This is the first lecture on the topic of binary tree.

We talked about binary tree earlier when we introduced linked list.

This slide is a review. A binary tree is composed of nodes that have two pointers.

This slide is also a review. A binary tree is fundamentally different from a linked list because 2 can be the base to express any positive number. However, one cannot do that.

This slide reviews the difference between a linked list and a binary tree. For a linked list, if we want to go the middle of the list, we have to pass many nodes. This problem is the same regardless whether the linked list is a doubly linked list or not. A doubly linked list is still a one-dimensional structure.

In contrast, a binary tree is a two-dimensional structure. It is possible arranging the data so that half the data is on one side and half of the data is on the other side. By this structure, it is possible to eliminate half of the data in a single step by choosing left or right.

When we talk about a tree, we usually think of the root at the bottom and the leaves at the top. For a binary tree, it is upside down. The root is at the top and the leaves are at the bottom. This is the convention.

Let’s introduce some definitions. A tree has nodes and edges. The nodes represent the allocated memory. Each edge means a pointer storing the addresses of the memory of the other nodes.

Usually, we do not draw the edge if it points to NULL.

Usually, the edges go downwards only. In most cases, tree nodes have no upward pointers.

If node ei has a pointer to the memory of node bee, then node ei is the parent of node bee. In this example, node ei is the parent of both nodes bee and cee. Bee and Cee are called ei’s child nodes. More specifically, bee is ei’s left child node. Cee is ei’s right child node. By convention, we call the nodes parents and children. We do not call them father node, mother node, son node, or daughter node.

Since bee and cee have the same parent node, ei, bee and cee are called siblings.

In a binary tree, each node may have zero, one, and at most two child nodes.

If a node has no parent node, this node is the root of the tree. A tree can have only one root node.

If a node has no child node, this node is a leaf node. A tree can have many leaf nodes. .

If ei is bee’s parent, then ei is bee’s ancestor. If ei is bee’s parent and bee is dee’s ancestor, then ei is also dee’s ancestor. Please notice that this is a recursive definition.

If ei is bee’s ancestor, then bee is ei’s offspring. .

A path is the sequence of edges from an ancestor node to an offspring node. Please notice that in a tree, there is exactly one path between a pair of an ancestor and an offspring nodes. .

The height of a node is the length of the longest path to a leaf node. If a node is a leaf node, by definition, its height is zero.

In this example, E is a leaf node and its height is zero. Dee has one child H and H is a leaf node. Thus, Dee’s height is one. Bee’s height is 2. .

The height of a tree is the height of the root node. In this example, the tree’s height is 3.

The depth of a node is the distance to the root. The depth of cee is 1. The depth of J is 3.

If a binary tree is full, then a node has either two children or no child. .

A binary tree is perfect if it is a full tree and all leaf nodes have the same length to the root. .

Bee is A’s left child. Bee and bee’s offsprings are ei’s left subtree.

Cee is A’s right child. Cee and cee’s offsprings are ei’s right subtree.

Binary tree is very important because of its performance. The concept is the following: logarithm grows very slowly. Exponent grows very fast.

If a perfect binary tree has height n, then the tree has 2 to the n power plus one minus one node.

If a tree has height zero, there is only one node, the root node. .

If a tree has height of one, the tree has the root and its two children, totally three nodes.

In this example, the tree has height of two, the three has seven nodes. .

Why is binary tree so important? Because in a single step, we can discard half of the data by choosing to go left or to go right. Based on a condition, a program goes left or right. .

Let us consider a simple condition, like comparing the values of two integers. This is a very simple operation inside a computer.

Consider a mobile phone. It has a processor inside. This processor can compare two integers within a nanosecond. That is ten to the power minus 9 second.

In one second, this processor can execute more than one billion comparisons of two integers.

Each comparison can select going left or going right. Thus, in a single second, the processor in a mobile phone can go through a tree with height of a billion.

If this binary tree is perfect, then the tree has approximately two to the one billion power nodes.

How large is this number? What does two to the power of one billion mean?

Scientists estimate that the universe has about ten to the 82 power atoms.

Two to the 3rd power 3 is 8 and it is smaller than 10.

Two to the power 4 is 16 and it is greater than 10.

Thus, ten to the power 82 is between two to the power 3 times 82, and two to the power 4 times 82.

Let’s simply pick the bigger one.

The universe has approximately two to the power 328 atoms. That is a pretty big number. However, it is not comparable with two to the power of one billion.

Within one second, the processor in a phone could go from the top to the bottom of a binary tree that has two to the power of one billion nodes.

This is an extremely large number, much large than the number of atoms in this universe. Of course, it is not possible to build such a large tree since we need atoms to store data. .

The purpose of this discussion is to explain the performance of binary tree. If data can be organized so that half of data can be discarded in a single step, the data structure is very efficient. .

Binary trees can be used to solve many types of problems. One type of problems involves ordering data. This type of binary trees is called binary search trees.

For these problems, each node in a binary tree store a piece of data called key. .

The keys must be totally ordered. What does that mean? Consider three values, ei, bee, and cee, in a set.

The set is totally ordered if the following conditions hold:

If ei is less than or equal to bee, and at the same time bee is less than or equal to ei, then ei and bee must be equal. .

If ei is less than or equal to bee, and at the same time bee is less than or equal to cee, then ei must be less than or equal to cee. This is called transitivity. .

Any two elements in the set must meet either of the two conditions: either ei is less than or equal to bee, or bee is less than or equal to ei.

.

Integers and real numbers are totally ordered sets. Thus, integers and real numbers can be keys for a binary search tree.

In contrast, complex numbers are not totally ordered. .

There are many other cases when data is not totally ordered. For example, it is not obvious how to order students. Instead, we have to choose representations of students so that students can be ordered. For example, we may order students by their ID numbers because ID numbers are integers. .

After selecting the keys to represent the data, a binary search tree has the following properties:

For every node, the following property holds: .

The key of every node in the left subtree must be smaller than the node’s key.

The key of every node in the right subtree must be greater than the node’s key.

Usually the nodes store distinct keys.

In other words, the same key must not be stored in two different nodes.

Let’s see how a binary search tree can be used in search.

First, let’s check whether this is a binary search tree.

The root stores 27. The left subtree stores 9, 3, 15, and 6. The key in every node in the left subtree is smaller than 27.

Next, we check the right subtree. The right subtree stores 38, 32, 46, 29 and 36. All of them are greater than 27.

Thus, for the root node, the property of binary search tree is satisfied.

Next, we need to check node 9. Its left subtree has values 3 and 6. Both are smaller than 9. The right subtree of node 9 stores 15. This is greater than 9.

So far, the property of binary search tree is satisfied.

We go further to check node 3. It has no left subtree. Its right subtree stores value 6 and it is greater than 3. .

Next, we check node 38. Its left subtree stores 32, 29, and 36. All of them are smaller than 38. Thus, the property of binary search tree is still satisfied.

For node 32, the left subtree has 29 and it is smaller than 32. The right subtree has value 36 and it is greater than 32.

We have checked every node. The property of binary search tree is satisfied. This is a binary search tree.

Next, we consider a case that is not a binary search tree. In this example, 25 is at the right subtree of node 27. Since 25 is smaller than 27, 25 should be in the left subtree.

This is another example that violates the property of binary search tree.

The left child of node 30 is 32. Since 32 is greater than 30, 32 should not be in the left subtree of 30.

A binary search tree, like a linked list, is another example of container structure. A container structure supports the following feature: insert data, delete data, search for data, and delete everything.

Let’s see a few examples how a binary search tree can be used to search whether a value is stored in the binary tree. .

Consider this binary search tree. Consider these numbers are the students’ I. Ds in a student database. We want to know whether 13 is stored in the tree. First, we compare 13 with the root node. The root stores number 27 and it is greater than 13. Thus, it is impossible finding 13 in the right subtree. We will search 13 in the left subtree only.

Next, compare 13 with 9. 13 is greater than 9. It is impossible finding 13 in the left subtree. We will search 13 in the right subtree only.

Compare 13 with 15. 13 is smaller. However, node 15 is a leaf node and has no child. Thus, 13 is not stored in this binary search tree.

In the second example, we want to search 32.

First, we compare 32 with the root. The root stores value 27 and it is smaller than 32. Thus, we will search 32 in the right subtree.

Next, we compare 32 with 38. 32 is smaller than 38. Thus, we will search 32 in the left subtree.

We have found 32 in the tree. The value 32 is stored in this tree.

What is the structure for a binary tree? It is similar to linked list or doubly linked list. Each has two pointers. The two pointers, by convention, are called left and right for pointing to the left child and right child.

A binary tree is a container class. Each node can store complex data. The data can be a structure.

For simplicity, this example uses an integer. This is a new data type called tree node.

The search function takes a pointer of tree node and an integer. The search function returns the address of the node that stores this integer value. If this value is not stored in the tree, the search function returns NULL.

This is how the search function works.

At the top, we first check whether the tree has any node. If the tree has nothing, it not possible to find the value and the function returns NULL. This is the stop condition of recursion.

Next, we compare the value stored in this node with the value we are searching. If the two values match, we have found the node. This function returns the address of this node.

If the value we are searching is smaller, then we search the left subtree.

Otherwise, we search the right subtree.

Please remember that the keys are totally ordered. Thus, two values must be in one of three possible scenarios:

The first scenario is that the two values are the same.

The second scenario is that this node’s value is greater.

The third scenario is that this node’s value is smaller. .

Every step moves down the tree. Thus, this recursive call will either find a node whose value matches, or the value cannot be found.