Name - Yogesh Agarwala

EE19B130

Assignment - 4

① Show how to implement a stack using two queues. Analyze the running time of the stack operations.

Sol^n: To construct a stack using two queues (q_1, q_2) we need to simulate the stack operations by using queue operations.

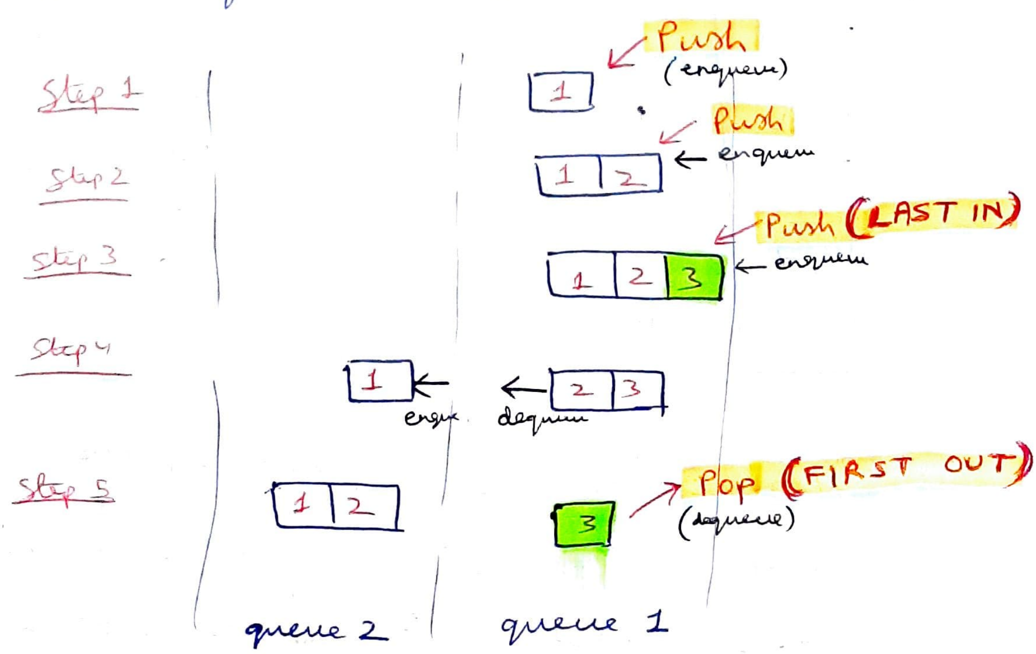
- stack operations
  - Push (insert at top)
  - Pop (remove from top)
- queue operations
  - Enqueue (insert at rear)
  - Dequeue (remove from front)

ⓐ Push : O(1)

Enque each element in queue 1, as we push it.

ⓑ Pop : O(n)

Then to do a pop, we dequeue each element from queue 1 and place it in queue 2, but stop before the last element. Then return the single element left in the original queue (queue 1)

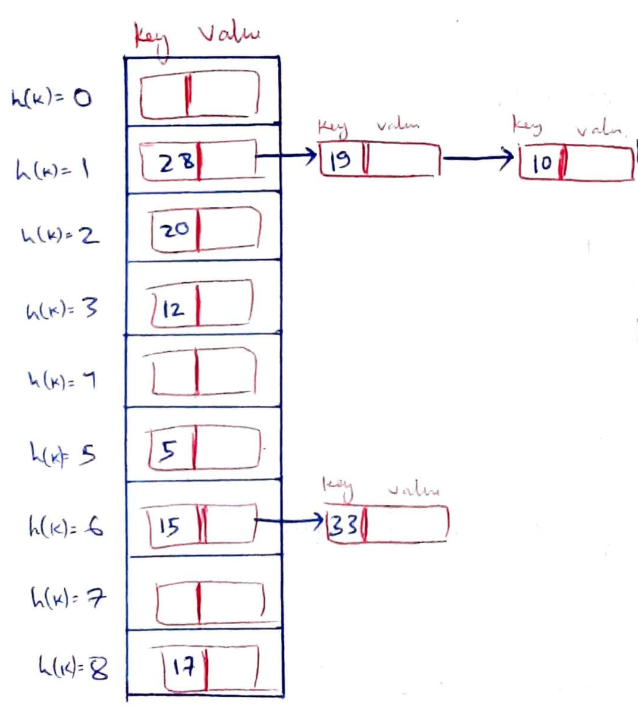


Running time :

Push operation is $O(1)$

but Pop operation is $O(n)$ since we need to dequeue each element from queue 1 to queue 2

② Demonstrate what happens when we insert the keys 5, 28, 19, 15, 20, 33, 12, 17, 10 into a hash table with collisions resolved by chaining. Let the table have 9 slots, and let the hash function be $h(K) = K \bmod 9$



$h(5) = 5 \bmod 9 = 5$

$h(28) = 28 \bmod 9 = 1$

$h(19) = 19 \bmod 9 = 1$

$h(15) = 15 \bmod 9 = 6$

$h(20) = 20 \bmod 9 = 2$

$h(33) = 33 \bmod 9 = 6$

$h(12) = 12 \bmod 9 = 3$

$h(17) = 17 \bmod 9 = 8$

$h(10) = 10 \bmod 9 = 1$

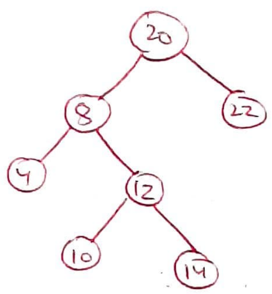
Here collision is resolved by chaining, that is each cell of hash table points to a linked list of records that have the same hash function value.

(3) Consider a binary search tree T whose keys are distinct. Show that if the right subtree of a node x in T is empty and x has a successor y, then y is the lowest ancestor of x whose left child is also an ancestor of x.

Successor → Node with smallest key greater than given node. To find Successor of a node →

→ Leftmost node in the right subtree of the given node, is its successor

e.g successor of 8 is 10
" " 20 is 22
" " 12 is 14



→ If the node don't have right subtree, then successor is the lowest ancestor whose left child is also an ancestor.

To prove

e.g To find succeson of 14

Step 1: lowest ancestor = 12
but its left child = 10
which is not an ancestor of 14
⇒ 12 is not successor

step 2: lowest ancestor = 8
but its left child = 4
which is not ancestor of 14
⇒ 8 is not successor

step 3: lowest ancestor = 20
& its left child = 8
is also ancestor of 14
⇒ 20 is successor of 14

Ancestor → In above tree ancestor of the nodes are given →

4 is 4, 8, 20 but not 12, 22, 10, 14
10 is 10, 12, 8, 20 but not 4, 22, 14
14 is 14, 12, 8, 20 but not 10, 4, 22

Note: Every node is both ancestor & descendant of itself

if y is ancestor of x then x is descendant of y

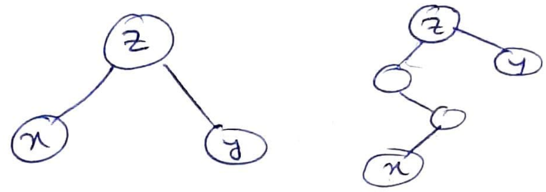
Proof: (1) First we prove that y must be ancestor of x.

To prove y must be ancestor of x

$\boxed{\text{Let } y \text{ is not an ancestor of } x}$

& z denote common ancestor of x and y

(a)

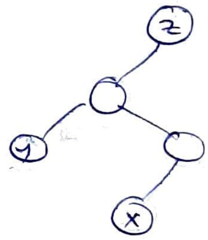


So by BST property, $x < z < y$,

so $\boxed{y \text{ cannot be successor}}$ as it is not the smallest key ~~smaller th~~ greater than x

$\Rightarrow \boxed{y \text{ must be ancestor of } x}$

(b)



In this case $y < x$ so y cannot be succesor

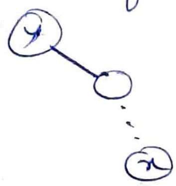
$\Rightarrow \boxed{y \text{ must be ancestor of } x}$

(2) Now $y.left$ must be ancestor of x because if it weren't, then $y.right$ would be ancestor of x $\Rightarrow$ $x > y$

$\Downarrow$

y cannot be successor of x

To prove left child of y must also be ancestor of x



(3) suppose y is not the lowest of x whose left child is also an ancestor of x.

To prove y is the lowest ancestor whose left child is also ancestor.

Let z denotes this lowest ancestor, then z must be in the left subtree of y, which implies $z < y$

also x must be in $z.left$ $\Rightarrow$ $x < z$

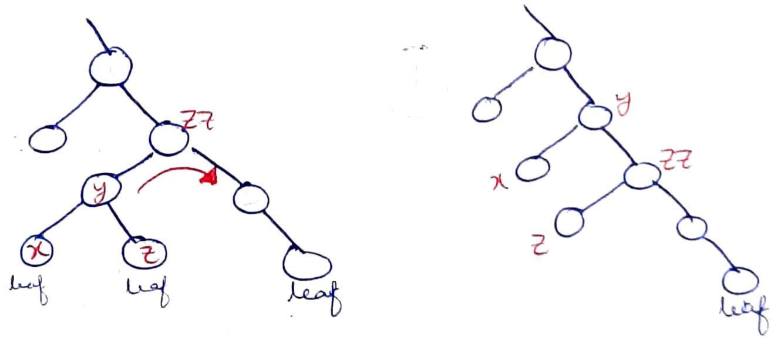
$\rightarrow$ $x < z < y$

So y won't be successor as it is not the smallest key greater than x

(5) Show that any n-node binary tree can be converted to any other n-node binary tree using O(n) rotations

(a) Step-1 : Binary search tree to right-going chain

It takes O(n) rotations to turn an arbitrary binary search tree in to a right-going chain (for all nodes in the tree they only have right child)



From the level above leaf to root, we check
Since leaf don't have any child, so no need to check leaves
if the node have a left child, if yes, then we perform right rotation so the node only have right child, we basically have to check all the nodes except leaves, so it is (n-k) steps, where k is the number of leaves. In worst case there will be only 1 leaf, so it will take (n-1) rotations to convert the BST into right-going chain → O(n) rotations

In other words, start by picking the lowest node on the rightmost chain which has a left ancestor, then perform right rotation to add the node to the right most chain. Now since initially the rightmost chain can contain at least 1 leaf, so we need to perform atmost (n-1) rotations to convert BST to right chain

in above e.g it will be node zz

in above e.g it will be node y

(b) Step-2 : Right-going chain to any arbitrary BST

$$\text{Binary tree 1} \xrightarrow[\underset{O(n)}{\shortparallel}]{r_1, r_2, \ldots, r_k \text{ rotations}} \text{Right-going chain} \xleftarrow[\underset{O(n)}{\shortparallel}]{s_1, s_2, \ldots, s_m \text{ rotations}} \text{Binary tree 2}$$

$$\text{Binary tree 1} \xrightarrow[\underset{O(n) \text{ rotations}}{\shortparallel}]{r_1, r_2, \ldots, r_k \text{ rotation}} \text{Right-going chain} \xrightarrow[\underset{O(n) \text{ rotations}}{\shortparallel}]{s_1', s_2', s_3' \ldots s_m' \text{ rotations}} \text{Binary tree 2}$$

where s_i' is the opposite rotation of s_i

$\therefore$ Total time complexity $= O(n) + O(n)$

$= O(n)$ rotations

Q4 Describe a non-recursive algorithm for enumerating all permutations of the numbers $\{1, 2, \ldots, n\}$ using an explicit stack.

Sol^n **Concept used:**

We can use a stack to reduce the problem to that of enumerating all permutations of the numbers $\{1, 2, \ldots, n-1\}$

Since if we have ~~for~~ all permutations of $\{1, 2, \ldots, n-1\}$ in one stack, we can produce all the permutations of $\{1, 2, \ldots, n\}$ in another stack. Code in next page →

Q.4 Describe a non-recursive algorithm for enumerating all permutations of the numbers {1, 2, ...,n} using an explicit stack

In [1]:

```
"""
Concept: If you have all the permutations of {1,2,...,n-1} in one
stack, then we can produce all the permutations
of {1, 2, . . . , n - 1, n} in another stack

We start with the given list {1,2,....n}, then pop one element fr
om it, so our list will have n-1 numbers.

Base case is when no number is left in the list

"""


def permutations_using_stack(nums):
    stack = []

    for num in nums:
        stack.append(([num], nums-set([num])))

    while len(stack)!= 0:

        """step1"""
        list1, remaining = stack.pop()

        """base-case"""
        if len(remaining) == 0:
            print (list1)

        else:
            """step2"""
            for n in remaining:
                list2 = list1.copy()
                list2.append(n)
                stack.append((list2, nums-set(list2)))
```

In [2]:

```
"""code verification"""
n = 3

nums = {x for x in range(1, n+1)}

permutations_using_stack(nums)
```

```
[3, 2, 1]
[3, 1, 2]
[2, 3, 1]
[2, 1, 3]
[1, 3, 2]
[1, 2, 3]
```