EECS 281, Week 6: Monday

Trees

A tree is _____.

In a _____ tree, each node can have zero or more children. Each node has one parent, except for the root, which has no parents.

A _____ is a node with no children.

An _____ is a node that is not a leaf.

Two nodes are _____ if they have the same parent.
The ancestors of a node *d* are the nodes on the path from *d* to the root. The root is an ancestor of every node in the tree.

If *a* is an _____ of *d*, then *d* is a _____ of *a*.

The length of a path is the number of edges in the path.

The _____ of a node *n* is the length of the path from *n* to the root.

The depth of the root is ____. The depth of any node is _____.

The _____ of a node *n* is the length of the path from n to its deepest descendant.

The height of a leaf node is ____. The height of an internal node is _____.

Binary Trees

A binary tree is _____.

A common way to represent binary trees:

```
template <typename T>
class BinaryTreeNode {
    T item;
    BinaryTreeNode *parent;
    BinaryTreeNode *left;
    BinaryTreeNode *right;
};

template <typename T>
class BinaryTree {
    BinaryTreeNode<T> *root;
    int size;
};
```

EECS 281, Week 6: Monday

Traversals

Pre-order traversal: _____

Post-order traversal: _____

In-order traversal: _____

Level-order traversal: _____

Binary Search Trees

EECS 281, Week 6: Monday

AVL Trees

EECS 281, Week 6: Monday

Tries

A trie is _____.

Draw a trie that stores the words “cat”, “cats”, “catdog”, “cheese”, “code”, “dog” and “dolphin”.

An easy way to represent binary tries:

```
struct TrieNode {
    <SomeType> data;
    TrieNode *children[ALPHABET_SIZE];
}

class Trie {
    TrieNode *root;
    int size;
};
```

How would we then represent a trie that stores the same words as above?

Complexity: _____.

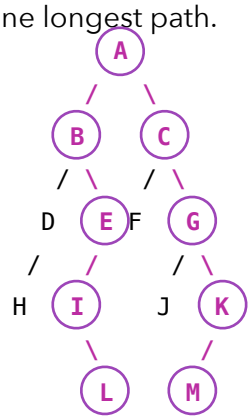
How to reduce memory used?

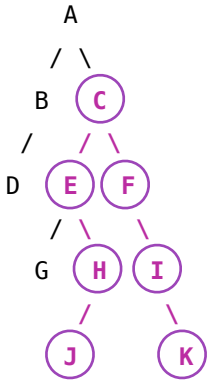
Tree Diameter

Consider this `BinaryTreeNode` class:

```
class BinaryTreeNode
{
public:
    BinaryTreeNode* left;
    BinaryTreeNode* right;
    int value;
    BinaryTreeNode(int n) : value(n), left(nullptr), right(nullptr) {}
};
```

The *diameter* of a tree is the maximum number of edges on any path connecting two nodes of the tree. For example, here are two sample trees and their diameters. In each case the longest path is shown with by circling the codes and shown in purple. Note that there can be more than one longest path.





Implement the function `diameter` that computes the diameter of a binary tree represented by a pointer to an object of `BinaryTreeNode` class.

```
int diameter(const BinaryTreeNode* tree) {
    // ...
}
```

Assume that `nullptr` represents an empty tree or a missing child. Do not modify the definition of `BinaryTreeNode` class, but you may write helper functions. Implement `diameter` as efficiently as possible.

Continued on the next page.

What is the worst case time complexity of `diameter`? Explain.

How can you change the `BinaryTreeNode` class to improve the worst time complexity of `diameter`?

Challenge question: In general, what is the diameter of a complete binary tree with *N* nodes? Express your answer in terms of *N*.

When finished, take pictures of your answers on pages 4 and 5 and email them to Maxim.