

CS106B

Instructor: Cynthia Lee

Spring 2016

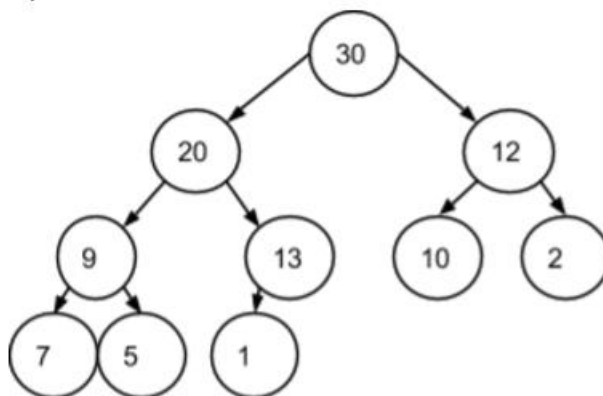
Solutions

PRACTICE FINAL EXAM 3 - SOLUTIONS

1. Sorting

9	4	8	5	1	2	3	7	6
4	9	8	5	1	2	3	7	6
4	8	9	5	1	2	3	7	6
4	5	8	9	1	2	3	7	6
1	4	5	8	9	2	3	7	6
1	2	4	5	8	9	3	7	6
1	2	3	4	5	8	9	7	6
1	2	3	4	5	7	8	9	6
1	2	3	4	5	6	7	8	9

2. Heap



Circle: YES

3. BST

(5,2)	(5,5)
<pre> (5,5) \ (9,3) </pre>	<pre> (5,5) \ (9,3) / (6,12) </pre>
<pre> (5,5) \ (9,3) / (6,12) \ (7,5) </pre>	<pre> (5,5) \ (9,3) / (6,12) \ (7,5) \ (8,1) </pre>

4. Trees

```

Node* makeTree(Stack<Node*>& expression) {
    Node* tree = NULL;
    if (!expression.isEmpty()) {
        tree = expression.pop();
        if (current->operator == '+') {
            tree->left = makeTree(expression);
            tree->right = makeTree(expression);
        }
    }
    return tree;
}

```

5. Hashing

```

struct node {
    int val;
    node* left;
    node* right;
};
const int P = 715827883;
const int Q = 2796203;

Map<int, Vector<node*>> simulateTreeHash(Vector<node*> trees, int k) {
    Map<int, Vector<node*>> result;
    for (node* root : trees) {

```

```

        int key = hash(root, k);
        result[key] += root;
    }
    return result;
}

int hash(node *root, int k) {
    if(root == NULL) return 1;
    return (P * hash(root->left, k) * h(root->val)
        + Q * hash(root->right, k)) % k;
}

```

6. Graphs and Classes

/* This solution uses some structs and a Map to implement its own version of
 * Graph ADT, using the Adjacency List style of Graph representation that we
 * learned in class (instead of the full BasicGraph class). It is also
 * reasonable to solve it with BasicGraph.
 */

```

class LifeGame {
public:
    void initializeCells(string filename);
    void updateCells();
    void printCells();
private:    //this may be more space than you need
    struct edge {
        int start;
        int end;
        int weight;
    }
    struct cell {
        Vector<edge> friends;
        bool alive;
    }
    Map<int, cell> graph;

    getFriends(int center, int dist, Set<int> result);
    countFriends(int center);
};

```

```
static const int MAXDIST = 20;
```

```
#include <iostream>
```

```
void LifeGame::initializeCells(string filename){
```

```

    ifstream myfile;
    myfile.open(filename);
    int numCells;
    myfile >> numCells;
    for (int i = 0; i < numCells; i++) {
        graph[i].alive = getRandomInteger(0, 1);
        for (int j = 0; j < numCells; j++) {
            int weight;
            myfile >> weight;
            if (weight > 0) {
                edge e = {i, j, weight};
                graph[i].friends += e;
            }
        }
    }
}

void LifeGame::updateCells() {
    Map<int, int> counts;
    for (int key : graph) {
        counts[key] = countFriends(key);
    }
    for (int key : graph) {
        if (counts[key] < 2 || counts[key] > 3) {
            graph[key].alive = false;
        } else if (counts[key] == 3) {
            graph[key].alive = true;
        }
    }
}

void LifeGame::getFriends(int center, int dist, Set<int>& result){
    if (result.contains(center)) return; // We've been here before
    result.add(center);
    for (edge e : graph[center].neighbors) {
        if (dist >= e.weight) {
            getFriends(e.end, dist - e.weight, result);
        }
    }
}

int LifeGame::countFriends(int center) {
    Set<int> friends;
    getFriends(center, MAXDIST, friends);
    int count = 0;
    for (int friendID : friends) {

```

```
        if (graph[friendID].alive) count++;  
    }  
    return count;  
}
```
