Chapter-2 (20)
Review of Abstract Data Type

1 i) Briefly explain a stack data structure. Write a algorithm to add and remove an element from the stack. and compute the complexity of your algorithm.

Ans: A stack is an ordered collection of homogeneous data elements, where the insertion and deletion takes place at one end, known as the top of the stack.

-> The stack follows LIFO (Las-In-First-Out). It means the ~~item~~ element which is inserted at last must be deleted first.

-> Stack is a non-primitive linear data structure.

Operations performed in stack

1) PUSH: Insertion or adding element into stack.
2) POP: Deletion of element from the stack.
3) IsEmpty: Reports whether the stack is empty or not.
4) IsFull: Reports whether the stack exceeds limit or not.

Applications:

1) Reverse a word 2) Backtracking

Scanned with CamScanner

Algorithm for PUSH (Stack)

Suppose STACK[SIZE] is a one-dimensional array for implementing the stack, which will hold the data items. TOP is the pointer that points to the top most element of the stack. Let DATA is the data item to be pushed.

1. If TOP = SIZE - 1, then: full
   a) Display "The stack is in overflow condition".
   b) EXIT
2. TOP = TOP + 1
3. STACK[TOP] = DATA
4. EXIT

Algorithm for POP (Stack)

Suppose STACK[SIZE] is a one-dimensional array for implementing the stack, which will hold the data items. TOP is the pointer that points to the top most element of the stack. DATA is the popped (or deleted) data item from the top of the stack.

1. If TOP = -1
   a) Display "The stack is empty (underflow)"
   b) EXIT
2. Else remove the top most element
   a) DATA = STACK[TOP]
   b) TOP = TOP - 1
3. EXIT

→ The time complexity of both operation is O(1).

Scanned with CamScanner

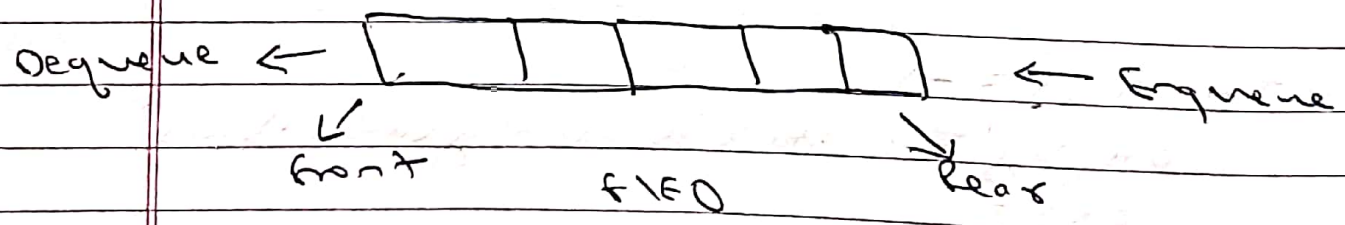
1 3) Briefly explain a Queue data Structure.

-> Queue is a non-primitive linear data structure.
-> It is an homogeneous collection of elements in which new elements are added at one end called the rear end, and the existing elemented are deleted from other end called the front end.
-> It is also called FIFO (First-In-First-Out) because the first added element will be the first to be removed from the queue.
-> Insertion and deletion happens on different end.
-> In queue every insert operation rear is incremented by one.

$$R = R + 1$$

-> and every deleted operation front is increased by one.

$$F = F + 1$$



-> Basic Operations in Queue
i) enqueue (): add an element at the rear end of the queue.
ii) dequeue (): remove an element from front end of the queue.

Scanned with CamScanner

(Not asked in exam)

Linear Queue

Implementation of Enqueue

```
int enqueue (int data)
{
    if (isfull())
        return 0;
    rear = rear + 1;
    queue[rear] = data;
    return 1;
}
```

Implementation of Dequeue

```
int dequeue ()
{
    if (isEmpty())
    {
        return 0;
    }
    int data = queue[front];
    front = front+1;
    return data;
}
```

→ The worst case time complexity of enqueue and dequeue operation is O(1).

Scanned with CamScanner

classmate
Date
Page

III 3) Briefly explain a circular queue. Write algorithm to add and remove an element from the circular queue and compute the complexity of your algorithm.

Ans: A circular queue is one in which the insertion of a new element is done at very first location of the queue if the last location of the queue is full.

→ If we delete an ~~item~~ element from the linear queue then empty space is created there, but we cannot insert any new element into this empty space because in a linear queue the insertion has to be done from one side and deletion has to be done from another side.

→ In circular Queue, once the queue is full the first element of the queue becomes the rear most element if front has move forward otherwise overflow occurs.

→ It will start overwriting the items from beginning so that new item get inserted.

→ Simply in circular queue the front and rear are adjacent to each other that means the insertion and deletion point is same. So if any element is deleted from the circular queue we can insert

Scanned with CamScanner

a new element to this position.

In this way, a circular queue overcomes the problem of unutilized space in linear queue.

→ The operations performed in circular queue are:-
i) enqueue() : inserting an element into the queue
ii) dequeue() : deleting an element from the queue.

P.T.O →

→ The time complexity of enqueue() & dequeue() in circular queue is O(1).

Scanned with CamScanner

Algorithm for insertion (Enqueue)

Step 1: Check if the queue is full
if (front == (Rear+1) % MaxSize)
display "Queue is full"
EXIT

Step 2: Check if the queue is empty, and set both front and rear to 0
if (front == -1)
set ~~front~~ front = 0
Rear = 0

Step 3: If Rear = MaxSize-1 && front != 0
(rear pointer is at the end of the queue and front is not at the 0th index)
then, set Rear = 0

Step 4: Otherwise, set Rear = (Rear+1) % MaxSize

Step 5: Inser an element into the queue
Queue[Rear] = value

Step 6: EXIT

Scanned with CamScanner

Algorithm for deletion (Dequeue)

Step 1: Check if the queue is empty
if (Front = -1 & Rear = -1)
display "Queue is empty"

Step 2: Otherwise,
item = Queue [front]

Step 3: If there is only one element in a queue,
set both front and rear to -1
if (front == Rear)
set front = -1
set Rear = -1

Step 4: And if front = MaxSize - 1
set front = 0

Step 5: Otherwise,
set front = front 1

Step 6: EXIT

Scanned with CamScanner

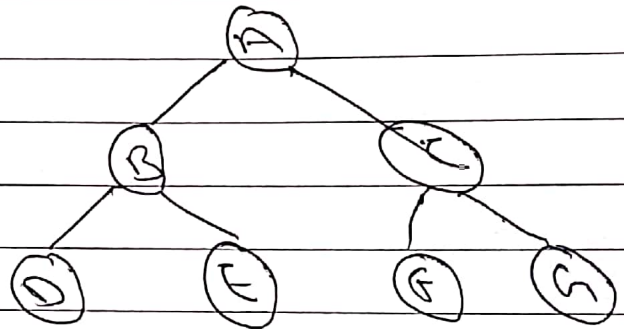
4) Define Binary Tree

-> It is a tree where :-

i) Each node can have atmost 2 childrens

ii) A node can have 0,1,2 subtrees, these subtrees are designated as left subtree & right subtree.

Eg.



(Total 7 questions)

15) Binary Search Tree (BST)

or

1 Write an algorithm to insert & delete an item from binary search tree. Also compare those operations in terms of their time and space complexity.

or

Write an algorithm for binary Search Tree

or

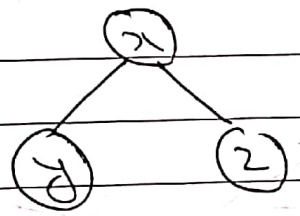
1 Devise an algorithm to delete an element x from a binary tree T. What is the time complexity of your algorithm.

Scanned with CamScanner

Ans: In BST, nodes left child must have values less than it & nodes right child must have values greater than it.

-> BST is a node-based binary tree in which nodes are assigned values with some restrictions like
1) no duplication of value
2) leftsubtree < root < right subtree
3) The left & right subtree of a node is also a BST.

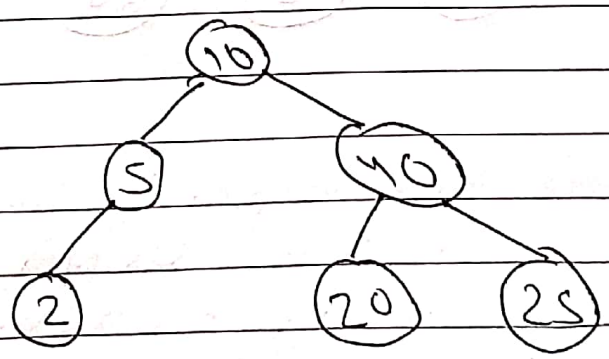
Example 1: smaller value in left subtree
larger value in right subtree



$x > y$

$z > x$

Example 2:

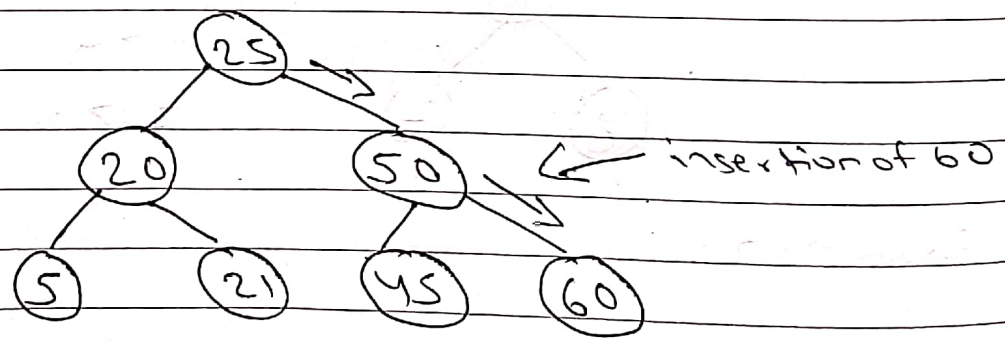


Scanned with CamScanner

Algorithm for Insertion in BST

Step 1: Always insert new node as leaf node
Step 2: Start at root node as current node
Step 3: If new node's value < current value
  Step 3.1: If current node has a left child, search left
  Step 3.2: else add new node as current's left child
Step 4: If new node's value > current value
  Step 4.1: If current node has a right child, search right.
  Step 4.2: Else add new node as current right node.

Example: Insert 60 in the tree



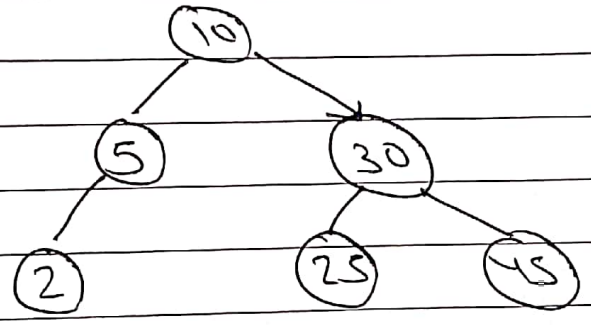
→ The worst case time complexity for insertion & deletion from BST is O(N)
→ The space complexity for both operation is also O(N).

Scanned with CamScanner

Algorithm of Deletion from BST

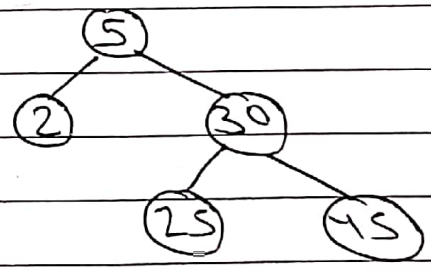
Step 1: Perform search for value x
Step 2: If x is a leaf, delete x
Step 3: Else //internal node has to be deleted
  Step 3.1: ~~replace~~ Delete internal node x and replace it with largest value y from left subtree or smallest value z from right subtree
  Step 3.2: delete replacement value (y or z) from subtree.

Example:

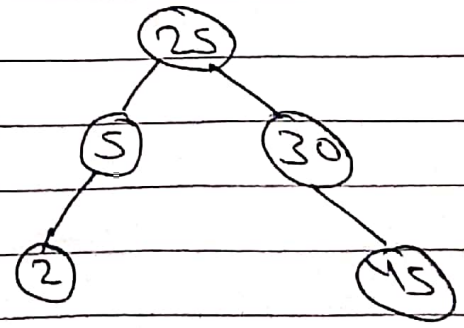


→ ~~Replacing~~ Delete 10

→ Replacing largest value from left subtree i.e 5 (inorder predecessor)



→ Replacing smallest value from right subtree i.e 25 (inorder sucessor



Scanned with CamScanner

Date
Page

6) Why recursion and iteration is used in an algorithm. Explain its use.

1) Iteration

-> Iterative algorithm will use looping statements such as for loop, while loop or do-while loop to repeat the same steps.

-> An iteration terminate while the loop condition fails.

-> An iteration doesnot use stack so it's faster than recursion.

-> It consumes less memory but the code is longer

2) Recursion

-> In Recursive algorithm, a function calls itself again and again till the base condition (stopping condition) is satisfied.

-> it is used to solve complicated problems when its application is easy and effective.

-> it is slower than iteration due to the overhead of maintaing the stack.

-> It uses more memory than iteration

-> Recursion makes the code smaller.

Scanned with CamScanner

7) Recurrence Relation

-> A recurrence is an equation or inequality that describes a function in terms of its value on smaller inputs.

-> To solve a recurrence relation means to obtain a function defined on the natural numbers that satisfy the recurrence.

-> Recurrence relations are very useful in calculating the running time complexities of recursive algorithms.

There are 3 methods for solving Recurrence

i) Substitution Method

ii) Recursion Tree Method

iii) Master Method

Scanned with CamScanner

3) Explain the recursion tree method for solving the recurrence relations

Ans. In this method, we draw a recurrence tree and calculate the time taken by every level of tree. Finally we sum the work done at all levels. To draw the recurrence tree, we start from the given recurrence and keep drawing till we find a pattern among levels.
The pattern is typically a arithmetic or geometric series.

Scanned with CamScanner

9) Solve the following recurrence relations using this method.

i) $T(n) = T(n/3) + T(2n/3) + n$

ii) $T(n) = T(n/4) + T(n/2) + cn^2$

Scanned with CamScanner

1 10) Explain the Master method for solving the recurrence relations.

Ans. Master method is a direct way to get the solution of recurrence relations.

The Master method is used for solving following types of recurrence.

$$\boxed{T(n) = aT\left(\frac{n}{b}\right) + f(n)}$$

with $a \geq 1$ and $b \geq 1$ be constant
$f(n)$ is a function
n is the size of the problem.

$f(n) = \Theta(n^k \log^p n)$ $\quad k \geq 0$

① $\log^a b$ ② k

Case 1: if $\log^a b > k$ then $\Theta(n^{\log^a b})$

case 2: if $\log^a b = k$ then
if $p > -1$ $\Theta(n^k \log^{p+1} n)$
if $p = -1$ $\Theta(n^k \log \log n)$
if $p < -1$ $\Theta(n^k)$

Case 3: if $\log^a b < k$
if $p \geq 0$ $\Theta(n^k \log^p n)$
if $p < 0$ $\Theta(n^k)$

Scanned with CamScanner

4) Solve the following recurrence relations using Master's method

i) $T(n) = 2T(n/4) + \sqrt{n}$

ii) $T(n) = 3T(n/2) + n$

iii) $T(n) = 4T(n/3) + n^2$

i) $T(n) = 2T(n/4) + \sqrt{n}$

i.e $T(n) = 2T(n/4) + n^{1/2}$

comparing with

$$T(n) = aT\left(\frac{n}{b}\right) + \theta(n^k \log^p n)$$

$\therefore a = 2 \quad b = 4 \quad k = \frac{1}{2} \quad p = 0$

Calculating

$\log_b a = \frac{1}{2}$

Here, $\log_b a = 1$ and $p > -1$

$\therefore \theta(n^k \log^{p+1} n)$

or, $\theta(n^{1/2} \log^1 n)$

$\therefore \theta(\sqrt{n} \log n)$

Scanned with CamScanner

1 iii) $T(n) = 3T(\frac{n}{2}) + n$

Comparing with

$$T(n) = aT(\frac{n}{b}) + \theta(n^k \log^p n)$$

$\therefore a = 3 \quad \therefore b = 2 \quad k = 1 \quad p = 0$

Calculating

$\log_b a = \log_2 3 = 1.585$

$\log_b a > k$

then $\theta(n^{\log_b a})$

$\therefore \theta(n^{1.585})$ Ans

Scanned with CamScanner

iii) $T(n) = 4T(n/3) + n^2$

comparing with

$$T(n) = aT\left(\frac{n}{b}\right) + \theta(n^k \log^p n)$$

$a = 4 \quad b = 3 \quad k = 2 \quad p = 0$

Calculate

$\log_b^a = \log_3^4 = 1.262$

$\therefore (\log_b^a > k)$

Then $\theta(n^{\log_b^a})$

$\therefore \theta(n^{1.262})$ Ans