Data Structures for Dynamic Sets

Algorithms operate on data, which can be thought of as forming a set S. The data sets manipulated by algorithms are dynamic, meaning they can grow, shrink, or otherwise change over time.

Data Structures are structured ways to represent finite dynamic sets. Different data structures support different kinds of data manipulations, e.g.,

- dictionary: insert, delete, test for membership
- priority queue: insert, extract-max

Operations on Dynamic Sets

- INSERT(S, x) adds element pointed to by x to S
- DELETE(S, x) removes element pointed to by x from S
- SEARCH(S, k) returns pointer to element x with key[x] = k (or returns nil)
- MINIMUM(S) returns element with the smallest key
- MAXIMUM(S) returns element with the largest key
- SUCCESSOR(S, x) returns element with the next key larger than key[x]
- PREDECESSOR(S, x) returns element with the next key smaller than key[x]

Running Time of a dynamic set operation is usually measured in terms of the size of the set (i.e., number of elements currently in the set).

Elementary Data Structures (Ch. 10)

These elementary data structures should be a review from CS102:

- arrays and linked lists (singly linked, doubly linked)
- stacks (e.g., implemented with arrays and lists)
- queues (e.g., implemented with arrays and lists)
- rooted trees (e.g., arbitrary trees using pointers, complete d-ary trees using arrays)

 BST Traversals

- The BST property allows us to print out all keys in sorted order using a simple recursive algorithm called an inorder tree walk. Strategy: visit left(x), visit x, visit right(x)

    Inorder-Tree-Walk(x)  // start at root
    1. if x ≠ NIL then
    2. Inorder-Tree-Walk(left(x))
    3. print key[x]
    4. Inorder-Tree-Walk(right(x))

    Running time = θ(n) (each node must be visited at least once)
   - The algorithms for pre-order and post-order traversals of BSTs are covered in a homework problem.

 BST Search

The iterative version is more efficient, in terms of space used, on most computers.

Both have running times of O(h), where h is the height of the tree.
**BST Min & Max**

The minimum element in a BST can always be found by following left child pointers to a leaf (until a NIL left child pointer is encountered). Likewise, the maximum element can be found by following right child pointers to a leaf.

Tree-Minimum(x)
1. while left[x] ≠ NIL do
2. x ← left[x]
3. return x

Both have running times of O(h), where h is the height of the tree.

**BST Successor & Predecessor**

- If x has a right child, then successor(x) is the smallest node in the subtree rooted at right(x).
- If x has no right child, then successor(x) is the nearest ancestor of x whose left child is also an ancestor of x (or x itself).

Tree-Successor(x)
1. if right[x] = NIL then
2. return Tree-Minimum(right[x])
3. temp ← parent[x]
4. while (temp ≠ NIL and x = right[temp])
5. x ← temp
6. temp ← parent[temp]
7. return temp

- If x has a left child, then predecessor(x) is the largest node in the subtree rooted at left(x).
- If x has no left child, then predecessor(x) is the nearest ancestor of x whose right child is also an ancestor of x (or x itself).

Tree-Predecessor(x)
1. if left[x] = NIL then
2. return Tree-Maximum(left[x])
3. temp ← parent[x]
4. while (temp ≠ NIL and x = left[temp])
5. x ← temp
6. temp ← parent[temp]
7. return temp

**BST Insert**

Insert(T, x)
1. y ← NIL
2. x ← root[T]
3. while x ≠ NIL do
4. y ← x
5. if key[x] < key[x]
6. x ← left[x]
7. else x ← right[x]
8. parent[x] ← y
9. if y = NIL then
10. root[T] ← x

**BST Delete**

Delete(T, x)
1. if left[x] = NIL or right[x] = NIL then
2. y ← x
3. else y ← TREE-SUCCESSOR(x)
4. if left[y] = NIL then
5. x ← left[y]
6. else x ← right[y]
7. if x = NIL then
8. parent[x] ← parent[y]
9. if parent[y] = NIL then
10. root[T] ← x
11. else if y = left[parent[y]] then
12. left[parent[y]] ← x
13. else if y = right[parent[y]] then
14. y ← x
15. key[y] ← key[x]
16. copy y’s satellite data tp x
17. return y

**BST Successor**

Case where right[x] not equal to NIL

Cases where right[x] equal to NIL

**BST Predecessor**

Case where left[x] not equal to NIL

Cases where left[x] equal to NIL

**What is running time of:  **

- Successor(x)?
- Predecessor(x)?
- Insert(T, x)?
- Delete(T, x)?