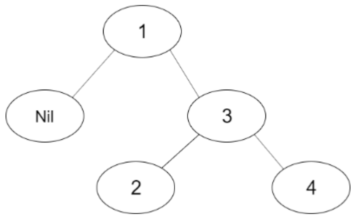
Samip Shah UTA ID # 1001709873
samipjanish.shah@mavs.uta.edu CSE5311

(1) Write the TREE-PREDECESSOR procedure.

```
TREE-PREDECESSOR(x)
if left[x] != EMPTY
        return TREE-MAX (left[x])
y ← p(x)
while y!= EMPTY && x=left(y)
        do x←y
        y←p[y]
return y
```

(2) Professor Bunyan thinks he has discovered a remarkable property of binary search trees. Suppose that the search for key k in a binary search tree ends up in a leaf. Consider three sets: A, the keys to the left of the search path; B, the keys on the search path; and C, the keys to the right of the search path. Professor Bunyan claims that any three keys a 2 A, b 2 B, and c 2 C must satisfy a < b < c. Give a smallest possible counterexample to the professor's claim.

Suppose we form a tree to search following example, then let A = {2}, B={2,4,5}, C = { }. Thus, Professors claimed can be proved false as 2<3



(3) Give a recursive version of the TREE-INSERT procedure.

```
TREE-INSERT(a,b)
if a = EMPTY
        then key[a]←b
        left[a]←EMPTY
        right[a]←EMPTY
else
        if b<key[a]
                then TREE-INSERT(left[a],b)
                else TREE-INSERT(right[a],b)
```

(4) Binary search trees with equal keys

Equal keys pose a problem for the implementation of binary search trees.

a. What is the asymptotic performance of TREE-INSERT when used to insert n items with identical keys into an initially empty binary search tree?

Reference: https://sites.math.rutgers.edu/~ajl213/CLRS/Ch13.pdf
https://github.com/gzc/CLRS/blob/master/C13-Red-Black-Trees/13.3.md
https://walkccc.github.io/CLRS/Chap12/Problems/12-1/

Samip Shah UTA ID # 1001709873
samipjanish.shah@mavs.uta.edu CSE5311

We propose to improve TREE-INSERT by testing before line 5 to determine whether z.key = x.key and by testing before line 11 to determine whether z.key = y.key. If equality holds, we implement one of the following strategies. For each strategy, find the asymptotic performance of inserting n items with identical keys into an initially empty binary search tree. (The strategies are described for line 5, in which we compare the keys of z and x. Substitute y for x to arrive at the strategies for line 11.)

Whenever, we will insert the element it will be inserted to right most position on right most leaf node because of inequality assumption we used in line 11 is always false. This will change runtime to $\sum_{i=1}^{n} i$ belongs to **Θ** (n^2).

b. Keep a boolean flag x.b at node x, and set x to either x.left or x.right based on the value of x.b, which alternates between FALSE and TRUE each time we visit x while inserting a node with the same key as x.

This will will result in two children subtrees with difference in size at one or more places. This means that the height will be Θ(lg n). So, the total runtime will be $\sum_{i=1}^{n} lg\ n$ belongs to **Θ** (n lg n)

c. Keep a list of nodes with equal keys at x and insert z into the list.

This will take linear time since the tree will be of height 0, and a single insertion in a list will be done in constant time.

d. Randomly set x to either x.left or x.right. (Give the worst-case performance and informally derive the expected running time.)

- Worst-case: All the choices are to the right or left then this will result in the same behaviour as problem 'a' which is $\Theta(n^2)$.
- Expected running time: when we choose randomly, we will pick left most of the time, so, the tree will be balanced, so, we have that the depth of lg (n), Θ(n lg n).

(5) Show the red-black trees that result after successively inserting the keys 41; 38; 31; 12; 19; 8 into an initially empty red-black tree.

Reference: https://sites.math.rutgers.edu/~ajl213/CLRS/Ch13.pdf
https://github.com/gzc/CLRS/blob/master/C13-Red-Black-Trees/13.3.md
https://walkccc.github.io/CLRS/Chap12/Problems/12-1/