C200 Programming Assignment № 8: Classes, Algorithms

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

Indiana University, Bloomington, IN, USA

April 14, 2022

Introduction

In this homework, you'll work on translating critical thinking to programming. This homework is due on **Thursday, April, 21 at 10:59 PM EST**.

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:

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

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

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

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

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

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

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

Here is another way to describe this:

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

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

The random walk also describes investing in, say, stock.

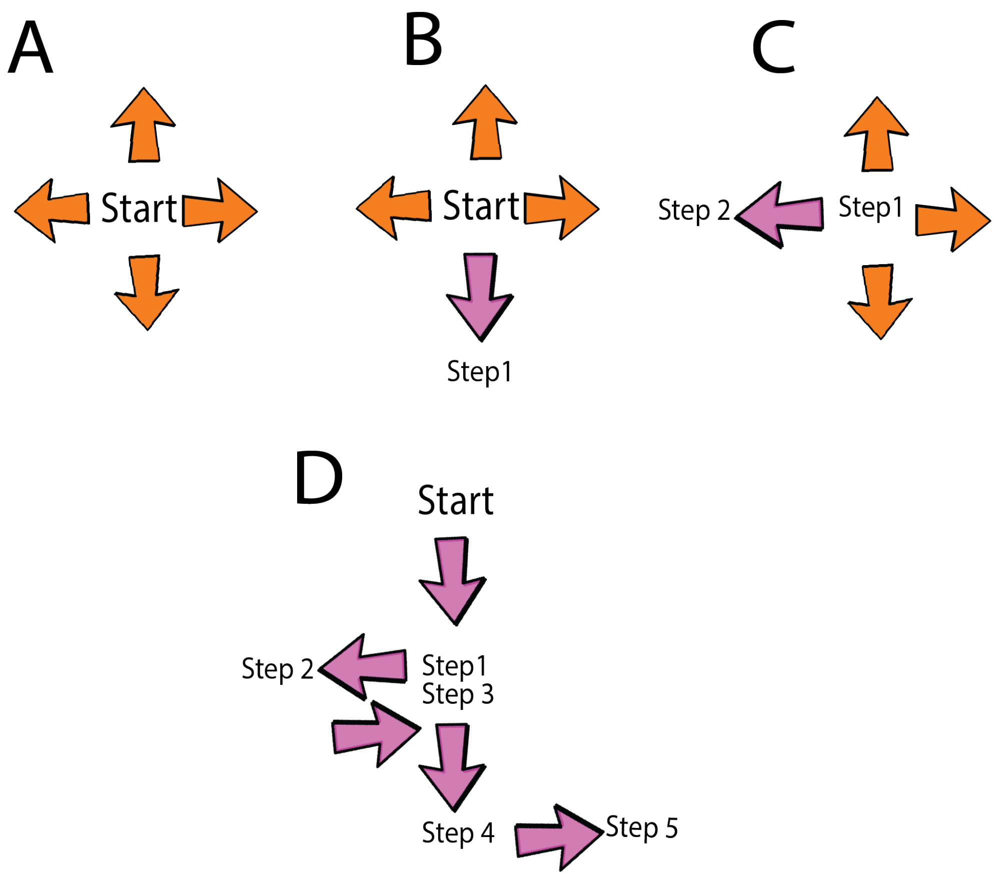

Figure 1: The first few steps of a random walk.

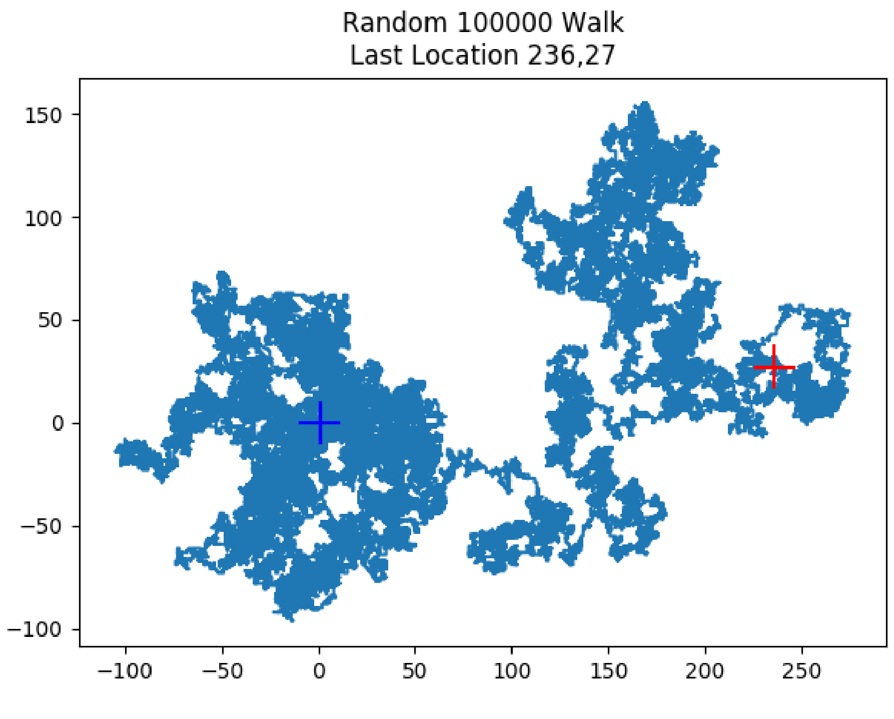

Figure 2: The blue cross is where we started and the red cross is where we ended. Since each run is random, and we have 100,000 steps, your plot will not look similar to mine.

Programming Problem 1: Random Walk

- Complete the `step` function.
- These are *random walks*; hence, the outputs will be different .

Problem 2: 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:

$$d(p_i, p_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \tag{3}$$

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:

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

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

$$d((2,2),(1,1)) = \sqrt{(2-1)^2 + (2-1)^2} = \sqrt{1+1} \approx 1.41 \tag{6}$$

$$d((5,1),(1,1)) = \sqrt{(5-1)^2 + (1-1)^2} = \sqrt{4} = 2 \tag{7}$$

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.*, $d(\mathtt{p1}, \mathtt{p2})$. When this code is run:

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

- Implement both 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 pair, y is the second pair, 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 3: No Silver Bullet

Please read the *No Silver Bullet Essay* pushed to your git. This famous essay discusses the difficulty of designing and implementing software. Your task is two-fold with this problem. In the essay, the author describes a productivity function of two variables, time and number of people. Assume time is forty hours. The function then becomes dependent only on number of people. The task you have here is to complete the function. I have implemented the Newton-Raphson method to find roots (critical values) values shown in Fig. 3.

When I ran this, my output was, including the plot,

```
The maximum productivity is P(62) ~ 896 person x hrs
```

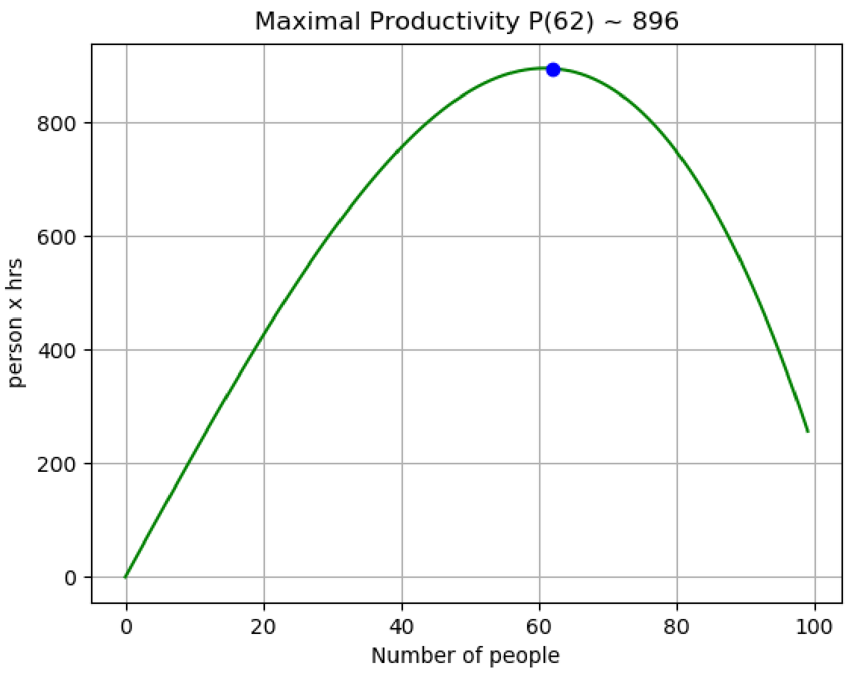

Figure 3: Plotting the productivity function assuming time = 40 hrs/wk as a function of people.

Deliverables for Problem 3

- Implement the ‘productivity‘ function from the paper (the function specifically is given on page 9/13 but the essay is fun to read as well).
- Note that, we have already implemented the Newton() and fp() functions and these are used in the code to plot the graph.

Problem 4: Permutations

In this problem you will take a list of numbers as input and return a list of lists that has all the permutations. For example,

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

returns

```
[[1, 2, 3], [1, 3, 2], [2, 1, 3], [2, 3, 1], [3, 1, 2], [3, 2, 1]]
```

$3! = 6$ which is the number of permutations. To remind you, a permutation from some alphabet $|A| = n$ means $n!$ lists of every possible sequence. For example, if $\mathcal{A} = \{1, 2, 3\}$, then there will be 6 lists (shown above). Adding one more makes 24:

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

gives

```
[[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, 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]]
```

The order of the list itself isn’t important – what’s important is that every possible list is present. You are free to implement this non-recursively or recursively.

Deliverable for Problem 4: Permutations

- Implement the permutation function – you <u>cannot</u> use any modules.
- Hint: Make small toy examples to find the pattern and then you can use that to basically make it work for list of any length.

Problem 5: Vector Class

In this problem, you'll implement a mathematical vector class. While the vectors are depicted as a column, you'll use a tuple. A vector of size $n \in \{1, 2, 3, \ldots\}$ is a collection of real numbers:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \tag{8}$$

Given $\mathbf{x}, \mathbf{y}$ are vectors of size n and z is a real number-we can define the following operations

$$\mathbf{x} + \mathbf{y} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} + \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} x_1 + y_1 \\ x_2 + y_2 \\ \vdots \\ x_n + y_n \end{bmatrix} \tag{9}$$

$$\mathbf{x} - \mathbf{y} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} - \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} x_1 - y_1 \\ x_2 - y_2 \\ \vdots \\ x_n - y_n \end{bmatrix} \tag{10}$$

$$\mathbf{x} \times \mathbf{y} = \sum_{i=1}^{n} x_i y_i \tag{11}$$

$$z \times \mathbf{x} = z \times \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} z \times x_1 \\ z \times x_2 \\ \vdots \\ z \times x_n \end{bmatrix} \tag{12}$$

$$-\mathbf{x} = \begin{bmatrix} -x_1 \\ -x_2 \\ \vdots \\ -x_n \end{bmatrix} \tag{13}$$

$$|\mathbf{x}| = \sqrt{\mathbf{x} \times \mathbf{x}} \tag{14}$$

$$\tag{15}$$

Two vectors $\mathbf{x}, \mathbf{y}$ are equal if $x_i = y_i$ for $1 \leq i \leq n$. The class you'll build will work for any non-zero length vector. The class constructor is:

```python
class Vector:
    def __init__(self, *x):
        self.__v = x
```

The *x means you are making the arguments a tuple–further, a tuple can be sent if it has *. For example,

```python
w = Vector(*(10,10))
```

will send the tuple as a tuple–this makes arguments easier. For a vector $\mathbf{x}$ of length n, print($\mathbf{x}$) should display

```
<x1,x2,...,xn>
```

Here are instances and outputs:

```python
x,y,w = Vector(1,2),Vector(3,-1),Vector(*(10,10))
z,a = Vector(0,3,2),Vector(-1,-1,-1)
print(x,y,z,a)
print(x+y,z+a)
print(x*y,z*a)
print(5*x,5*z)
print(abs(x),abs(z))
print(-x,-z)
print(x - y + y == x, 2 * z == z + z)
```

with output:

```
<1, 2> <3, -1> <0, 3, 2> <-1, -1, -1>
<4, 1> <-1, 2, 1>
1 -5
<5, 10> <0, 15, 10>
2.23606797749979 3.605551275463989
<-1, -2> <0, -3, -2>
True True
```

Deliverable for Problem 5: Vectors

- Complete the class–use a tuple to represent the mathematical vector.
- You can use Python's sum function.
- You are free to use any Python string class method.

Paired Programming

For this assignment, here is your partner. If you find your partner is unresponsive, you are free to join another group.
mabdayem@iu.edu, shadoshi@iu.edu
ahnabrah@iu.edu, bbacso@iu.edu
adamsjf@iu.edu, raia@iu.edu
dadeyeye@iu.edu, mattroac@iu.edu
cmaguila@iu.edu, luilmill@iu.edu
ahmedrr@iu.edu, gabradle@iu.edu
malshama@iu.edu, timbogun@iu.edu
olalbert@iu.edu, cbylciw@iu.edu
nalemanm@iu.edu, linweix@iu.edu
faysalza@iu.edu, asaokho@iu.edu
rnameen@iu.edu, vramkum@iu.edu
svamin@iu.edu, joecool@iu.edu
jaygul@iu.edu, evewalsh@iu.edu
rbajaj@iu.edu, akaushal@iu.edu
cbalbuen@iu.edu, pheile@iu.edu
ikbanist@iu.edu, sahmir@iu.edu
zsbanks@iu.edu, dylomall@iu.edu
mbarrant@iu.edu, johnslia@iu.edu
tymbarre@iu.edu, jjwelp@iu.edu
dcblakle@iu.edu, yangyuc@iu.edu
mlboukal@iu.edu, anniye@iu.edu
logbrads@iu.edu, kamdelmo@iu.edu
lburrola@iu.edu, dazamora@iu.edu
aidcarli@iu.edu, gillenj@iu.edu
dcaspers@iu.edu, vimadhav@iu.edu
mathchen@iu.edu, fraustom@iu.edu
ccoriag@iu.edu, sowvemul@iu.edu
gcruzcor@iu.edu, kjwalapu@iu.edu
ddahodu@iu.edu, tymath@iu.edu
rpdeady@iu.edu, ibnash@iu.edu
cadelaga@iu.edu, yjan@iu.edu
emdelph@iu.edu, yl181@iu.edu
edepke@iu.edu, mvincen@iu.edu
shrdesai@iu.edu, yudsingh@iu.edu
eeconomo@iu.edu, eluthra@iu.edu
ereilar@iu.edu, jpenrigh@iu.edu
jaespin@iu.edu, gkarnuta@iu.edu

mfanous@iu.edu, rosavy@iu.edu
jayfish@iu.edu, mschauss@iu.edu
sydfoste@iu.edu, owenaj@iu.edu
ethfrago@iu.edu, camitong@iu.edu
nfrische@iu.edu, sothor@iu.edu
gaoxinl@iu.edu, egmorley@iu.edu
ggivan@iu.edu, silmudee@iu.edu
bgloor@iu.edu, chlzhang@iu.edu
noahgrah@iu.edu, jthurd@iu.edu
halejd@iu.edu, ejharms@iu.edu
ehallor@iu.edu, evtomak@iu.edu
jchobbs@iu.edu, sj110@iu.edu
binyhu@iu.edu, jk130@iu.edu
srimmadi@iu.edu, chsand@iu.edu
aj110@iu.edu, cl101@iu.edu
gjarrold@iu.edu, hdwatter@iu.edu
mjerrell@iu.edu, kevko@iu.edu
yuljiao@iu.edu, actoney@iu.edu
njindra@iu.edu, mdtanner@iu.edu
cjohanns@iu.edu, gkyoung@iu.edu
fkanmogn@iu.edu, mppan@iu.edu
sskauvei@iu.edu, lancswar@iu.edu
phklein@iu.edu, apoellab@iu.edu
jtkrug@iu.edu, tclady@iu.edu
jhlazar@iu.edu, ashankwi@iu.edu
wlegear@iu.edu, bolabanj@iu.edu
jleverty@iu.edu, remarche@iu.edu
lopezis@iu.edu, jamoya@iu.edu
mlumbant@iu.edu, ndvanbur@iu.edu
pmanolis@iu.edu, rlmcdani@iu.edu
mmansoo@iu.edu, eliserr@iu.edu
mooralec@iu.edu, daparent@iu.edu
jsm13@iu.edu, ryou@iu.edu
hnasar@iu.edu, perkcaan@iu.edu
apapaioa@iu.edu, nps1@iu.edu
divpatel@iu.edu, powelchr@iu.edu
shevphil@iu.edu, grtalley@iu.edu
srpothir@iu.edu, vvictori@iu.edu
sprabhak@iu.edu, cadwilco@iu.edu
jwrohn@iu.edu, lzinn@iu.edu
msronan@iu.edu, ssalama@iu.edu

sasaluja@iu.edu, astrouf@iu.edu
ansiva@iu.edu, cy30@iu.edu
amystaff@iu.edu, gtutton@iu.edu
rwan@iu.edu, weidzhen@iu.edu