CS211 Spring 2007
Final Exam
May 14, 2007

Solutions

Instructions

Write your name and Cornell netid above. There are 9 questions on 10 numbered pages. Check now that you have all the pages. Write your answers in the boxes provided. Use the back of the pages for workspace. Ambiguous answers will be considered incorrect. The exam is closed book and closed notes. Do not begin until instructed. You have $2\frac{1}{2}$ hours. **Good luck!**

	1	2	3	4	5
Score	/14	/6	/10	/14	/8
Grader					

	6	7	8	9	Σ
Score	/10	/8	/15	/15	/100
Grader					

1. (14 points) Fill in the following table of asymptoptic complexities for each of the given data structures and operations.

	Sorted singly-linked list	Sorted ArrayList	Sorted doubly-linked list
finding the smallest element	$O(1)$	$O(1)$	$O(1)$
finding the largest element	$O(n)$	$O(1)$	$O(1)$
searching for a given element	$O(n)$	$O(\log n)$	$O(n)$
deleting a given element	$O(1)$ or $O(n)$	$O(n)$	$O(1)$
finding the median	$O(n)$	$O(1)$	$O(n)$

In the sorted singly-linked list, assume the elements are ordered from smallest to largest, and there is no tail pointer. For deleting a given element, do not include the cost of finding it.

2. (6 points) For each of the following sets of class definitions, say what would be printed by the program

```
C x = new C();
System.out.println(x.foo());
```

If the program gives an error, say what that error is.

(a)
```
abstract class A {
   abstract int foo();
   abstract int bar(A a);
}
class B extends A {
   int bar(A a) { return 2; }
   int foo() { return bar(this); }
}
class C extends B {
   int bar(A a) { return 3; }
}
```

3

(b)
```
abstract class A {
   int bar(A a) { return 0; }
   int bar(B b) { return 1; }
   int bar(C c) { return 2; }
}
class B extends A {
   int bar(A a) { return 3; }
   int bar(B b) { return 4; }
   int bar(C c) { return 5; }
}
class C extends B {
   int foo() { return ((A)this).bar((A)this); }
}
```

3

3. (10 points) Say we are given a weighted directed graph G with strictly positive edge weights $w(u, v)$ and a source node s in G. We would like to modify Dijkstra's shortest-path algorithm to produce a *count* of the number of different minimal-length paths from s to v for every node v.

   Recall that Dijkstra's algorithm builds a set X of nodes incrementally, starting with $X = \{s\}$. Normally, the algorithm maintains a value $D(v)$ for each node v giving the length of the shortest path from s to v through only intermediate nodes in X.

   We will modify the algorithm to maintain an extra value $N(v)$ giving the number of paths of length $D(v)$ from s to v through only intermediate nodes in X.

   Describe how to initialize $D(v)$ and $N(v)$ and how to update the values of $D(v)$ and $N(v)$ when a new node u is added to X. Describe your modifications in words, do not write any code.

Initialize $D(v)$ as in Dijkstra's algorithm. Initialize $N(s) = 1$, $N(v) = 1$ for all edges (s, v), and $N(v) = 0$ for all other nodes.

To update N when a node u is added to X, for each edge (u, v),

- if $D(u) + w(u, v) < D(v)$, set $N(v) = N(u)$;
- if $D(u) + w(u, v) = D(v)$, set $N(v) = N(u) + N(v)$;
- if $D(u) + w(u, v) > D(v)$, do nothing.

Update $D(v)$ for all edges (u, v) as in Dijkstra's algorithm.

4. (14 points)

(a) What is a spanning tree in an undirected graph? Give an accurate definition.

A spanning tree is a connected subgraph that contains all the nodes but has no cycles.

(b) Do Prim's and Kruskal's algorithms still work if the edge weights are allowed to be negative?

yes

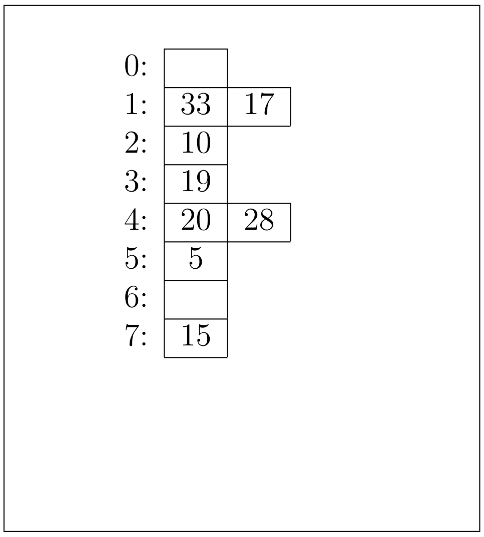
Suppose that we have already computed a minimum-weight spanning tree M in a weighted undirected graph G. Now suppose we want to either (c) *increase* the weight of one of the edges in M or (d) *decrease* the weight of one of the edges of G *not* in M. In either case M may no longer be of minimum weight, and we may have to compute a new minimum-weight spanning tree. Describe linear-time ($O(n + m)$) algorithms for each of these two problems. For (c), assume that the edges not in M are sorted by weight. Say what to do in high-level terms, no code.

(c) Let (u, v) be the edge in M whose weight increased. Using DFS or BFS, mark all the nodes reachable from u in M without going through v and call this set A. Similarly, mark all the nodes reachable from v without going through u and call this set B. Now go through the sorted list of edges not in M and find the smallest weight edge e connecting a node in A with a node in B. If the weight of e is less than that of (u, v), replace (u, v) in M with e.

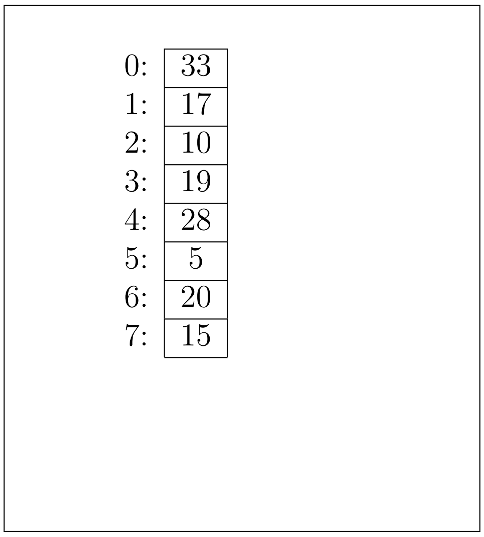
(d) Let (u, v) be the edge not in M whose weight decreased. Using DFS or BFS, find the unique simple path from u to v in M. Find an edge e of maximal weight on that path. If the weight of e is greater than that of (u, v), replace e in M with (u, v).

5. (8 points) Draw the final result after inserting keys 5, 19, 28, 15, 20, 17, 10, 33 into a hash table with collisions resolved by (a) chaining, (b) linear probing. Let the table have eight slots with addresses starting at 0, and let the hash function be $h(k) = k \bmod 8$.

(a)



(b)



6. (10 points) True or false?

(a) T $\boxed{\text{F}}$ In Java, each event such as a button click or keystroke spawns a new thread to handle the event.

(b) $\boxed{\text{T}}$ F Insertion sort, selection sort, and bubblesort all have the same asymptotic worst-case complexity.

(c) T $\boxed{\text{F}}$ A planar graph is one with no arrowheads on the edges.

(d) $\boxed{\text{T}}$ F Dijkstra's algorithm can be used to find a minimum spanning tree.

(e) $\boxed{\text{T}}$ F Static methods may not refer to `this`.

(f) $\boxed{\text{T}}$ F If class `B` is a subclass of class `A` and `A` implements interface `C`, then `B` automatically implements `C` even if it does not provide explicit implementations of the methods of `C`.

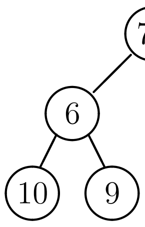
(g) $\boxed{\text{T}}$ F In a Java GUI, a `Container` would use a `LayoutManager` to organize the layout of its `Components`.

(h) T $\boxed{\text{F}}$ When implementing quicksort, the first element of an interval is always a good choice for the pivot.

(i) T $\boxed{\text{F}}$ At runtime, a successful downcast from type `A` to type `B` changes the dynamic type of the object from `A` to `B`.

(j) $\boxed{\text{T}}$ F $n^2 \log n + n(\log n)^2$ is $O(n^2(\log n)^2)$.

7. (8 points) Consider the following complete binary tree.

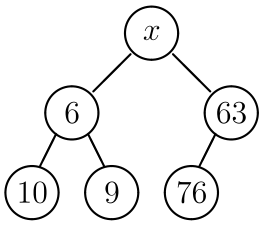


(a) The tree represents a heap implementing a priority queue, but the heap is in an inconsistent intermediate state; a priority queue operation is currently being performed, but the operation is not yet finished. Which operation?

Extract min.

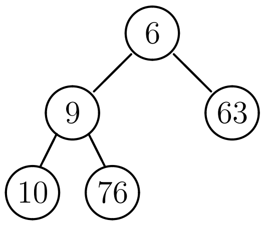
(b) Draw the original tree before the operation started and the final tree after the operation is complete. If there are several possibilities, just pick one.

Before:

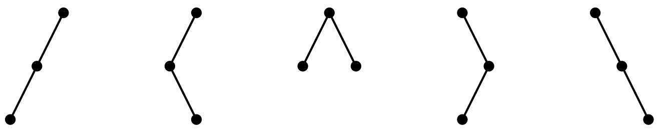


x can be any value ≤ 6.

After:



8. (15 points) Recall that a *binary tree* is a tree in which each node has 0, 1, or 2 children. There are exactly five binary trees with three nodes, namely



(a) Write a recursive method `int nTrees(int n)` that calculates the number of different binary trees with n nodes for $n \geq 0$. (*Hint.* A binary tree with n nodes consists of a root node, a left subtree with m nodes, and a right subtree with $n - 1 - m$ nodes for some $0 \leq m \leq n - 1$.)

```
public static int nTrees(int n) {
   if (n == 0) return 1;
   int s = 0;
   for (int m = 0; m < n; m++) {
      s += nTrees(m)*nTrees(n-1-m);
   }
   return s;
}
```

(b) Your recursive algorithm is probably quite inefficient, because during the course of the computation on a large value of n, it calls itself on smaller values of n many times over. It can be made much more efficient by caching values already computed in a data structure and just consulting the data structure to see if the value has already been computed. Describe briefly how and where you would modify your code to take advantage of this idea. Say what data structure(s) you would use and how they would be defined in Java.

Cache the answers in an `ArrayList<Integer> ans`. Here `ans` should be a static field defined outside the method `nTrees`, but in the same class. Before calling `nTrees` the first time, set the size of `ans` to 0. In the first line of `nTrees(n)`, check whether `n < ans.size()`. If so, just return the value that is there; it has already been computed. If not, compute the value recursively as in part (a), but before returning it, add it to `ans`.

The difference is remarkable. Calculating `nTrees(10)` without caching requires 59049 recursive calls as compared to 111 with caching.

9. (15 points) Let $T(n)$ be the number of binary trees with n nodes that you calculated in the previous exercise. Show that $T(n)$ is exponential by proving by induction that there exists $c > 1$ such that $T(n) \geq c^{n-1}$ for all $n \geq 0$. (*Hints.* Use the recurrence $T(n) = \sum_{m=0}^{n-1} T(m) \cdot T(n-1-m)$. Take $c = \sqrt{2}$. Do two base cases. Use strong induction.)

Let $c = \sqrt{2}$ and let $T(n)$ be the unique solution to the recurrence $T(0) = 1$ and $T(n) = \sum_{m=0}^{n-1} T(m) \cdot T(n-1-m)$ for $n \geq 1$. We show that $T(n) \geq c^{n-1}$ for all $n \geq 0$.

The proof is by induction on n.

Basis: $n = 0$ and $n = 1$. $T(0) = 1 \geq c^{-1}$ and $T(1) = 1 \geq c^0$.

Induction step: $n \geq 2$. Unwinding the recurrence once,

$$\begin{aligned}
T(n) &= \sum_{m=0}^{n-1} T(m) \cdot T(n-1-m) \\
&\geq \sum_{m=0}^{n-1} c^{m-1} \cdot c^{n-1-m-1} && \text{by the strong induction hypothesis} \\
&= \sum_{m=0}^{n-1} c^{m-1+n-1-m-1} \\
&= n \cdot c^{n-3} \\
&\geq c^{n-1} && \text{since } n \geq c^2.
\end{aligned}$$

End of Exam