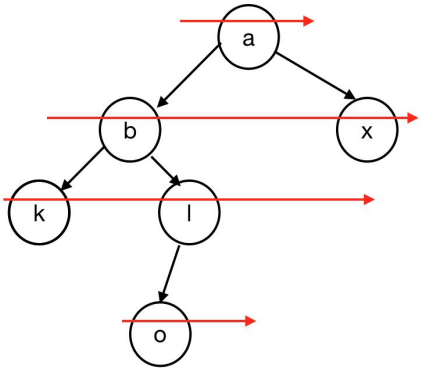
CS 225 Spring 2019 :: TA Lecture Notes 2/20 BST

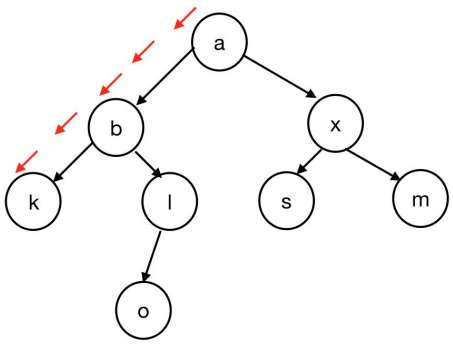
By Wenjie

- **Traversal vs Search:**
  - Traverse: to visit each node exactly one time for a tree
  - Search: to find one of nodes specifically in a tree

- **BFS**
  - The idea is to visit nearby nodes quickly by each level. First the current node, then all its children, and then all the children of the children. In other words, visit the direct descendants quicker. This is what we called level-order traversal.



- **DFS**
  - The idea is to find the endpoint of a path quickly, and to move deeper into the tree as quickly as possible. In other words, we visit the leaves as soon as possible.



CS 225 Spring 2019 :: TA Lecture Notes 2/20 BST

By Wenjie

- **Running time**
  - Normally find function takes O(1) time in the best case scenario (the node we are looking for is at the root) and O(n) in the worst case scenario (the node we are looking for is the very last node we visit).

- **Dictionary ADT**
  - Parallel with lists which can be considered as a set of `index, value` pairs (in dictionaries keys are similar to indices in the arrays.)
  - Unique keys : Complex Data
  - Keys and values don't have to be of the exactly the same type, but all keys must be of consistent type and all values have to be of consistent type.
    - For example, a key can be of type string (all keys are of type string) and a value can be a list (all values are lists - all keys map to lists).
  - Templated class with two generics:
    `template<typename K, typename V>`
  - Basic operations for a dictionary :
    - `V & find(K & key);`
    - `void insert(K & key, V & value)`
    - `V & remove(K & key)`
    - `// We also need an iterator.`

- **BST**
  - **BST Definition:**
    - **$T = \{\}$ or $T = \{d, T_L, T_R\}$, where $\forall x, x \in T_L, x < d$**
      $x \in T_R, x > d$
  - In another words, this means everything to the left of the root is less than the root, and everything to the right of the root is greater than the root.
  - The properties are recursively → true for every node.

- **Find function in BST**
  - It starts at the root with sanity check.

CS 225 Spring 2019 :: TA Lecture Notes 2/20 BST

By Wenjie

- Then we look at the value at the current node.
  - If the value we are looking for is smaller than current, go for the left subtree recursively.
  - If the value we are looking for is bigger, go for the right.

BST.cpp

```
5   TreeNode * BST::_find(TreeNode *& root, const K & key) const
6   {
7        If (root == NULL || key == root→key) {
8             return root;  //root is null when we cannot find
9        }
10       If (key < root→key) {
11            return _find(root→left, key);
12       }
13       If (key > root→key) {
14            return _find(root→right, key);
15       }
     }
```

- **Insert in BST**
  - Find a position to `insert → _find(root, key)`.
    Create a new node, and insert at the position (always be inserted at the leaf)