C200 Programming Assignment №7 Representation, Comprehension, & Graphics

Dr. M.M. Dalkilic

Computer Science

School of Informatics, Computing, and Engineering

Indiana University, Bloomington, IN, USA

March 18, 2025

Introduction

Due Date: 5:00 PM, Friday, March 28, 2025. Please note that the assignment will need to be submitted to Gradescope and **must be done before the deadline. We do not accept late submissions**. Any form of cheating or plagiarism will be dealt with seriously. If this results in a grade change, we are compelled by the University to send a formal notification to the Dean, please review the policy on academic misconduct in the syllabus.

As always, all the work should be with you and your partner; but *both* of you should contribute. Please remember:

- You will be submitting on Gradescope
- Your highest score of **10 submissions** will be used for your final score.
- Any function that is defined requires a docstring, please refer to lab 03 for what a docstring should have.
- Do NOT change the function names, or the parameters.
- Finally, manual grading will be done, so if you receive a 50 out of 100 on the autograder, it means the lowest grade you can receive is 50. The rest will come from manual grading of the problems.

If your timestamp is 5:01PM or later, the homework will not be graded. So **do not wait until 4:59 PM** to submit, make sure to also plan accordingly. If you have questions or problems with Gradescope, please visit office hours or make a post on Inscribe ahead of time. Since you are working in pairs, your paired partner is shown on canvas. Esure that you are using only features that we have discussed in lectures and labs.

1: Caeser Cipher

The Caeser Cipher (CC) is a means of encoding a message. Please visit: `https://en.wikipedia.org/wiki/Caesar_cipher`
to read more about it (though it's not as complete as one would hope). We have decided to encode messages for this class using the CC, but with a slight modification. We'll only use the lower case alphabet, and we'll include space as an actual symbol. If you look at the character after 'z' in ASCII, you'll see the left brace "{".

```
>>> for i in range(ord('x'), ord('x') + 4):
...     print(f"{i} is {chr(i)} in ASCII")
...
'120 is x in ASCII'
'121 is y in ASCII'
'122 is z in ASCII'
'123 is { in ASCII'
>>>
```

We are including "{" in our alphabet after z as the symbol for space. You'll write two functions, encode(msg, shift) and decode(msg, shift) that encode a message and decode the message using shift as discussed in the wikipedia page, except our alphabet is one symbol larger. The following code:

```
data = ["abc xyz","the cat", "i love ctwohundred"]
for i,j in enumerate(data,start=2):
    print(f"original msg {j}")
    print(f"encoded  msg {encode(j,i)}")
    print(f"decoded  msg {decode(encode(j,i),i)}")

secret_msg = encode("the quick brown fox jumps over the lazy dog", 24)
print(secret_msg)
print(decode(secret_msg,24))
```

produces:

```
original msg abc xyz
encoded  msg cdebz{a
decoded  msg abc xyz
original msg the cat
encoded  msg wkhcfdw
decoded  msg the cat
original msg i love ctwohundred
encoded  msg mdpszidgx{slyrhvih
decoded  msg i love ctwohundred
qebxnrf{hxzoltkxcluxgrjmpxlsboxqebxiywvxald
```

```
11 the quick brown fox jumps over the lazy dog
```

Our output is obviously different from the wikipedia example, since we're expanding our alphabet to consider space a letter and including that in the shift.

Programming Problem 1: Caeser Cipher

- Complete the functions.
- You can use any normal string function that is associated with the class.

2: Tiling

In this problem, you're given a collection of tiles as numbers, for example, [1,2,3]. You're given a space to fit the tiles (linearly), for example 6. Your function gives all the possible patterns where the tiles add exactly to the fit. For example,

```
1   n = 6
2   v = [1,2,3]
3   print(tiles(n,v,[[i] for i in v]))
4   for i in tiles(n,v,[[i] for i in v]):
5         print(sum(i), end="")
```

produces:

```
1 [[3, 3], [3, 2, 1], [2, 3, 1], [3, 1, 2], [2, 2, 2], [1, 3, 2], [2, 1,↩
      3],
2  [1, 2, 3], [3, 1, 1, 1], [2, 2, 1, 1], [1, 3, 1, 1], [2, 1, 2, 1],
3  [1, 2, 2, 1], [1, 1, 3, 1], [2, 1, 1, 2], [1, 2, 1, 2], [1, 1, 2, 2],
4  [1, 1, 1, 3], [2, 1, 1, 1, 1], [1, 2, 1, 1, 1], [1, 1, 2, 1, 1],
5  [1, 1, 1, 2, 1], [1, 1, 1, 1, 2], [1, 1, 1, 1, 1, 1]]
6 666666666666666666666666
```

All the possible configurations and all equal 6. For numbers [1,2] and size = 4, we have:

```
1 [[2, 2], [2, 1, 1], [1, 2, 1], [1, 1, 2], [1, 1, 1, 1]]
2 44444
```

The function tiles takes n (the total size), v (the different numbers), and the starter tiles. When approaching this problem, you should inspect the output–it is helpful wrt what the recursion does.

Programming Problem 2: Tiling

- Complete the function
- You are free to use any form of looping, but recursion with comprehension seems to be the simplest approach

3: Base 17

In this problem, you'll write a function that takes a number in base 17 (as a string) and return the decimal value. We'll simply extend base 16 by adding another digit G. Since F represents 15, G will represent 16. Interestingly, Python can interpret base 17 as we've imagined:

```
>>> int("G", 17)
16
>>> int("E2",17)
240
>>> int("10",17)
17
```

Programming Problem 3: Base 17

- Complete the function.
- You **cannot** use the integer function int(x,y) to convert to decimal. You must build the decimal value from powers of 17. You can still use int(x). Revising the lectures slides on conversion between number system can help to understand and solve this problem.

Problem 4: City Block Distance

For this problem, we will only consider integer values. When calculating distance, typically Euclidean distance (yielding a decimal) is used:

$$d(p,q) = \sqrt{\sum_{1}^{n}(p_i - q_i)^2} \tag{1}$$

In practical settings, however, we need to adhere to the context. For travel in a city, for example, we must use blocks. In this problem we assume that blocks are squares. Each intersection represents the joining of four roads. Distance as city blocks is:

$$d(p,q) = \sum_{1}^{n}|p_i - q_i| \tag{2}$$

Consider Fig. 1. that shows the city block distance between the post office (red) and the recently chapter 11 TGIF (green). The path length is three. There are several routes that are three blocks, but that is distance you'll have to travel. In this problem, you'll implement three functions:

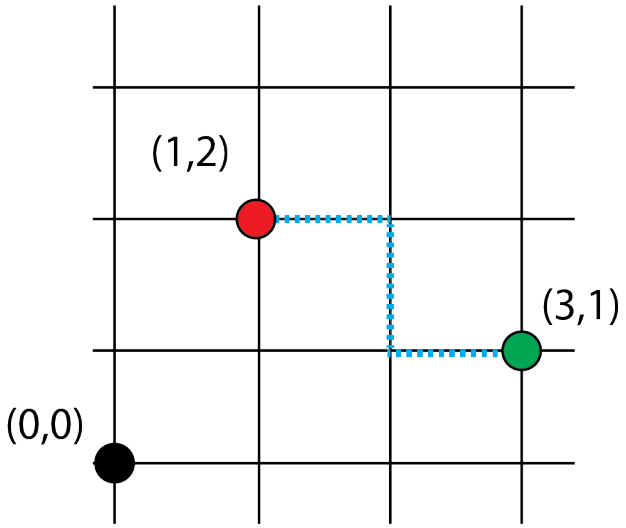

Figure 1: We have a city with straight, square roads. The center of town is at the origin (0,0). The city block distance between red and green is $d(p,q) = |1-3| + |2-1| = 3$. The blue dotted lines gives one of these paths; there are several so long as the number of blocks is three.

- `block_distance(p0,p1)` that gives city block distance between points `p0,p1`;
- `get_points(center,bd)` that generates all the points whose city block distance is `bd` or less;
- `intersection(x,y)` that yields the shared points in the lists `x,y` of points.

We're including a visualization to make the problem easier. The following code:

```
1
2 A = ((0,-1),2)
```

```
B = ((0,1),1)
C = ((4,4),1)
p = get_points(*A)
q = get_points(*B)
r = intersection(p,q)
s = get_points(*C)
t = intersection(s,q)

for points in p,q,r,s:
    print(points)

#begin matplotlib code
color = 'rgbmy'

for i,pts in enumerate([p,q,r,s,t]):
    plt.plot([x for x,_ in pts],[y for _,y in pts],color[i] + 'o')

plt.gca().legend(("A: ((0,-1),2)", "B: ((0,1),1)", r"$\mathsf{A}\cap\mathsf{B}$","C: ((0,1),1)", r"$\mathsf{B}\cap\mathsf{C}$"))
plt.axis([-7, 7, -7, 7])
plt.grid()
plt.gca().set_aspect("equal")

plt.grid(True)
plt.title("City with square streets.")
plt.show()
#end matplotlib code
```

produces:

```
[(-2, -1), (-1, -2), (-1, -1), (-1, 0), (0, -3), (0, -2), (0, -1), (0, 0), (0, 1), (1, -2), (1, -1), (1, 0), (2, -1)]
[(-1, 1), (0, 0), (0, 1), (0, 2), (1, 1)]
[(0, 0), (0, 1)]
[(3, 4), (4, 3), (4, 4), (4, 5), (5, 4)]
```

with visualization:

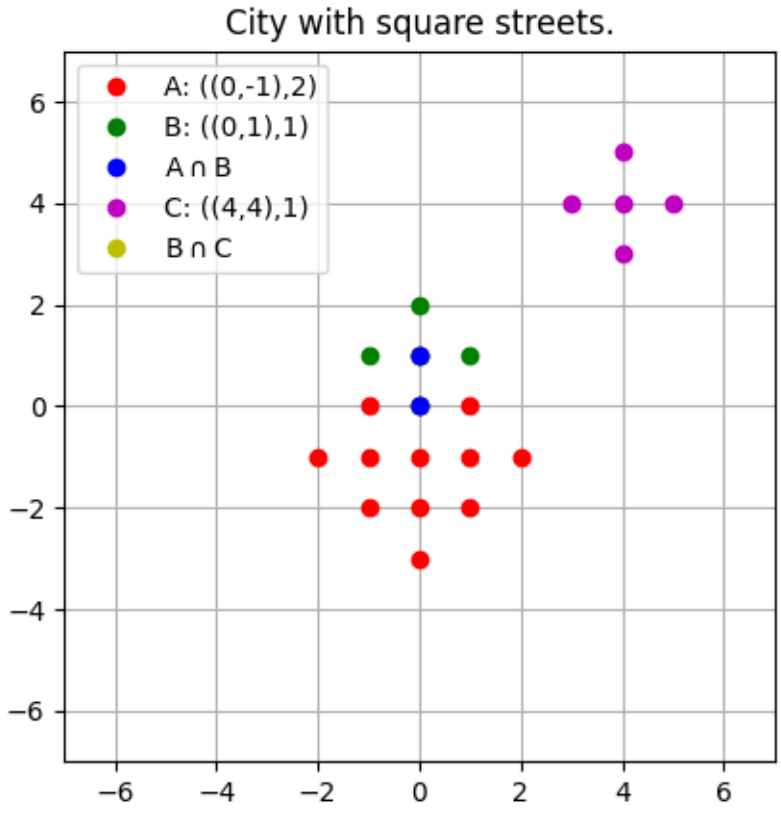

Figure 2: Observe no intersection between B,C.

Programming Problem 4: City Block Distance

- Complete the three functions.
- Implement `intersection(x,y)` as a single `return` statement using list comprehension. The function will look like this:

```
1 def intersection(x,y):
2     return [] #list comprehension
```

- Hint: *It is likely easier to generate more points then use the city block distance to select the ones that are within the distance than to code generating the exact points.*
- The order of the points is not important.