CS 61B Spring 2021

Homework 3

May 5, 2021

Note: this homework is graded as a Gradescope Quiz, so input your final answers there.

1 Heaps of fun

For this problem, we will use approach 3B from lecture to represent our heaps. Recall that this representation uses an array where we keep index 0 empty, leaving the root of the heap in index 1.

(a) Starting with an empty min heap, give the resulting heap in array format after carrying out the 12 operations below. Only give the heap that results after the final operation, i.e. you do not need to provide the array for intermediate steps.

Assume that the internal array has an initial size of 8. You will not have to worry about resizing in this problem. If the value of the array at some index is null/out of bounds for the heap, write `-` in it's place. This means that if the heap has k items in it, the array will have $8 - k$ `-` elements in it. So, initially, the heap is:

```
[-, -, -, -, -, -, -, -]
```

Write a single space after each comma and include the `[` and `]` symbols. **Index 0 will always be empty**, i.e. occupied with a `-`

```
1  MinHeap<Integer> h = new MinHeap<>();
2  h.add(1);
3  h.add(2);
4  h.add(20);
5  h.add(0);
6  h.add(4);
7  h.removeMin();
8  h.removeMin();
9  h.add(5)
10 h.add(3)
11 h.add(10)
12 h.add(1)
13 h.removeMin();
```

The next parts will make statements that are either always true, sometimes true, or never true. Provide a brief written justification or counter-example. **We will only grade your choice of answer, not your justification**, but the justification is an important study tool so it behooves you to actually try to justify your answers!

All heaps in these parts are min heaps and contain N items total unless explicitly noted otherwise. **Assume the internal array does not resize for any of the specified operations.**

(b) A `removeMin` operation takes $\Theta(\log N)$ time.

(c) An `add` operation's time complexity is $\Omega(1)$ and $O(\log N)$.

(d) Given a min heap with N **distinct** items, if you add an item that is smaller than every item in the heap, then immediately call `removeMin` on that heap, you'll end up with the original heap before you added the item.

(e) Given a min heap with N items (**not necessarily unique**), if you add an item that is smaller than every item in the heap, then immediately call `removeMin` on that heap, you'll end up with the original heap before you added the item. This is the same question as the previous part, but now we may have duplicate items in the heap. Assume that we break ties when sinking by swapping with the left child.

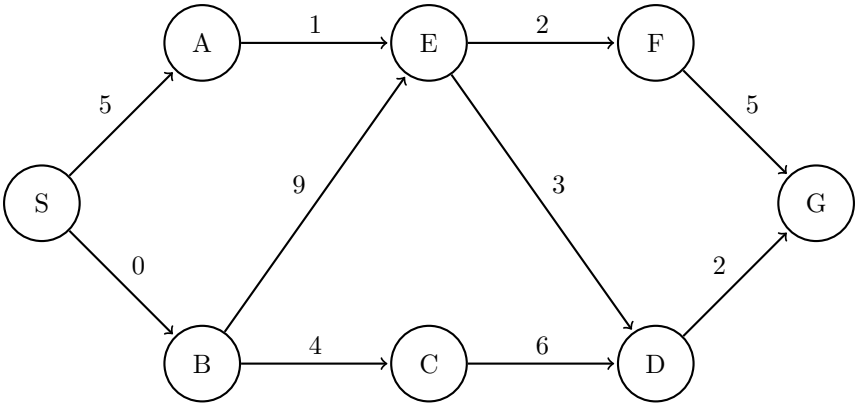
(f) The array representation of any min heap is in sorted order.

(g) If you swap every node's left child with its right child and vice-versa, you'll end up with a valid min heap. This would amount to the following pseudocode:

```
1  public void swapRecursive(Node root) {
2      if (root == null) {return;}
3      Node tmp = root.right;
4      root.right = root.left;
5      root.left = tmp;
6      swapRecursive(root.left);
7      swapRecursive(root.right);
8  }
```

2 Shortest Paths

Examine the following graph. We will be running Dijkstra's algorithm starting at the node labeled S.



We're using the version of Dijkstra's algorithm described here in lecture: note that the `fringe` and `distTo` data structures are always changed at the same time.

Give the resulting `edgeTo` and distTo maps after vertex B is visited (i.e. its outgoing edges to C and E have been relaxed). Also give the resulting `edgeTo` and distTo maps after Dijkstra's algorithm has completely finished execution. Initialize the `edgeTo` for each vertex as `-` which will represent `null` for us. Initialize the `distTo` for each vertex as `inf` which we'll use to represent ∞, except for S where it should be 0.

So, before the first iteration, the values of these variables are:

```
edgeTo = {A:-, B:-, C:-, D:-, E:-, F:-, G:-, S:-}
distTo = {A:inf, B:inf, C:inf, D:inf, E:inf, F:inf, G:inf, S:0}
```

The maps **must be in this order**. If you have the same mappings but in a different order (i.e. the mapping for S comes first), Gradescope will say your answer is incorrect.

(a) Give `edgeTo` and `distTo` after B is visited.

(b) Give `edgeTo` and `distTo` after Dijkstra's is finished.

(c) What is the shortest **path** from S to G? Format your answer as a comma separated list of vertices with spaces. For example, if the answer was `A -> B -> C`, then write `A, B, C`. Your answer will start with S and end with G.

(d) Give the vertices in the order that they are visited by Dijkstra's. What do you notice about the ordering vs. the `distTo` map?

The next parts will involve answering questions about shortest paths in general. As before, provide a brief written justification or counter-example. **We will only grade your choice of answer, not your justification**.

(e) True or False: Dijkstra's algorithm will correctly generate a Shortest Paths Tree for some graphs that have negative edge weights.

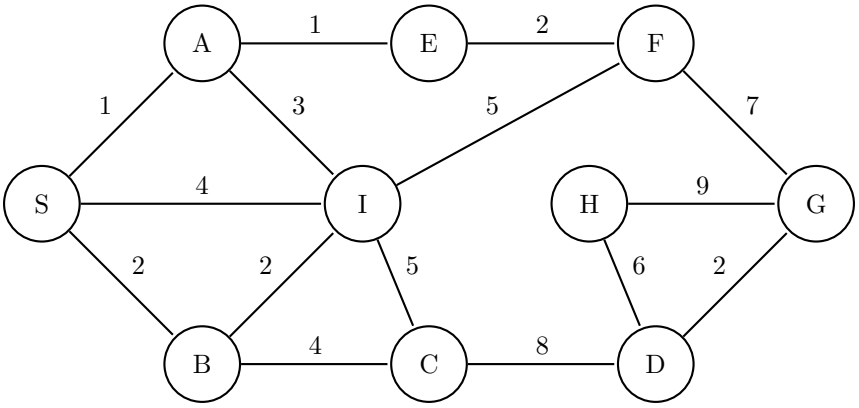
(f) True or False: Multiplying every edge in a graph with positive edge weights by some positive constant k will not change the Shortest Paths Tree that Dijkstra's algorithm generates.

(g) True or False: for graphs that happen to be trees, there is a Shortest Paths Tree algorithm which is asymptotically faster than Dijkstra's algorithm.

(h) True or False: For any graph **with distinct edge weights** and a node S in the graph, there is only one possible Shortest Paths Tree that you can generate from S.

3 MSTs

We will be referring to the following graph in the questions below



Suppose we start Prim's algorithm from S on this graph. Write down the state of the Priority Queue being used as the `fringe` after S is visited, after B is visited, and after the algorithm is completed. Each value in the priority queue should be an integer. Resolve ties by alphabetical order of the vertices. For example: if A and C have the same priority, then you should first pop A since it comes alphabetically first.

Initialize the priority of each vertex as `inf` which we'll use to represent ∞. So, before the first iteration, the values of the `fringe` are:

```
fringe = {S:0, A:inf, B:inf, C:inf, D:inf, E:inf, F:inf, G:inf, H:inf,
I:inf}
```

Order your fringe by priority values. You should use the same alphabetical tie-breaking scheme when choosing the order of the `fringe`. Make sure you include the `{}` symbols in your answer

(a) Give the state of the priority queue after S is visited in the format shown above.

(b) Give the state of the priority queue after B is visited.

(c) Give the state of the priority queue after Prim's is complete.

(d) For the graph above, what is the order in which edges are added to the MST when running Kruskal's Algorithm. Your answer should be in the form of a comma separated list with spaces.

In the case of a tie, choose the edge which comes first in alphabetical order i.e. if you had to choose between AS and AE, then you would choose AE first.

Edges themselves are labeled alphabetically: this means that the edge that connects S to A is labeled AS since A comes before S in the alphebet.

Here is a list of all the available edges:
`AS BS IS AI AE BI BC CD CI FI EF FG DG GH DH`

The next parts will involve answering questions about MSTs in general. As before, provide a brief written justification or counter-example. **We will only**

grade your choice of answer, not your justification.

(e) Given any graph with V vertices and E edges, how many edges will the graph's MST have? Write your answer in terms of V and E with no spaces. Assume the graph is connected.

(f) True or False. Prim's algorithm will work with negative edge weights.

(g) True or False. It's impossible for the MST of a graph to contain the largest weighted edge.

(h) True or False. The Shortest Paths Tree returned by Dijkstra's will never be a correct MST.

(i) True or False. A graph with unique edge weights will have exactly one MST. If you're stuck, you mind find it useful to know that Kruskal's algorithm can generate any MST depending on its tie-breaking scheme.

(j) True or False. A graph with non unique edge weights will always have a non unique MST.

(k) True or False. If you take any graph G **with positive edge weights** and square all the edge weights and turn it into the graph G' will G and G' have all the same MSTs.

(l) True or False. The minimum weight edge of any cycle in a graph G will be part of any MST of G.

4 Sorting

Consider the following unsorted array.

[234, 634, 1234, 210, 123, 542, 1021, 909, 321, 552, 135, 432, 1943, 53]

In each column below, we give the intermediate results for some sort that has not yet completed. The left most column a is the original array (as shown above), and the last column i is the sorted result.

0234	0909	1021	0053	0123	0123	0234	0123	0053
0634	0210	0909	0123	0135	0210	0634	0053	0123
1234	1021	0542	0135	0210	0234	0210	0135	0135
0210	0321	0634	0210	0234	0542	0123	0210	0210
0123	0123	0552	0234	0321	0634	0542	0234	0234
0542	0432	0432	0321	0432	1021	0909	0542	0321
1021	0234	0053	0432	0542	1234	0321	1021	0432
0909	0634	0210	0542	0552	0321	0552	0909	0542
0321	1234	0321	0552	0634	0552	0135	0321	0552
0552	0135	0123	0634	0909	0909	0432	0552	0634
0135	0542	0135	0909	1021	0053	0053	1234	0909
0432	1943	0234	1021	1234	0135	1234	0432	1021
1943	0552	1234	1943	1943	0432	1021	1943	1234
0053	0053	1943	1234	0053	1943	1943	0634	1943
a	b	c	d	e	f	g	h	i

Identify each sort from the list: Insertion sort, Selection sort, Mergesort, Quicksort, Heapsort, LSD sort, MSD sort. Each is used exactly once.

(a) Which sort is insertion sort? Give your answer as a letter, e.g. if column b is insertion sort, pick b.

(b) Which sort is selection sort?

(c) Which sort is Mergesort?

(d) Which sort is Quicksort? Assume we are using the leftmost item as the pivot. Assume we do not shuffle. Assume we use Tony Hoare style partitioning.

(e) Which sort is Heapsort? Assume we use bottom-up heapification. Assume we use a max heap.

(f) Which sort is LSD sort?

(g) Which sort is MSD sort?

(h) Which sort most likely has the fastest best case runtime on the array above?

The next parts will involve answering questions about sorting in general. As before, provide a brief written justification or counter-example. **We will only grade your choice of answer, not your justification**.

(i) True or False: Suppose we run Quicksort on a sorted array of distinct elements. Suppose our pivot selection is always the last element. Suppose we do not shuf-

fle. Suppose we use Tony Hoare style partitioning. Given these suppositions, Quicksort will take N^2 time.

(j) True or False: Quicksort can be made stable using a partitioning scheme which involves 3 different arrays, one array for items less than the pivot, one array for items equal to the pivot, one array for items greater than the pivot.

(k) True or False: Heapsort is empirically just as fast as mergesort.

(l) True or False: Finding and using the median element of every partition as the pivot will usually result in an empirically faster quicksort than a quicksort that uses a random pivot selection strategy.

(m) True or False: The following sort is stable: We split an array up into two halves and run insertion sort on each half. Then we merge the halves together like we do in merge sort.