Problem Set 5

Due before the start of lecture on December 12, 2018.
No submissions will be accepted after Sunday, December 16, at 11:59 p.m. so that we can post solutions before the final exam.

Preliminaries

Homework is due prior to the start of lecture. If it is submitted more than 10 minutes after the start of lecture, it will be considered a full day late. There will be a 10% deduction for late submissions that are made by 11:59 p.m. on the Sunday after the deadline. ***For this assignment, no submissions will be accepted after Sunday, December 16, at 11:59 p.m. so that we can post solutions before the final exam.***

In your work on this assignment, make sure to abide by the policies on academic conduct described in the syllabus.

If you have questions while working on this assignment, please attend office hours, post them on Piazza, or email `cscie22@fas.harvard.edu`.

The Problems

100 points total

Notes:

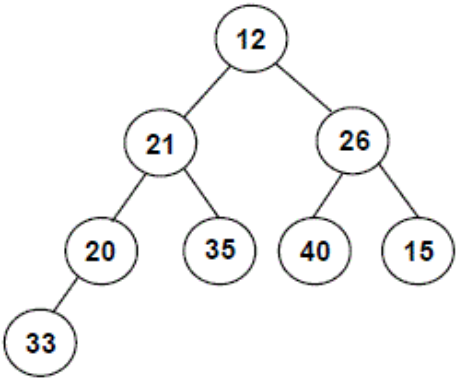
- There are no extra grad-credit problems for this problem set.
- There is only one part to this problem set, and all of your answers should be submitted in a single file. Although there are problems that require you to write a Java method, you should submit those methods in the same file as your other work.

- Because we are asking you to draw diagrams, you may submit *either* a plain-text file or a PDF file. Give it the name `ps5.txt` *or* `ps5.pdf`.

Problem 1: Heaps and heapsort

10 points total

Consider the following complete tree:



1. (3 points) Illustrate the process of turning this tree into a max-at-top heap. Show the tree after each sift operation.
2. (2 points) What is the array representation of the max-at-top heap that you obtain in part 1?
3. (5 points) Heapsort begins by turning the array to be sorted into a heap. Assume that your answer to part 2 is the result of this process of turning the original array into a heap. Illustrate the remaining steps in using heapsort on this array. Show the contents of the array after each element is put into its final position – i.e., at the end of each iteration of the `while` loop in the `heapSort` method from lecture.

Problem 2: Hash tables

12 points total; 4 points each part

The following sequence of keys is to be inserted into an initially empty hash table of size 8:

```
the, my, an, by, do, we, if, to, go
```

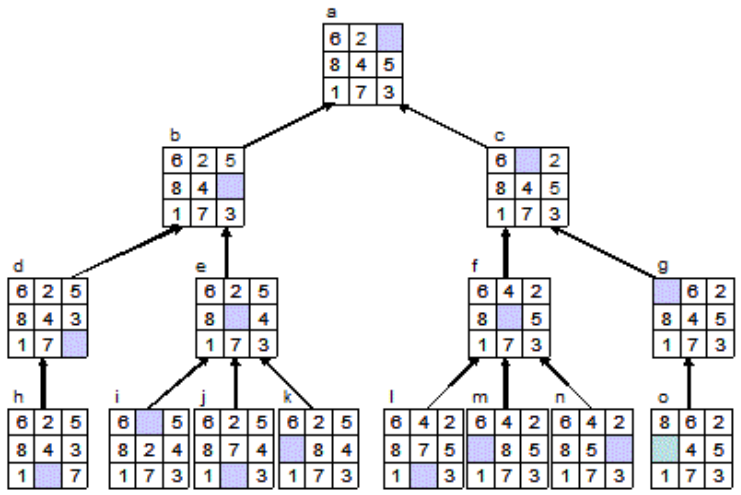
The hash function assigns to each key the number of characters in the key. For example, `h("cat")` is 3, because `"cat"` has 3 characters.

1. Assume that *linear probing* is used to insert the keys. Determine which key causes overflow, and show the table at that point.
2. Now assume that *quadratic probing* is used. Determine which key causes overflow, and show the table at that point.
3. Finally, assume that *double hashing* is used, with the second hash function assigning 1 to a noun or pronoun (we, my), 2 to a verb (do, go), 3 to an article (the, an), and 4 to any other part of speech (the rest of the words). Determine which key causes overflow, and show the table at that point.

Problem 3: Informed state-space search

8 points total; 4 points each part

Below is a portion of a state-space search tree for the Eight Puzzle whose initial configuration is shown at the root of the tree. The individual states have been labeled with lower-case letters to make it easier to refer to them.



1. What are the priorities that greedy search would assign to the following states: state a, state b, state d, and state h? Assume that the Manhattan distance heuristic from lecture is used to estimate the remaining cost.

2. What are the priorities that A* search would assign to the same four states? Assume again that the Manhattan distance heuristic from lecture is used to estimate the remaining cost.

Problem 4: Determining if an array is a heap

10 points total; 5 points each part

Consider the following static methods, which are meant to determine if an array of integers corresponds to a heap:

```
public static boolean isHeap(int[] arr) {
    if (arr == null) {
        return false;
    }

    return isHeapTree(arr, 0);
}

private static boolean isHeapTree(int[] arr, int i) {
    // Implement this method.

}
```

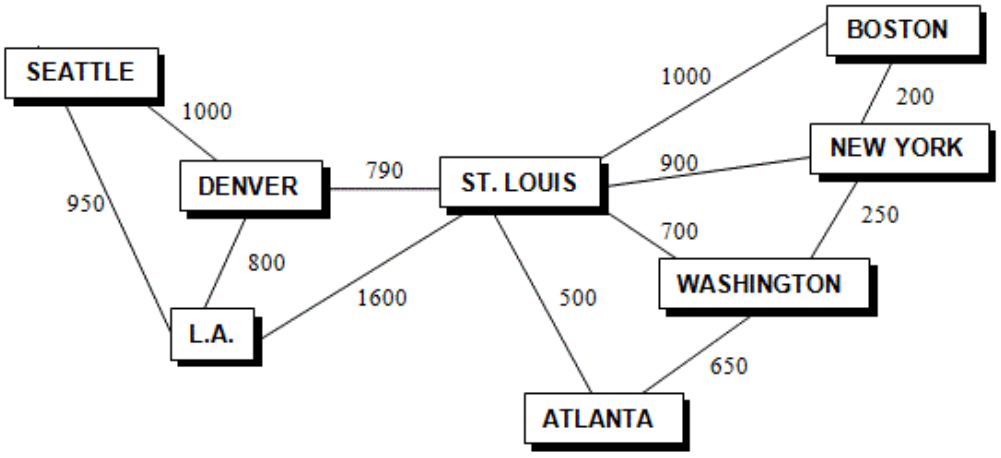
1. Implement the second method so that it uses recursion to process the complete tree/subtree whose root is at position `i` in the array `arr` – i.e., the complete tree/subtree given by the array/subarray

   ```
   {arr[i], arr[i+1], arr[i+2], … arr[arr.length-1]}
   ```

   The method should return `true` if that tree/subtree is a heap, and `false` if it is not a heap. (Note that the first method – which is a public wrapper method for the method that you will write – makes the initial call to your method with a value of `0` for `i`, which means that it will determine if the complete tree represented by the full array is a heap.) In implementing your method, you may find it helpful to review the arithmetic rules for navigating through a complete tree that is stored in an array.

2. What is the efficiency of determining if an array of length `n` represents a heap in the best case? In the worst case? Use big-O notation, and explain your answers briefly.

Problems 5-7 refer to the following graph:



Problem 5: Graph traversals

8 points; 2 points each part

Suppose that you have purchased a pass that allows you to fly anywhere you wish along the routes that are shown in the diagram above.

1. List the order in which you will visit the cities if you start from L.A. and do a breadth-first traversal. You should assume that the edges of each vertex are stored in order of increasing distance, as we did in the lecture examples.

2. What is the path from L.A. to New York in the breadth-first spanning tree? Give the path in the form

   ```
   A -> B -> C ->  etc.
   ```

   where `A`, `B`, and `C` are vertices.

3. List the order in which you will visit the cities if you start from L.A. and do a depth-first traversal. You should assume that the edges of each vertex are stored in order of increasing distance, as we did in the lecture examples.

4. What is the path from L.A. to New York in the depth-first spanning tree? Give the path in the form

   ```
   A -> B -> C ->  etc.
   ```

   where `A`, `B`, and `C` are vertices.

Problem 6: Minimal spanning tree

8 points

A cheaper version of the same pass requires that you confine yourself to flights that are part of a minimal spanning tree for the graph. List the order in which flights/edges will be added to this tree if you build it using Prim's algorithm, starting from L.A. Use the form (city1, city2) when specifying an edge.

Problem 7: Dijkstra's shortest-path algorithm

8 points total

Suppose you set out to use Dijkstra's algorithm to determine the shortest distance from L.A. to every other city in the graph.

1. (5 points) Make a table showing the order in which the cities are finalized and the minimum distance to each.
2. (3 points) What path(s) does the algorithm discover from L.A. to Boston? Include both the final shortest path and any temporary estimates for the shortest path that are later replaced. Give the paths in the form

   ```
   A -> B -> C ->  etc.
   ```

   where `A`, `B`, and `C` are vertices.

Problem 8: Comparing data structures

8 points

A supermarket chain wants you to implement an in-memory database that can be used to access facts about the products they sell. Although a snapshot of this database will be periodically copied to disk, its contents fit in memory, and your component of the application will operate only on data stored in memory.

Here are the requirements specified by the managers of the supermarket chain:

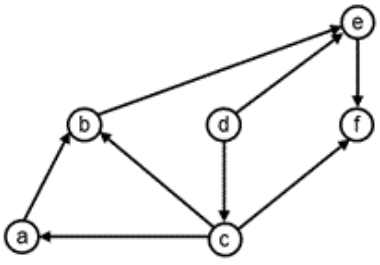
- They want to be able to retrieve product records by specifying the name of the product.
- They want to be able to specify the first n characters of a product name and to retrieve all records that begin with those characters.
- They want the time required to retrieve a record to be as efficient as possible – on the order of 20 operations per retrieval, given a database of approximately one million records.
- They want to be able to increase the size of the database – adding large sets of new records – without taking the system offline.

Given this list of requirements, which data structure would be the better choice for this application, a balanced search tree or a hash table – or would these two data structures work equally well? Explain your decision, specifying as many reasons as possible for the choice that you make.

Problem 9: Directed graphs and topological sort

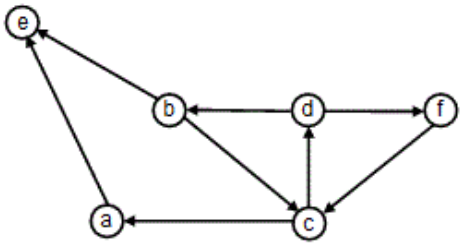
10 points total; 5 points each part

1. Is the graph below a directed acyclic graph (a DAG)?

   If it isn't a DAG, specify *all* cycles that are present in the graph. If it is a DAG, use topological sort to find *one* of the possible topological orderings of the vertices in the graph.

2. Repeat the process outlined above on the graph below.



Problem 10: Determining if two vertices are adjacent

10 points total

1. (3 points) Given the graph representation used in the `Graph` class from lecture, write a simple Java method that takes two `Vertex` objects as parameters and determines if the vertices are adjacent. The method should return `true` if the vertices are adjacent, and `false` if they are not adjacent. Keep in mind that the edges in the graph may be directed, which means that you may need to check both vertices' adjacency lists.

2. (3 points) If a graph with *V* vertices were represented using the `Graph` class from lecture, what would be the worst-case time efficiency of the algorithm from part a? Use big-O notation, and explain your answer briefly.

3. (4 points) We can make the process of determining if two vertices are adjacent more efficient if we modify the representation of the graph. Describe a modification that will achieve this goal, and explain how your algorithm would be changed to take advantage of the changed representation. State the worst-case time efficiency of the more efficient algorithm using big-O notation, and explain your answer briefly.

Problem 11: Alternative MST algorithm

8 points

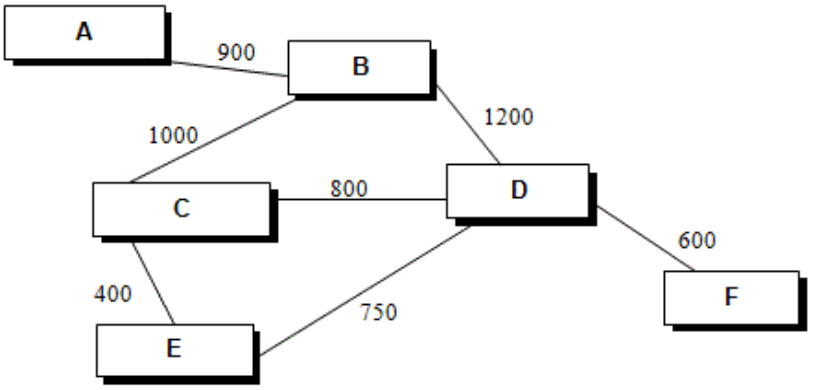
Prim's algorithm is just one possible algorithm for finding a minimum spanning tree. Another such algorithm was developed by Kruskal:

```
kruskal_MST(Graph g) {
    put each of g's vertices in its own set

    while (there is more than one set) {
        let e = the minimum-cost edge that hasn't been considered

        if (e connects vertices that are in different sets) {
            add e to the MST
            merge the sets containing the vertices connected by e
        }
    }
}
```

Note that this algorithm considers the edges in order of increasing cost. In addition, at an intermediate stage of the algorithm, there may be multiple trees that are not connected to each other, although they will ultimately be joined together to form a single MST.



For example, if we applied Kruskal's algorithm to the graph above, we would start out with the following sets:

```
{A}, {B}, {C}, {D}, {E}, {F}
```

We would consider the edge (C, E) first, because it has the lowest cost (400). Because it connects vertices in different sets, we would add this edge to the tree and merge the sets involved to get:

```
{A}, {B}, {C, E}, {D}, {F}
```

We would next consider the edge (D, F), because it has the smallest remaining cost. Because it connects vertices in different sets, we would add this edge to the tree and merge the sets involved to get:

```
{A}, {B}, {C, E}, {D, F}
```

We would next consider the edge (D, E), because it has the smallest remaining cost. Because it connects vertices in different sets, we would add this edge to the tree and merge the sets involved to get:

```
{A}, {B}, {C, D, E, F}
```

We would next consider the edge (C, D), because it has the smallest remaining cost. Because it connects vertices that are *already in the same set*, we would *not* add this edge to the tree.

We would next consider the edge (A, B), because it has the smallest remaining cost. Because it connects vertices in different sets, we would add this edge to the tree and merge the sets involved to get:

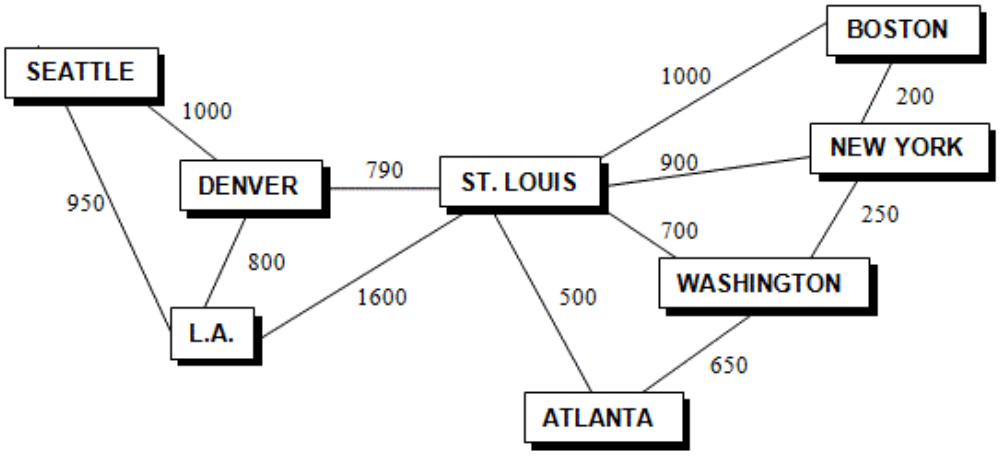
```
{A, B}, {C, D, E, F}
```

We would next consider the edge (B, C), because it has the smallest remaining cost. Because it connects vertices in different sets, we would add this edge to the tree and merge the sets involved to get:

```
{A, B, C, D, E, F}
```

Finally, we would consider the edge (B, D). Because it connects vertices in the same set, we would *not* add this edge to the tree.

Apply Kruskal's algorithm to the airline graph from earlier in the problem set, which is reproduced below. List the edges of the MST in the order in which the algorithm would add them, using the format (city1, city2) for an edge.



Submitting Your Work

You should use Canvas to submit your `ps5.txt` or `ps5.pdf` file.

Here are the steps you should take to submit your work:

1. Go to the page for submitting assignments (logging in as needed).
2. Click on the link for `ps5`.
3. Click on the *Submit Assignment* link near the upper-right corner of the screen. (If you have already submitted something for this assignment, click on *Re-submit Assignment* instead.)
4. Use the *Choose File* button to select the file to be submitted.
5. Once you have chosen the file that you need to submit, click on the *Submit Assignment* button.
6. After submitting the assignment, you should check your submission carefully. In particular, you should:
   - Check to make sure that you have a green checkmark symbol labeled *Turned In!* in the upper-right corner of the submission page (where the *Submit Assignment* link used to be), along with the names of all of the files from the part of the assignment that you are submitting.
   - **Click on the link for your file to download it, and view the downloaded file so that you can ensure that you submitted the correct file.**

Warning

We will not accept any files after the fact, so please check your submission carefully following the instructions in Step 6.

Important

- You must submit all of the files for the assignment at the same time. If you need to resubmit a file for some reason, you should also resubmit any other files from that part of the assignment.
- If you re-submit a file in Canvas, it will append a version number to the file name. You do *not* need to worry about this. Our grading scripts will remove the version number before we attempt to grade your work.
- There is a *Comments* box that accompanies each submission, but we do **not** read anything that you write in that space. If you need to inform us of something about your submission, please email `cscie22@fas.harvard.edu`.

- If you encounter problems submitting your files, close your browser and start again, or try again later if you still have time. If you are unable to submit and it is close to the deadline, email your homework before the deadline to `cscie22@fas.harvard.edu`.