Mid-Term Exam 1 Solutions

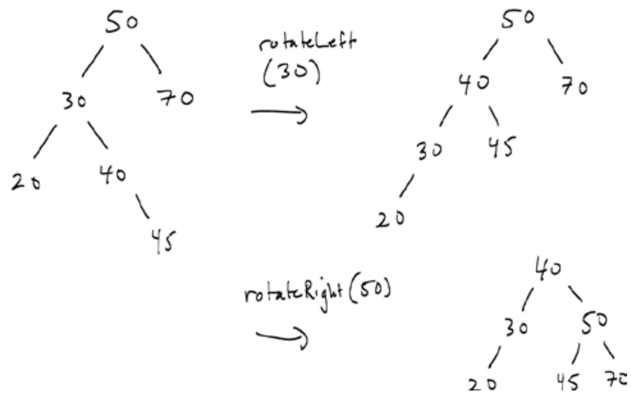
COMP 251 Algorithms and Data Structures

Tues. Feb. 4, 2014

Prof. Michael Langer

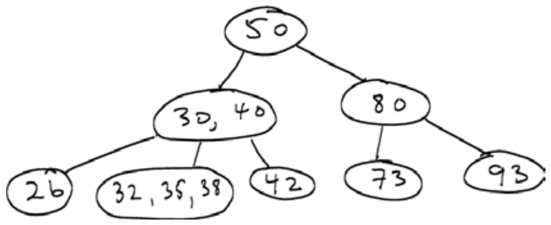
1)

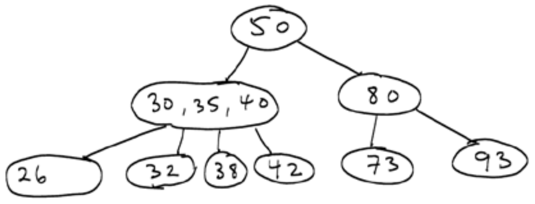
a)

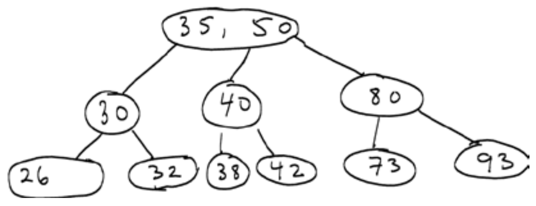


GRADING SCHEME: 2 points total. We have 0.5 for inserting the 45 at the correct place. This makes the tree unbalanced. The first rotation gets another 0.5. The second rotation gets 1 point.

b)



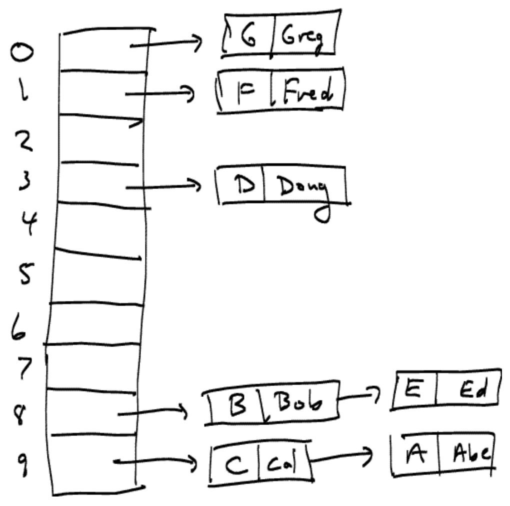




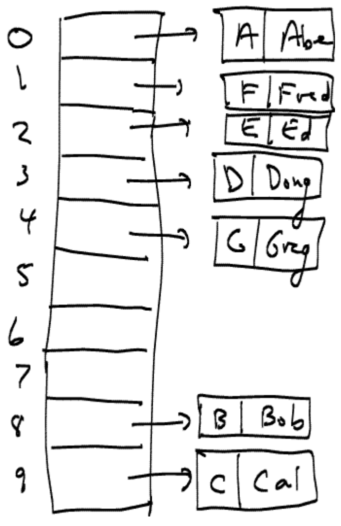
GRADING SCHEME: 2 points total. Many students inserted the 38 in the middle layer, in node (30,40). That is not how 2-3 tree inserts work. You always insert at the bottom layer. We gave 1 point only if you made this mistake (assuming that you also did the proper bubbling up operation for 2-3 trees).

2)

a)



b)



In the figure on the right, you can draw the key,values right in the boxes as I did in the lecture. I just drew them separately here to show that's possible too.

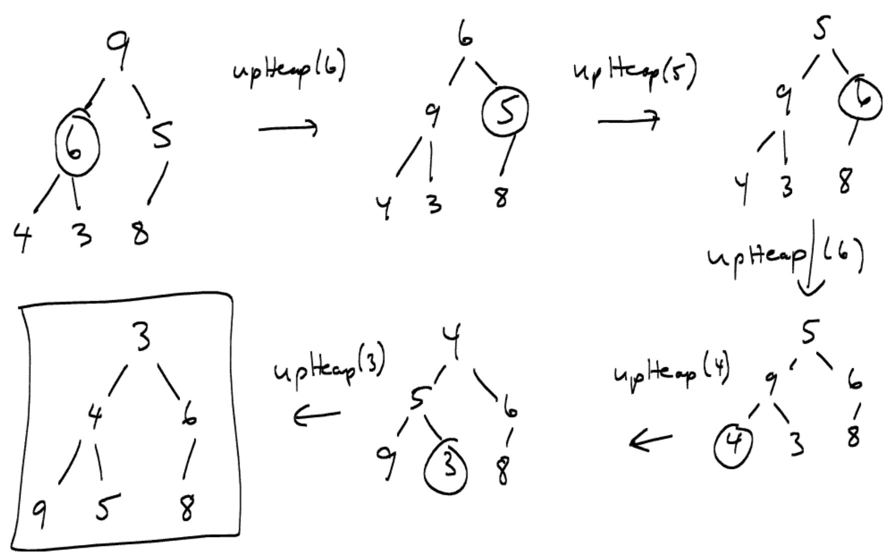
GRADING SCHEME: 2 points for (a) and 2 points for (b).

- Some students put the hashcode in the table rather than putting the keys and values. We originally gave 0.5/2 for each, assuming the hashcodes were put in the correct spot. However, some students said afterwards that they felt this was too harsh since they had demonstrated they understood how the two hashing schemes worked and that was the main point. **I accept that argument, and will give 1 point each (a and b) if you make this mistake. Feel free to bring me your exam.**
- Some students put only the keys in, or only the values in. In that case, we took off points (unless you wrote explicitly "I am only putting in the keys in, but the values are there too"). The reason we took off points is that we don't know if you understand that both need to be there – some students don't yet understand how hash tables work. Unfortunately, the grading was inconsistent on this particular mistake. Some graders took off 1 point for each of (a) and (b), and others took off 0.5 points. (This is assuming, of course, that the things were put in the correct position.) **If you had 2 points (1+1) taken off for this error and you want that penalty reduced, feel free to bring me your exam.)**

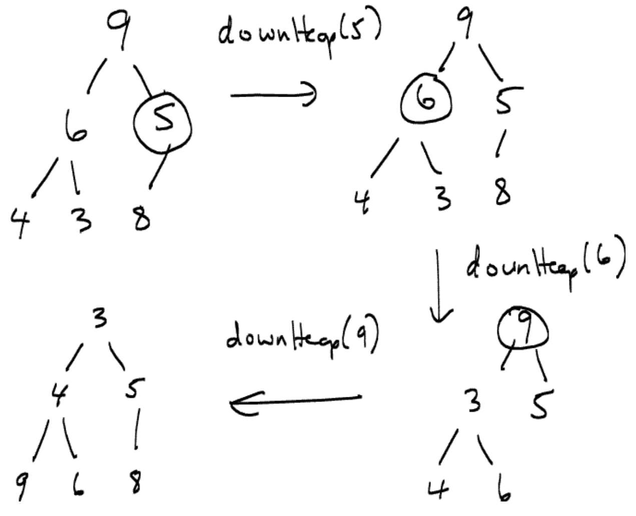
- Another mistake that a few students made is that they wrongly computed modulus, by taking the first digit, not the second. (I would hope this was just an exam panic mistake.) When grading, we didn't realize what the mistake was and we gave 0. If we had noticed, we would have taken off just one point for this error, assuming all the rest was done correctly of course. **(If you made this mistake and got 0, feel free to bring me your exam.)**

3) .

a)



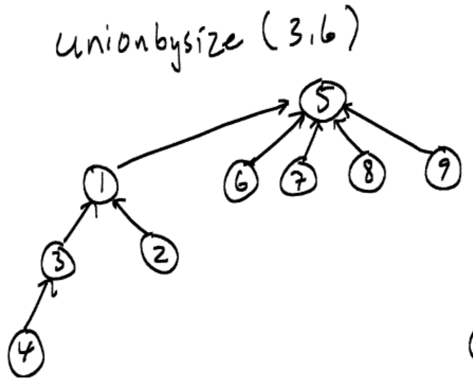
b)

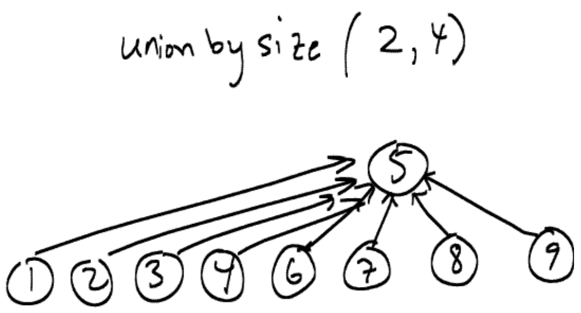


GRADING SCHEME: 2 points for (a) and (b) each.

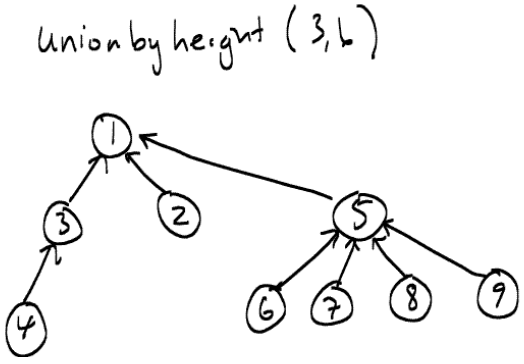
4)

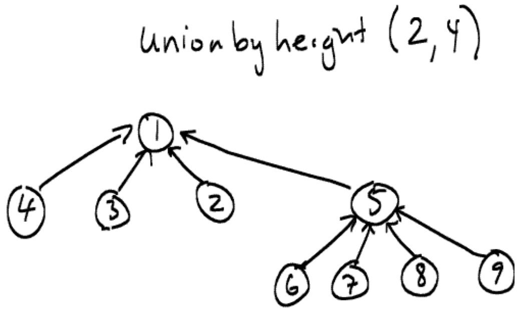
a) unionbysize(3,6) find the roots of 3 and 6 (namely 1 and 5 respectively) and compares the sizes (5's tree has five elements, 1's tree has four elements), and points 1 to 5 (smaller to bigger). No path compression was done there since 3 was already pointing at the root and 5 was already the root. Unionbysize(2,4) finds the roots (which turn out to be common in this case) and in doing so, compresses the paths from 2 to the root and from 4 to the root.





b) Unionbyheight(3,6) finds the roots as in the previous question (again no path compression) but now compares by height (node 5 has height 1 and node 1 has height 2), so node 5 points to 1. Unionbyheight(2,4) then compresses the path from 4.



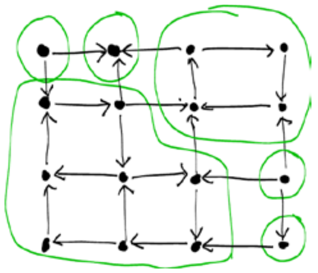


GRADING SCHEME: We gave 1 point for each of the four drawings above. The main problem students had with this question seems to be in understanding when path compression occurs. Path compression occurs in the find operations, but since union calls find, there is path compression occurs even though the finds are not explicit.

5)

GRADING SCHEME FOR THIS QUESTION: 2 points for each of a, b, c.

a)

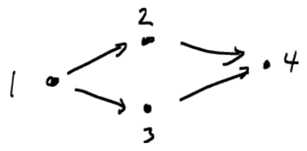


b) The answer is n. A strongly connected component is a set of vertices that are mutually reachable. So, any SCC with more than one component would have a cycle with more than one vertex. A DAG has no cycles, every vertex is its own SCC.

GRADING: Many students thought that an SCC needs to have at least one vertex and gave an answer "0" claiming that there are no cycles in a DAG and therefore no SCCs. However, if you look at page 2 of the lecture slides, I gave an example of a graph with six vertices in which each vertex is its own SCC. Also, on page 1 of that lecture, I mentioned that every vertex is reachable from itself by a path of length 0. GRADING: I gave 1 point if you made this mistake, assuming you justified it as mentioned above.

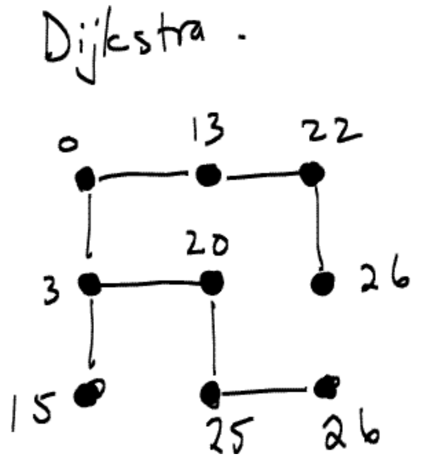
c) A path from v to w, by definition, means there is a sequence of vertices such that neighboring vertices are connected by a directed edge. If there is a topological ordering on the graph then, by definition, we have a numbering of the vertices such that if (vi, vj) is an edge then i < j. The path from v to w thus defines an sequence of increasing numbers. In particular, the first number in the sequence (v's number) is less than the last number in the sequence (w's number).

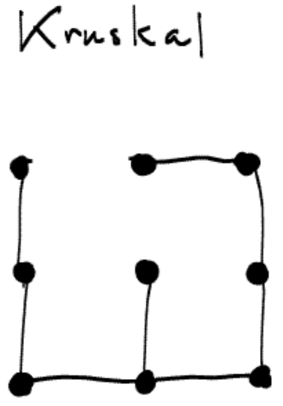
Many students argued something like this: "Since it is a DAG, if w came before v, *then this would imply that there is a path from w to v,* which would imply there is a cycle (contradiction). " This argument is incorrect, however. Specifically the italicized part is incorrect. If w comes before v in a topological ordering, this does not imply there is a path from w to v. For example, in the graph (and topological ordering) below, 2 comes before 3 but this does not imply that there is a path from 2 to 3. (We gave 0.5 points if you made this argument.)



Others argued that the claim is already true, *by definition*. But this is also false. The definition of a topological order specifies a constraint on individual edges. The fact that this constraint extends to paths is precisely what I am asking for. (We gave 0.5 points for that.)

6)





7) If you use a balanced binary search tree (e.g. Java TreeMap) to index into the priority queue, then it takes O(log n) to find an item in the BST. This is slower in the O( ) sense than the hash map which is O(1). However, since we are changing the priority, this itself takes O(log n) to update the heap. So using the BST doesn't slow things down in the O( ) sense, since we are using O(log n) anyhow.

GRADING SCHEME: Many students wrote that O(1) for hashmap is less than O(log n) for BST and so hashmpa is faster. We gave 0.5 for this.

Some wrote that both methods of implementing an indexed priority queue to perform changePriority are O(log n), but that the BST is actually slower since hashmap case is O(1) + O(log n) wherase the balanced BST case O(log n) + O(log n). However, I tend to reject this argument as there are many constants hidden in the two methods, and its hard to say which really would be better or worse. The question asks about O( ) so we shouldn't be trying to guess the constants. [feb 13] I believe we gave 1.5 points for this.

8) The priority queue would be size O(|E|). Removing an element from the priority queue would take O(log |E|). Removing all |E| edges would take O(|E| log |E|). So it neither help nor hurt us to use a priority queue instead of sorting.