One of the tree applications in Chapter 10 is **binary search trees**. In Chapter 10, binary search trees are used to implement bags and sets. This presentation illustrates how another data type called a **dictionary** is implemented with binary search trees.

A dictionary is a collection of **items**, similar to a bag. But unlike a bag, each item has a string attached to it, called the item's **key**.

Example: The **items** I am storing are records containing data about a state.

The insertion procedure for a dictionary has two parameters.

```c
void Dictionary::insert(string key, Item item);
```

When you want to retrieve an item, you specify the **key**.

```c
Item Dictionary::retrieve(string key);
```
The Dictionary Data Type

When you want to retrieve an item, you specify the key... ... and the retrieval procedure returns the item.

The Dictionary Data Type

We'll look at how a binary tree can be used as the internal storage mechanism for the dictionary.

A Binary Search Tree of States

The data in the dictionary will be stored in a binary tree, with each node containing an item and a key.

Storage rules:
1. Every key to the left of a node is alphabetically before the key of the node.
Example: 'Massachusetts' and 'New Hampshire' are alphabetically before 'Oklahoma'

A Binary Search Tree of States

Storage rules:
1. Every key to the left of a node is alphabetically before the key of the node.
2. Every key to the right of a node is alphabetically after the key of the node.
A Binary Search Tree of States

Storage rules:
1. Every key to the left of a node is alphabetically before the key of the node.
2. Every key to the right of a node is alphabetically after the key of the node.

Retrieving Data

Start at the root.
1. If the current node has the key, then stop and retrieve the data.
2. If the current node's key is too large, move left and repeat 1-3.
3. If the current node's key is too small, move right and repeat 1-3.

Retrieve 'New Hampshire'

Start at the root.
1. If the current node has the key, then stop and retrieve the data.
2. If the current node's key is too large, move left and repeat 1-3.
3. If the current node's key is too small, move right and repeat 1-3.
Adding a New Item with a Given Key

1. Pretend that you are trying to find the key, but stop when there is no node to move to.
2. Add the new node at the spot where you would have moved to if there had been a node.

Adding

1. Pretend that you are trying to find the key, but stop when there is no node to move to.
2. Add the new node at the spot where you would have moved to if there had been a node.

Adding

1. Pretend that you are trying to find the key, but stop when there is no node to move to.
2. Add the new node at the spot where you would have moved to if there had been a node.

Adding

1. Pretend that you are trying to find the key, but stop when there is no node to move to.
2. Add the new node at the spot where you would have moved to if there had been a node.

Adding

1. Pretend that you are trying to find the key, but stop when there is no node to move to.
2. Add the new node at the spot where you would have moved to if there had been a node.
Adding

1. Pretend that you are trying to find the key, but stop when there is no node to move to.
2. Add the new node at the spot where you would have moved to if there had been a node.

Removing 'Florida'

1. Find the item.

Removing an Item with a Given Key

1. Find the item.
2. If necessary, swap the item with one that is easier to remove.
3. Remove the item.

Kazakhstan is the new right child of Iowa?

Florida cannot be removed at the moment...
... because removing Florida would break the tree into two pieces.

The problem of breaking the tree happens because Florida has 2 children.

For the rearranging, take the **smallest** item in the right subtree...

...copy that smallest item onto the item that we're removing...

... and then remove the extra copy of the item we copied...

... and reconnect the tree
Removing 'Florida'

Why did I choose the smallest item in the right subtree?

Removing an Item with a Given Key

1. Find the item.
2. If the item has a right child, rearrange the tree:
   - Find smallest item in the right subtree
   - Copy that smallest item onto the one that you want to remove
   - Remove the extra copy of the smallest item (making sure that you keep the tree connected)
   - else just remove the item.

Summary

- Binary search trees are a good implementation of data types such as sets, bags, and dictionaries.
- Searching for an item is generally quick since you move from the root to the item, without looking at many other items.
- Adding and deleting items is also quick.
- But as you'll see later, it is possible for the quickness to fail in some cases -- can you see why?