CS106B | Instructor: Cynthia Lee

Spring 2016 | Solutions

PRACTICE FINAL EXAM 1 - SOLUTIONS

---

1. Graphs

```
Vertex* findLargestTree(BasicGraph& graph) {
    int largestTreeSize = 0;
    Vertex* largestTreeRoot = NULL;

    for (Vertex* v : graph.getVertexSet()) {
        graph.resetData();

        int treeSize = findLargestTree(v, graph);
        if (treeSize > largestTreeSize) {
            largestTreeRoot = v;
            largestTreeSize = treeSize;
        }
    }

    return largestTreeRoot;
}

int findLargestTree(Vertex* v, BasicGraph& graph) {
    if (v == NULL) return 0;
    if (v->visited) return -1;

    v->visited = true;
    int treeSize = 1;
    for (Edge* e : v->edges) {
        int subTreeSize = findLargestTree(e->finish, graph);
        if (subTreeSize < 0) return -1;
        treeSize += subTreeSize;
    }
    return treeSize;
}
```

2. Pointers and Linked Lists

```
struct listnode {
   int val;
   listnode * next;
};

bool contains(listnode* list, listnode* sub) {
    if (sub == NULL) {
        return true;
    } else if (list == NULL) {
        return false;
    }

    if (list->val == sub->val) {
        return contains(list->next, sub->next);
    } else {
        return contains(list->next, sub);
    }
}
```

3. Recursion

```
Set<int> maxSumSubset (treenode* root) {

    if (root == NULL) return Set<int>();

    Set<int> childSet = maxSumSubset(root->left) +
                        maxSumSubset(root->middle) +
                        maxSumSubset(root->right);

    int childSum = 0;
    for (int i : childSet) {
        childSum += i;
    }

    if (childSum > root->key) {
        return childSet;
    } else {
        Set<int> us;
        us += root->key;
        return us;
    }
}
```

4. BSTs and Heaps

Diagram after inserting (25,2):

This one is completed for you.

Diagram after inserting (17,1):

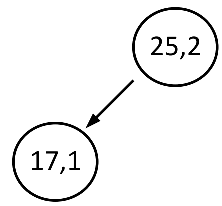


Diagram after inserting (29,0):

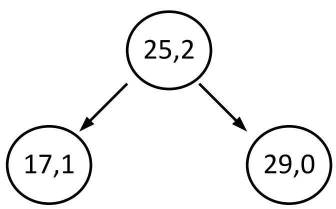


Diagram after inserting (55,1):

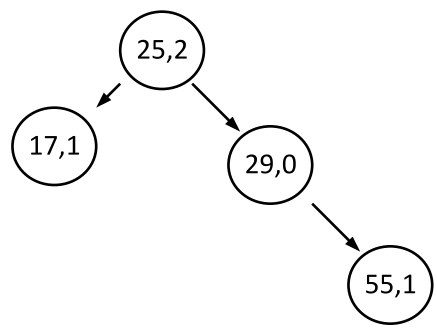


Diagram after inserting (45,7):

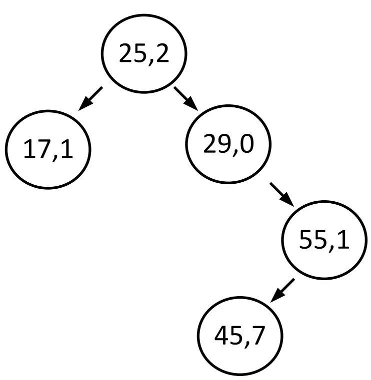


Diagram after inserting (29,3):

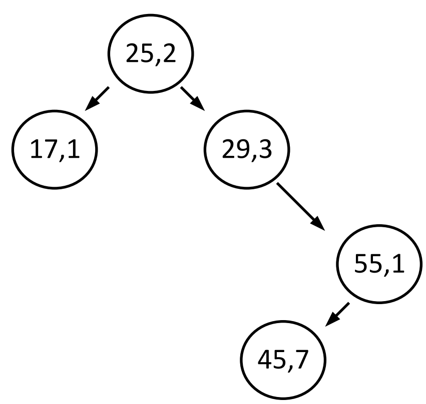
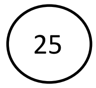


Diagram after inserting 25:



This one is completed for you.

Diagram after inserting 37:

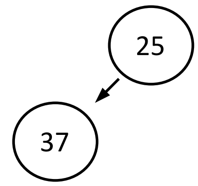


Diagram after inserting 28:

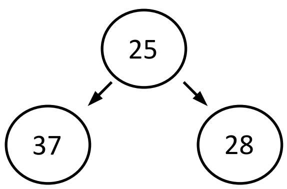


Diagram after inserting 12:

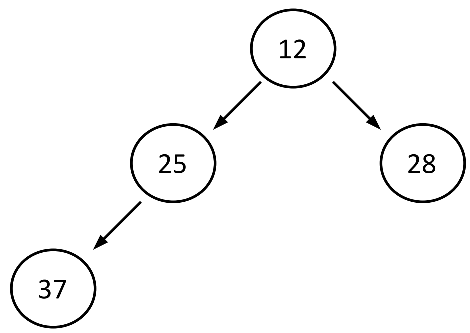


Diagram after inserting 30:

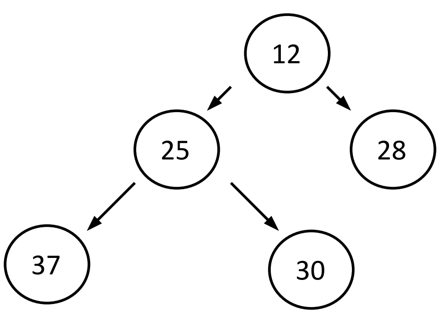
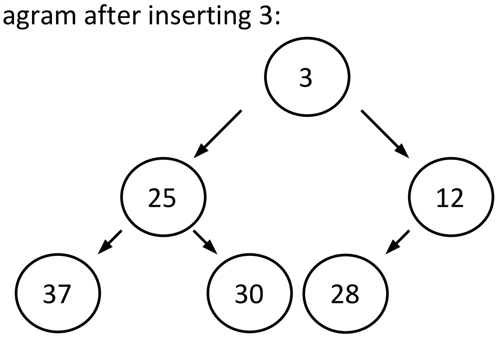


Diagram after inserting 3:



5. Inheritance

```
class Byron {
public:
    virtual void m3() {
        cout << "B 3" << endl;
        m1();
    }

    virtual void m1() {
        cout << "B 1" << endl;
    }
};

class Yeats : public Byron {
public:
    virtual void m3() {
        Byron::m3();
        cout << "Y 3" << endl;
    }

    virtual void m4() {
        cout << "Y 4" << endl;
    }
};

class Plath : public Yeats {
public:
    virtual void m1() {
        cout << "P 1" << endl;
        Yeats::m1();
    }

    void m3() {
        cout << "P 3" << endl;
    }
};

class Angelou : public Plath {
public:
    virtual void m4() {
        cout << "A 4" << endl;
        m3();
    }

    void m3() {
        cout << "A 3" << endl;
    }
};
```

Now assume that the following variables are defined:

```
Byron* var1 = new Plath();
Yeats* var2 = new Angelou();
Byron* var3 = new Byron();
Byron* var4 = new Yeats();
Yeats* var5 = new Plath();
```

In the table below, indicate in the right-hand column the output produced by the statement in the left-hand column. If the statement produces more than one line of output, indicate the **line breaks with slashes** as in "x / y / z" to indicate three lines of output with "x" followed by "y" followed by "z".

If the statement does not compile, write "**compiler error**". If a statement would crash at runtime or cause unpredictable behavior, write "**crash**".

Statement	**Output**
`var4->m3();`	`B 3 / B 1 / Y 3`
`var4->m1();`	`B 1`
`var4->m4();`	**COMPILER ERROR**
`var2->m3();`	`A 3`
`var2->m1();`	`P 1 / B 1`
`var2->m4();`	`A 4 / A 3`
`var1->m4();`	**COMPILER ERROR**
`var1->m3();`	`P 3`
`var1->m1();`	`P 1 / B 1`
`var5->m1();`	`P 1 / B 1`
`var5->m4();`	`Y 4`
`var5->m3();`	`P 3`

Statement	**Output**
`((Yeats*) var4)->m3();`	`B 3 / B 1 / Y 3`
`((Yeats*) var4)->m4();`	`Y 4`
`((Angelou*) var3)->m4();`	**CRASH**
`((Byron*) var5)->m4();`	**COMPILER ERROR**
`((Plath*) var2)->m3();`	`A 3`
`((Angelou*) var2)->m3();`	`A 3`

6. Algorithms

(a)

O(log n)

If n is even, we divide by two, otherwise we add one to n. Clearly, **Binky** cannot add one to n twice in a row. There must therefore be at least as many steps where we divide n by 2 as there can be steps where we add one to n. As n gets large, the number of times we have to divide n by two will be the factor that determines how quickly we approach zero or one. There can be at most log(n) of those steps, so the running time is therefore O(log n)

(b)

Does this strategy work? YES **NO** (circle) Briefly explain why or why not:

If we are deleting the last cell in the list, **ptr->next** is **NULL**. When try to access assign to ***(ptr)** in the next line, the right hand side will dereference **NULL** and crash.

(c)

	Worst-case big-O
1.	**O(n^2)**
2.	**O(n log(n))**
3.	**O(n^2)**