Justin Ciocoi

Nov. 27, 2023

CSCI 377 Textbook Notes

Chapter 12: Binary Search Trees

- The search tree data structure supports many dynamic-set operations, including search, minimum, maximum, predecessor, successor, insert and delete
- Basic operations conducted on a binary search tree will take time proportional to the height of the tree
  - For a complete binary tree with n nodes, these operations run in $\Theta(\log(n))$ worst-case time
- **12.1: What is a Binary Search Tree?**
  - A binary search tree is a binary tree represented by a linked data structure in which each node is an object and contains data as well as *left, right, and parent* attributes which point to the node's left child, right child, and parent respectively
  - If the child or parent is missing a value, the appropriate attribute contains the value NIL
  - Only the root node has no parent node
  - A binary search tree is stored in such a way as to satisfy the *binary-search-tree property:*

    Let x be a node in a binary search tree. If y is a node in the left subtree of x, then $y.key \leq x.key$. If y is a node in the right subtree of x, then $y.key \geq x.key$
  - This property allows us to print out the values of a binary search tree in sorted order using a fairly simple recursive algorithm called the *in-order tree walk*
    - This algorithm prints a parent in between printing the values in the left sub-tree and right sub-tree
  - A *pre-order* tree walk will print out the root before either subtree, whereas a *post-order* tree walk will print out the root after either subtree

```
In-Order-Tree-Walk(x)
    if x != NIL
        In-Order-Tree-Walk(x.left)
        print x.key
        In-Order-Tree-Walk(x.right)
```

- **Theorem 12.1**
    - If x is the root of an n-node sub-tree, then the function call `In-Order-Tree-Walk(x)` takes $\Theta(n)$ time

- **12.2: Querying a Binary Search Tree**
    - In this section the minimum, maximum, successor, and predecessor functions will be examined, including implementations such that they will be supported in $O(h)$ time in any binary search tree of height h
    - **Tree Search**

```
Tree-Search(x, k)
    if x==NIL or k==x.key
        return x
    if k < x.key
        return Tree-Search(x.left, k)
    else
        return Tree-Search(x.right, k)
```

- It will begin searching at the root, and for each node make a decision on which subtree to follow until the root of the `Tree-Search()` call is equal to the k value passed into the function

- **Minimum and Maximum**

```
Tree-Minimum(x)
    while x.left != NIL
        x = x.left
    return x
```

```
Tree-Maximum(x)
    while x.right != NIL
        x = x.right
    return x
```

- **Tree Successor**

  ```
  Tree-Successor(x)
      if x.right != NIL
          return Tree-Minimum(x.right)
      y = x.p
      while y != NIL and x == y.right
          x = y
          y = y.p
      return y
  ```

- **Insertion and Deletion**
  - *Insert(T, z)*
    - Walk through the tree starting at the root
    - Find the leaf position where z fits
  - *Delete(T, z)*
    - 3 Possible Cases
      1. If z has no children, delete z and modify its parent's link by setting it to NIL
      2. If z has one child, elevate the child to take z's position in the tree by replacing the parent's link to point at z's child
      3. If z has 2 children, we need to find the successor, y, to take z's place and attach z's right sub-tree to y's right sub-tree as well as z's left sub-tree to y's left sub-tree