

CS106B

Instructor: Cynthia Lee

Spring 2016

Solutions

PRACTICE FINAL EXAM #4 (DRESS REHEARSAL) - SOLUTIONS

1. MST


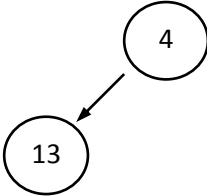
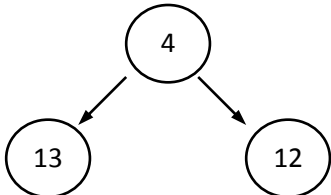
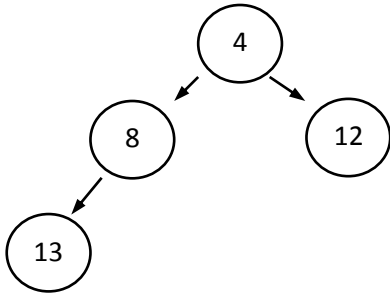
AC, CD, AB, DF, AE

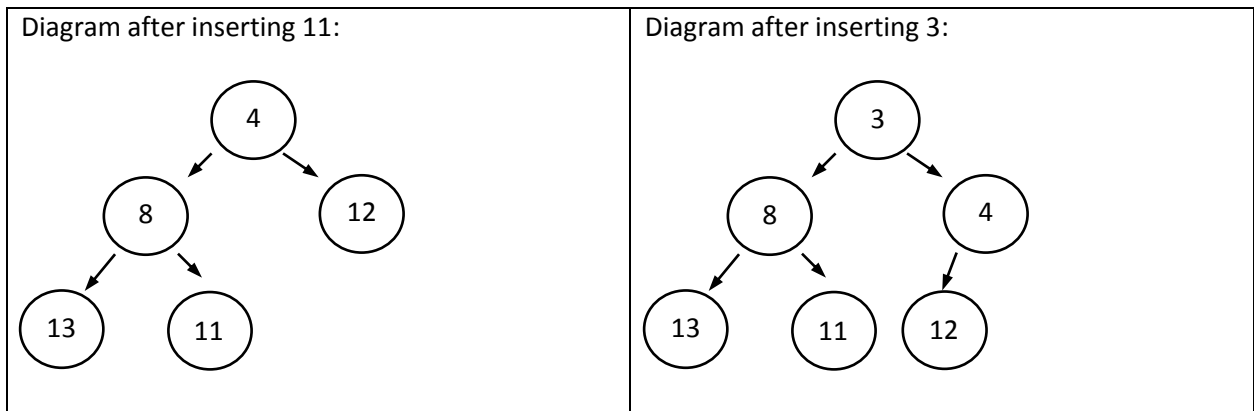
2. Linked Lists

```
delete list2->next;
list2->next = list->next;
list->next = NULL;
list2->next->next = list;
list = list2;
```

3. Heaps

(a)

<p>Diagram after inserting 13:</p>  <p><i>This one is completed for you.</i></p>	<p>Diagram after inserting 4:</p> 
<p>Diagram after inserting 12:</p> 	<p>Diagram after inserting 8:</p> 



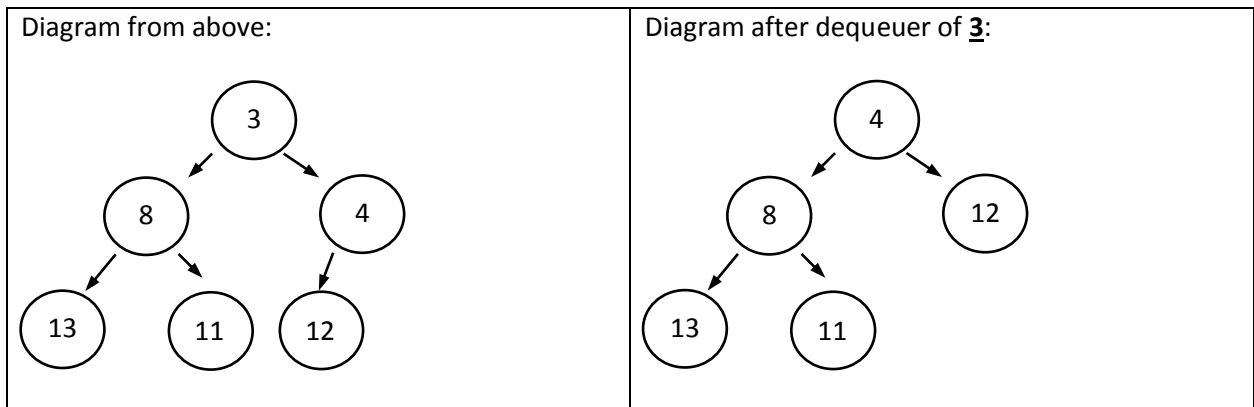
(b)

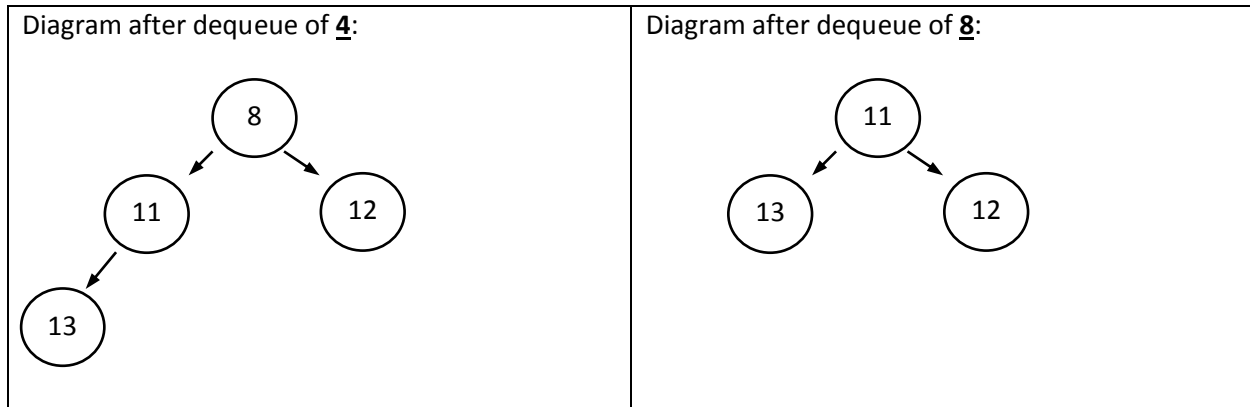
0	1	2	3	4	5	6	7	8	9
3	8	4	13	11	12				

Or:

0	1	2	3	4	5	6	7	8	9
	3	8	4	13	11	12			

(c)





4. Graphs

```

bool findEulerPathHelper(BasicGraph& graph, Vertex* v, Vector<Vertex*>& path) {
    if (path.size() == graph.getEdgeSet().size() / 2 + 1) {
        return true; // base case: all the edges have been used (this is
                    // half the total edges because graph is undirected)
    } else {
        // recursive case: for each unused edge: choose, explore, unchoose
        for (Edge* edge : graph.getEdgeSet(v)) {
            if (!edge->visited) {
                // choose
                Edge* partner = graph.getEdge(edge->finish, edge->start);
                edge->visited = true; // mark this edge and its corresponding
                partner->visited = true; // directed edge in the reverse direction
                path.add(edge->finish);

                // explore
                if (findEulerPathHelper(graph, edge->finish, path)) {
                    return true;
                }

                // unchoose
                edge->visited = false;
                partner->visited = false;
                path.remove(path.size() - 1);
            }
        }
        return false;
    }
}

Vector<Vertex*> findEulerPath(BasicGraph& graph) {
    Vector<Vertex*> path;
    if (graph.isEmpty() || graph.getEdgeSet().isEmpty()) {
        return path; // special case: return empty path
    }
}

```

```

for (Vertex* v : graph) {
    graph.resetData();
    path.add(v);
    if (findEulerPathHelper(graph, v, path)) {
        break;
    } else {
        path.clear(); //prepare for next attempt
    }
}
return path;
}

```

5. BSTs

```

void partitionTree(BSTnode *tree, Vector<int>& lessThan, Vector<int>& greaterThan){
    bool sawSentinel = false; // false must be stored in a named variable before
    // passing as arg because it's pass by reference
    partitionTree(tree, lessThan, greaterThan, sawSentinel);
}

```

```

// In order traversal; keep track of when we saw sentinel
void partitionTree(BSTnode *tree, Vector<int>& lessThan, Vector<int>& greaterThan,
    bool& sawSentinel){
    // base case
    if (tree == NULL) {
        return;
    }
    // first traverse left
    partitionTree(tree->left, lessThan, greaterThan, sawSentinel);

    // handle visit of self by adding self key to correct vector (or setting
    // flag if self is the sentinel)
    if (tree->key == -1) {
        sawSentinel = true;
    } else {
        if (sawSentinel) {
            greaterThan.add(tree->key);
        } else {
            lessThan.add(tree->key);
        }
    }

    // then traverse right
    partitionTree(tree->right, lessThan, greaterThan, sawSentinel);
}

```

6. Inheritance

```

var1->m1();           // H1 B3 B1
var1->m2();           // COMPILER ERROR
var2->m2();           // COMPILER ERROR
var3->m1();           // H1 B3 B1
var3->m4();           // COMPILER ERROR
var4->m2();           // G2
var4->m4();           // G4
var4->m1();           // H1 B3 B1
var5->m2();           // G2
var5->m4();           // B3 E4
var5->m1();           // H1 B3 B1

((Hamilton*) var4)->m2(); // COMPILER ERROR
((Eliza*) var4)->m4();   // B3 E4
((Eliza*) var1)->m1();  // H1 B3 B1
((Eliza*) var2)->m2();  // CRASH!
((Hamilton*) var4)->m1(); // H1 B3 B1
((Eliza*) var3)->m4();  // B3 E4

```

7. ADTs

```

// uses inclusion/exclusion pattern of recursive backtracking
bool canSell(Map<string,offer>& customerOffers, int nTix, int minCash,
             Vector<string>& acceptedOffers) {
    if (customerOffers.size() == 0) {
        return false;
    }
    if (nTix <= 0) {
        return false;
    }
    if (minCash <= 0) {
        return true;
    }

    // choose an offer
    string custName;
    for (string s : customerOffers) { custName = s; break; }
    offer cust = customerOffers[custName];
    customerOffers.remove(custName);

    // try for a solution WITH this offer
    acceptedOffers.add(custName);
    if (canSell(customerOffers, nTix - cust.seats, minCash - cust.price,
                acceptedOffers)) {
        return true;
    }
    acceptedOffers.remove(acceptedOffers.size()-1);
}

```

```
// try for a solution WITHOUT this offer
if (canSell(customerOffers, nTix, minCash, acceptedOffers)) {
    return true;
}

return false;
}
```
