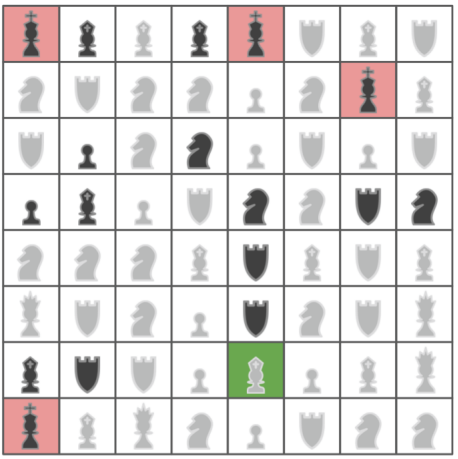
If you are on a rook, move one space up, down, left or right.

If you are on a queen, move one space in any direction (horizontal, vertical or diagonal).

Chess board with 64 pieces on it. From left to right in each row, and then top to bottom, the pieces are (b = black, w = white):

For example, suppose your first move from the bishop is diagonally down and to the right. Now you're at a white rook, so your possible moves are left or up to a pawn or right to a knight.

Your objective is to reach one of the four black kings on the grid. However, at no point can you land on any of the other black pieces. (Knights are allowed to hop over the black pieces.)



What sequence of moves will allow you to reach a king?

Solution

Here is the Python code for solving the problem using directed graphs. Every square on the chessboard is a node. All squares that are reachable from any given square subject to the movement constraints of the piece on that square are neighbours of the given square. The neighbours of a square are connected by directed edges whose source is the original square and the destination is a neighbour. Given this representation, the above problem reduces to finding a directed path in the graph from the source(i.e. the green square) to the destination (i.e. one the sqares with a black king).

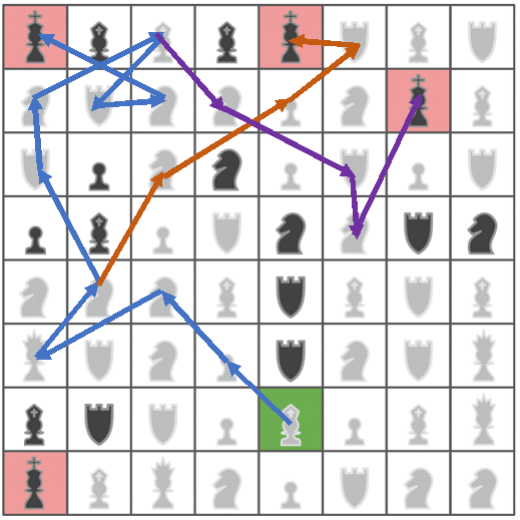
```
def moves(piece, i, j, size=8):
    def is_valid(pos):
        i,j = pos
        return True if i >= 0 and i < size and j >= 0 and j < size else False
    piece_moves = {
        "p": [(i+1,j+1),(i-1,j+1)],
        "b": [(i+1,j+1),(i-1,j+1),(i-1,j-1),(i+1,j-1)],
        "r": [(i,j+1),(i-1,j),(i+1,j),(i,j-1)],
        "h": [(i-1,j+2), (i+1,j+2), (i-1,j-2),(i+1,j-2),
            (i-2,j-1), (i-2,j+1),(i+2,j-1),(i+2,j+1)],
        "q": [(i,j+1),(i-1,j),(i+1,j),(i,j-1),(i+1,j+1),
            (i-1,j+1),(i-1,j-1),(i+1,j-1)]
    }
    return list(filter(is_valid, piece_moves[piece]))
```

```
def paths(board, source, targets):
    import networkx as nx
    pieces = {}
    for i, line in enumerate(reversed(board)):
        for j, piece in enumerate(line):
            pieces[(j,i)] = piece.strip()
    G = nx.DiGraph()
    for (i,j), piece in pieces.items():
        if piece[0] == "w":
            for move in moves(piece[1],i,j):
                if pieces[move][0] == "w" or pieces[move] == "bk":
                    G.add_edge((i,j), move)
    paths = []
    for target in targets:
        path = None
        try:
            path = nx.shortest_path(G, source, target)
        except:
            pass
        if path:
            paths.append(path)
    return paths

board1 = [
    ["bk", "bb", "wb", "bb", "bk", "wr", "wb", "wr"],
    ["wh", "wr", "wh", "wh", "wp", "wh", "bk", "wb"],
    ["wr", "bp", "wh", "bh", "wp", "wr", "wp", "wr"],
    ["bp", "bb", "wp", "wr", "bh", "wh", "br", "bh"],
    ["wh", "wh", "wh", "wb", "br", "wb", "wr", "wb"],
    ["wq", "wr", "wh", "wp", "br", "wh", "wr", "wq"],
    ["bb", "br", "wr", "wp", "wb", "wp", "wb", "wq"],
    ["bk", "wb", "wq", "wh", "wp", "wr", "wh", "wh"]
]
print(paths(board1, (4,1), [(0,0),(0,7),(4,7),(6,6)]))
```

Riddler Classic

Today marks the beginning of the Summer Olympics! One of the brand-new events this year is sport climbing, in which competitors try their hands (and feet) at lead climbing, speed climbing and bouldering.

Suppose the event's organizers accidentally forgot to place all the climbing holds on and had to do it last-minute for their 10-meter wall (the regulation height for the purposes of this riddle). Climbers won't have any trouble moving horizontally



along the wall. However, climbers can't move between holds that are more than 1 meter apart vertically.

In a rush, the organizers place climbing holds randomly until there are no vertical gaps between climbing holds (including the bottom and top of the wall). Once they are done placing the holds, how many will there be on average (not including the bottom and top of the wall)?

Extra credit: Now suppose climbers find it just as difficult to move horizontally as vertically, meaning they can't move between any two holds that are more than 1 meter apart in any direction. Suppose also that the climbing wall is a 10-by-10 meter square. If the organizers again place the holds randomly, how many have to be placed on average until it's possible to climb the wall?

Computational Solution 1

Start with an array $[0, height = 10]$ and repeatedly add uniform random values from $\mathcal{U}[0, height]$ to this array (while keeping it **sorted**) until all consecutive differences are less than $d = 1$. Average length of the array across multiple runs gives us the average number of holds that need to be placed. From the simulation below we see that the average number of holds is $\approx$ **43**.

```
from random import uniform

def avg_holds_1d(height, d, runs = 10000):
    sum_hold_cnts = 0
    for _ in range(runs):
        holds = [0, height]
        while any([holds[i+1] - holds[i] > d for i in range(len(holds)-1)]):
            new_hold = uniform(0, height)
            i = 0
            while holds[i] < new_hold:
                i += 1
            holds.insert(i, new_hold)
        sum_hold_cnts += len(holds)-2
    return sum_hold_cnts/runs

print(avg_holds_1d(10.0,1.0))
```

Computational Solution 2

The computational trick to increase the speed of simulation is to use the **Union-Find** algorithm to keep track of all *climbable paths* (i.e. paths without any gaps) identified so far as new holds are randomly added. When we find a climbable path that contains the top and bottom holds, we stop the simulation run.

Using the simulation code below, we see that the average number of holds when there are no vertical gaps greater than 1m is $\approx$ **43** and the average number of holds when there are no gaps(in any direction) greater than 1m is $\approx$ **143**.

```
from networkx.utils.union_find import UnionFind
from random import uniform
from math import sqrt

def dist(p1, p2):
    return sqrt((p1[0]-p2[0])**2 + (p1[1]-p2[1])**2)

def vdist(p1, p2):
    return abs(p1[1]-p2[1])

def avg_holds(height, d, climb_type="1d", runs = 1000):
    dist_fun = vdist if climb_type == "1d" else dist
    sum_holds_cnt = 0
    for _ in range(runs):
        holds, top, bot, cnt = {}, 1, 2, 2
        holds[top], holds[bot] = (0,0), (0,height)
        climbable_paths = UnionFind()
```

```
        climbable_paths.union(top)
        climbable_paths.union(bot)
        while True:
            cnt, new_hold = cnt+1, (uniform(0, height), uniform(0,height))
            for i, hold in holds.items():
                if i == top or i == bot:
                    if vdist(hold, new_hold) <= d:
                        climbable_paths.union(i, cnt)
                elif dist_fun(hold, new_hold) <= d:
                        climbable_paths.union(i, cnt)
            climbable_paths.union(cnt)
            holds[cnt] = new_hold
            if climbable_paths[top]==climbable_paths[bot]:
                break
        sum_holds_cnt += cnt-2
    return sum_holds_cnt/runs

print(avg_holds(10, 1, "1d"))
print(avg_holds(10, 1, "2d"))
```

Chapter 68

Can You Win The Penalty Shootout?

Riddler Express

I have three dogs: Fatch, Fetch and Fitch. Yesterday, I found a brown 12-inch stick for them to play with. I marked the top and bottom of the stick and then threw it for Fatch. Fatch, a Dalmatian, bit it in a random spot — leaving a mark — and returned it to me. In her honor, I painted the stick black from the top to the bite and white from the bottom to the bite.

I subsequently threw the stick for Fetch and then for Fitch, each of whom retrieved the stick by biting a random spot. What is the probability that Fetch and Fitch both bit the same color (i.e., both black or both white)?

Solution

The three bite marks can be permuted in $3! = 6$ ways all of which are *equally* likely because of symmetry. Out of the 6 permutations, there are 4 permutations where the bite marks of Fitch and Fetch lie on one side of the bite mark of Fatch. Therefore, the probability that Fetch and Fitch both bit the same color is **2/3**.

From the simulation below, we see that the probability that Fetch and Fitch both bit the same color is indeed **0.667**.

```
from random import random

def prob(runs = 1000000):
    succ_cnt = 0
    for _ in range(runs):
        fa, fe, fi = random(), random(), random()
```

```
        if (fe < fa and fi < fa) or (fe > fa and fi > fa):
            succ_cnt += 1
    return succ_cnt/runs

print(prob())
```

Riddler Classic

Italy defeated England in a heartbreaking (for England) European Championship that came down to a penalty shootout. In a shootout, teams alternate taking shots over the course of five rounds. If, at any point, a team is guaranteed to have outscored its opponent after five rounds, the shootout ends prematurely, even if each side has not yet taken five shots. If teams are tied after five rounds, they continue one round at a time until one team scores and another misses.

If each player has a 70 percent chance of making any given penalty shot, then how many total shots will be taken on average?

Solution

From the simulation below, we see that the total shots taken on average is **10.47**.

```
def avg_total_shots(p, runs = 1000000):
    from random import random
    shot = lambda p: 1 if random() < p else 0
    def game_over(t1, t2, g1, g2):
        return 1 if max(5-t1, 0) + g1 < g2 or max(5-t2, t1-t2) + g2 < g1 else 0

    sum_ts = 0
    for _ in range(runs):
        t1, t2, g1, g2 = 0, 0, 0, 0
        while True:
            t1, g1 = t1 + 1, g1 + shot(p)
            if game_over(t1, t2, g1, g2):
                break
            t2, g2 = t2 + 1, g2 + shot(p)
            if game_over(t1, t2, g1, g2):
                break
        sum_ts += t1+t2
    return sum_ts/runs

print(avg_total_shots(0.7))
```

Chapter 69

Can You Solve This Astronomical Enigma?

Riddler Express

Earlier this year, a new generation of Brood X cicadas had emerged in many parts of the U.S. This particular brood emerges every 17 years, while some other cicada broods emerge every 13 years. Both 13 and 17 are prime numbers — and relatively prime with one another — which means these broods are rarely in phase with other predators or each other. In fact, cicadas following a 13-year cycle and cicadas following a 17-year cycle will only emerge in the same season once every 221 (i.e., 13 times 17) years!

Now, suppose there are two broods of cicadas, with periods of A and B years, that have just emerged in the same season. However, these two broods can also interfere with each other one year after they emerge due to a resulting lack of available food. For example, if A is 5 and B is 7, then B's emergence in year 14 (i.e., 2 times 7) means that when A emerges in year 15 (i.e., 3 times 5) there won't be enough food to go around.

If both A and B are relatively prime and are both less than or equal to 20, what is the longest stretch these two broods can go without interfering with one another's cycle? (Remember, both broods just emerged this year.) For example, if A is 5 and B is 7, then the interference would occur in year 15.

Solution

Using the code below, we see that the longest stretch the two broods can go without interfering with one another's cycle is **153**.

```
from math import gcd

def max_non_intf(n):
    def min_non_intf(a,b):
        for x in range(0, a*b+1, a):
            for y in range(0, a*b+1, b):
                if abs(x-y)==1:
                    return max(x,y)

    valid_tuples = [(a, b) for a in range(1, n) for b in range(a, n+1)
                    if gcd(a,b) == 1]
    return max([min_non_intf(a,b) for a,b in valid_tuples])

print(max_non_intf(20))
```

Riddler Classic

The astronomers of Planet Xiddler are back!

This time, they have identified three planets that circularly orbit a neighboring star. Planet A is three astronomical units away from its star and completes its orbit in three years. Planet B is four astronomical units away from the star and completes its orbit in four years. Finally, Planet C is five astronomical units away from the star and completes its orbit in five years. They report their findings to Xiddler's Grand Minister, along with the auspicious news that all three planets are currently lined up (i.e., they are collinear) with their star. However, the Grand Minister is far more interested in the three planets than the star and wants to know how long it will be until the planets are next aligned.

How many years will it be until the three planets are again collinear (not necessarily including the star)?

Solution

Let r_1, r_2, r_3 and $\omega_1, \omega_2, \omega_3$ be the radii and the angular velocities of the three planets. The cartesian coordinates of the

planets are $(r_i \cos \omega_i t, r_i \sin \omega_i t)$ where $i \in \{1, 2, 3\}$. When the three planets are **collinear** then

$$\begin{vmatrix} r_1 \cos \omega_1 t & r_1 \sin \omega_1 t & 1 \\ r_2 \cos \omega_2 t & r_2 \sin \omega_2 t & 1 \\ r_3 \cos \omega_3 t & r_3 \sin \omega_3 t & 1 \end{vmatrix} = 0$$

For,

$$(r_1, \omega_1) = (3, 2\pi/3)$$
$$(r_2, \omega_2) = (4, 2\pi/4)$$
$$(r_3, \omega_3) = (5, 2\pi/5)$$

the above determinant reduces to

$$15 \sin \frac{4\pi t}{15} - 12 \sin \frac{\pi t}{6} - 20 \sin \frac{\pi t}{10} = 0$$

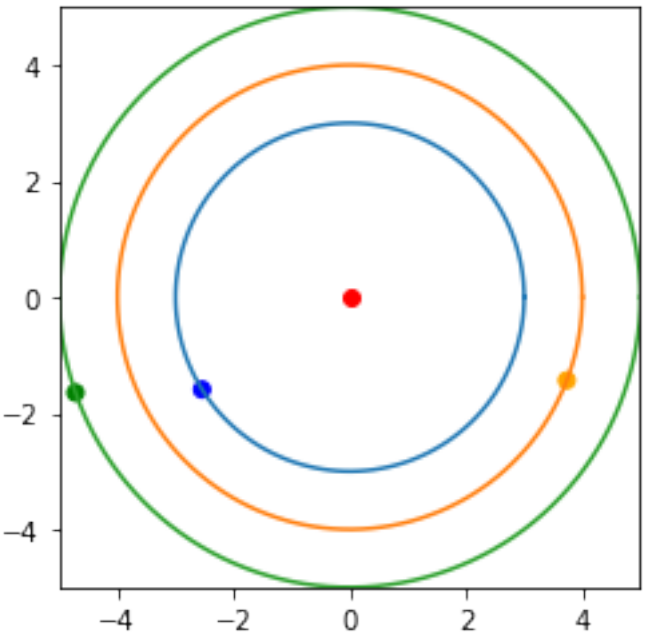
Solving the above equation numerically for t, we get the time period until the three planets are collinear next to be **7.766** years.

The Python code for numerically solving the above equation and plotting the positions of the three planets is given below.

```
from scipy.optimize import fsolve
from math import sin, cos, pi
import numpy as np
import matplotlib.pyplot as plt

def polToCart(r, w, t):
    return (r*cos(w*t), r*sin(w*t))

def circle(radius):
    angle = np.linspace(0, 2*np.pi, 150)
    x = radius*np.cos(angle)
    y = radius*np.sin(angle)
    return (x,y)

planets_r_w = [(3, 2*pi/3), (4, 2*pi/4), (5, 2*pi/5)]

def planetsPositions():
    def func(x):
        return [15.0*sin(4*pi*x/15.0)-20.0*sin(pi*x/10.0)-12.0*sin(pi*x/6.0)]
    root = fsolve(func, [10.0])
    return [polToCart(r, w, root[0]) for r, w in planets_r_w]
```

```
def plotPlanetsPositions(planets_xy):
    figure, axes = plt.subplots(1)
    axes.set_xlim([-5, 5])
    axes.set_ylim([-5, 5])
    axes.set_aspect(1)
    axes.scatter([0], [0], color='red')

    for r,_ in planets_r_w:
        axes.plot(*circle(r))

    planets_x, planets_y = list(zip(*planets_xy))
    axes.scatter(planets_x, planets_y, color=['blue', 'orange', 'green'])
    plt.show()

plotPlanetsPositions(planetsPositions())
```

The diagram below shows the positions of the planets when they are collinear next.



Chapter 70

Can You Decipher The Secret Message?

Riddler Express

Max the Mathemagician is calling for volunteers. He has a magic wand of length 10 that can be broken anywhere along its length (fractional and decimal lengths are allowed). After the volunteer chooses these breakpoints, Max will multiply the lengths of the resulting pieces. For example, if they break the wand near its midpoint and nowhere else, the resulting product is 55, or 25. If the product is the largest possible, they will win a free backstage pass to his next show. (Amazing, right?)

You raise your hand to volunteer, and you and Max briefly make eye contact. As he calls you up to the stage, you know you have this in the bag. What is the maximum product you can achieve?

Extra credit: Zax the Mathemagician (no relation to Max) has the same routine in his show, only the wand has a length of 100. What is the maximum product now?

Solution

Let L be the length of the wand and let $x_1, x_2, \ldots, x_n$ be the lengths of the n resulting pieces after the volunteer chooses the breakpoints. Using the famous **Arithmetic-Geometric mean inequality** $AM \geq GM$, we have

$$\frac{x_1 + \cdots + x_n}{n} \geq (x_1 x_2 \cdots x_n)^{1/n}$$
$$\implies \left(\frac{L}{n}\right)^n \geq x_1 x_2 \cdots x_n$$

The equality is achieved when all the pieces are equal.

The function $f(x) = \left(\frac{L}{x}\right)^x$ attains its maximum value at $x = \frac{L}{\mathrm{e}}$. This can be verified using basic calculus.

As the wand can only be broken into an integral number of pieces, the maximum product is obtained at $n^* = \left\lfloor \frac{L}{\mathrm{e}} \right\rfloor$ or $n^* = \left\lceil \frac{L}{\mathrm{e}} \right\rceil$.

The number of equal pieces required to achieve the maximum product when $L = 10$ is $n^* = \left\lceil \frac{10}{\mathrm{e}} \right\rceil = 4$ and the maximum product that can be achieved is $(10/4)^4 = 39.0625$.

The number of equal pieces required to achieve the maximum product when $L = 100$ is $n^* = \left\lceil \frac{100}{\mathrm{e}} \right\rceil = 37$ and the maximum product that can be achieved is $(100/37)^{37} \approx 9474061716781824$.

Riddler Classic

This week's Classic comes courtesy of Alexander Zhang of Lynbrook High School, California. Alexander won first place in the mathematics category at this year's International Science and Engineering Fair for his work at the intersection of topology and medicine. He developed his own highly efficient algorithms to detect and remove defects (like "handles" or "tunnels") from three-dimensional scans (e.g., MRI). Alexander has long had an interest in topology, which just might be related to his submitted puzzle.

Consider the following image showing a particular uppercase sans serif font:

Alexander thinks many of these letters are equivalent, but he leaves it to you to figure out how and why. He also has a message for you:

It may not look like much, but Alexander assures me that it is equivalent to exactly one word in the English language.

What is Alexander's message?

ABCDEFGHIJKLM

NOPQRSTUVWXYZ

YIRTHA

Solution

Partitioning scheme 1

If alphabets are partitioned into equivalence classes such that two alphabets are in the same class when one can be **continuously deformed** into another, we have the following equivalence classes:

```
1 continous line - C,G,I,J,M,N,S,U,V,W,Z
1 hole with 1 dead end - P,Q
1 hole with 2 dead ends - A,R
2 holes - B
1 hole - D,O
3 dead ends - E,F,T,Y
4 dead ends - H,K,X
```

Partitioning scheme 2

If alphabets are partitioned into equivalence classes based on the number of **non-overlapping continous strokes**, we have the following three equivalence classes:

```
1 continuous stroke - C,D,G,I,J,L,M,N,O,P,Q,S,U,V,W,Z
2 non-overlapping continuous strokes - A,B,E,F,R,T,X,Y
3 non-overlapping continuous strokes - H,K
```

Using the code below, we see that **YIRTHA** can be decoded as **EUREKA** under partitioning scheme 1. Un-

der paritioning scheme 2, **YIRTHA** can be decoded as *APATHY*, *TWEAKY* or *EUREKA*.

```
import urllib.request
DICT_URL = "https://norvig.com/ngrams/enable1.txt"

def load_dictionary(url):
    response = urllib.request.urlopen(url)
    return [l.decode('utf-8').strip().upper() for l in response.readlines()]

ALL_WORDS = load_dictionary(DICT_URL)

target_word = "YIRTHA"
relevant_words = [w for w in ALL_WORDS if len(w)==len(target_word)]
scheme1 = ['AR','B','CGIJMNSUVWZ','DO','EFTY','HKX','PQ']
scheme2 = ['ABEFRTXY','CDGIJLMNOPQSUVWZ','HK']

def decode(grouping):
    target_subgroups = []
    for l in target_word:
        for sg in grouping:
            if l in sg:
                target_subgroups.append(sg)
    matches = set()
    for word in relevant_words:
        if all([c in sg for c, sg in zip(word, target_subgroups)]):
                matches.add(word)
    return matches

print(decode(scheme1))
print(decode(scheme2))
```

Can You Crack The Case Of The Crystal Key?

Riddler Classic

Dakota Jones is back in action! To gain access to a hidden temple deep in the Riddlerian Jungle, she needs a crystal key.

She already knows the crystal is a polyhedron. And according to an ancient text, it has exactly six edges, five of which are 1 inch long. Cryptically, the text does not specify the length of the sixth edge. Instead, it says that the key is the largest such polyhedron (i.e., with six edges, five of which have length 1) by volume.

Once again, Dakota Jones needs your help. What is the volume of the crystal key?

Solution 1

Given the distances between the vertices of a tetrahedron the volume can be computed using the **Cayley–Menger determinant**:

$$288 \cdot V^2 = \begin{vmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & d_{12}^2 & d_{13}^2 & d_{14}^2 \\ 1 & d_{12}^2 & 0 & d_{23}^2 & d_{24}^2 \\ 1 & d_{13}^2 & d_{23}^2 & 0 & d_{34}^2 \\ 1 & d_{14}^2 & d_{24}^2 & d_{34}^2 & 0 \end{vmatrix}$$

where the subscripts $i, j \in 1, 2, 3, 4$ represent the vertices and d_{ij} is the pairwise distance

between them – i.e., the length of the edge connecting the two vertices.

If a, b, c be the three edges that meet at a point, and x, y, z the opposite edges. The volume V of the tetrahedron is given by

$$V = \frac{\sqrt{4a^2b^2c^2 - a^2X^2 - b^2Y^2 - c^2Z^2 + XYZ}}{12}$$

where

$$X = b^2 + c^2 - x^2$$
$$Y = a^2 + c^2 - y^2$$
$$Z = a^2 + b^2 - z^2$$

In our case, let $a = b = c = y = z = 1$, we have

$$X = 2 - x^2$$
$$Y = 1$$
$$Z = 1$$

We need to find the value of x that maximizes

$$V = \frac{\sqrt{4 - (2 - x^2)^2 - 1 - 1 + (2 - x^2)}}{12} = \frac{x\sqrt{-x^2 + 3}}{12}$$

V attains the maximum value of $\frac{1}{8}$ when $x = \sqrt{\frac{3}{2}}$.

Solution 2

A polyhedron with 6 edges has to be a tetrahedron. In this particular case, we have a tetrahedron where two faces are equilateral triangles of side length 1 which share a common edge. The volume of tetrahedron (which is a triangular pyramid where the base is an equilateral triangle) is given by

$$Vol = \frac{1}{3} \times base \times height$$
$$= \frac{1}{3} \times \frac{\sqrt{3}}{4} \times height$$

The volume is maximized when the height is maximized i.e. when the second equilateral triangular face which shares an

edge with the base is perpendicular to the base. The height of an equilateral triangle of side 1 is $\frac{\sqrt{3}}{2}$.

Therefore the volume of the crystal key is

$$Vol_{max} = \frac{1}{3} \times \frac{\sqrt{3}}{4} \times \frac{\sqrt{3}}{2} = \frac{1}{8}$$

References

Wikipedia-Tetrahedron

Chapter 72

No Isosceles Triangles For You!

Riddler Classic

This week's Classic is being served up by Jordan Ellenberg, whose new book, "Shape," goes on sale on May 25.

One of the many geometers who appears in "Shape" is the great Paul Erdős. In 1946, Erdős himself posed a riddle about what are called "isosceles sets": How many points can you place in d-dimensional space such that any three of them form an isosceles triangle?

In two dimensions, the largest isosceles set has exactly six points, with five forming the vertices of a regular pentagon and the sixth point in the middle. In three dimensions, the largest isosceles set has eight points; in four dimensions, the largest set has 11 points. And in dimensions higher than eight, no one knows what the largest isosceles sets might be!

But this week, Jordan is asking about what he calls anti-isosceles sets. Consider a NN grid of points. What is the greatest subset of these N^2 points such that no three of them form an isosceles triangle? (Note: Degenerate triangles, or triangles with zero area, do count as triangles here, as long as their three vertices are distinct.)

How many points are in the largest anti-isosceles set for a 44 grid?

Extra credit: What about a 5×5 grid, a 6×6 grid, or even larger square grids? Can you find an expression (or bounds) for the size of the largest anti-isosceles set for the general $N \times N$ grid? (If you figure out anything about the general case, Jordan would love to hear about it!)

Solution

```
from itertools import product, combinations
from copy import deepcopy

def isoceles(t):
    def d(p1, p2):
        return (p1[0]-p2[0])**2 + (p1[1]-p2[1])**2
    d1, d2, d3 = d(t[0], t[1]), d(t[1], t[2]), d(t[2], t[0])
    if d1 == d2 or d2 == d3 or d3 == d1:
        return True
    else:
        return False

n = 5
grid = product(*[range(n)]*2)
max_size = 6
cur_size = max_size + 1
max_pts_set = []
while True:
    cur_size_largest = False
    pts_set = []
    for pts in combinations(grid, cur_size):
        if all([not isoceles(t) for t in combinations(pts, 3)]):
            cur_size_largest = True
            pts_set.append(pts)
    if not cur_size_largest:
        break
    else:
        max_size = cur_size
        cur_size += 1
        max_pts_set = deepcopy(pts_set)
print(max_size, max_pts_set)
```

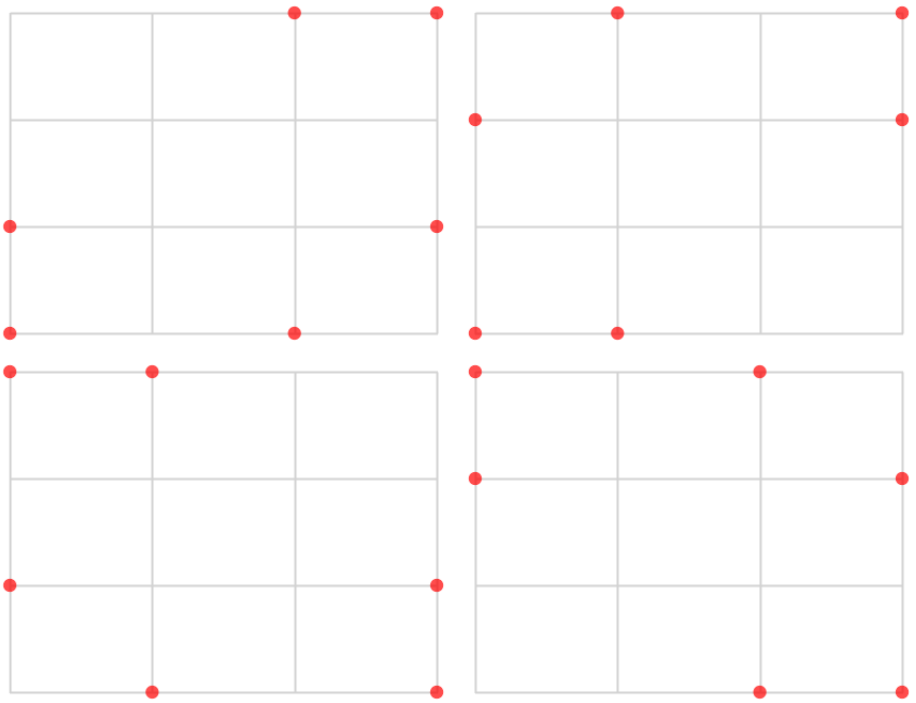


Figure 72.1: Anti-isoceles sets for 4×4 grid

Chapter 73

Can You Systematically Solve A Friday Crossword?

Riddler Express

You're playing a game of cornhole with your friends, and it's your turn to toss the four bean bags. For every bean bag you toss onto your opponents' board, you get 1 point. For every bag that goes through the hole on their board, you get 3 points. And for any bags that don't land on the board or through the hole, you get 0 points.

Your opponents had a terrible round, missing the board with all their throws. Meanwhile, your team currently has 18 points — just 3 points away from victory at 21. You're also playing with a special house rule: To win, you must score exactly 21 points, without going over.

Based on your history, you know there are three kinds of throws you can make:

- An **aggressive throw**, which has a 40 percent chance of going in the hole, a 30 percent chance of landing on the board and a 30 percent chance of missing the board and hole.
- A **conservative throw**, which has a 10 percent chance of going in the hole, an 80 percent chance of landing on the board and a 10 percent chance of missing the board and hole.
- A **wasted throw**, which has a 100 percent chance of missing the board and hole.

For each bean bag, you can choose any of these three throws. Your goal is to maximize your chances of scoring exactly 3 points with your four tosses. What is the probability that your team will finish the round with exactly 21 points and declare victory?

Solution

Let us consider the general case of an N-throw game, and let the state be the *number of points*. The optimal probability $P_k(x_k)$ at the start of the k^{th} throw as a function of the number of points x_k is given by the dynamic programming recursion

$$\begin{aligned} P_k(x_k) = \max[&a_h P_{k+1}(x_k+3) + a_b P_{k+1}(x_k+1) + a_m P_{k+1}(x_k), \\ &c_h P_{k+1}(x_k+3) + c_b P_{k+1}(x_k+1) + c_m P_{k+1}(x_k), \\ &P_{k+1}(x_k)] \end{aligned}$$

where a_h, a_b, a_m and c_h, c_b, c_m are the probabilities of a throw landing in the hole, on the board and missing the board under *aggressive* and *conservative* throws.

The maximum above is taken over the three possible decisions:

- Aggressive throw, which keeps the score at x_k with probability a_m, changes it to $x_k + 3$ with probability a_h and $x_k + 1$ with probability a_b.
- Conservative throw, which keeps the score at x_k with probability c_m, changes it to $x_k + 3$ with probability c_h and $x_k + 1$ with probability c_b.
- Wasted throw, which keeps the score at x_k with probability 1.

The required probability is obtained by calculating $P_1(0)$ given $N = 4$ and the following terminal conditions:

$$\begin{aligned} P_4(0) &= 0.4 \\ P_4(1) &= 0 \end{aligned}$$

$$P_4(2) = 0.8$$
$$P_4(3) = 1$$

From the code below, the probability that the team will finish the round with exactly 21 points and declare victory is 85.48%.

Computational solution

Chapter 74

Can You Cut The Perfect Pancake?

Riddler Classic

Riddler Nation's neighbor to the west, Enigmerica, is holding an election between two candidates, A and B. Assume every person in Enigmerica votes randomly and independently, and that the number of voters is very, very large. Moreover, due to health precautions, 20 percent of the population decides to vote early by mail.

On election night, the results of the 80 percent who voted on Election Day are reported out. Over the next several days, the remaining 20 percent of the votes are then tallied.

What is the probability that the candidate who had fewer votes tallied on election night ultimately wins the race?

Solution

The probability that the candidate who had fewer votes tallied on election night ultimately wins the race is approximately 14.82.

```
using Distributions
runs = 100000
frac_day = 0.8
frac_mail = 1 - frac_day
num_total = 1000000
num_day, num_mail = trunc(Int32, frac_day*num_total), trunc(Int32, frac_mail*num_total)
succ = 0
for i in 1:runs
        votes = rand(Bernoulli(0.5), num_total)
        if sum(votes[1:num_day]) < 0.5*num_day && sum(votes) > 0.5*num_total
                succ += 1
        end
```

```
end
print(2*succ/runs)
```

Chapter 75

Can You Crack The Case Of The Crescent Moon?

Riddler Express

You are creating a variation of a Romulan pixmit deck. Each card is an equilateral triangle, with one of the digits 0 through 9 (written in Romulan, of course) at the base of each side of the card. No number appears more than once on each card. Furthermore, every card in the deck is unique, meaning no card can be rotated so that it matches (i.e., can be superimposed on) any other card.

What is the greatest number of cards your pixmit deck can have?

Extra credit: Suppose you allow numbers to appear two or three times on a given card. Once again, no card can be rotated so that it matches any other card. Now what is the greatest number of cards your pixmit deck can have?

Solution

The number of ways of choosing 3 distinct digits from the 10 digits for each card is $\binom{10}{3}$.Once three digits have been chosen for a card, they can be permuted in $3! = 6$ ways. These 6 permutations can be grouped into two equivalence classes of three permutations each such that the permutations in each class are rotations of other permutations in that class. Therefore, the greatest number of cards in the deck is $2\binom{10}{3} = 240$.

Extra credit

Case 1: When a digit can be repeated thrice in each card.

We choose one digit for each card and this can be done in 10 ways.

Case 2: When two digits are chosen for each card and one of them is repeated twice.

Number of ways of choosing 2 digits from 10 digits is $\binom{10}{2}$. Once two digits have chosen for a card, there is only one way of assigning digits to each side of the card (ignoring rotations). Therefore, the number of cards in this case is $2\binom{10}{2}$.

The greatest number of cards in the deck after allowing the digits to be repeated is $240 + 10 + 90 = 340$.

Chapter 76

Can You Navigate The One-Way Streets?

Riddler Express

You have two 16-ounce cups — cup A and cup B. Both cups initially have 8 ounces of water in them. You take half of the water in cup A and pour it into cup B. Then, you take half of the water in cup B and pour it back into cup A. You do this again. And again. And again. And then many, many, many more times — always pouring half the contents of A into B, and then half of B back into A. When you finally pause for a breather, what fraction of the total water is in cup A?

Extra credit: Now suppose both cups initially have somewhere between 0 and 8 ounces of water in them. You don't know the precise amount in each cup, but you know that both cups are not empty. Again, you pour half the water from cup A into cup B, and then half from cup B back to A. You do this many, many times. When you again finally pause for a breather, what fraction of the total water is in cup A?

Solution

The fraction of the total water in Cup A is always $\frac{2}{3}$.

```
function fracA(a, b)
        for i in 1:1000000
                a = a/2
                b += a
                b = b/2
                a += b
        end
```

```
        (a, b, a/(a+b))
end

println(fracA(8,8))
println(fracA(8*(1-rand()),8*(1-rand())))
```

Riddler Classic

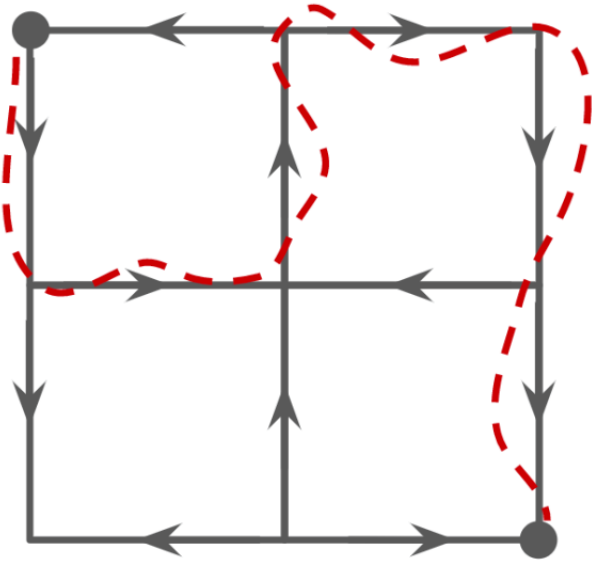
In Riddler City, all the streets are currently two-way streets. But in an effort to make the metropolis friendlier for pedestrians and cyclists, the mayor has decreed that all streets should be one-way. Meanwhile, the civil engineer overseeing this transition is not particularly invested in the project and will be randomly assigning every block of each street a random direction.

For your daily commute to work, you drive a car two blocks east and two blocks south, as shown in the diagram below. What is the probability that, after each block is randomly assigned a one-way direction, there will still be a way for you to commute to work while staying within this two-by-two block region (i.e., sticking to the 12 streets you see in the diagram)? Here is one such arrangement of one-way streets that lets you commute to work:

Solution

The probability that, after each block is randomly assigned a one-way direction, there will still be a way for you to commute to work while staying within this two-by-two block region is 0.277. The probability of being able to perform a round trip is 0.0415.

```
using LightGraphs
import Base.Iterators: product

road_combs = product([(1,2),(2,1)],[(2,3),(3,2)],[(1,4),(4,1)],
                     [(2,5),(5,2)],[(3,6),(6,3)],[(4,5),(5,4)],
                     [(5,6),(6,5)],[(4,7),(7,4)],[(7,8),(8,7)],
                     [(5,8),(8,5)],[(6,9),(9,6)],[(8,9),(9,8)])
cnt, succ, succ_rt = 0, 0, 0
for roads in road_combs
        g = SimpleDiGraph(9)
        for i in 1:12
```

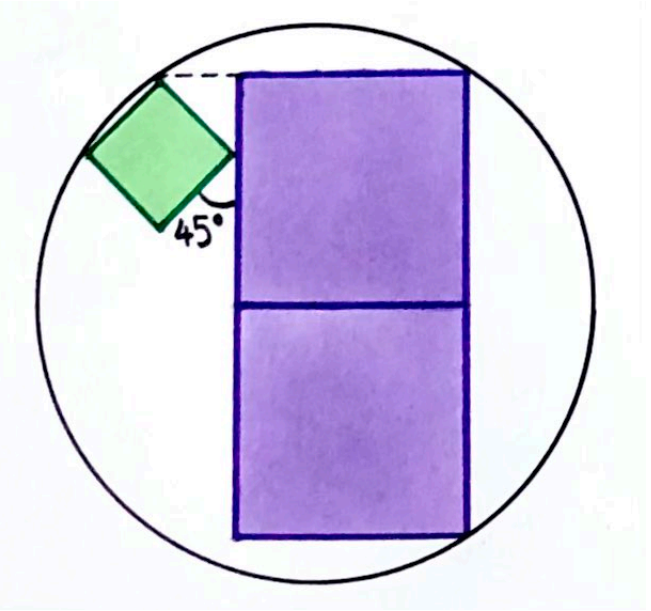


```
                add_edge!(g, roads[i][1], roads[i][2])
        end
        cnt +=1
        if has_path(g, 1, 9)
                succ += 1
                if has_path(g, 9, 1)
                        succ_rt += 1
                end
        end
end
println(succ/cnt,"\t", succ_rt/cnt)
```

Chapter 77

Can You Find An Extra Perfect Square?

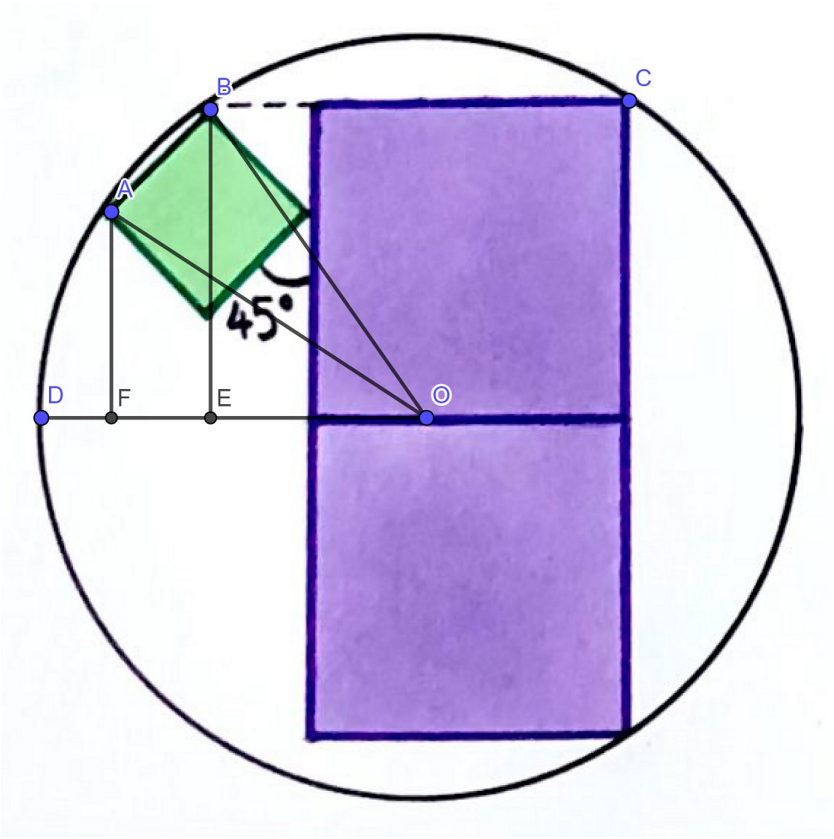
Riddler Express



The two larger squares are congruent, and the smaller square makes a 45 degree angle with one of the larger squares. Both larger squares touch the circle at one corner, while the smaller square touches the circle at two corners.

How many times greater is the area of one of the larger squares than the area of the smaller square?

Solution



Let O be the center of the circle, x be the length of the side of the larger square and y be the length of the side of the smaller square.

We have the following equations:

$$OE = BC/2 = \frac{1}{2}(x+\frac{y}{\sqrt{2}})BE = xAF = BE-\frac{y}{\sqrt{2}}OF = OE+\frac{y}{\sqrt{2}}OA = OBOA^2 = OF^2+AF^2C$$

From the above, we have

$$-\frac{y^2}{2} + 2x\frac{y}{\sqrt{2}} = \frac{y^2}{2} + 2\frac{1}{2}(x + \frac{y}{\sqrt{2}})\frac{y}{\sqrt{2}}$$
$$\implies \frac{x}{y} = \frac{3}{\sqrt{2}}$$

$$\implies \frac{x^2}{y^2} = \frac{9}{2}$$

Riddler Classic

For some perfect squares, when you remove the last digit, you get another perfect square. For example, when you remove the last digit from $256(16^2)$, you get $25(5^2)$.

The first few squares for which this happens are 16, 49, 169, 256 and 361. What are the next three squares for which you can remove the last digit and get a different perfect square? How many more can you find?

Extra credit: In the list above, $169(13^2)$ is a little different from the other numbers. Not only when you remove the last digit do you get a perfect square, $16(4^2)$, but when you remove the last two digits, you again get a perfect square: $1(1^2)$. Can you find another square with both of these properties?

Solution

Three more perfect squares that satisfy the conditions of the problem are 1444, 3249 and 18496.

```
from math import sqrt
cnt = 1
for i in range(20, 1000):
    if cnt <= 3 and sqrt(int(str(i*i)[:-1])).is_integer():
        print(i*i)
        cnt += 1
```

Chapter 78

Can You Bake The Biggest Pie?

Riddler Classic

This Sunday, March 14, is Pi Day! To celebrate, you are planning to bake a pie. You have a sheet of crust laid out in front of you. After baking, your pie crust will be a cylinder of uniform thickness (or rather, thinness) with delicious filling inside.

To maximize the volume of your pie, what fraction of your crust should you use to make the circular base (i.e., the bottom) of the pie?

Solution

Assuming we use the entire sheet of crust without wasting to create the hollow cylinder with uniform thickness. We need to maximize the volume $\pi r^2 h$ subject to the constraint $2\pi rh + 2\pi r^2$ is constant.

We have the Lagrangian,

$$\mathcal{L}(r,h) = \pi r^2 h - \lambda(2\pi rh + 2\pi r^2 - k) \tag{78.1}$$

Taking the partial derivatives of the Lagrangian w.r.t h and r and setting them to 0, we have

$$\frac{\partial \mathcal{L}(r,h)}{\partial r} = 2\pi rh - \lambda(2\pi h + 4\pi r) = 0 \tag{78.2}$$

$$\frac{\partial \mathcal{L}(r,h)}{\partial h} = \pi r^2 - 2\lambda\pi r = 0 \tag{78.3}$$

Solving the above simultaneous equations, we have

$$\lambda = \frac{r}{2} \tag{78.4}$$

$$h = 2r \tag{78.5}$$

The fraction of the crust that should be used to make the circular base so as to maximize the volume of the pie is $\frac{\pi r^2}{2\pi rh+2\pi r^2} = \frac{1}{6}$.

Chapter 79

Can You Bat .299 In 299 Games?

Riddler Express

You have three coins in your pocket, each of which can be a penny, nickel, dime or quarter with equal probability. You might have three different coins, three of the same coin or two coins that are the same and one that is different.

Each of these coins can buy you a string whose length in centimeters equals the value of the coin in cents, i.e., the penny buys 1 cm of string, the nickel buys 5 cm of string, etc. After purchasing your three lengths of string, what is the probability that they can be the side lengths of a triangle?

Solution 1

We first look at all possible combinations of three coins. As all the combinations are equally likely, we divide the number of combinations where the three coin values form a triangle by the total number of combinations. The probability that the purchased strings can be the side lengths of a triangle is 0.34375.

```
from itertools import product

coins = [1,5,10,25]
succ = 0
coin_combs = list(product(*[coins]*3))
for x,y,z in coin_combs:
    if x + y > z and x + z > y and y + z > x:
        succ += 1
print(succ/len(coin_combs))
```

Solution 2

For each run of the Monte Carlo simulation, we choose 3 coins randomly with replacement from the set $\{1, 5, 10, 25\}$ and check if a triangle can be formed with the coin values. The average count of successes is the probability that the purchased strings can be the side lengths of a triangle. The probability based on the simulation below is 0.343.

```
from random import choices

coins = [1,5,10,25]
runs = 100000
succ = 0
for _ in range(runs):
    x,y,z = choices(coins, k=3)
    if x + y > z and x + z > y and y + z > x:
        succ += 1
print(succ/runs)
```

Riddler Classic

Suppose a baseball player has four at-bats per game (not including walks), so their batting average is the number of hits they got divided by four times the number of games they played. For many games, it's possible to have a corresponding batting average that, when rounded to three digits, equals the number of games divided by $1,000$. For example, if a player typically gets one hit per game in their at-bats, then they could very well have a $.250$ average over 250 games.

What is the greatest number of games for which it is not possible to have a matching rounded batting average? Again, assume four at-bats per game.

Solution

As per the code below, the number of games for which it is not possible to have a matching rounding batting average is 239.

```
for g in range(1000, 1, -1):
    if all([round(h/(4*g),3) != g/1000 for h in range(0, 4*g+1)]):
        break
print(g)
```

Chapter 80

How Many Ways Can You Build A Staircase?

Riddler Express

This is the fourth and final week of CrossProduct™ puzzles — for now. This time, there are four four-digit numbers — each belongs in a row of the table below, with one digit per cell. The products of the four digits of each number are shown in the rightmost column. Meanwhile, the products of the digits in the thousands, hundreds, tens and ones places, respectively, are shown in the bottom row. Can you find all four four-digit numbers and complete the table?

Solution

```
from z3 import *

class MysteriousNumbersSolver:
    def __init__(self):
        self.X = [[Int("self.X_%s_%s" % (i, j)) for j in range(4)]
                      for i in range(4)]
        self.s = Solver()
        self.s.add([And(0 <= self.X[i][j], self.X[i][j]<= 9) for i in range(4) for j in ran
        self.s.add([And(0 <= self.X[i][0], self.X[i][0]<= 9) for i in range(4)])

    def set_constraints(self):
        self.s.add(self.X[0][0]*self.X[0][1]*self.X[0][2]*self.X[0][3]==1458)
        self.s.add(self.X[1][0]*self.X[1][1]*self.X[1][2]*self.X[1][3]==128)
        self.s.add(self.X[2][0]*self.X[2][1]*self.X[2][2]*self.X[2][3]==2688)
        self.s.add(self.X[3][0]*self.X[3][1]*self.X[3][2]*self.X[3][3]==125)
        self.s.add(self.X[0][0]*self.X[1][0]*self.X[2][0]*self.X[3][0]==960)
```

```
            self.s.add(self.X[0][1]*self.X[1][1]*self.X[2][1]*self.X[3][1]==384)
            self.s.add(self.X[0][2]*self.X[1][2]*self.X[2][2]*self.X[3][2]==630)
            self.s.add(self.X[0][3]*self.X[1][3]*self.X[2][3]*self.X[3][3]==270)

        def output_solution(self):
            m = self.s.model()
            for i in range(4):
                print(" ".join([str(m.evaluate(self.X[i][j])) for j in range(4)]))

        def solve(self):
            self.set_constraints()
            if self.s.check() == sat:
                self.output_solution()
            else:
                print(self.s)
                print("Failed to solve.")

s = MysteriousNumbersSolver()
s.solve()
```

Here is the solution:

```
3 6 9 9
8 8 2 1
8 8 7 6
5 1 5 5
```

Riddler Classic

You have 10 blocks with which to build four steps against a wall. The first step is one block high, the second is two blocks high, the third is three blocks high and the fourth is four blocks high.

However, the ground ever-so-slightly slopes down toward the wall, and both the floor and the blocks are a little bit slippery. As a result, whenever you place a block at ground level, it slides toward the wall until it hits the wall or another block. And when you place a block atop another block, it will similarly slide toward the wall until it hits the wall or another block.

Suppose the four blocks in the bottom row are labeled A, the three blocks in the second row are labeled B, the two blocks in the next row are labeled C and the topmost block is labeled D. One way to build the steps would be to place the blocks in the following order, one row at a time: A-A-A-A-B-B-B-C-C-D. You could alternatively place the blocks one column at a time:

A-B-C-D-A-B-C-A-B-A. But you could not place them in the order A-B-B-A-A-A-B-C-C-D because that would mean at one point you have more blocks in the second row, B, than in the bottom row, A — a physical impossibility!

How many distinct ways are there to build these four steps using the 10 blocks?

Extra credit: Suppose you have precisely enough blocks to build a staircase with N stairs. How many distinct ways are there to build this staircase?

Solution

Here is a brute force computational solution which gives 768 ways to build the staircase with 4 steps and 10 blocks.

```
from itertools import permutations

def valid_block_seqs(n):
    blocks = [(i,j) for i in range(1,n+2) for j in range(1, n-i+2)]
    sequences = []
    for seq in permutations(blocks[1:]):
        used = set([(1,1)] + [(0,i) for i in range(1, n+1)] + [(i,0) for i in range(1,n+1)]
        for (x,y) in seq:
            if (x-1, y) in used and (x, y-1) in used:
                used.add((x, y))
            else:
                break
        if len(used) == len(blocks) + 2*n:
            sequences.append([(1,1)] + list(seq))
    return sequences

print(len(valid_block_seqs(4)))
```

Can You Win Riddler Jenga?

Riddler Express

This time, there are seven three-digit numbers — each belongs in a row of the table below, with one digit per cell. The products of the three digits of each number are shown in the rightmost column. Meanwhile, the products of the digits in the hundreds, tens and ones places, respectively, are shown in the bottom row.

183,708	245,760	117,600	

Can you find all seven three-digit numbers and complete the table?

Solution

```
from z3 import *

class MysteriousNumbersSolver:
    def __init__(self):
        self.X = [[Int("self.X_%s_%s" % (i, j)) for j in range(3)]
```

```
                    for i in range(7)]
        self.s = Solver()
        self.s.add([And(0 <= self.X[i][j], self.X[i][j]<= 9) for i in range(6) for j in ran
        self.s.add([And(0 <= self.X[i][0], self.X[i][0]<= 9) for i in range(6)])

    def set_constraints(self):
        self.s.add(self.X[0][0]*self.X[0][1]*self.X[0][2]==280)
        self.s.add(self.X[1][0]*self.X[1][1]*self.X[1][2]==168)
        self.s.add(self.X[2][0]*self.X[2][1]*self.X[2][2]==162)
        self.s.add(self.X[3][0]*self.X[3][1]*self.X[3][2]==360)
        self.s.add(self.X[4][0]*self.X[4][1]*self.X[4][2]==60)
        self.s.add(self.X[5][0]*self.X[5][1]*self.X[5][2]==256)
        self.s.add(self.X[6][0]*self.X[6][1]*self.X[6][2]==126)
        self.s.add(self.X[0][0]*self.X[1][0]*self.X[2][0]*self.X[3][0]*self.X[4][0]*self.X[
        self.s.add(self.X[0][1]*self.X[1][1]*self.X[2][1]*self.X[3][1]*self.X[4][1]*self.X[
        self.s.add(self.X[0][2]*self.X[1][2]*self.X[2][2]*self.X[3][2]*self.X[4][2]*self.X[

    def output_solution(self):
        m = self.s.model()
        for i in range(7):
            print(" ".join([str(m.evaluate(self.X[i][j])) for j in range(3)]))

    def solve(self):
        self.set_constraints()
        if self.s.check() == sat:
            self.output_solution()
        else:
            print(self.s)
            print("Failed to solve.")

s = MysteriousNumbersSolver()
s.solve()
```

Here is the solution:

```
7 8 5
3 8 7
9 6 3
9 8 5
3 5 4
4 8 8
9 2 7
```

Riddler Classic

In the game of Jenga, you build a tower and then remove its blocks, one at a time, until the tower collapses. But in Riddler

Jenga, you start with one block and then place more blocks on top of it, one at a time.

All the blocks have the same alignment (e.g., east-west). Importantly, whenever you place a block, its center is picked randomly along the block directly beneath it. On average, how many blocks must you place so that your tower collapses — that is, until at least one block falls off?

(Note: This problem is not asking for the average height of the tower after any unbalanced blocks have fallen off. It is asking for the average number of blocks added in order to make the tower collapse in the first place.)

Solution

The key observation is that at some point when the next jenga block is placed on top of the current *stable* set of blocks, you get a *composite block* (comprising of two or more contiguous blocks from the top) such that the x-coordinate of its center of mass lies outside of the *base*, i.e. falls outside of the block just below the composite block. The resulting torque topples *all* the blocks which are a part of the composite block. As per the simulation below, the average number of blocks that one must place in the tower such that atleast one block falls of is 7.1.

```
from random import uniform

runs = 100000
total_len = 0
for _ in range(runs):
    run_len = 1
    centers = [0]
    brk = False
    while True:
        centers.append(centers[-1] + uniform(-1,1))
        run_len += 1
        for i in range(run_len-1, 0, -1):
            cm_block = sum(centers[i-run_len:])/(run_len-i)
            if  cm_block < centers[i-1]-1 or cm_block > centers[i-1]+1:
                total_len += run_len
                brk = True
                break
        if brk:
            break
```

```
print(total_len/runs)
```

Chapter 82

Can you find the luckiest coin?

Riddler Express

It's time for a random number duel! You and I will both use random number generators, which should give you random real numbers between 0 and 1. Whoever's number is greater wins the duel!

There's just one problem. I've hacked your random number generator. Instead of giving you a random number between 0 and 1, it gives you a random number between 0.1 and 0.8.

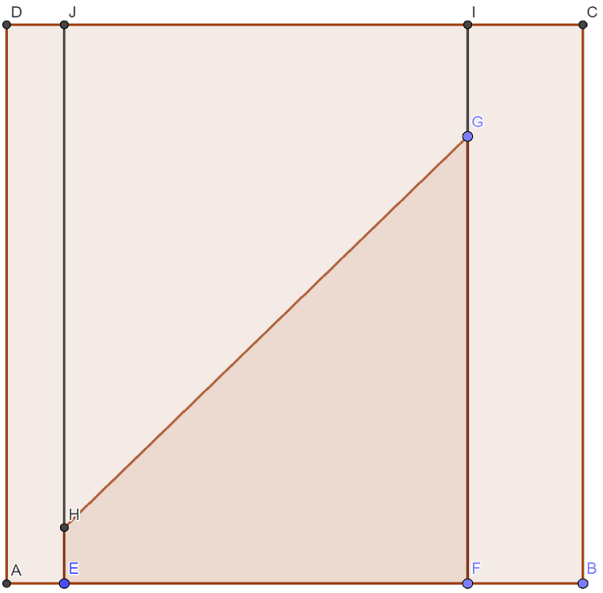
What are your chances of winning the duel?

Solution

The random generator that I have generates numbers according to $X \sim \mathcal{U}[a, b]$ where $0 \leq a \leq b \leq 1$ and the random generator that you have (in the general case) generates numbers according to $Y \sim \mathcal{U}[0, 1]$. We need the probability, $\mathbb{P}[X < Y]$. This is given by the area of the trapezium $EFGH$ divided by the area of the rectangle $EFIJ$. We have $|EF| = |IJ| = b - a$, $|FG| = b$, $|EH| = a$ and $|EJ| = 1$ in the figure below

$$\mathbb{P}[X < Y] = \frac{\frac{1}{2}(b-a)(b+a)}{(b-a)} = \frac{b+a}{2}.$$

In our particular case $a = 0.1$ and $b = 0.8$, therefore the probability of me winning is **0.45**.



Computational validation

The probability of me winning the duel as per the simulation below is **0.45** which validates the result we got earlier.

```
from numpy.random import uniform

def prob_win(p1l, p1h, p2l, p2h, runs = 1000000):
    total_wins = 0
    for _ in range(runs):
        x, y = uniform(p1l, p1h), uniform(p2l, p2h)
        if x > y:
            total_wins += 1
    return total_wins/runs

print(prob_win(0.1,0.8,0,1))
```

Riddler Classic

I have in my possession 1 million fair coins. Before you ask, these are not legal tender. Among these, I want to find the "luckiest" coin.

I first flip all 1 million coins simultaneously (I'm great at multitasking like that), discarding any coins that come up

tails. I flip all the coins that come up heads a second time, and I again discard any of these coins that come up tails. I repeat this process, over and over again. If at any point I am left with one coin, I declare that to be the "luckiest" coin.

But getting to one coin is no sure thing. For example, I might find myself with two coins, flip both of them and have both come up tails. Then I would have zero coins, never having had exactly one coin.

What is the probability that I will — at some point — have exactly one "luckiest" coin?

Solution

Let $\mathbb{P}[N]$ be the probability that we will have exactly one 'luckiest' coin starting with N coins. We have the following recurrence relation:

$$\begin{aligned} \mathbb{P}[N] &= \sum_{i=1}^{N} \mathbb{P}[i] \frac{\binom{N}{i}}{2^N} \\ \implies \mathbb{P}[N](1 - \frac{1}{2^N}) &= \sum_{i=1}^{N-1} \mathbb{P}[i] \frac{\binom{N}{i}}{2^N} \\ \implies \mathbb{P}[N] &= \frac{1}{2^N - 1} \sum_{i=1}^{N-1} \mathbb{P}[i] \binom{N}{i} \end{aligned}$$

because the probability of ending up with i heads when you flip N coins is $\frac{\binom{N}{i}}{2^N}$ and then it is equivalent to starting the game with i coins. When $N = 1$, $\mathbb{P}[N] = 1$.

Computational solution

From the code below, we get $\mathbb{P}[100] \approx \mathbf{0.7214}$. Assuming convergence, $\mathbb{P}[1000000] \approx \mathbf{0.7214}$

```
from functools import lru_cache
from math import comb

def prob_luckiest_coin(n):
    @lru_cache()
    def prob(n):
```

```
        if n == 1:
            return 1
        else:
            total_prob = 0
            for i in range(1, n):
                total_prob += prob(i)*comb(n,i)
        return total_prob/(2**n-1)
    return prob(n)

print(prob_luckiest_coin(100))
```

Chapter 83

Work A Shift In The Riddler Gift Shop

Riddler Express

You take half of a vitamin every morning. The vitamins are sold in a bottle of 100 (whole) tablets, so at first you have to cut the tablets in half. Every day you randomly pull one thing from the bottle — if it's a whole tablet, you cut it in half and put the leftover half back in the bottle. If it's a half-tablet, you take the vitamin. You just bought a fresh bottle. How many days, on average, will it be before you pull a half-tablet out of the bottle?

Extra credit: What if the halves are less likely to come up than the full tablets? They are smaller, after all.

Solution

```
from random import random

runs = 100000
init_full_tablets = 100
num_days_till_half_tablet = 0

for _ in range(runs):
    num_full_tablets = init_full_tablets
    num_half_tablets = 0
    while(True):
        prob_full_tablet = num_full_tablets/(num_full_tablets + num_half_tabl
        prob_half_tablet = 1 - prob_full_tablet
        num_days_till_half_tablet += 1
        if random() <= prob_full_tablet:
            num_full_tablets -= 1
            num_half_tablets += 1
        else:
```

```
            num_half_tablets -= 1
            break

print(num_days_till_half_tablet/runs)
```

On average, it will take 13.2 days to pull a half-tablet out of the bottle.

Chapter 84

It's Elementary, My Dear Riddler!

You have four fair tetrahedral dice whose four sides are numbered 1 through 4.

You play a game in which you roll them all and divide them into two groups: those whose values are unique, and those which are duplicates. For example, if you roll a 1, 2, 2 and 4, then the 1 and 4 will go into the "unique" group, while the 2*s* will go into the "duplicate" group.

Next, you reroll all the dice in the duplicate pool and sort all the dice again. Continuing the previous example, that would mean you reroll the 2*s*. If the result happens to be 1 and 3, then the "unique" group will now consist of 3 and 4, while the "duplicate" group will have two 1s.

You continue rerolling the duplicate pool and sorting all the dice until all the dice are members of the same group. If all four dice are in the "unique" group, you win. If all four are in the "duplicate" group, you lose.

What is your probability of winning the game?

Solution

The probability of winning the game is **0.45**.

```
from random import choice
from collections import Counter

def win_prob(n, s, runs=100000):
    def roll(s,n):
        return [choice(range(1, s+1)) for _ in range(n)]

    wins = 0
    for i in range(runs):
```

```
        throw = roll(s, n)
        while True:
            ctr = Counter(throw)
            if len(ctr.keys()) == n:
                wins += 1
                break
            if all([True if c > 1 else False for c in ctr.values()]):
                break
            throw = []
            for k,v in ctr.items():
                if v == 1:
                    throw.append(k)
                else:
                    throw += roll(s, v)
    return wins/runs
print(win_prob(4, 4))
```

Chapter 85

Can You Fold All Your Socks?

In my laundry basket, I have 14 pairs of socks that I need to pair up. To do this, I use a chair that can fit nine socks, at most. I randomly draw one clean sock at a time from the basket. If its matching counterpart is not already on the chair, then I place it in one of the nine spots. But if its counterpart is already on the chair, then I remove it from the chair (making that spot once again unoccupied) and place the folded pair in my drawer.

What is the probability I can fold all 14 pairs without ever running out of room on my chair?

Extra credit: What if I change the number of pairs of socks I own, as well as the number of socks that can fit on my chair?

Computational Solution

The probability that you can fold all 14 pairs without ever running out of room on my chair is **0.7**.

```
from random import choice

def succ_prob(n_pairs, n_slots, runs=100000):
    fail_cnt = 0
    for _ in range(runs):
        basket = set((i,t) for i in range(n_pairs) for t in [0,1])
        chair = set()
        while basket:
            (i,t) = choice(list(basket))
            basket.remove((i,t))
            if (i,1-t) in chair:
                chair.remove((i,1-t))
            else:
                chair.add((i,t))
            if len(chair)>n_slots:
                fail_cnt += 1
```

```
                break
    return 1- (fail_cnt/runs)

print(succ_prob(14,9))
```

Chapter 86

Can You Salvage Your Rug?

✠

Today I happen to be celebrating the birthday of a family member, which got me wondering about how likely it is for two people in a room to have the same birthday.

Suppose people walk into a room, one at a time. Their birthdays happen to be randomly distributed throughout the 365 days of the year (and no one was born on a leap day). The moment two people in the room have the same birthday, no more people enter the room and everyone inside celebrates by eating cake, regardless of whether that common birthday happens to be today.

On average, what is the expected number of people in the room when they eat cake?

Extra credit: Suppose everyone eats cake the moment three people in the room have the same birthday. On average, what is this expected number of people?

Computational Solution

The expected number of people in the room when they eat cake is **23.67**.

```
from random import choice

def avg_num_people(runs=100000):
    ppl_cnt = 0
    for _ in range(runs):
        people_in = set()
        while True:
            new_person = choice(range(365))
            if new_person in people_in:
                break
```

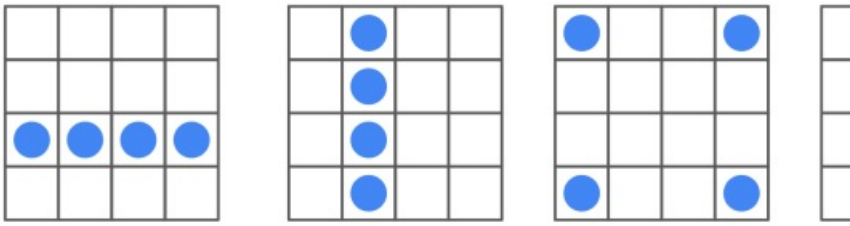
```
            people_in.add(new_person)
        ppl_cnt += len(people_in)
    return ppl_cnt/runs
print(avg_num_people())
```

Chapter 87

Can You Make The Fidget Spinner Go Backwards?

A thousand people are playing Lotería, also known as Mexican bingo. The game consists of a deck of 54 cards, each with a unique picture. Each player has a board with 16 of the 54 pictures, arranged in a 4-by-4 grid. The boards are randomly generated, such that each board has 16 distinct pictures that are equally likely to be any of the 54.

During the game, one card from the deck is drawn at a time, and anyone whose board includes that card's picture marks it on their board. A player wins by marking four pictures that form one of four patterns, as exemplified below: any entire row, any entire column, the four corners of the grid and any 2-by-2 square.



Four four-by-four grids are shown. In the first grid, the third row has four blue markers. In the second grid, the second column has four blue markers. In the third grid, the four corner squares are marked. And in the fourth grid, the two middle squares in the third and fourth columns are marked, forming a smaller two-by-two square. After the

fourth card has been drawn, there are no winners. What is the probability that there will be exactly one winner when the fifth card is drawn?

Solution

Using the Python code below for Monte Carlo simulation, we see that the required probability is **0.181**.

```
from numpy import array, vstack, apply_along_axis
from numpy.random import choice

def cond_probability(n=1000, d=5, runs=1000):
    def win(board, cards):
        win_patches = [[0,1,2,3], [4,5,6,7], [8,9,10,11], [12,13,14,15],
                       [0,4,8,12], [1,5,9,13],[2,6,10,14], [3,7,11,15],
                       [0,3,12,15], [0,1,4,5], [1,2,5,6], [2,3,6,7],
                       [4,5,8,9], [5,6,9,10], [6,7,10,11], [8,9,12,13],
                       [9,10,13,14], [10,11,14,15]]
        return any([set(board[tiles]).issubset(cards) for tiles in win_patches])

    cnt_not_match4, cnt_match5 = 0, 0
    for _ in range(runs):
        rand_boards = vstack([choice(range(54), 16, replace=False) for _ in range(n)])
        drawn_cards = choice(range(54), d, replace=False)
        if not any(apply_along_axis(win, 1, rand_boards, drawn_cards[0:4])):
            cnt_not_match4 += 1
            if sum(apply_along_axis(win, 1, rand_boards, drawn_cards))==1:
                cnt_match5 += 1
    return cnt_match5/cnt_not_match4

print(cond_probability())
```

Chapter 88

When Will The Fall Colors Peak?

Riddler Express

The end of daylight saving time here on the East Coast of the U.S. got me thinking more generally about the calendar year. Each solar year consists of approximately 365.24217 mean solar days. That's pretty close to 365.25, which is why it makes sense to have an extra day every four years. However, the Gregorian calendar is a little more precise: There are 97 leap years every 400 years, averaging out to 365.2425 days per year. Can you make a better approximation than the Gregorian calendar? Find numbers L and N (where N is less than 400) such that if every cycle of N years includes L leap years, the average number of days per year is as close as possible to 365.24217.

Solution

The brute-force search using the code below gives us $N = 351$ and $L = 85$.

```
min_err, N, L = 1, None, None
for n in range(1, 400):
    for l in range(1,n):
        err = abs(365.24217*n - 365*n - l)
        if  err < min_err:
            min_err, N, L = err, n, l
print(N, L, min_err)
```

Riddler Classic

It's peak fall foliage season in Riddler Nation, where the trees change color in a rather particular way. Each tree independently begins changing color at a random time between the autumnal equinox and the winter solstice. Then, at a random later time for each tree — between when that tree's leaves began changing color and the winter solstice — the leaves of that tree will all fall off at once.At a certain time of year, the fraction of trees with changing leaves will peak. What is this maximal fraction?

Solution

The maximal fraction as per the simulation below is **.367**.

```
import numpy as np
from random import choice
def maximal_fraction(n_tps = 10000, n_trees= 100000):
    trees = np.zeros((n_trees, n_tps))
    for i in range(n_trees):
        s = choice(range(n_tps-1))
        e = choice(range(s+1, n_tps))
        trees[i,s:e] = 1
    return np.max(np.sum(trees, axis=0))/n_trees

print(maximal_fraction())
```

Chapter 89

Can you knock down the gates?

I have made a square peanut butter and jelly sandwich, and now it's time to slice it. But rather than making a standard horizontal or diagonal cut, I instead pick two random points along the perimeter of the sandwich and make a straight cut from one point to the other. (These points can be on the same side.) My slice is "reasonable" if I cut the square into two pieces and the smaller resulting piece has an area that is at least one-quarter of the whole area. What is the probability that my slice is reasonable?

Solution

Without loss of generality, we can assume that the square is a unit square. To select two random points on the perimeter of the square, we first select two sides with **replacement** and then select a random point on each side. We now have to consider three cases.

Two points on the same side

The probability of this event is $\frac{4}{16} = \frac{1}{4}$. It is easy to see that in this case, the slice can never be reasonable.

Two points on adjacent sides

The probability of this event is $\frac{8}{16} = \frac{1}{2}$. In this case, the smaller piece is a **triangle** whose vertices are the two selected points and the vertex of the square that is common to the two selected adjacent sides. We need the probability that the area of this

triangle is greater than $\frac{1}{4}$,

$\mathbb{P}\left[\frac{1}{2}XY \geq \frac{1}{4}\right]$, where $X \sim \mathcal{U}[0,1], Y \sim \mathcal{U}[0,1]$ and X, Y are independent

The CDF of XY is given by

$$F_{XY}(z) = \mathbb{P}[XY \leq z] = z - z\log z, 0 < z \leq 1$$

Therefore the required probability is

$$1 - F\left(\frac{1}{2}\right) = \frac{1}{2} + \frac{1}{2}\log\frac{1}{2}$$

Two points on opposite sides

The probability of this event is $\frac{4}{16} = \frac{1}{4}$. In this case, the smaller piece is a **trapezium** whose area is between $\frac{1}{4}$ and $\frac{1}{2}$. We need the probability,

$$\mathbb{P}\left[\frac{1}{4} \leq \frac{1}{2}(X+Y) < \frac{1}{2}\right] \text{ or } \mathbb{P}\left[\frac{1}{4} \leq \frac{1}{2}(1-X+1-Y) < \frac{1}{2}\right]$$

where $X \sim \mathcal{U}[0,1], Y \sim \mathcal{U}[0,1]$ and X, Y are independent.

The CDF of $X + Y$ is given by

$$F_{X+Y}(z) = \mathbb{P}[X+Y \leq z] = \frac{z^2}{2}, 0 \leq z \leq 1$$

From symmetry, the required probability is

$$2\left(F(1) - F\left(\frac{1}{2}\right)\right) = \frac{3}{4}$$

Total probability

Therefore, the probability that the slice is reasonable is given by

$$\frac{1}{2}\left(\frac{1}{2} + \frac{1}{2}\log\frac{1}{2}\right) + \frac{1}{4}\cdot\frac{3}{4} = \frac{7}{16} - \frac{\log 2}{4} \approx 0.2642$$

Computational verification

The simulation below gives the probability of a reasonable slice to be 0.2645.

```
from random import choices, random
```

```
def prob_reasonable_slice(runs = 1000000):
    sides = ["a","b","c","d"]
    adj_sides = ["ab","bc","cd","ad"]
    opp_sides = ["ac","bd"]
    succ = 0
    for _ in range(runs):
        s1, s2 = choices(sides, k=2)
        x, y = random(), random()
        if s1 != s2:
            if "".join(sorted([s1,s2])) in adj_sides:
                if x*y/2 >= 0.25:
                    succ += 1
            if "".join(sorted([s1,s2])) in opp_sides:
                if min((x+y)/2, (1-x+1-y)/2) >= 0.25:
                    succ += 1
    return succ/runs

print(prob_reasonable_slice())
```

Chapter 90

How Many Turkey Trotters Can You Pass?

I have five kinds of fair Platonic dice: tetrahedra (whose faces are numbered $1-4$), cubes (numbered $1-6$), octahedra (numbered $1-8$), dodecahedra (numbered $1-12$) and icosahedra (numbered $1-20$). When I roll two of the cubes, there is a single most likely sum: seven. But when I roll one cube and two tetrahedra, there is no single most likely sum — eight and nine are both equally likely.

Which whole numbers are never the single most likely sum, no matter which combinations of dice I pick?

Solution

Can You Win The Riddler Football Playoff?

Riddler Express

World Cup group play consists of eight groups, each with four teams. The four teams in a group all play each other once (for a total of six matches), earning three points for a win, one point for a draw and zero points for a loss.

After group play in a particular group, all four teams have different numbers of points. The first-place team has A points, the second-place team B points, the third place team C points and the last-place team D points. Find all possible quadruples (A, B, C, D).

Solution

The **thirteen** quadruples satisfying the conditions of the problem are:

- (5, 4, 3, 2)
- (6, 5, 4, 1)
- (7, 4, 3, 1)
- (7, 4, 3, 2)
- (7, 5, 2, 1)
- (7, 5, 3, 1)
- (7, 5, 4, 0)

- (7, 6, 2, 1)
- (7, 6, 3, 1)
- (7, 6, 4, 0)
- (9, 4, 2, 1)
- (9, 4, 3, 1)
- (9, 6, 3, 0)

The code below was used for identifying the quadruples.

```
import numpy as np
from itertools import combinations, product

def scores(i, j):
    scores = [np.zeros(4) for _ in range(3)]
    scores[0][i], scores[1][j], scores[2][i], scores[2][j] = 3, 3, 1, 1
    return scores

uniq_scores = set()
for all_scores in product(*[scores(i,j) for i,j in combinations(range(4),2)]):
    a,b,c,d = tuple(sorted(np.sum(np.vstack(all_scores), axis=0), reverse=Tru
    if a != b != c != d:
        uniq_scores.add((a,b,c,d))
print(len(uniq_scores), uniq_scores)
```

Riddler Classic

Speaking of "football," the Riddler Football Playoff (RFP) consists of four teams. Each team is assigned a random real number between 0 and 1, representing the "quality" of the team. If team A has quality a and team B has quality b, then the probability that team A will defeat team B in a game is $a/(a+b)$.

In the semifinal games of the playoff, the team with the highest quality (the "1 seed") plays the team with the lowest quality (the "4 seed"), while the other two teams play each other as well. The two teams that win their respective semifinal games then play each other in the final.

On average, what is the quality of the RFP champion?

Solution

From the simulation below, we see that the average quality of the RFP champion is 0.674.

```
from random import random

def avg_quality(runs=1000000):
    win = lambda p, q: p if random() < p/(p+q) else q
    sum_q = 0
    for _ in range(runs):
        a, b, c, d = sorted([random() for _  in range(4)])
        sum_q += win(win(a,d), win(b,c))
    return sum_q/runs
```

```
print(avg_quality())
```

Chapter 92

Random Walk from the Centre of a Unit Square and Cube

Riddler Express

You start at the centre of the unit square and pick a random direction to move in, with all directions equally likely. You move along this direction until you reach a point on the perimeter of the square. On average, how far can you expect to have travelled?

Solution

Place the square with corners at $(0,0)$, $(1,0)$, $(1,1)$, $(0,1)$. By symmetry, the mean distance over all directions equals the mean over the triangle with vertices $(1/2,1/2)$, $(1,1/2)$, $(1,1)$. Pick a direction $\theta \in [0,\pi/4]$; the distance to the right edge is $d = 1/(2\cos\theta)$.

Weighting by the uniform density on the fundamental sector,

$$\mathbb{E}[d] \;=\; \int_0^{\pi/4} \frac{1}{2\cos\theta}\cdot\frac{4}{\pi}\,d\theta \;=\; \frac{2}{\pi}\int_0^{\pi/4} \sec\theta\,d\theta.$$

The antiderivative of $\sec\theta$ is $\ln|\sec\theta+\tan\theta|$, so

$$\mathbb{E}[d] \;=\; \frac{2}{\pi}\,\ln(\sqrt{2}+1) \;=\; \frac{\ln(3+2\sqrt{2})}{\pi} \;\approx\; 0.5611.$$

Why the $\int \sec x\,dx$ step works: four derivations

Method 1: multiplication by a clever form of 1.

Multiply by $(\sec x + \tan x)/(\sec x + \tan x) = 1$:

$$\int \sec x\,dx \;=\; \int \frac{\sec^2 x + \sec x \tan x}{\sec x + \tan x}\,dx.$$

With $u = \sec x + \tan x$, the numerator equals du, so the integral equals $\ln|u| + C = \ln|\sec x + \tan x| + C$.

Method 2: Weierstrass substitution.

With $t = \tan(x/2)$, the half-angle formulas give $\sin x = 2t/(1+t^2)$, $\cos x = (1-t^2)/(1+t^2)$, and $dx = 2\,dt/(1+t^2)$. Then

$$\int \sec x\,dx \;=\; \int \frac{2}{1-t^2}\,dt \;=\; \ln\left|\frac{1+t}{1-t}\right| + C \;=\; \ln\left|\tan\left(\tfrac{\pi}{4} + \tfrac{x}{2}\right)\right| + C.$$

Method 3: complex exponentials.

Since $\cos x = (e^{ix} + e^{-ix})/2$,

$$\int \sec x\,dx \;=\; \int \frac{2\,e^{ix}}{e^{2ix} + 1}\,dx.$$

Substitute $u = e^{ix}$, giving $\int 2/(i(u^2+1))\,du = -2i\arctan(e^{ix}) + C$, which on the real line reduces to $\ln|\sec x + \tan x| + C$.

Method 4: via hyperbolic functions.

Setting $x = it$, $\cos(it) = \cosh t$, so

$$\int \sec x\,dx \;=\; i\int \operatorname{sech} t\,dt \;=\; 2i\arctan\bigl(\tanh(t/2)\bigr) + C,$$

which again equals $\ln|\sec x + \tan x| + C$ after substituting $t = -ix$.

Monte Carlo verification

```
import numpy as np

def simulate_random_walk_2d(num_simulations=1_000_000):
    """Simulate random walks from the centre of the unit square."""
```

```
    x0, y0 = 0.5, 0.5
    theta = np.random.uniform(0, 2 * np.pi, num_simulations)
    dx, dy = np.cos(theta), np.sin(theta)

    t_right  = np.where(dx > 0, (1 - x0) / dx, np.inf)
    t_left   = np.where(dx < 0,  -x0 / dx,     np.inf)
    t_top    = np.where(dy > 0, (1 - y0) / dy, np.inf)
    t_bottom = np.where(dy < 0,  -y0 / dy,     np.inf)

    return np.minimum(np.minimum(t_right, t_left),
                      np.minimum(t_top,   t_bottom))

np.random.seed(42)
exact = np.log(3 + 2 * np.sqrt(2)) / np.pi
d = simulate_random_walk_2d(10**6)
print(f"analytical: {exact:.6f}")
print(f"simulated:  {d.mean():.6f}")
# analytical: 0.561100
# simulated:  0.561103
```

Riddler Classic

Now, you start at the centre of the unit cube. Again, you pick a random direction to move in, with all directions being equally likely. You move along this direction until you reach a point on the surface of the cube. On average, how far can you expect to have travelled?

Solution

Place the cube with corners at $(0,0,0)$ and $(1,1,1)$. A random direction on the sphere is written as $\vec{v} = (\sin\varphi\cos\theta, \sin\varphi\sin\theta, \cos\varphi)$ with $\theta \in [0, 2\pi)$ and $\varphi \in [0, \pi]$; the uniform measure on the sphere is $(4\pi)^{-1}\sin\varphi\, d\varphi\, d\theta$. From the centre, the distance along $\vec{v}$ to the boundary is

$$d = \frac{1}{2\,m}, \qquad m = \max\bigl(|\sin\varphi\cos\theta|,\ |\sin\varphi\sin\theta|,\ |\cos\varphi|\bigr).$$

The cube is invariant under 48 symmetries: eight sign combinations of the three axes, combined with six permutations of the coordinates. Restrict to the fundamental region where the direction lies in the first octant and the x-component is the largest in magnitude:

$$\theta \in \left[0, \tfrac{\pi}{4}\right], \qquad \varphi \in \left[\arctan(\sec\theta),\ \tfrac{\pi}{2}\right].$$

In this region, $d = 1/(2\sin\varphi\cos\theta)$, and the $\sin\varphi$ in the measure cancels the $\sin\varphi$ in the denominator of d. Multiplying by 48 and dividing by 4π:

$$\mathbb{E}[d] = \frac{6}{\pi}\int_0^{\pi/4} \frac{\frac{\pi}{2} - \arctan(\sec\theta)}{\cos\theta}\, d\theta = \frac{6}{\pi}\int_0^{\pi/4} \frac{\arctan(\cos\theta)}{\cos\theta}\, d\theta,$$

using $\arctan x + \arctan(1/x) = \pi/2$ for $x > 0$.

The integral does not collapse to elementary constants. Numerically,

$$\int_0^{\pi/4} \frac{\arctan(\cos\theta)}{\cos\theta}\, d\theta \approx 0.31980,$$

giving

$$\mathbb{E}[d] \approx 0.6106.$$

Distance to each face

From the centre $(1/2, 1/2, 1/2)$, the distance to each face in direction $(\sin\varphi\cos\theta, \sin\varphi\sin\theta, \cos\varphi)$ is:

- right $(x = 1)$: $t = 1/(2\sin\varphi\cos\theta)$ if $\sin\varphi\cos\theta > 0$
- left $(x = 0)$: $t = 1/(2|\sin\varphi\cos\theta|)$ if $\sin\varphi\cos\theta < 0$
- front $(y = 1)$: $t = 1/(2\sin\varphi\sin\theta)$ if $\sin\varphi\sin\theta > 0$
- back $(y = 0)$: $t = 1/(2|\sin\varphi\sin\theta|)$ if $\sin\varphi\sin\theta < 0$
- top $(z = 1)$: $t = 1/(2\cos\varphi)$ if $\cos\varphi > 0$
- bottom $(z = 0)$: $t = 1/(2|\cos\varphi|)$ if $\cos\varphi < 0$

The actual distance travelled is the minimum of these six values.

Monte Carlo verification

```
import numpy as np

def simulate_random_walk_3d(num_simulations=1_000_000):
    """Simulate random walks from the centre of the unit cube."""
    x0, y0, z0 = 0.5, 0.5, 0.5

    theta = np.random.uniform(0, 2 * np.pi, num_simulations)
    u     = np.random.uniform(-1, 1, num_simulations)   # cos(phi), uniform
    s     = np.sqrt(1 - u * u)                          # sin(phi)
    dx, dy, dz = s * np.cos(theta), s * np.sin(theta), u

    t_right  = np.where(dx > 0, (1 - x0) / dx, np.inf)
    t_left   = np.where(dx < 0,  -x0 / dx,     np.inf)
    t_front  = np.where(dy > 0, (1 - y0) / dy, np.inf)
    t_back   = np.where(dy < 0,  -y0 / dy,     np.inf)
    t_top    = np.where(dz > 0, (1 - z0) / dz, np.inf)
    t_bottom = np.where(dz < 0,  -z0 / dz,     np.inf)

    return np.minimum.reduce([t_right, t_left, t_front,
                              t_back,  t_top,  t_bottom])

np.random.seed(42)
d = simulate_random_walk_3d(10**6)
print(f"simulated: {d.mean():.6f}")
# simulated: 0.610596
```

Convergence

Running Monte Carlo for varying sample sizes:

Sample size	Mean	Std. error
10^3	0.6133	0.0053
10^4	0.6109	0.0017
10^5	0.6105	0.00053
10^6	0.6106	0.00017
10^7	0.61060	0.00005
Integral	0.6106	—

Comparison of the two answers

Dimension	Exact form	Numerical
2D (square)	$\dfrac{\ln(3+2\sqrt{2})}{\pi}$	0.5611
3D (cube)	$\dfrac{6}{\pi}\displaystyle\int_0^{\pi/4} \dfrac{\arctan(\cos\theta)}{\cos\theta}\,d\theta$	0.6106

The expected distance grows only modestly (by about 9%) from 2D to 3D. Although an extra dimension opens longer "body-diagonal" directions (up to $\sqrt{3}/2 \approx 0.866$), the cube's six faces cut most rays off quickly, so the uniform-direction average is only a little above the inradius 0.5.