Tutorial

Question 1

What are binary search trees for?

The purpose of binary search trees is to maintain a dictionary set on which a set of operations can be applied. These operations include insert, delete, successor, predecessor, maximum and minimum.

Question 2

Assume that $a_1, a_2, \ldots, a_n$ is the node sequence obtained by a preorder tree traversal in a binary search tree, construct the binary search tree.

1. Given a sequence $A = a_1, a_2, \ldots, a_n$, we know that the first element, a_1, in the sequence must be the root of the binary search tree (see Figure 1).
2. Next, we find the element a_j such that $a_j > a_1$. Thus the possibly empty subsequence $a_2, a_3, \ldots, a_{j-1}$ are elements less than or equal to a_1; and whose root a_2 is the root of a_1's left subtree.
3. Conversely, the elements from the possibly empty sequence $a_j, a_{j+1}, \ldots, a_n$ are elements whose root a_j is the root of a_1's right subtree.

By using this approach recursively, the original binary tree can be reconstructed.

Question 3

Given a binary search tree where all keys are distinct, write pseudo-code to find the predecessor of a node in the tree.

Focusing on the sub-tree rooted at node x in Figure 2, we can say that all the nodes in area B are less than x, and all the nodes in area C are greater than x:

$$b < x < c, \forall b \in B, \forall c \in C.$$

Moving up the hierarchy, all the nodes in area D are greater than node x, and nodes in areas B and C. Next, all the nodes in area A are less than all the nodes in areas B, C, D, and node x. This gives us the relationship:

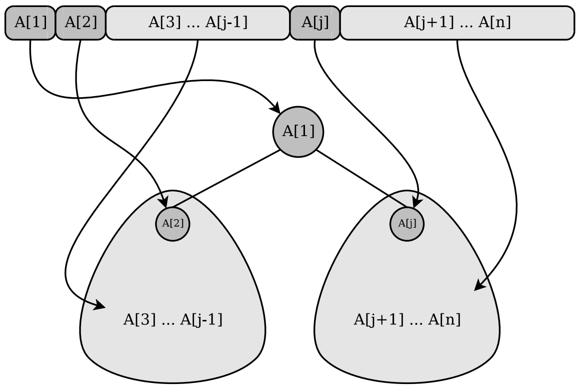

Figure 1: Mapping the preorder sequence of elements to the binary tree.

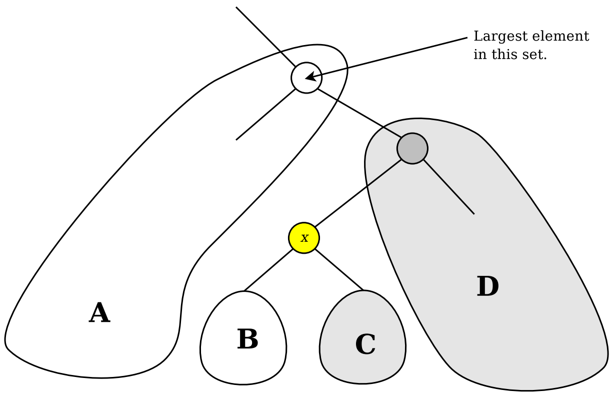

Figure 2: Generic binary tree.

$$a < b < x < c < d, \forall a \in A, \forall b \in B, \forall c \in C, \forall d \in D.$$

Since the predecessor of x is essentially the node with the largest key that is less than $key[x]$, then the predecessor must be either in area B, or if that does not exist, in area A.

The pseudo code below works as follows. If there x has a left sub-tree (area B), then return the maximum of the tree rooted at $left[x]$. Otherwise, traverse up the tree to find the top node of area A.

Note that the top node of area A is not necessarily the root of the tree.

```
def tree_predecessor(x):
    if left[x] != NIL:
        return tree_maximum(left[x])
    z = x
    y = parent[z]
    while y != NIL and z == left[y]:
        z = y
        y = parent[y]
    return y
```

Question 4

What is the difference between the binary search tree and the red-black tree?

The difference between the binary search tree and the red-black tree is that the height of binary search trees are unbounded, which means its height may be $O(n)$, while the height of a red-black tree is always bounded by $O(\lg n)$, which means each of the operations on it takes $O(\lg n)$ time.

Question 5

What is left/right rotation? What is it purpose when applied in red-black trees?

The left/right rotation at a node is used to reduce the height difference between its left and right subtrees, while maintaining the binary search tree property; that is, the sequences resulting from an inorder walk along both the original and resulting trees are the same.

For red-black trees, the left/right rotation technique is applied to the tree to maintain its red-black properties during insertion and deletion.

Question 6

Describe two heuristics used to improve the running time of the Find-Set *and* Union *operations in a forest representing disjoint-sets.*

1. **Union by rank**

   Used in the Link operation. We link the forest with the smaller rank to the forest with the larger rank. If both forests have the same rank, then we link forest x to y and increment the rank of y by 1.

```
def link(x, y):
  if rank[x] > rank[y]:
    p[y] = x
  else
    p[x] = y
    if rank[x] == rank[y]:
      rank[y] = rank[y] + 1
```

2. **Path compression**

   Path compression takes place within FIND-SET. It is based on the observation that a node is in the same set as its parent. So after finding the set representative, all the nodes we have searched through can update its parent to the set representative directly. In the FIND-SET algorithm, this is done post-recursion.

```
def find_set(x):
  if x != p[x]:
    p[x] = find_set(p[x])
  return p[x]
```