DSA [04.10.24]

BINARY SEARCH TREE

"DYNAMIC SET OPERATIONS"

DICTIONARY: INSERT, SEARCH, DELETE

WHAT IF WE WANT MORE:
- → MIN/MAX
- → ELEMENT BEFORE OR AFTER
- → TRAVERSAL

WE WANT
$O(\lg n)$
WORST CASE:
$O(n)$

TYPICALLY:
$O(h)$ WHERE h IS HEIGHT

BEST:

WORST:

ENDS UP LIKE LINKED LIST

TREES WITH SELF-HEALING
1. AVL TREE
2. RED-BLACK TREE

BST → BINARY TREE THAT RESPECTS THE BST PROPERTY

RELATES THE VALUES OF A NODE TO IT'S CHILDREN
ASSUMING UNIQUE KEYS:
- X.LEFT.KEY < X.KEY
- X.RIGHT.KEY > X.KEY

DSA [04.10.24]

BST FUNCTIONS:
- → RECURSIVE?
- → MAY HAVE INTERNAL POINTER OR REFERENCE AS PARAM.
- → WE MAY NEED A PRIVATE AND PUBLIC PART FOR EACH FUNCTION.
- → MIGHT WANT WRAPPER FUNCTIONS:

POINTER — KEY VALUE

PRIVATE:
```
NODE* SEARCH(NODE* NODE, INT KEY){ // O(h)
  IF(NODE==NULL || NODE.KEY == KEY) // FOUND & NOT FOUND
    RETURN NODE;
  IF (KEY < NODE.KEY)
    RETURN SEARCH(NODE.LEFT.KEY);
  ELSE
    RETURN SEARCH(NODE.RIGHT.KEY);
}
```

PUBLIC:
```
BOOL FOUND(INT KEY){
  RETURN (SEARCH(ROOT,KEY) != NULL);
}
```

PRIVATE: ← STARTING NODE
```
NODE MIN(NODE* NODE){ // O(h)
  IF (NODE==NULL) // BOUNDS
    RETURN NULL;
  WHILE(NODE.LEFT != NULL)
    NODE = NODE.LEFT;
  RETURN NODE;
}

NODE MAX(NODE* NODE); // SAME AS MIN BUT MOVE RIGHT.
```

5.
```
SUCCESSOR(NODE*NODE)
  IF (NODE == NULL) RETURN NULL; // BOUNDS CHECK
  IF (NODE.RIGHT != NULL) RETURN MIN(NODE.RIGHT); // CASE #1
```

```
NODE SUCCESSOR (NODE * NODE) {
  IF (NODE == NULL) RETURN NULL;
  IF (NODE.RIGHT != NULL)                    } CASE #1
    RETURN MIN(NODE.RIGHT);                  }
  NODE TEMP = NODE.PARENT;
  WHILE (TEMP != NULL && NODE != TEMP.RIGHT) {
    NODE = TEMP
    TEMP = TEMP.PARENT
  }
  RETURN TEMP
}
```

FIND LOWEST ANCESTOR WHOSE LEFT CHILD IS ALSO AN ANCESTOR OF MINE.