Introduction

C200 Programming Assignment № 9

Dr. M.M. Dalkilic
Computer Science
School of Informatics, Computing, and Engineering

Indiana University, Bloomington, IN, USA

April 15, 2023

The HW is due on **Friday, April, 21 at 10:59 PM EST**. Please commit, push and submit your work to the Autograder before the deadline.

In this homework, you'll work on translating critical thinking to programming. For those whose interest in finance or statistics, a random walk describes a remarkable outcome given the function: one step back and forth or up and down and you'll travel a great distance. You'll also start a fraction class. Permutations are used to check all possible inputs.

1. Make sure that you are **following the instructions** in the PDF, especially the format of output returned by the functions. For example, if a function is expected to return a numerical value, then make sure that a numerical value is returned (not a list or a dictionary). Similarly, if a list is expected to be returned then return a list (not a tuple, set or dictionary).

2. Test **debug** the code well (syntax, logical and implementation errors), before submitting to the Autograder. These errors can be easily fixed by running the code in VSC and watching for unexpected behavior such as, program failing with syntax error or not returning correct output.

3. Make sure that the **code does not have infinite loop (that never exits) or an endless recursion (that never completes)** before submitting to the Autograder. You can easily check for this by running in VSC and watching for program output, if it terminates timely or not.

4. Given that you already tried points 1-3, if you see that Autograder does not do anything (after you press 'submit') and waited for a while (30 seconds to 50 seconds), try refreshing the page or using a different browser.

5. Once you are done testing your code, comment out the tests i.e. the code under the __name__ == "__main__" section.

Problem 1: Random Walk

A random walk is a *stochastic process*. A stochastic process is a series of values that are not determined functionally, but probabilistically. The random walk is supposed to describe an inebriated person who, starting from the bar, intends to walk home, but because of intoxication instead randomly takes single steps either forward or backward, left or right. The person has no memory of any steps taken, so theoretically, the person shouldn't move too far from where he or she starts. Random walks are used to model many phenomena, like the size of the web or changes in financial instruments. We will model a 2D random walk with two arrays x and y where x represents moving left or right and y represents forward or backward. The index i will represent the step and x[i],y[i] will represent the location at step i. So, for i=0, we have x[0],y[0] (starting place). Using random we choose from the list [1,2,3,4]. If the value is one then we move right from the previous x position:

```
1 x[i] = x[i-1] + 1
2 y[i] = y[i-1]
```

If the value is two, then we move left from the previous x position:

```
1 x[i] = x[i-1] - 1
2 y[i] = y[i-1]
```

If the value is three, we move up from the previous y position:

```
1 x[i] = x[i-1]
2 y[i] = y[i-1] + 1
```

And when the value is four, we move down from the previous y position:

```
1 x[i] = x[i-1]
2 y[i] = y[i-1] - 1
```

Here is another way to describe this:

$$step(0) = 0 \tag{1}$$

$$step_i(n) = \begin{cases} \text{left} & step(n-1),\ i=1 \\ \text{right} & step(n-1),\ i=2 \\ \text{up} & step(n-1),\ i=3 \\ \text{down} & step(n-1),\ i=4 \end{cases} \tag{2}$$

The following code:

```
1 # #number of steps
```

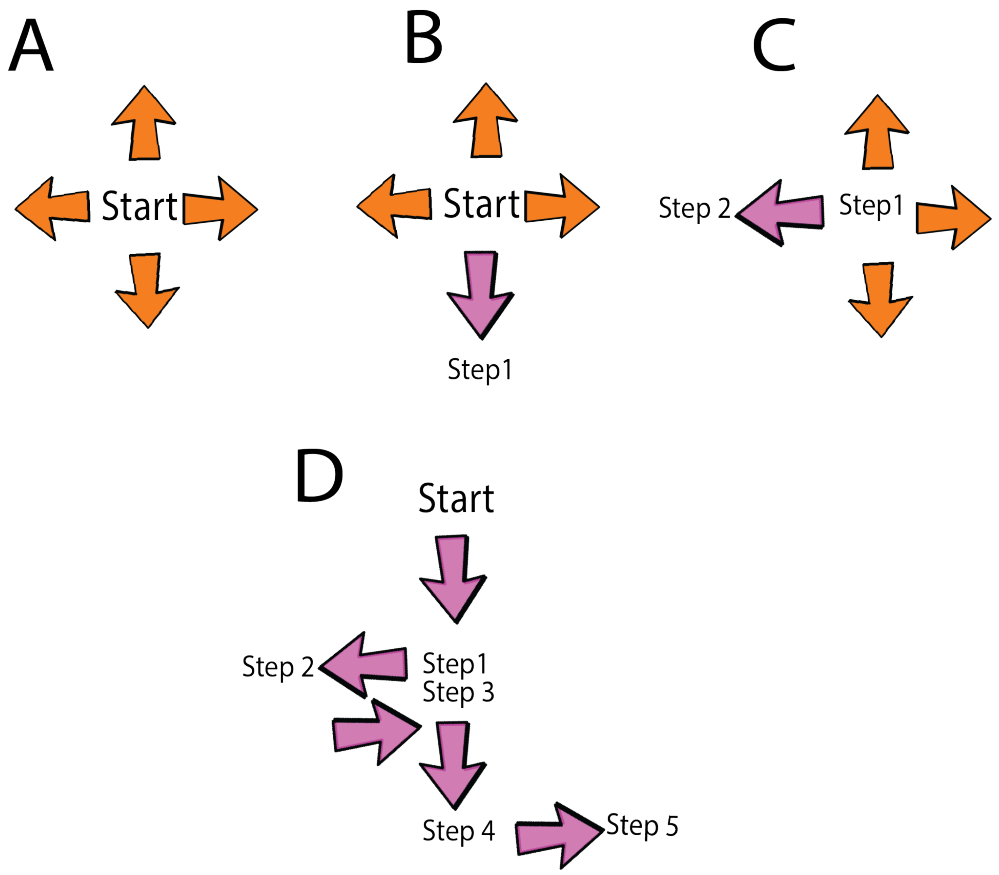

Figure 1: The first few steps of a random walk.

```
2       n = 100000    # You should change the number of steps to see
3                     # how it affects the plot
4       x = np.zeros(n)
5       y = np.zeros(n)
6
7     #fill array with step values
8       for i in range(1,n):
9           step(x,y,i)
10      graphit(x,y,n)
```

has Fig. 2 as output.

Programming Problem 1: Random Walk

- Complete the `step` function.
- These are random walks; these outputs will be different.
- Experiment with different numbers of steps to see how it generates different walks.

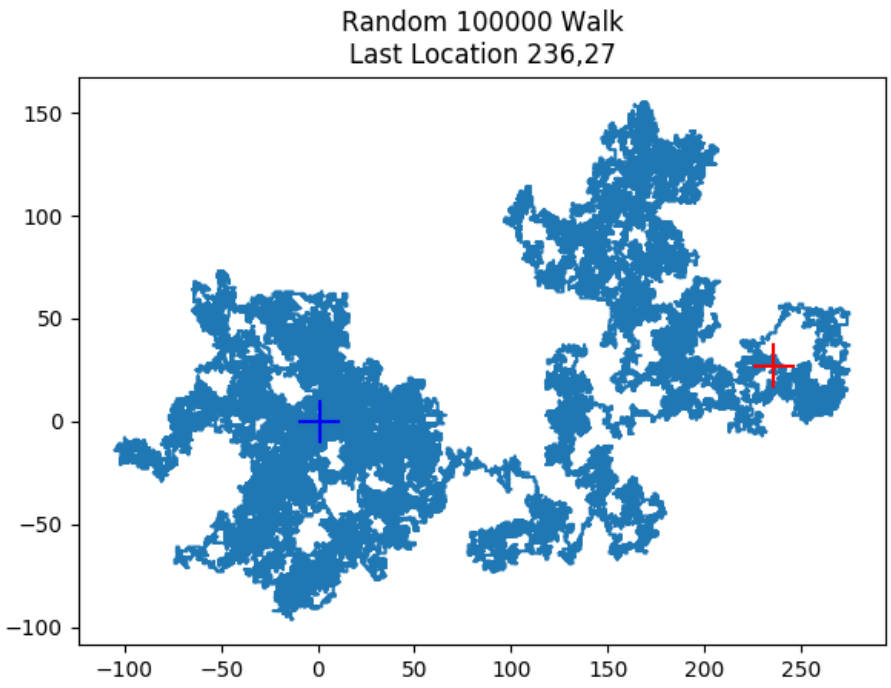

Figure 2: The blue cross is where we started and the red cross is where we ended.

Problem 2: Fraction Class

For this problem, you'll be given *some* of the methods of a fraction class. You are responsible for building the required functionality. We define the Fraction class as an ordered pair of positive integers x, y. The first is called the numerator and the second the denominator. When we create an instance, the fraction is **reduced**, *i.e.*, common factors are removed. For example,

$$(2 \times 3 \times 5, 2 \times 3 \times 7) \quad \rightarrow \quad (5, 7) \tag{3}$$

We have only two operations: addition + and multiplication *. We also rewrite __str()__ to make the display (x/y).

Note that when we print the instance (object of a class), the __str__ function will be called automatically, and instance of the class is returned, not a value. So, x, y, z, and a are all instances of the fraction class and so when we print them, the __str__ function is called, that returns the representation in the format: numerator/denominator.

When run, the following code:

```
1 x = fraction(2*3*4,4*3*5)
2 y = fraction(2*7,-7*2)
3 z = fraction(-13,-14)
4 a = fraction(-13*2*7,14)
5 print(x,y,z,a)
6 print(f"{x} + {y} == {x + y}")
7 print(f"{x}*{y} == {x * y}")
8 b,c = fraction(1,2),fraction(3,5)
```

```
9 print(f"{b} + {c} == {b + c}")
```

has output:

```
1 (2/5) (1/-1) (-13/-14) (-182/14)
2 (2/5) + (1/-1) == (3/-5)
3 (2/5)*(1/-1) == (2/-5)
4 (1/2) + (3/5) == (11/10)
```

Deliverables Problem 2

- Complete the Fraction class
- In class we looked at Euclid's Algorithm for greatest common divisor `https://en.wikipedia.org/wiki/Euclidean_algorithm`. This is a useful algorithm to reduce the fractions, but you're free to do it as you'd like.
- For this initial attempt, do not consider division by zero.

Problem 3: Permutations

Given a list of objects, a permutation is a sequence, without replacement, drawn from the list. The set of permuations is every possible unique sequence. For the permutations of [1,2,3] are:

```
[1] + permutations ([2,3]); [1] + permutations([3,2])
[2] + permutations ([1,3]); [2] + permutations([3,1])
[3] + permutations ([1,2]); [3] + permutations([2,1])
gives
[1,2,3] [1,3,2]
[2,1,3] [2,3,1]
[3,1,2] [3,2,1]
```

This can be described recursively–a single item is the single item. Given more than a single item, form the list of permutations taking the first item of the orignal list and the permutations of the remainder. The following code:

```
1 print(permutation([1,2,3]))
2 print(permutation([1,2,3,4]))
```

has output:

```
1 [[1, 2, 3], [1, 3, 2], [2, 1, 3], [2, 3, 1], [3, 1, 2], [3, 2, 1]]
2 [[1, 2, 3, 4], [1, 2, 4, 3], [1, 3, 2, 4], [1, 3, 4, 2], [1, 4, 2, 3],↩
      [1, 4, 3, 2], [2, 1, 3, 4], [2, 1, 4, 3], [2, 3, 1, 4], [2, 3,
```

```
3 4, 1], [2, 4, 1, 3], [2, 4, 3, 1], [3, 1, 2, 4], [3, 1, 4, 2], [3, 2, ↩
    1, 4], [3, 2, 4, 1], [3, 4, 1, 2], [3, 4, 2, 1], [4, 1, 2, 3], [4, ↩
    1, 3, 2], [4, 2, 1, 3], [4, 2, 3, 1], [4, 3, 1, 2], [4, 3, 2, 1]]
```

Hint: You will have to think about this! Important thing is to note that, recursion has to be implemented locally within the function, it can't done by calling **permutation()** recursively, but by creating a local function within **permutation()** and calling that recursively.

Deliverables Problem 3

- Complete the permutation class
- Unsurprisingly, you cannot use any libraries.

Problem 4: Closest Pair

In this problem, you'll use write a *brute force* algorithm that, given a list of points in the 2D Euclidean plane, $P = [(x_0, y_0), (x_1, y_1), \ldots (x_n, y_n)]$, find the pair $p_i = (x_i, y_i)$ and $p_j = (x_j, y_j)$ such that the distance d between the two:

$$d(p_i, p_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \tag{4}$$

is the smallest possible value of any possible distance d between any pair and P has at least two pairs. For example, say $P = [(5, 1), (2, 2), (1, 1)]$, then:

$$d(p_i, p_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \tag{5}$$

$$d((5, 1), (2, 2)) = \sqrt{(5 - 2)^2 + (1 - 2)^2} = \sqrt{9 + 1} \approx 3.16 \tag{6}$$

$$d((2, 2), (1, 1)) = \sqrt{(2 - 1)^2 + (2 - 1)^2} = \sqrt{1 + 1} \approx 1.41 \tag{7}$$

$$d((5, 1), (1, 1)) = \sqrt{(5 - 1)^2 + (1 - 1)^2} = \sqrt{4} = 2 \tag{8}$$

So the closest pair is [(2,2), (1,1)]. Since d is communtative, it doesn't matter whether we choose p_i first or p_j first.

To make it easier for someone to use our code, the function will return [p1, p2, distance] where p1 is the first point as a tuple, p2 is the second point as a tuple, and distance is the distance between them, *i.e.*, $d(\text{p1}, \text{p2})$. When this code is run:

```
1 x = [(rn.randint(1,50), rn.randint(1,50)) for _ in range(10)]
2 print(x)
3 print(brute(x))
```

this output was produced: When I ran this, my output was

```
[(34, 34), (28, 38), (24, 3), (18, 17), (33, 33),
 (16, 1), (32, 45), (50, 24), (5, 35), (48, 30)]
[(34, 34), (33, 33), 1.4142135623730951]
```

Deliverables for Problem 4

- Implement the distance and brute functions.
- Observe that the function brute(x) takes a list of pairs of numbers and returns a list [x,y,d], where x is the first point, y is the second point, and d is the smallest distance between x and y. An observation–d might not be necessarily unique–there might be two, or even several, ways to differently obtain this value.
- if you find that there are more than 1 pair with the same minimum distance then just return the first such pair.

Problem 5: Haversine Distance

Finding the distance between two points $p_1 = (x_1, y_1), p_2 = (x_2, y_2)$, $d(p_1, p_2)$ in a 2D plane is straightforward:

$$d(p_1, p_2) = [(x_1 - x_2)^2 + (y_1 - y_2)^2]^{1/2} \tag{9}$$

When calculating the distance on the Earth, however, we have to take into account Earth's shape. To that end, we can use the Haversine formula, named by Inman for the function:

$$h(\theta) = \sin^2\left(\frac{\theta}{2}\right) \tag{10}$$

called the *haversine*. If p_1, p_2 are on a Earth (sphere) we have:

$$h\left(\frac{d}{r}\right) = h(y_2 - y_1) + \cos(x_1)\cos(y_1)h(x_2 - x_1) \tag{11}$$

where d is the distance between the points for a sphere whose radius is r. To solve for d we take the inverse $\sin$ and muliply by r:

$$d = 2r \arcsin\left(h(\frac{d}{r})^{1/2}\right) \tag{12}$$

Assume the points p_1, p_2 are latitude, longitude pairs. These pair describe a point through intersection: The latitude runs parallel to the equator and longitude runs perpendicular to the equator. They intersect at a point. We now show how to calculate distance between two points on the Earth using $p_1 = (lat_1, lon_1), p_2 = (lat_2, lon_2)$ latitude and longitude:

$$lat_d = \frac{lat_2 - lat_1}{2} \tag{13}$$

$$lon_d = \frac{lon_2 - lon_1}{2} \tag{14}$$

$$d_0 = \sin(lat_d)^2 + \cos(lat_1)\cos(lat_2)\sin(lon_d)^2 \tag{15}$$

$$d_1 = 2r \arcsin(d_0^{1/2}) \tag{16}$$

$$r = 3961 \text{ mi.} \tag{17}$$

When implementing in Python, you'll have to make sure to convert to radians. $\arcsin$ is implemented in math as `asin()`.

The following code:

```
#Lindley Hall Home of Computer Science for decades
#south side of campus
l1 = (39.165341,-86.523588)

#Luddy Hall the new Luddy building new home of Computer Science
l2 = (39.172725,-86.523295)

#Distance between Lindley and Luddy
print("haversine", round(hd(l1,l2),4), "mi")
print("Euclidean", round(eu_distance(l1,l2),4), "mi")
print("Approximate", round(dd(l1,l2),4), "mi")
```

has output:

```
haversine 0.5107 mi
Euclidean 0.0074 mi
Approximate 0.5126 mi
```

You can see even small distances are erroneously computed for Euclidean distance.

Deliverables Programming Problem 5: Haversine

- Complete the functions.
- Don't round your answer, we have shown the rounded-off values for the sake of explanation.
- Here is a URL that provides a different, but equivalent formula for finding distance `https://andrew.hedges.name/experiments/haversine/`.
- We've provide code for traditional Euclidean distance eu_d() and a shortcut for haversine (dd) that recognizes the amount of error as we move north from the equator.
- You don't have to proivde the answer to the following questions but they will help you to think bit deeper about this problem:
  - What would be the benefits of dd() over hd()?
  - What would be the drawbacks?
  - What do you suppose this means for what we *remember* as distance?

Programming Pairs

abolad@iu.edu, btpfeil@iu.edu
jacbooth@iu.edu, bpoddut@iu.edu
megofolz@iu.edu, lmamidip@iu.edu
mdgamble@iu.edu, mw154@iu.edu
aketcha@iu.edu, bashih@iu.edu
rythudso@iu.edu, tshore@iu.edu
qhodgman@iu.edu, yijwei@iu.edu
eddykim@iu.edu, jwescott@iu.edu
pkasarla@iu.edu, alexschu@iu.edu
jolawre@iu.edu, johwarre@iu.edu
akundur@iu.edu, isarmoss@iu.edu
jl335@iu.edu, mszczas@iu.edu
bradhutc@iu.edu, sjvaleo@iu.edu
vilokale@iu.edu, cjromine@iu.edu
achimeba@iu.edu, sabola@iu.edu
arykota@iu.edu, kvanever@iu.edu
agoldswo@iu.edu, hvelidi@iu.edu
moalnass@iu.edu, saseiber@iu.edu
sijatto@iu.edu, nireilly@iu.edu
joecool@iu.edu, alescarb@iu.edu
sakinolu@iu.edu, elyperry@iu.edu
abdufall@iu.edu, nkmeade@iu.edu
lgflynn@iu.edu, omilden@iu.edu
marbelli@iu.edu, schwani@iu.edu
jackssar@iu.edu, jvalleci@iu.edu
sjkallub@iu.edu, jmom@iu.edu
sbehman@iu.edu, pvinod@iu.edu
nokebark@iu.edu, macsvobo@iu.edu
stfashir@iu.edu, ajneel@iu.edu
lsherbst@iu.edu, cartmull@iu.edu
tyfeldm@iu.edu, almoelle@iu.edu
ivycai@iu.edu, edshipp@iu.edu
dud@iu.edu, psaggar@iu.edu
chnbalta@iu.edu, patel88@iu.edu
nbakken@iu.edu, ss126@iu.edu
moesan@iu.edu, sousingh@iu.edu
siqidong@iu.edu, hnichin@iu.edu
aibitner@iu.edu, jcpilche@iu.edu
swcolson@iu.edu, jmissey@iu.edu

arjbhar@iu.edu, rair@iu.edu
klongfie@iu.edu, aw149@iu.edu
jl263@iu.edu, jonvance@iu.edu
brakin@iu.edu, danwils@iu.edu
aadidogr@iu.edu, criekki@iu.edu
katzjor@iu.edu, sowmo@iu.edu
kbchiu@iu.edu, roelreye@iu.edu
nbulgare@iu.edu, shahneh@iu.edu
sihamza@iu.edu, mostrodt@iu.edu
mgorals@iu.edu, shethsu@iu.edu
aaberma@iu.edu, wuyul@iu.edu
jaybaity@iu.edu, bensokol@iu.edu
mgambett@iu.edu, rpmahesh@iu.edu
jacklian@iu.edu, caitreye@iu.edu
ogift@iu.edu, grschenc@iu.edu
joracobb@iu.edu, ornash@iu.edu
tifhuang@iu.edu, crfmarti@iu.edu
dk80@iu.edu, webejack@iu.edu
hollidaa@iu.edu, arnpate@iu.edu
nwbarret@iu.edu, dannwint@iu.edu
anderblm@iu.edu, jorzhang@iu.edu
sl92@iu.edu, esmmcder@iu.edu
aalesh@iu.edu, jyamarti@iu.edu
ljianghe@iu.edu, imalhan@iu.edu
egillig@iu.edu, snuthala@iu.edu
dilkang@iu.edu, dy11@iu.edu
neybrito@iu.edu, trewoo@iu.edu
ecastano@iu.edu, vthakka@iu.edu
davgourl@iu.edu, cmulgrew@iu.edu
dombish@iu.edu, sjxavier@iu.edu
wdoub@iu.edu, cartwolf@iu.edu
sgcolett@iu.edu, kalomart@iu.edu
jonhick@iu.edu, yasmpate@iu.edu
actonm@iu.edu, zisun@iu.edu
anthoang@iu.edu, yz145@iu.edu
aladkhan@iu.edu, nmonberg@iu.edu, zhaozhe@iu.edu
bgabbert@iu.edu, svaidhy@iu.edu
cheng47@iu.edu, avpeda@iu.edu
ekkumar@iu.edu, chnico@iu.edu
georliu@iu.edu, lenonti@iu.edu
joshbrin@iu.edu, davthorn@iu.edu

jasoluca@iu.edu, leplata@iu.edu
hc51@iu.edu, parisbel@iu.edu
jegillar@iu.edu, jarymeln@iu.edu
coldjone@iu.edu, dpepping@iu.edu