Greetings!

I do not have any professional publish work to showcase. However, I do have a few math and logic puzzles that I would like to share with you. I have a small little channel inside a big gaming clan online where I share math puzzles and problems to several clan members. The clan members vary greatly in their technical backgrounds. Some of these problems involve programming, while others are more logic based. I try to present several problems that would excite the members and lead to interesting discussions. A lot of these problems were modified from several sources, are hybridization of multiple items, or are unique problems from me. The solutions are all original from me.

I have attached a link on the **last page** with a Data Science Project I did that might be of interest to you. I also added several more questions in case you were curious of some of the problems I gave my clanmates. The problems were never meant to be shared beyond a small circle of friends, so they aren't the most polished.

Question 2

Henrique owns a parking lot for vehicles. A vehicle is either a motorcycle with two wheels or a car with four wheels. There are 100 vehicles in the parking lot. The total number of wheels in Henrique's parking lot is 326. If Henrique charges $1.00 for each motorcycle and $2.00 for each car per day, what is the total revenue for today?

Solution

There are a myriad of methods to solve this. The commonality in all of these methods is we need to solve for a system of equations. Let m be the number of motorcycles and c the number of cars on the lot. We end with two equations based on the above scenario. You can use substitution or any method to solve this system.

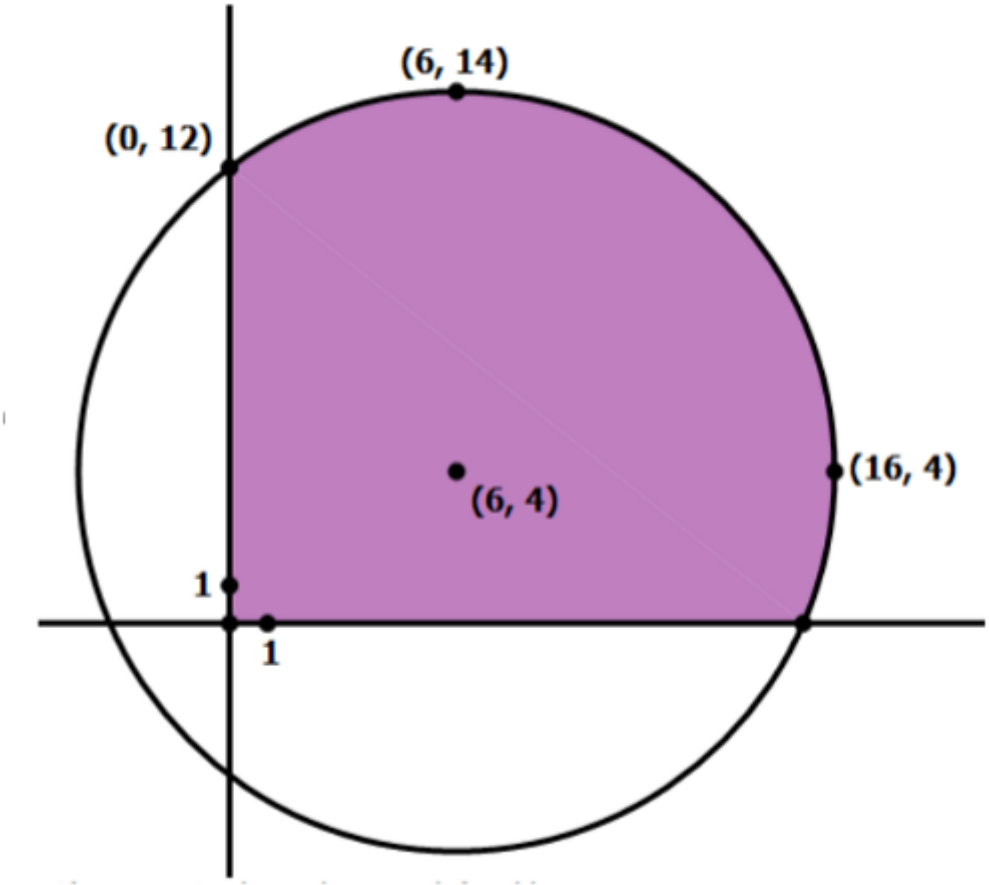
$$\begin{aligned} m + c &= 100 \quad \# \, of \, vehicles \\ 2m + 4c &= 326 \quad \# \, of \, wheels \end{aligned}$$

We can also use matrices to perform the above operation. This isn't particularly time saving for this particular question, but for future questions, this will likely be the preferred operation. We extract the coefficients from equations above and translate them into matrices. Don't worry if you aren't familiar with the techniques presented below. This is called Linear Algebra, and we will cover this in greater detail in the future.

$$\begin{pmatrix} 1 & 1 & | 100 \\ 2 & 4 & | 326 \end{pmatrix} \longrightarrow \begin{pmatrix} 1 & 1 & | 100 \\ 1/2(2 & 4 & | 326) \end{pmatrix} \longrightarrow \begin{pmatrix} 1 & 1 & | 100 \\ 1 & 2 & | 163 \end{pmatrix} \longrightarrow \begin{pmatrix} 1 & 1 & | 100 \\ 0 & 1 & | 63 \end{pmatrix}$$

So we are left with the number of cars $c = 63$, and $m = 100 - 63 = 37$ motorcycles. So the amount of money Henrique makes is $\$1 * m + \$2 * c = \$1 * 37 + \$2 * 63 = \$163\$$

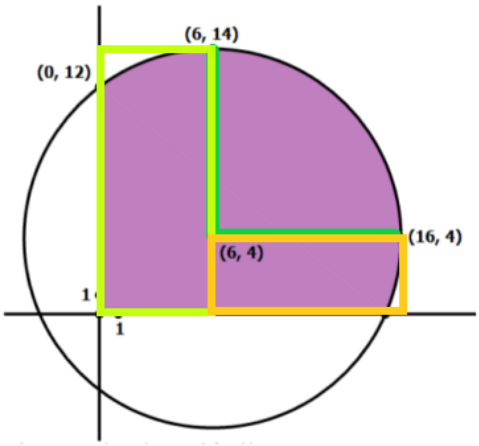
Question 8

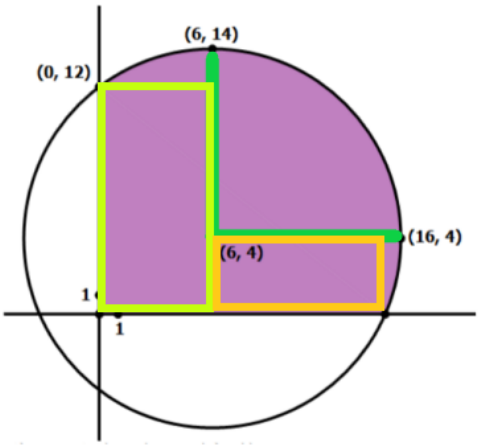


The shaded region is defined by the circle with origin $(6, 4)$, $y \geq 0$, and $x \geq 0$. If the $Area_{circle} \leq 100$, what is the approximate area of the shaded region?

- [ ] less than 75
- [ ] between 75 and 125
- [ ] between 125 and 175
- [ ] between 175 and 225
- [ ] between 225 and 275
- [ ] between 275 and 325
- [ ] more than 325

Solution





We can break the shape of the shaded region approximately into the sum of 2 rectangles and one-quarter of a circle. We can create an upper boundary and a lower boundary by looking at the above figure. Clearly the figure on the left will have a larger area than the figure on the right since the rectangles extend pass the radius of the circle at certain points.

The radius of the circle

$$y_i - y_{origin} = 14 - 4 = 10$$

The upper bound

$$\frac{1}{4}Area_{circle} = \pi r^2 \longrightarrow \frac{1}{4}\pi(10)^2 = 25\pi$$
$$Area_{rectangle_1} = base * height = (6-0)(14-0) = 84$$
$$Area_{rectangle_2} = base * height = (16-6)(4-0) = 40$$
$$\frac{1}{4}Area_{circle} + Area_{rectangle_1} + Area_{rectangle_2}$$
$$25\pi + 84 + 40$$
$$25\pi + 124$$

The lower bound

$$\frac{1}{4}Area_{circle} = \pi r^2 \longrightarrow \frac{1}{4}\pi(10)^2 = 25\pi$$
$$Area_{rectangle_1} = base * height = (6-0)(12-0) = 72$$
$$Area_{rectangle_2} = base * height = (15.1-6)(4-0) \approx 36.4$$
$$\frac{1}{4}Area_{circle} + Area_{rectangle_1} + Area_{rectangle_2}$$
$$25\pi + 72 + 36.4$$
$$25\pi + 108.4$$

To find the base of the horizontal rectangle, we need to find the x-intercept of the circle. We can do this by setting $y = 0$ in the circle formula.

$$(x-6)^2 + (y-4)^2 \leq 100$$
$$(x-6)^2 + (0-4)^2 \leq 100$$
$$(x-6)^2 \leq 100 - 16$$
$$\sqrt{(x-6)^2} \leq \sqrt{84}$$
$$\sqrt{(x-6)^2} \lessapprox 9.1$$
$$x \approx 15.1 \quad or \quad x \approx 3.1$$

We can approximate $\sqrt{84}$ as $\sqrt{81+e}$, where $e > 0$, so we get 9+a small fraction. 9.1 is close enough for our purposes. $x \approx 3.1$ does not make sense in this context so we only need $x \approx 15.1$

The shaded region must be between our upper and lower bounds

$$25\pi + 108.4 < shaded\ region < 25\pi + 124$$
$$25(3) + 108.4 < shaded\ region < 25(3) + 124$$
$$25(3) + 108.4 < shaded\ region < 25(3) + 124$$
$$183.4 < shaded\ region < 199$$

Question 12

Tired of losing to that MMO boss purely by chance? Why not embrace randomness, and play a game where our score is determined by a random number generator — all hail the RNG god!

Our random number generator produces integers from 0 to 9 inclusively. The first number generated gets added to the decimal place of our score. So if we roll a 5, our current score becomes 0.5. Each time we generate a random number, if the number generated is less than or equal to the previous number, we append it to our score. If the number generated is 0, we end our score. Anything else gets skipped.

For example, suppose we generated the following sequence: 7, 5, 4, 2, 5, 6, 1, 1, 0. Our score for that round would be 0.754211.

What will be the average final score produced?

Solution

```
0.4737855368144297
```

I find working on these sorts of problem easier to handle computationally rather than constructing a pure mathematical proof. For this game, I construct a simple Python script. First, we create a function that represents the random number generator. It is a good idea to separate this component, since we can more easily modify the script in the future by keeping functions modular.

```
def generate_int():
    return random.randint(0,9)
```

Our function `generate_int()` outputs a random integer from 0 to 9 inclusively and uniformly. In the future, we can modify the random number generator to follow a Gaussian normal distribution or Chi-Squared distribution. However, for our purposes, a good old uniform distribution is acceptable.

```
def roll():
    running_score = []
    is_True = True
    while is_True:
        digit = generate_int()
        if digit == 0:
            is_True = False
        elif (running_score == []) or (digit <= running_score[-1]):
            running_score.append(digit)

    digits_to_string = [str(i) for i in running_score]

    return float("0." + "".join(digits_to_string))
```

The function `roll()` simulates each round of the game. The crux of the function is a **while loop**. Because we don't know how long any particular game will run, a while loop gives us the flexibility that we need to have games of undetermined length. We will store our temporary results in a list called `running_score` to make our lives easier.

Inside the loop, we have a sequence of conditions. These are the conditions that represent the rules of our game. The `if` statement is our stop condition. Recall that our game is over at any point that the RNG generates a 0, so this is why this condition is first. The following conditions adjusts our `running_score`. The first part of the `elif` statement checks if our `running_score` is empty. This should only occur once when we initialize the score with a valid digit. The second part of the `elif` statements checks to see if the digit generated is less than or equal to the

previously appended digit. If that condition is true, we append the current generated digit to our `running_score`.

After the while loop, we format the integers in our `running_score` to a string using a **list comprehension**. We then join those elements together and add a "0." in front to return our final float. This gives us our final score for that round.

There are a few idiosyncratic things to note. In the second part of the `elif` statement we used `==` as the comparison operator as opposed to `is`. Can you explain the reason why? Try replacing `==` with `is` and see what the script outputs.

After simulating one round of our game, why not simulate another? What about simulating 10,000 games? The beauty of a digital simulation, is that we can easily run multiple subsequent scenarios. The `test_rolls()` function repeats the `roll()` function simulating thousands of scenarios and returns an average score of each round. Try a few simulations and see what average scores you obtain. Please note that your average score might differ slightly from the answer given in this solution, but it should be fairly close.

```
0.4737855368144297
```

```
import random

def generate_int():
    return random.randint(0,9)

def roll():
    digits = []
    is_True = True
    while is_True:
        digit = generate_int()
        if digit == 0:
            is_True = False
        elif (digits == []) or (digit <= digits[-1]):
            digits.append(digit)

    digits_to_string = [str(i) for i in digits]

    return float("0." + "".join(digits_to_string))

def test_rolls(n=100000):
    temp = []
    for i in range(n):
        if i%10000 ==0:
            print("Progress: {:0.2f}".format(i/n))
        x = roll()
        temp.append(x)
    average = sum(temp)/ len(temp)
    return average

%time test_rolls()
```

Question 4

The Battle of Winterfell has come and gone. Regardless of what your feelings are for the final season of Game of Thrones (I'm still not ready to come to grips with it), there is a lot of inspiration we can take from it. Lets take the Night King vs Jon Snow in the Battle of Winterfell. The army of the undead bombarded Jon's rag tag team of the obstinate living. Let's come up with a simulation to see how likely Jon's army would come out as triumphant.

The undead may have supernatural abilities, but Jon's army is full of battle-hardened experienced soldiers. Let's give them a 50-50 chance of beating the undead via a one-on-one battle. If the undead kills the living, not only does the living lose a member, but the undead army gains a new member as the Night King is continuously raising the dead. If the living defeats the undead, that undead is permanently out of commission. Lets assume that all members of the living are carrying dragonglass. What distribution of living vs undead yields yields what probability of either side coming out as triumphant?

Solution

There are multiple ways to attack this problem. I want to tackle this from a bottom-up approach. The following function will focus on the living.

```
def one_one_one(living, undead):
    if random.random()>.5:  # living wins
        undead -= 1
    else:
        undead += 1
        living -= 1
    return (living, undead)
```

The function `one_on_one()` takes in the living and undead and generates a random number. If the random number is greater than 0.5, the living soldier has vanquished the undead. If the random number is less than 0.5, the undead kills the living and recruits the newly risen dead into the army of the Night King.

```
def battle_of_winterfell(living, undead):
    long_night = True
    while long_night:
         if living and undead > 0:
             living, undead = one_one_one(living,undead)
         else:
             long_night = False
    if living> undead:  #returns 1 if Living wins
        return 1
    else:
        return 0
```

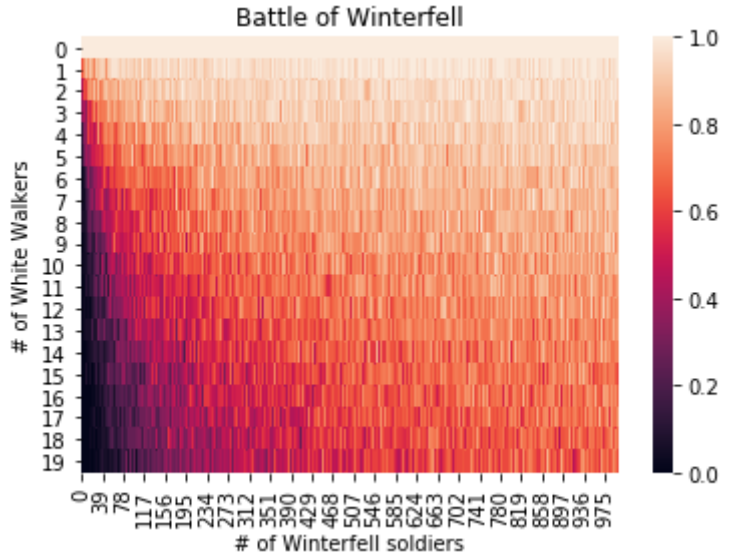
We continue these one-on-one battles as long as both the living and the undead have at least 1 member.

I decided to make a heatmap showing what is the probability of either side winning given the starting numbers for both sides.

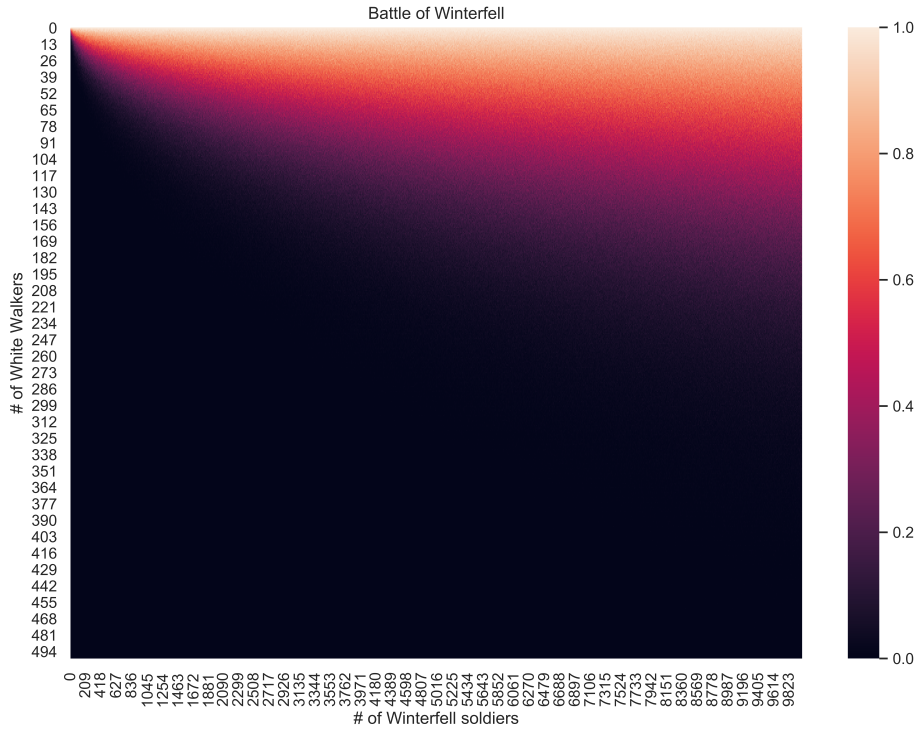
```
import random
import seaborn as sns
import pandas as pd
import matplotlib.pyplot as plt

```

```
living=1000
undead=20


def one_one_one(living, undead):
    if random.random()>.5: #living wins
        undead -= 1
    else:
        undead += 1
        living -= 1
    return (living,undead)


def battle_of_winterfell(living, undead):
    long_night = True
    while long_night:
         if living and undead > 0:
             living, undead = one_one_one(living,undead)
         else:
             long_night = False
    if living> undead:  #returns 1 if Living wins
        return 1
    else:
        return 0


def survival_rate(living, undead):
    n=50
    rate = []
    for trial in range(n):
        rate.append(battle_of_winterfell(living, undead))
    return sum(rate)/len(rate)


df = pd.DataFrame(0.0, index=range(undead), columns=range(living))

for x in range(living):
    for y in range(undead):
        df[x][y] = survival_rate(x, y)

p1 = sns.heatmap(df)
plt.title("Battle of Winterfell")
plt.ylabel('# of White Walkers')
plt.xlabel('# of Winterfell soldiers')
#plt.figure(figsize=(4,3))
plt.savefig('BattleOfWinterfell.png', dpi=600)
#plt.figure(dpi=600)
plt.show()

#df.to_csv()
```

This is a heatmap using the data as from above.



We can get a much smoother graph if we increase the values. Do note, that this will exponentially increase the time it takes to run. Can you think of ways to improve the efficiency of the code?



Question 19

Hello fellow Pokémon trainers!

I recently learned about Pokémon base stats. This got me thinking. The motto for Pokémon is *Gotta Catch 'Em All!* but just how many can you catch? How many Pokémon can Game Freak make that is competition valid?

There are 6 base stats categories ranging from 1 to 255 points–HP, Attack, Defense, Special Attack, Special Defense, and Speed. A simple way to gauge a Pokémon is to add up all 6 categories. Lets assume that every Pokémon is unique in its points distribution. The legendary status cutoff for base stats is 600 points; this means that if you want a Pokémon that is valid for competitive play, the total amount of points in all 6 categories must add up to 600 points or less.

How many ways can you distribute the points such that the Pokémon is valid for competitive play? You can arrange the points to be jack of all trades or really good in a couple of categories.

—Aspiring Pokémon trainer

Solution

I haven't typed up the solution yet, but feel free to contact me for a sneak preview.

Project

Sample Data Science Project1A.html