CS61B Spring 2015 Guerrilla Section 3 Worksheet SOLUTIONS

Leo Colobong, Nick Fong, Jasmine Giang, Laura Harker, Yujie Huang, Anusha Ramakuri, Nick Rose, Khalid Shakur, Jason Won, Fan Ye, Jason Zhang, Giulio Zhou

7 May 2015

Directions: In groups of 4-5, work on the following exercises. Do not proceed to the next exercise until everyone in your group has the answer and *understands why the answer is what it is*. Of course, a topic appearing on this worksheet does not imply that the topic will appear on the midterm, nor does a topic not appearing on this worksheet imply that the topic will not appear on the midterm.

1 Sorting with Heaps

Given an array of n elements, where each element is at most k away from the target position, sort it as efficiently as possible.

Hint: Consider using a priority queue and aim for a runtime of $O(n \log k)$.

Solution:

```
public void sortK(int arr[], int n, int k) {
    PriorityQueue<Integer> pq = new PriorityQueue<Integer>();
    for (int i = 0; i <= k && i < n; i++) {
        pq.add(arr[i]);
    }
    int ti = 0;
    for (int i = k+1; ti < n; i++) {
        arr[ti] = pq.poll();
        if (i < n) {
            pq.add(arr[i]);
        }
        ti++;
    }
}
```

STOP!

Don't proceed until everyone in your group has finished and understands all exercises in this section!

2 More Heaps

Design and code a data structure which supports adding `ints` in $O(\log n)$ time and finding the median in constant time, where n is the number of items in the data structure.

Solution:

```
import java.util.PriorityQueue;

public class MedianPQ {

    PriorityQueue<Integer> maxPQ; //Keeps track of the lower half
    PriorityQueue<Integer> minPQ; //Keeps track of the upper half

    public MedianPQ() {
        maxPQ = new PriorityQueue<Integer>();
        minPQ = new PriorityQueue<Integer>();
    }

    public void add(int x) {
        if (maxPQ.size() < minPQ.size()) {
            maxPQ.add(-x);
        } else {
            minPQ.add(x);
        }
        if (maxPQ.size() + minPQ.size() > 1 && -maxPQ.peek() > minPQ.peek()) {
            minPQ.add(-maxPQ.poll());
            maxPQ.add(-minPQ.poll());
        }
    }

    public Integer findMedian() {
        int maxSize = maxPQ.size();
        int minSize = minPQ.size();
        if(maxSize < minSize) {
            return maxPQ.peek();
        } else if (minSize < maxSize) {
            return -minPQ.peek();
        } else {
            return (-minPQ.peek() + maxPQ.peek())/2;
        }
    }
}
```

STOP!

Don't proceed until everyone in your group has finished and understands all exercises in this section!

3 Even More Heaps

Consider a valid minheap h.

1. If we add a constant c to every value in h, is h still a valid minheap?

2. If we multiply every value in h with a constant c, is h still a valid minheap?

1. Yes

2. No, if c is negative, then the ordering of numbers swaps. For example, $1 < 2$, but for $c = -1$, $-1 > -2$, so our heap would no longer be a valid minheap. (Note: it would be a valid maxheap, however).

STOP!

Don't proceed until everyone in your group has finished and understands all exercises in this section!

4 Sorts

Execution times that result from applying sorting methods to sort sequences of 2000 values into ascending order appear in the table below. The algorithms possibly used are selection sort, insertion sort, and merge sort, and tree sort (repeated insertions into a binary search tree with no attempt to balance, followed by traversal of the tree). Specify which times go with which sorting method.

Time to sort 2000 random values	Time to sort 2000 values already in increasing order	Time to sort 2000 values already in decreasing order	Sorting Method (Selection, Insertion, Merge, or Tree)
326	2	597	
52	5596	5209	
422	423	419	

Insertion, Tree, Selection

STOP!

Don't proceed until everyone in your group has finished and understands all exercises in this section!

5 Graphs

Youre given an undirected, weighted graph $G = (V, E)$, a list of start vertices, and a list of end vertices. Describe an efficient algorithm that returns the shortest path from some start vertex to some end vertex.

Solution: Create two new nodes, s and t. Add edges (s, $start_i$) for each start node and (end_i, t) for each end node, all with weight 0. Run A^* search from s to t.

STOP!

Don't proceed until everyone in your group has finished and understands all exercises in this section!

6 More Graphs

Given an undirected graph $G = (V, E)$ and an edge $e = (u, v)$ in G, create an $O(V + E)$ time algorithm to determine whether G has a cycle containing e.

Solution: Do a modified DFS. Say the edge is $e = (u, v)$, do DFS out from each of us neighbors except for v. If you reach v while searching then there is a cycle with e.

```
public class CycleDetective {
    private boolean[] marked;

    // Returns whether there is a cycle in G containing e=(u,v)
    public boolean cycleDetect(Graph G, int u, int v) {
        marked = new boolean[G.V()];
        for(int w = 0; w < G.V(); w++) {
            marked[w] = false;
        }
        marked[u] = true;
        for(int w : G.adj(u)) {
            if (!marked[w] && w != v) {
                if (cycleDetectDfs(G, w, v)) {
                    return true;
                }
            }
        }
        return false;
    }

    // returns whether v is reachable from w using only unmarked vertices
    private boolean cycleDetectDfs(Graph G, int u, int v) {
        for(int w : G.adj(u)) {
            if (w == v) {
                return true;
            }
            if (!marked[w]) {
                marked[w] = true;
                if( cycleDetectDfs(G, w, v)) {
                    return true;
                }
            }
        }
        return false;
    }
}
```

STOP!

Don't proceed until everyone in your group has finished and understands all exercises in this section!

7 Even More Graphs

Given a connected, undirected, weighted, graph, give a strategy to construct a set with as few edges as possible such that if those edges were removed, there would be no cycles in the remaining graph and that the sum of the weights of the edges you remove is as big as possible. This strategy must be as quick as possible. List any data structures you might need and how they would be utilized.

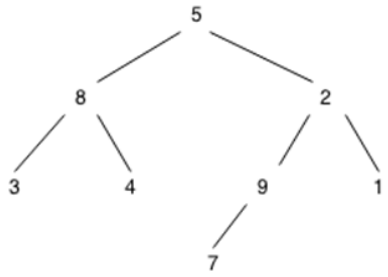
Solution: Use Prim's or Kruskal's since this problem is equivalent to finding a minimum spanning tree and subtracting those edges from the set of all edges. In either case, you will need the basic data structures necessary for either algorithm: a priority queue to keep track of potential edges to form your MST, some sort of list or set to keep track of edges in your MST, and disjoint sets (if using Kruskal) to help you detect that youre not adding an edge to your MST that would create a cycle. Note that this runs in $O(E \log E) = O(E \log V)$ time.

STOP!

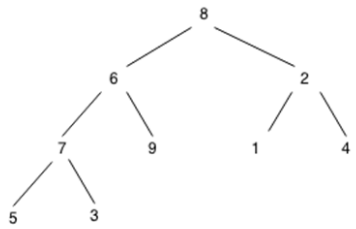
Don't proceed until everyone in your group has finished and understands all exercises in this section!

8 Tree Traversals

1. Draw the binary tree that corresponds to the traversals:
   `in-order: 3 8 4 5 7 9 2 1`
   `post-order: 3 4 8 7 9 1 2 5`
   Solution: We know that 5 is the root since it's the last to be visited in the post-order traversal. Then we can construct both of its subtrees.



2. Now suppose youre given the traversals for a full binary tree (i.e. each node has 0 or 2 children). Draw the corresponding tree:
   `pre-order: 8 6 7 5 3 9 2 1 4`
   `post-order: 5 3 7 9 6 1 4 2 8`
   Solution:



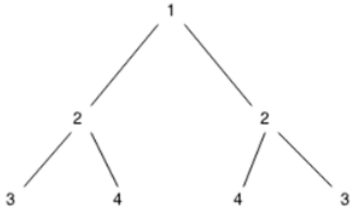
STOP!

Don't proceed until everyone in your group has finished and understands all exercises in this section!

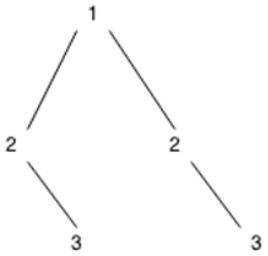
9 Even More Graphs

Given a binary tree, check whether it is a mirror of itself (i.e, symmetric around its center).

For example, this binary tree is symmetric:



But the following tree is not:



Given the following `TreeNode` implementation, fill in the `isSymmetric` method of the `Symmetric` class.

Solution:

```
public class Symmetric {
    public boolean isSymmetric(TreeNode root) {
        if(root == null) return true;
            return isSymmetric(root.left, root.right);
    }

    public boolean isSymmetric(TreeNode a, TreeNode b){
        if(a == null) return b == null;
        if(b == null) return false; // when only one of them is null
        if(a.val != b.val) return false;
        if(!isSymmetric(a.left, b.right)) return false;
        if(!isSymmetric(a.right, b.left)) return false;
        return true;
    }
}
```

STOP!

Don't proceed until everyone in your group has finished and understands all exercises in this section!

10 Hashing

1. What is the worst case runtime of a single call to insert on a hashtable with n items?
2. In what situations would we achieve the worst case runtime?

Solution:

1. $O(n)$
2. If all of the items hash into the same bucket, then we have to iterate through all of them to check if the key is already in the hashtable, which would take $O(n)$ time. This could occur if we have a bad hashcode function. Also, if we have to resize the array during this call, then we have to rehash all of the items into the new hashtable, which would also take $O(n)$ time.