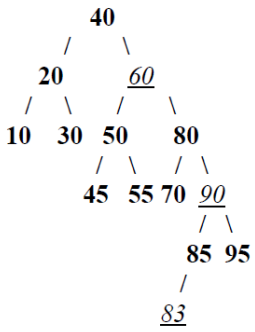
Data Structures and Programming

Spring 2019, Final Exam.

June 18, 2019

1. (6 pts) Draw an example of a Red-Black tree with 14 nodes that achieves the maximum height. Mark the red and the black nodes clearly.

Solution



2. (14 pts) Suppose that the keys shown on the left of the following figure are inserted into an initially empty *linear-probing* hash table of size 7, but not necessarily in the order given. The result is shown on the right.

key	*hash*
A	1
D	5
L	6
M	0
N	1
S	6
X	4

Hash function

0	1	2	3	4	5	6
S	M	N	A	X	D	L

Result

For each of the keys $A\ D\ L\ M\ N\ S\ X$, determine whether it could have been the last key inserted. Fill out the following table (with "yes" or "no").

No explanations needed. However, $score = max\{0, Right - \frac{1}{2}Wrong\}$

Solution

Key	A	D	L	M	N	S	X
Yes or No	O	O	X	X	X	X	O

- A could be last (L S M N X D A)
- D could be last (L S M N A X D)
- L could not be last (otherwise S would end up in 6)
- M could not be last (otherwise N or A would end up in 1)
- N could not be last (otherwise A would not end up after 2)
- S could not be last (otherwise M would end up in 0)
- X could be last (L S M N A D X)

3. (12 pts) In Union-Find data structures, we often use an array for the parent-link representation. Take array B in the following table for instance. The parent of node 0 is node 9, as $b[0] = 9$, and 9 is a root as $b[9] = 9$.

```
          i:  0  1  2  3  4  5  6  7  8  9
              ------------------------------
A.    a[i]:   1  2  3  0  1  1  1  4  4  5

B.    a[i]:   9  0  0  0  0  0  9  9  9  9

C.    a[i]:   1  2  3  4  5  6  7  8  9  9

D.    a[i]:   0  0  0  0  0  1  1  1  6  2

E.    a[i]:   0  0  0  0  0  1  1  1  6  8

F.    a[i]:   0  0  0  1  1  3  3  7  7  7
```

Fill out the table below with *possible* (P) or *impossible* (I) whether the respective array could possibly occur during the execution of a union-find data structure with *weighted union*.

No explanations needed. However, $score = max\{0, Right - \frac{1}{2}Wrong\}$.

Solution

Disjoint set rep.	A	B	C	D	E	F
possible or impossible	I	I/P	I	P	I	I

(A) Impossible: has a cycle 0-1, 1-2, 2-3, and 3-0 in the parent-link representation.

(B) You get full credit either you answered I or P.
Impossible (if by size): the nodes 1, 2, 3, 4, and 5 must link to 0 when 0 is a root; hence, 0 would not link to 9 because 0 is the root of the larger tree.
Possible (if by height)

(C) Impossible: tree rooted at 0 has height $9 > \log_2 10$.

(D) Possible: 8-6, 7-1, 5-1, 6-1, 9-2, 3-0, 4-0, 2-0, 1-0.

(E) Impossible: tree rooted at 0 has height $4 > \log_2 10$.

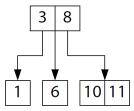
(F) Impossible: tree rooted at 0 has height $3 > \log_2 7$

4. (9 pts) In the potential method for the amortized analysis of splay trees, we define the *rank* of a node q in a tree T as $rank(q) = \lfloor \log_2 size(q) \rfloor$, and $size(q)$ equals the number of nodes (including q) in the subtree rooted at q.

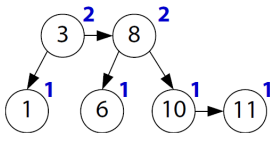
   (a) (4 pts) Define the potential of T discussed in class.
   Solution $\sum_{q \in T} rank(q)$

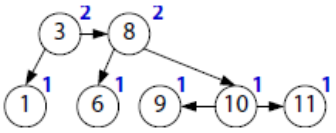
   (b) (5 pts) Recall that when we splay a node x in a splay tree T, the amortized cost of the splay operation is $\leq 3(rank(root) - rank(x)) + 1$. Suppose that at some point in time the amortized const of a splay is 34. What is the smallest possible number of nodes in the tree? Why.
   Solution 3(rank(root)-rank(x))=34, so rank(root) is at least 11 and the number of nodes is at least $2^{11} = 2048$

5. (10 pts) Look at the 2-3 tree on the right.

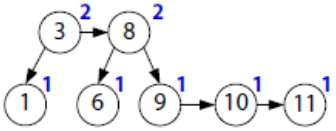


   (a) (4 pts) Draw the 2-3 tree as an AA tree. Mark each node with its level. Assume that a leaf node is of level 1.
   Solution

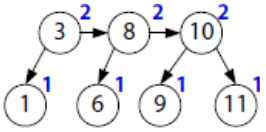




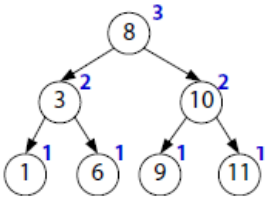
Now 10 has the same height as its left child, so we do a skew (rotation):



Now 10 has become a "4-node", so we do a split, which lifts 10 up:



Now 3 has become a "4-node", so we do another split, which lifts 8 up:



(b) (6 pts) Insert 9 into the *AA* tree using the *AA* tree insertion algorithm. Show all intermediate steps and write down the final tree.
Solution

6. (15 pts) In the figure below, the leftmost column is an array of strings to be sorted. The column to the far right gives the strings in sorted order. Each of the remaining columns (A)-(E) gives the contents of the array during some intermediate step of one of the algorithms listed below: *(1) Merge; (2) Quick; (3) Heap; (4) Bubble; (5) Selection; (6) Insertion; (7) Straight Radix; (8)Radix Exchange.*

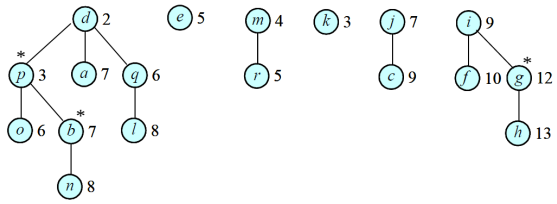
4873	1876	1874	1626	9573	2212	1626
1874	1874	1626	1874	7121	8917	1874
8917	2212	1876	1876	9132	7121	1876
1626	1626	1897	4873	6973	1626	1897
4982	3492	2212	4982	4982	9132	2212
9132	1897	3492	8917	8917	6152	3492
9573	4873	4873	9132	6152	4873	4873
1876	9573	4982	9573	1876	9573	4982
6973	6973	6973	1897	1626	6973	6152
1897	9132	6152	3492	1897	1874	6973
9587	9587	7121	6973	1874	1876	7121
3492	4982	8917	9587	3492	9877	8917
9877	9877	9132	2212	4873	4982	9132
2212	8917	9573	6152	2212	9587	9573
6152	6152	9587	7121	9587	3492	9587
7121	7121	9877	9877	9877	1897	9877
----	----	----	----	----	----	----
Input	(A)	(B)	(C)	(D)	(E)	Sorted

Match each column (A)-(E) with its corresponding algorithm. No penalty for wrong answers.

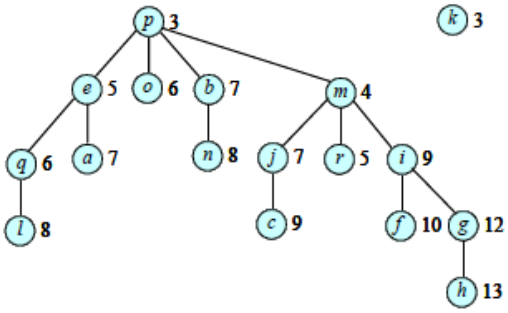
Solution

	A	B	C	D	E
	Quick (2)	Radix Exchange (8)	Merge (1)	Heap (3)	Straight Radix (7)

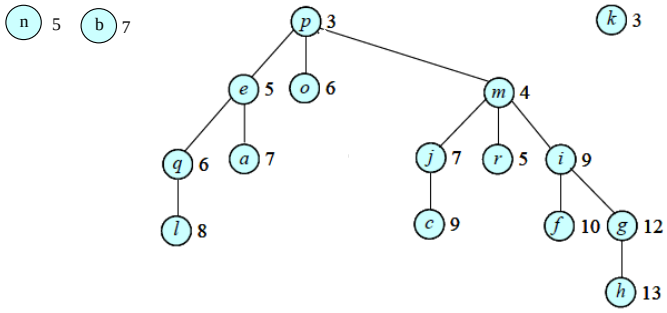
7. (10 pts) The figure below shows a single Fibonacci heap, in which a marked node is annotated with a *.



(1) (5 pts) Show how the heap structure changes following a delete-min operation. Perform the linking steps from left to right (as they appear in the figure) - that is always combine the leftmost pair of trees that are "eligible" to be combined.
Solution



(2) (5 pts) With respect to the resulting heap in step (1), perform a decrease-key $8 \rightarrow 5$ at node n (i.e., change the key from 8 to 5). Draw the resulting heap.
Solution



8. (15 pts) Recall that Dijkstra's algorithm finds the shortest paths in a weighted graph $G = (V, E)$ from some starting vertex s to all other vertices in the graph. Recall that the algorithm uses a priority queue Q which is empty initially.

(a) (10 pts) In terms of big-Oh, $|E|$ (the number of edges), and $|V|$ (the number of vertices), how many times does each of the following priority queue operations get called in one complete run of Dijkstra's algorithm?
(1) insertion (2) delete-min (3) is-empty (4) contain-key (5) decrease-key
where the contain-key operation is to ask whether a key x is in Q.
Solution

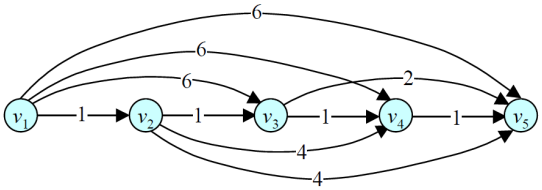
i. insertion: $O(|V|)$
ii. delete-min: $O(|V|)$

iii. is-empty: $O(|V|)$
(or $O(1)$ if the implementation is through "$for\ i = 1\ to\ n-1$", i.e., without checking for the emptiness of the priority queue.)

iv. contain-key: $O(|E|)$)
(or $O(1)$ if the identity of each node is not treated as a key in the priority queue.)

v. decrease-key: $O(|E|)$

(b) (5 pts) How many *decrease-key* operations are performed by Dijkstra's algorithm when executed on the graph shown below with vertex v_1 as the source vertex? Explain your answer in sufficient detail.



Solution: 3 decrease-key are performed when v_2 is removed from the heap, 2 are performed when v_3 is removed and 1 when v_4 is. So the total is 6.
Note: You still get full credit if you started with $D[v_2] = ... = D[v_4] = \infty$ which results in 4 more decrease-key operations.

9. (9 pts) Let $P_a(n), P_b(n), P_l(n), P_f(n)$ denote the running time of Prim's algorithm for the minimum spanning tree problem on graphs of n nodes using *sorted arrays*, *binomial heaps*, *leftist heaps*, and *Fibonacci heaps* as the implementation of priority queues, respectively. Is each of the following statements true? Justify your answers.
(1) $P_b(n) = O(P_f(n))$ when $m = 3n$; (2) $P_l(n) = O(P_f(n))$ when $m = \frac{n^2}{4}$; (3) $P_a(n) = O(P_b(n))$ when $m = n^{1.5}$, where m is the number of edges of the graph.

Solution First note that

- $P_a(n)$=$O(mn)$
- $P_b(n)$=$O(m \log n)$
- $P_l(n)$=$O(m \log n)$
- $P_f(n)$=$O(m + n \log n)$

Hence,

- $(m = 3n)$ $P_b(n)$=$O(n \log n)$; $P_f(n)$=$O(n \log n)$ – True.
- $(m = \frac{n^2}{4})$ $P_l(n)$=$O(n^2 \log n)$; $P_f(n)$=$O(n^2)$ – False.
- $(m = n^{1.5})$ $P_a(n)$=$O(n^{2.5})$; $P_b(n)$=$O(n^{1.5} \log n)$ – False.