Lec_27.md

Lecture 27

Nov. 18/2020

Announcements

Quiz 5

- Quiz 5 is on November 20th (This Friday)
- Covers
  - Lab 1 to 3
  - Covers all lecture material until **end of this lecture**
- Uses `"Classical Quizzes"` on Quercus
- Open textbook and open notes

Binary Search Trees

A **binary search tree** has the following properties

- Each node has a key
- The key of any node is greater than **any** key in the left subtree
- The key of any node is smaller than **any** key in the right subtree
- Descendants of the root node in a **BST** is also a **BST**

BST Traversal

For a tree:

> 14
>
> 10 16
>
> 8 11 15

1. LNR (**In-Order Traversal**)

- LNR traversal will print:
  - `8 10 11 14 15 16`
    - Notice that this is printing the tree **key's** in order of magnitude
  - This is how we have defined LNR (**In-Order**)

2. NLR
3. LRN

BST Searching

Objective: Given a BST with root N, find if the node with key `x` exists in the BST

Recall how **BST**'s are defined.

- Everything in the left subtree of a node has *smaller keys*
- Everything in the right subtree of a node has *larger keys*
- Descendants of the root node are also **BST**

Basic Search Algorithm:

1. Compare x to key(N)
2. If(x = key(N))

- x found in BST, return

3. If(x < key(N))

- x in left subtree, traverse left node

4. If(x > key(N))

- x in right subtree, traverse right node

BST Search Algorithm

```
treenode* SearchBST(treenode* N,int k){
  if(N==NULL){
    return NULL;
  }
  if(N->key==k){
    return N;
  }else if(N->key<k){
    return SearchBST(N->R,k);
  }else{
    return SearchBST(N->L,k);
  }
}
```

1. `if(N==NULL){ return NULL; }`

- This is the recursive **basis**
  - If the node doesn't exist, return `NULL`

2. `if(N->key==k){ return N; }`

- Node with key `k` found

3. `else if(N->key<k){ return SearchBST(N->R,k); }`

- Traverse right if the key `k` is larger than the key of node

4. `else{ return SearchBST(N->L,k); }`

- Traverse right if the key `k` is smaller than the key of node

BST Insertion

Objective: Given a BST with root N, where do we place a *new* node with key `k` ?

Really very similar to searching

- Instead of returning `NULL` if the key does not exist, we place the new node there.

Do you want to build a BST?

Build a BST from a sequence of nodes

For a set of numbers with same values but different order, you will get a different (unique) tree

BST Sorting

Can use a BST to sort a given sequence of keys.

1. Build BST from sequence of keys
2. Traverse the BST using **In-order Traversal**

BST Deletion

Objective: Given a BST with root N and key `k`, we want to delete the key `k` in the tree such that the tree is still a BST

1. Search for node with key `k`

- Denote this node `r`

2. If `r` is a leaf node

- Can *delete immediately*

3. If `r` is an interior node, with one subtree

- More complicated to deal with

4. If `r` is an interior node, with two subtrees

- Even more complicated

Case: If `r` is an interior node with one subtree

- Can delete `r`, and replace `r` with the **root** node of the one subtree under `r`
  - Since the subtree under `r` contains values that are *all larger* than the parent of `r`

Case: If `r` is an interior node with two subtrees

- Can delete `r`
- Need to combine the two subtree's under `r`
  - Remember that the because of the BST properties,
    - `key(x) < key(p) < key(r) < key(q)`
    - Denote `p` and `q` to be the **left subtree** and **right subtree** of `r` respectively
  - **4 ways** to dealing with combining these two subtrees:

1. Replace `r` with `p`, and make `q` the right child of the largest node in `p`
2. Replace `r` with `q`, and make `p` the left child of the smallest node in `p`
3. Replace `r` with the smallest node in `q`
4. Replace `r` with the largest node in `p`

Notice:

- Methods `1` and `2` **skew the tree** (make the tree deeper)
  - Tree is not nice and "balanced" like earlier
    - Balanced has a different meaning for BST's, so maybe we should use the phrase "the tree is not nice and stable like earlier"
- Methods `3` and `4` ***require the smallest/largest nodes to be leaf nodes***
  - Can only replace `r` with single nodes to maintain BST

- Need to implement a check to make sure they are **leaf nodes**