## 15-110 Hw4 - Written Portion

## Name:

## AndrewID:

---

Complete the following problems in the fillable PDF, or print out the PDF, write your answers by hand, and scan the results.

When you are finished, upload your hw4.pdf to **Hw4 - Written** on Gradescope, and upload your hw4.py file to **Hw4 - Programming** on Gradescope. Make sure to check the autograder feedback after you submit!

Don't forget that you can get three bonus points on Hw4 by filling out the midsemester surveys! Check Piazza for links to those surveys and further instructions.

# Written Problems

## #1 - Tracing Graph Code - 6pts

*Can attempt after Graphs lecture*

The function below takes a graph (in adjacency list format) and the value of a node in that graph. The graph represents a map, where nodes are cities and edge weights are distances between cities. The graph shown to the right could be an input to this function.

```
def mystery(g, node):
    a = None
    b = 1000
    for val in g[node]:
        c = val[1]
        if c < b:
            a = val[0]
            b = c
    return a
```

```
g = {
  "Detroit" :       [ ["New York City", 587],
                      ["Pittsburgh",    277],
                      ["Washington DC", 499] ],
  "New York City" : [ ["Detroit",       587],
                      ["Philadelphia",   88] ],
  "Pittsburgh" :    [ ["Detroit",       277],
                      ["Philadelphia",  289],
                      ["Baltimore",     224],
                      ["Washington DC", 223] ],
  "Philadelphia" :  [ ["New York City",  88],
                      ["Pittsburgh",    289],
                      ["Baltimore",      99] ],
  "Baltimore" :     [ ["Pittsburgh",    224],
                      ["Philadelphia",   99],
                      ["Washington DC",  36] ],
  "Washington DC" : [ ["Detroit",       499],
                      ["Pittsburgh",    223],
                      ["Baltimore",      36] ]
}
```

In the context of the map graph (using non-code terms), what is the relationship between the variable **node** and the variable **c** in a specific loop iteration?

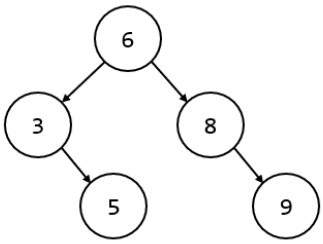
If you trace through the code with the graph shown above and the node set to the value "Philadelphia", what values do **a** and **b** hold at the end of the function call?

In general (abstract, non-code) terms, for any given map graph and city node, what does this function return?

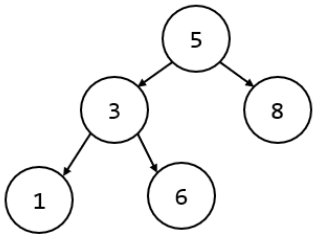
# #2 - Tree Identification - 5pts

*Can attempt after Search Algorithms II lecture*

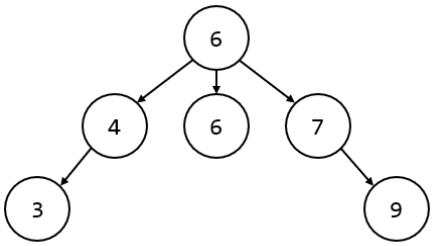
For each of the following trees, determine whether or not it is a Binary Search Tree.



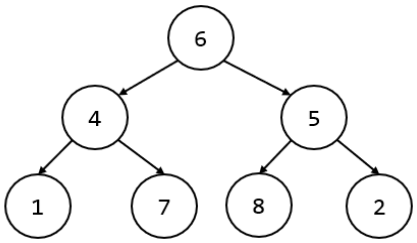

☐ BST
☐ Not BST



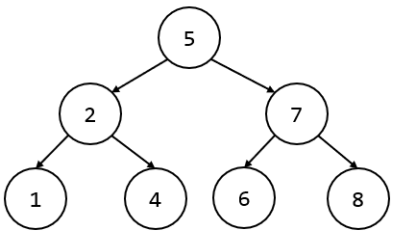

☐ BST
☐ Not BST




☐ BST
☐ Not BST




☐ BST
☐ Not BST

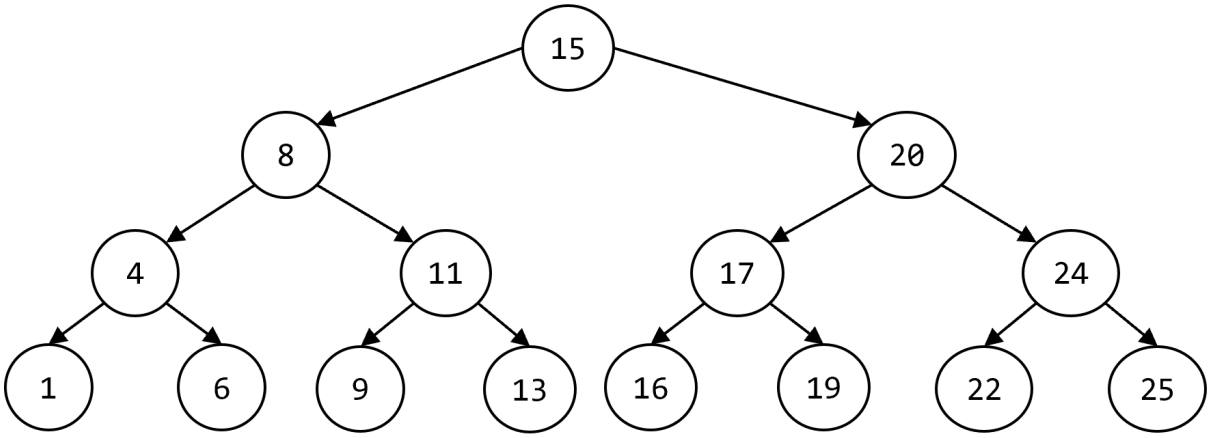



☐ BST
☐ Not BST

## #3 - Searching a BST - 9pts

*Can attempt after Search Algorithms II lecture*

Given the Binary Search Tree shown below:




What series of numbers would you visit if you ran a search algorithm that looked for **19**?

What series of numbers would you visit if you ran a search algorithm that looked for **4**?

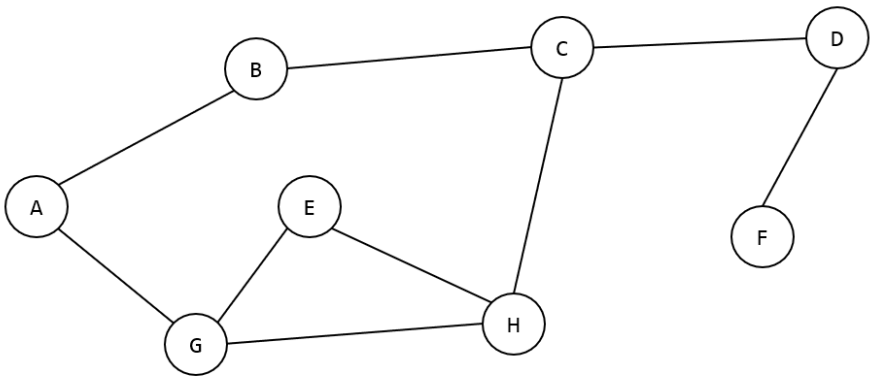
What series of numbers would you visit if you ran a search algorithm that looked for **10**?

## #4 - Searching a Graph - 12pts

*Can attempt after Search Algorithms II lecture*

For this problem, note that each prompt has multiple correct answers; you only need to include one. **We recommend that you visit neighbors in alphabetical order.**

Given the undirected graph shown below, where the letter **A** is the start node:




What series of letters could you visit if you ran **Depth-First Search** to find **H**?
(For this and the following problems, there is more than one correct answer.)

What series of letters could you visit if you ran **Breadth-First Search** to find **H**?

What series of letters could you visit if you ran **Depth-First Search** to find **J**?

What series of letters could you visit if you ran **Breadth-First Search** to find **J**?

# #5 - P and NP Identification - 10pts

*Can attempt after Tractability lecture*

For each of the following problems, identify whether the problem is in the complexity class P, NP, or neither. Please choose just one class (the one that **best** describes the problem), even if multiple classes are technically correct.

| | |
|---|---|
| Finding the smallest value in a tree | ☐ P<br>☐ NP<br>☐ Neither |
| Scheduling final exams for CMU so that there are no conflicts | ☐ P<br>☐ NP<br>☐ Neither |
| Determining if an item is in a list | ☐ P<br>☐ NP<br>☐ Neither |
| Finding the **best** (fastest) road route through Pittsburgh that takes you over every bridge. | ☐ P<br>☐ NP<br>☐ Neither |
| Determining if there is a set of inputs that makes a circuit output 1 | ☐ P<br>☐ NP<br>☐ Neither |

## #6 - Tractability - 6pts

*Can attempt after Tractability lecture*

Select whether each of the following function families is **tractable** or **intractable**.

| | | |
|---|---|---|
| O(n) | ☐ Tractable | ☐ Intractable |
| O(n!) | ☐ Tractable | ☐ Intractable |
| $O(2^n)$ | ☐ Tractable | ☐ Intractable |
| $O(n^2)$ | ☐ Tractable | ☐ Intractable |
| O(log n) | ☐ Tractable | ☐ Intractable |
| $O(n^{100})$ | ☐ Tractable | ☐ Intractable |

## #7 - P vs NP - 9pts

*Can attempt after Tractability lecture*

Which of the following is the *best* definition of the complexity class **P**?

- ☐ The set of problems that can be solved in polynomial time
- ☐ The set of problems that can be verified in polynomial time
- ☐ The set of problems we discussed in lecture (Puzzle solving, Subset Sum, etc)

Which of the following is the *best* definition of the complexity class **NP**?

- ☐ The set of problems that can be solved in polynomial time
- ☐ The set of problems that **cannot** be solved in polynomial time
- ☐ The set of problems that can be verified in polynomial time
- ☐ The set of problems that **cannot** be verified in polynomial time
- ☐ The set of problems we discussed in lecture (Puzzle solving, Subset Sum, etc)

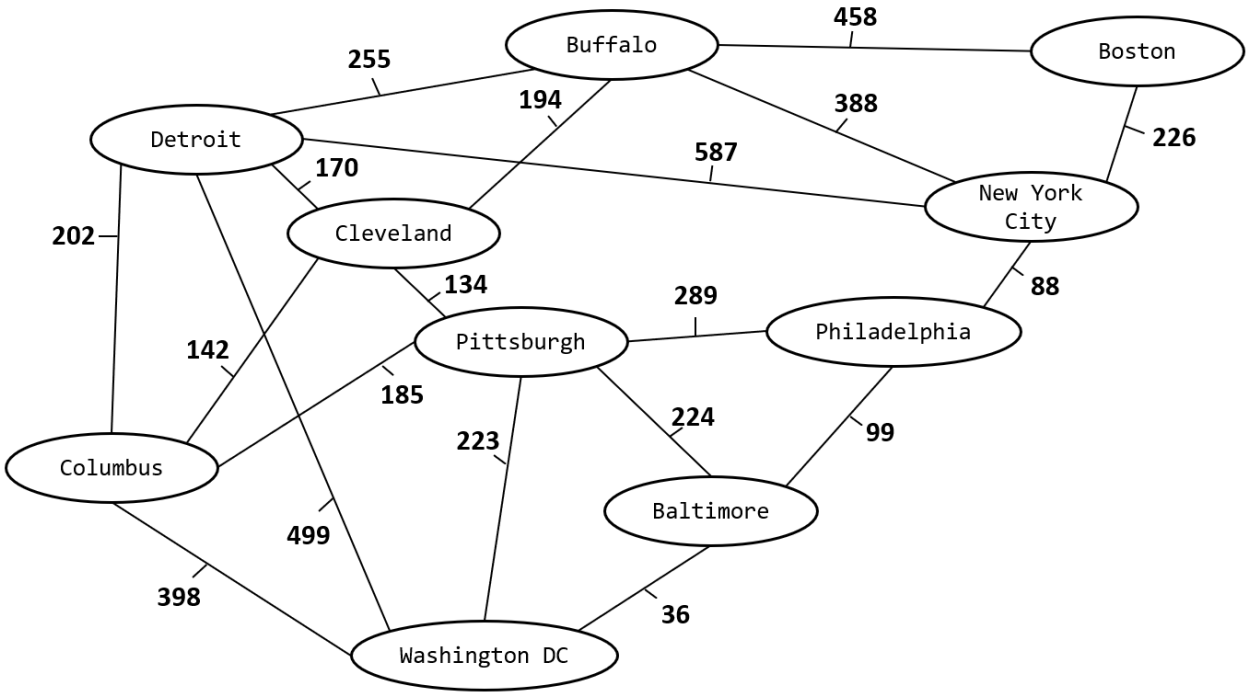
Why does it matter whether or not P = NP? Choose the *best* answer.

- ☐ If they are the same, we'll be able to solve hard and useful problems a lot faster
- ☐ If they are the same, we'll need to change how we implement some adversarial algorithms, like encryption, to keep them from being broken easily
- ☐ If they are not the same, we can spend less time trying to invent super-fast solutions to hard but useful problems
- ☐ All of the above

# #8 - Heuristics - 4pts

*Can attempt after Tractability lecture*

We want to apply the Travelling Salesperson algorithm to the graph shown here to find a short route that visits each city once, but we want to use a **heuristic** to get the answer quickly. Our heuristic is this: rank the neighbor cities that have not yet been visited based on the weight of the edge leading to them, and choose the lowest-weight edge (the shortest distance).




Say we want to start in New York City and visit each city once (returning to New York City at the end). What path would this heuristic generate?

# #9 - Recognizing Data Structures - 5pts

*Can attempt after Graphs lecture*

For each of the following types of data, choose the data structure that would be the **best/most natural choice** to represent the data

| | |
|---|---|
| Carnegie Mellon's organizational structure: ie, departments within each college, and majors within each department | ☐ 1D List<br>☐ 2D List<br>☐ Dictionary<br>☐ Tree<br>☐ Graph |
| A chess board that has pieces located at specific row-column positions | ☐ 1D List<br>☐ 2D List<br>☐ Dictionary<br>☐ Tree<br>☐ Graph |
| A set of chores you need to do over the weekend | ☐ 1D List<br>☐ 2D List<br>☐ Dictionary<br>☐ Tree<br>☐ Graph |
| The subway map for London | ☐ 1D List<br>☐ 2D List<br>☐ Dictionary<br>☐ Tree<br>☐ Graph |
| A deck of flashcards with words on one side and definitions on the other | ☐ 1D List<br>☐ 2D List<br>☐ Dictionary<br>☐ Tree<br>☐ Graph |

# #10 - Search Efficiencies - 8pts

*Can attempt after Tractability lecture*

For each of the following prompts, consider the search algorithms we've discussed in class to determine the optimal Big-O runtime for the given situation.

We store a music collection in a dictionary mapping artists to lists of their songs. What is the runtime to find whether a specific artist is in a collection if n is the size of the dictionary?

- ☐ O(1)
- ☐ O(log n)
- ☐ O(n)
- ☐ O(n log n)
- ☐ $O(n^2)$
- ☐ $O(2^n)$

We store a game of 20 questions in a binary tree where each node is a question whose left branch answers 'yes' and whose right branch answers 'no'. The leaves hold all eventual solutions. What is the runtime to determine whether *any* of the answer paths lead to the solution 'octopus' if n is the number of nodes in the tree?

- ☐ O(1)
- ☐ O(log n)
- ☐ O(n)
- ☐ O(n log n)
- ☐ $O(n^2)$
- ☐ $O(2^n)$

We store all current players in the Basketball Hall of Fame in a list, organized alphabetically by name. What is the runtime to find whether a specific player is in the Hall of Fame if n is the length of the list?

- ☐ O(1)
- ☐ O(log n)
- ☐ O(n)
- ☐ O(n log n)
- ☐ $O(n^2)$
- ☐ $O(2^n)$

We make a graph of everyone in the world where two people are connected if they have ever met. What is the runtime to find if there is a chain of introductions that links you to Oprah Winfrey if n is the number of nodes in the graph?

- ☐ O(1)
- ☐ O(log n)
- ☐ O(n)
- ☐ O(n log n)
- ☐ $O(n^2)$
- ☐ $O(2^n)$

# #11 - Optimizing for Search - 6pts

*Can attempt after Search Algorithms II lecture*

You have been given a very large dataset of temperatures (represented as floats), and your task is to find the most extreme temperatures that fall into a given temperature range (such as 40 degrees to 50 degrees, or 75.7 degrees to 78.2 degrees). To do this, you want to store the data in a data structure so that, given any range, you'll be able to:

- find the smallest value in the structure that falls in that range
- find the largest value in the structure that falls in that range

**You want to optimize how quickly you can run the algorithm shown above, assuming the data structure has already been created.** In other words, **you don't know what range you'll need to check when you create the structure.**

Choose the best search algorithm + data structure combination for the task. There might be multiple correct answers; you only need to choose one per question.

**Search Algorithm:**

- ☐ Linear Search
- ☐ Binary Search
- ☐ Hashed Search
- ☐ Breadth-First Search

**Data Structure:**

- ☐ Sorted List of degrees
- ☐ Dictionary mapping degree->count
- ☐ Binary Search Tree of degrees
- ☐ Graph connecting close degrees

# Programming Problems

For each of these problems (unless otherwise specified), write the needed code directly in the Python file, in the corresponding function definition.

All programming problems may also be checked by running 'Run File As Script' on the starter file, which calls the function `testAll()` to run test cases on all programs.

## #1 - `findParents(t, name)` - 10pts

*Can attempt after Search Algorithms II lecture*

Write the function `findParents(t, name)` which takes a genealogical family tree in the form of a binary tree (as discussed in class) and a string, the name of a person in that family tree. The function returns a list containing the named person's parents as shown by the tree.

You'll need to use if statements to separate the four possible arrangements of parents.

- Return an empty list if the parents are unknown
- Return a singleton list in the two cases where only one parent is known
- Return a two-element list if both parents are known.

You are guaranteed that every name in the tree only shows up once (to avoid conflicting information) and that the person being searched for is in the tree.

*(cont'd on next page)*

For example, consider the following family tree (as shown in the course slides):

```
t = { "contents" : "Arya",
      "left"  : { "contents" : "Ned",
                  "left"  : { "contents" : "Rickard",
                              "left"  : None, "right" : None },
                  "right" : { "contents" : "Lyarra",
                              "left"  : None, "right" : None } },
      "right" : { "contents" : "Catelyn",
                  "left"  : { "contents" : "Hoster",
                              "left"  : None, "right" : None },
                  "right" : { "contents" : "Minisa",
                              "left"  : None, "right" : None } } }
```

If we called `findParents(t, "Ned")`, the function would return the list `["Rickard", "Lyarra"]`.

If we called `findParents(t, "Rickard")`, the function would return `[ ]`.

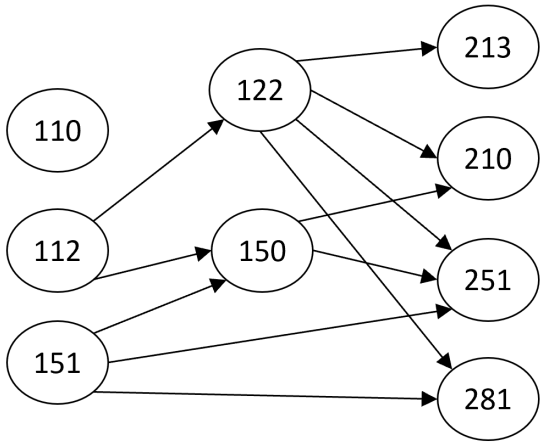
**Hint 1:** treat this like a recursive search problem (in fact, you might want to reference tree linear search and binary search if you get stuck). You'll need to make **two** base cases - one for when you find the person, one for when you reach a leaf or an empty tree. For this problem, it may be easier to set up the base case as an empty tree.

**Hint 2:** unlike recursive binary search, you need to check **both** branches of the tree to see if the person occurs in either branch. How can you combine their results? If the person is found in one of the branches, you should get a list with the parents; if they are *not* found, you'll get an empty list. Just concatenate the two results together to get the final result.

## #2 - `getPrereqs(g, course)` - 10pts

*Can attempt after Graphs lecture*

College course prerequisites are notoriously complicated. However, we can make them a little easier to understand by representing the course dependency system as a **directed graph**, where the nodes are courses and an edge leads from course A to course B if A is a prerequisite of B. For example, the core Computer Science courses (almost) produce the following prereq graph:




Which would be represented in code as:

```
g = { "110" : [],
      "112" : ["122", "150"],
      "122" : ["213", "210", "251", "281"],
      "151" : ["150", "251", "281"],
      "150" : ["210", "251"],
      "213" : [],
      "210" : [],
      "251" : [],
      "281" : [] }
```

Write the function `getPrereqs(g, course)` that takes a directed graph (in our adjacency list dictionary format, without weights) and a string (a course name) and returns a list of all the immediate prerequisites of the given course. If we called `getPrereqs` on our graph above and `"210"`, for example, the function should return `["122", "150"]`.

**Hint:** you can't just return the neighbors of the course, because the edges are going in the opposite direction! Instead, iterate over all the nodes to find those that have the course as a neighbor. Construct a new list out of these nodes as the result.