Assignment

Name :- Tanishq Chauhan

Roll No :- 21C3184

Subject :- Data Structure (CER3C3)

Semester :- 3rd Semester

Branch :- Computer Science & Engineering (CS-B)

Chauhan

Tanishq Chauhan
21C3184
CS-B

Chauhan
Date
Page 01

Data Structure (CER3C3)

Assignment

Q.1. What is Abstract Data Type? What is it's role in Data Structure? You must be knowing String data structure? Try to write ADT definition for String data type? Assume functions by your own.

Answer of Q. No. 1

Abstract Data Type

- An abstract data type (ADT) refers to a set of data values and associated operations that are specified accurately, ~~it~~ independent of any particular implementation.
- With an ADT, we know what a specific data type can do, but how it actually does it ~~it~~ is hidden.
- In broader terms, the ADT consists of a set of definitions that allow us to use the functions while hiding the implementation.

- An ADT is a mathematical model of a data structure that specifies the type of data stored, the operations supported on them, and the types of parameters of the operations.
- An ADT specifies what each operation does, but not how it does it.
- Typically, an ADT can be implemented using one of many different data structure, like array and linked-list.

Tanishq Chauhan
21C3184
CS-B

- A useful first step in deciding what data structure to use in a program is to specify an ADT for the program.
- In general, the steps of building ADT to data structure are:

(i) Understand and clarify the nature of the target information unit.
(ii) Identify and determine which data objects and operations to include in the model.
(iii) Express this property somewhat formally so that it can be understood and communicate well.
(iv) Translate this formal specification into proper language.
(v) In C++, this becomes a .h file. In Java, this is called "user interface".
(vi) Upon finalized specification, write necessary implementation.
(vii) This includes storage scheme and operational detail.
(viii) Operational detail is expressed as separate function (methods).

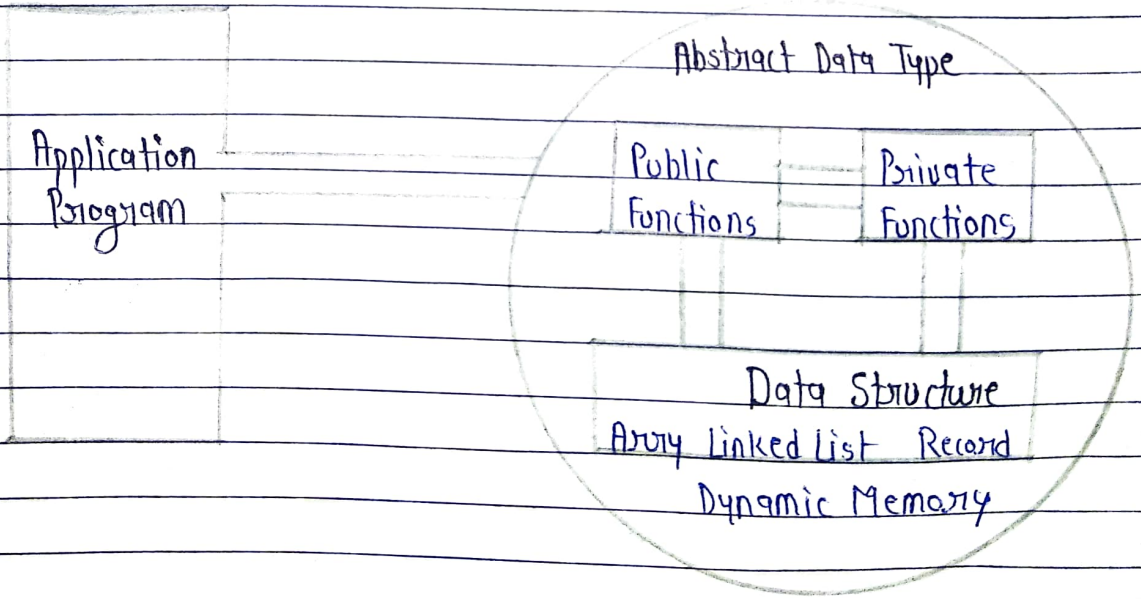


Figure 1.1 : ADT Model

Tanishq Chauhan
21C3184
CS-B

String :- Strings are defined as an array of characters. The difference between a character array and a string is the string is terminated with a special character '\0'.
Declaring a string is as simple as declaring a one dimensional array.

Strings as ADT

Most languages have strings as a built-in data type or a standard library, and a set of operations defined on ~~what~~ that type. Therefore, there is actually no need for us to create our own string ADT.

However, we can implement our own string data type, if required. The use of a data type in string processing applications should not depend on how it is implemented or whether it is built-in or user-defined. We only need to know what operations are allowed on the string.

Examples Operations :-

length : A function that returns the current length of the string

```
abstract typedef <<char>> STRING;

abstract length(s)
STRING s;
postcondition length == len(s);
```

Tanishq Chauhan
21C3184
CS-B

Chauhan
Date
Page 04

(ii) concat : a function that returns the concatenation of its two input strings.

```
abstract typedef <<char>> STRING;

abstract STRING concat (S1, S2)
STRING S1, S2;
postcondition  concat == S1 + S2;
```

(iii) substr : A function that returns a substring of a given string.

```
abstract STRING substr (S1, i, j)
STRING S1;
int i, j;

precondition  0 <= i < len (S1);
              0 <= j < len (S1) - i;
postcondition substr == sub (S1, i, j);
```

Tanishq Chauhan
21C3184
CS-B

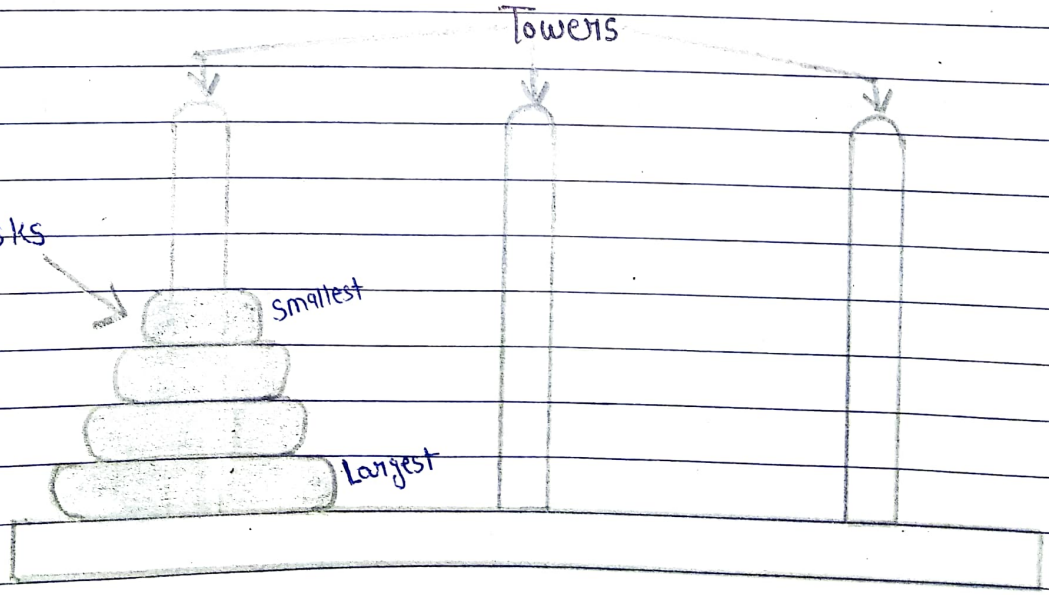
Q.2. What is Tower and Hanoi Problem? Try to enumerate a solution with 4 disks.

Answer of Q.No.2

Tower of Hanoi

Tower of Hanoi is a mathematical puzzle where we have three rods and n disks. The objective of the puzzle is to move the entire stack to another rod, obeying the following simple rules:

(i) Only one disk can be moved at a time.
(ii) Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack i.e. a disk can only be moved if it is the uppermost disk on a stack.
(iii) No disk may be placed on top of a smaller disk.



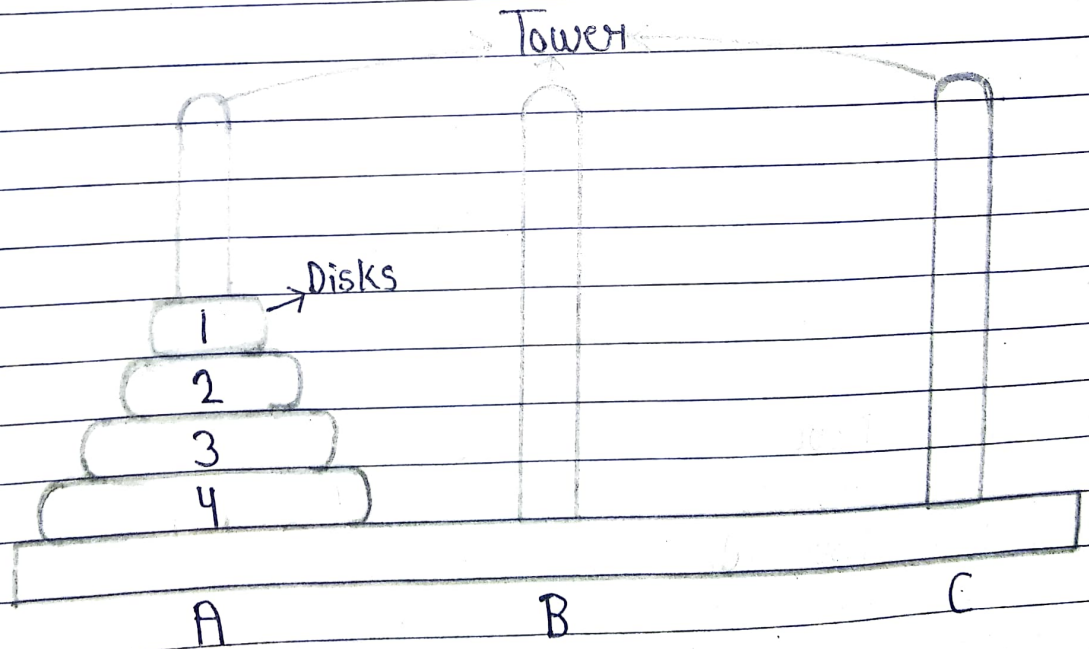
Tanishq Chauhan
21C3184
CS-B

- These rings are of different sizes and stacked upon in an ascending order, i.e. the smaller one sits over the larger one.
- There are other variations of the puzzle where the number of disks increase, but the tower count remain the same.

Enumeration of a Solution with 4 Disks.

- We have 3 tower and 4 disks named 1, 2, 3, 4 where 4 is the largest disk and 1 is the smallest disk.
- All 4 disks are placed in tower A and our task is to move all 4 disks to pole/tower C.

Initial Condition



Tanishq Chauhan
21C3184
CS-B

Chauhan
Date
Page 07

Step 1 : Move disk '1' from rod A to rod B

Step 2 : Move disk '2' from rod A to rod C

Step 3 : Move disk '1' from rod B to rod C

Step 4 : Move disk '3' from rod A to rod B

Step 5 : Move disk '1' from rod C to rod A

Step 6 : Move disk '2' from rod C to rod B

Step 7 : Move disk '1' from rod A to rod B

Step 8 : Move disk '4' from rod A to rod C

Step 9 : Move disk '1' from rod B to rod C

Step 10 : Move disk '2' from rod B to rod A

Step 11 : Move disk '1' from rod C to rod A

Step 12 : Move disk '3' from rod B to rod C

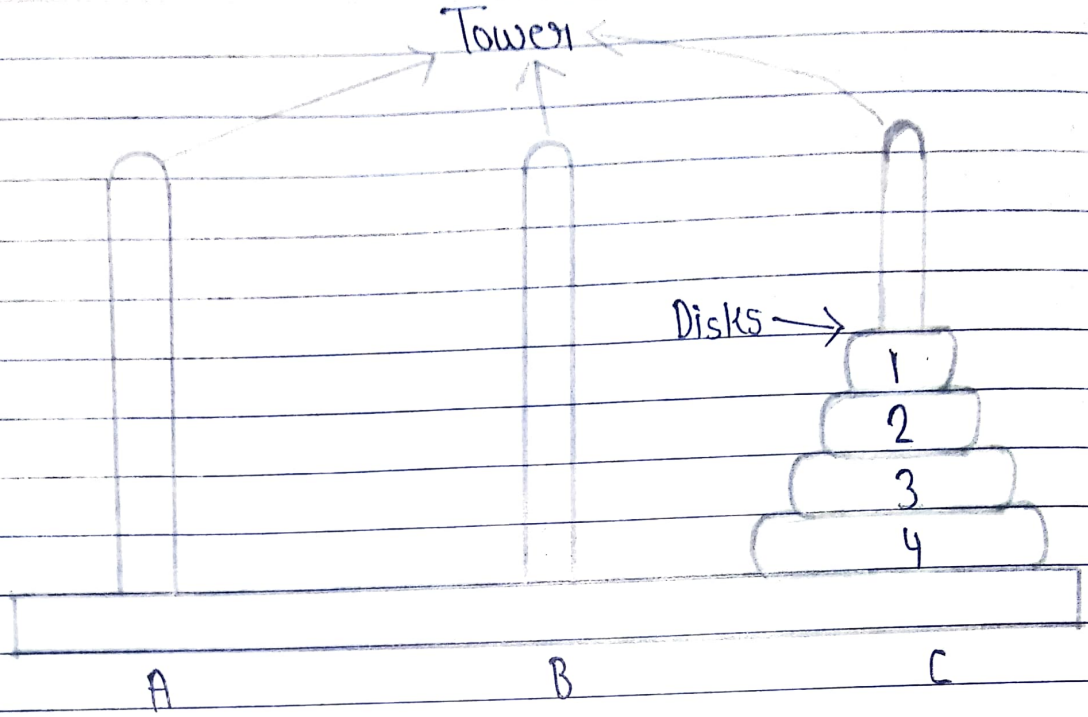
Step 13 : Move disk '1' from rod A to rod B

Step 14 : Move disk '2' from rod A to rod C

Step 15 : Move disk '1' from rod B to rod C

Tanishq Chauhan
21C3184
CS-B

Chauhan
08



After 15 moves all the disks placed to rod C from rod A

Tanishq Chauhan
21C3184
CS-B

Chauhan
Date
Page 09

Q.3. How would you implement two queues using a single array ? Write enqueue() and dequeue () function for the same.

Answer of Q. No. 3

- Create a data structure kQueues that represents k queue. Implementation of kQueues should use only one array i.e, k queues should use the same array for storing elements.
- Following functions must be supported by kQueues

enqueue (int x, int qn) → adds x to queue number 'qn' where qn is from 0 to k-1.

dequeue (int qn) → deletes an element from queue number 'qn' where qn is from 0 to k-1.

~~Method~~

We have two methods to implement two queues using a single array :-

Method 1 (Divide the array in slots of size n/k)

- A simple way to implement k queues is to divide the array in k slots of size n/k each, and fix the slots for different queues, i.e, use arr[0] to arr[n/k-1] for the first queue, and arr[n/k] to arr[2n/k-1] for queue 2 where arr[] is the array to be used to implement two queues and size of array be n.

Tanishq Chauhan
21C3184
CS-B

- The problem with this method is an inefficient use of array space.
An enqueue operation may result in overflow even if there is space available in arr[]
- For Example, consider K as 2 and array size n as 6. Let we enqueue 3 elements to first and do not enqueue anything to the second queue. When we enqueue the 4th element to the first queue, there will be overflow even if we have space for 3 more elements in the array.

Method 2 (A space efficient implementation)

- The idea is similar to the stack post, here we need to use three extra arrays.
- The more array is required because in queues, enqueue() and dequeue() operations are done at different ends.

Following are the three extra arrays are used:

(i) front[] : This is of size k and stores indexes of front elements in all queues.

(ii) rear[] : This is of size k and stores indexes of real elements in all queues.

(iii) next[] : This is of size n and stores indexes of next item for all items in array arr[].

Tanishq Chauhan
21C3184
CS-B

Chauhan
Date
Page 11

Here arr[] is the actual array that stores k stacks. Together with k queues, a stack of free slots in arr[] is also maintained. The top of this stack is stored in a variable 'free'.
All entries in front[] are initialized as -1 to indicate that all queues are empty. All entries next[i] are initialized as i+1 because all slots are free initially and pointing to the next slot.
Top of the free stack, 'free' is initialized as 0.

* enqueue() Function

```
// To enqueue an item in queue number 'qn' where qn is
from 0 to k-1

enqueue (int item, int qn)

{
    // Overflow check

    if (isFull())

    {
        cout << "\n Queue Overflow \n";
        return;
    }

    int i = free; // Store index of first free slot
// Update index of free slot to index of next slot in free list
free = next[i];
```

```
if (isEmpty (qn))
        front [qn] = i;
else
    next [rear[qn]] = i;

next [i] = -1;

// Update next of rear and then rear for queue
   number 'qn'

rear [qn] = i;

// Put the item in array
   arr [i] = item;
}
```

* dequeue() Function

// To dequeue an from queue number 'qn' where qn is from 0 to k-1.

```
dequeue (int qn)

{
    // Underflow Check
    if (isEmpty (qn))
    {
        cout << "In Queue Underflow In";
        return INT_MAX;
    }
```

Tanishq Chauhan
21C3184
CS-B

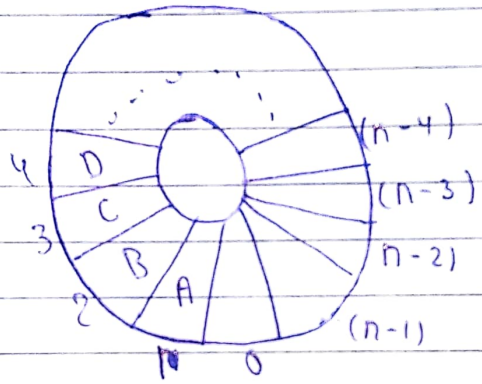

```
// Find the index of front item in queue number 'qn'
int i = front[qn];

front[qn] = next[i]; // Change top to store next
                        of previous top

// Attach the previous front to the beginning of free list

next[i] = free;
free = i;

// Return the previous front item

return arr[i];

}
```

Tanishq Chauhan
21C3184
CS-B

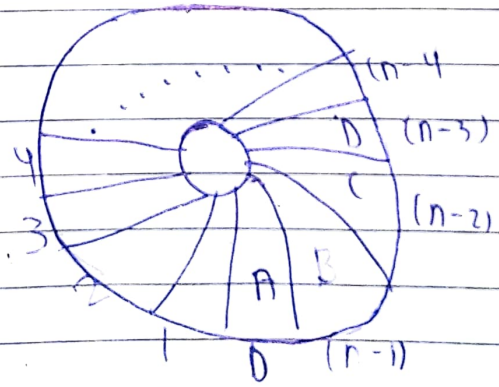
Q.4. A general circular queue supports n-1 elements. How would you allow it to store n elements.

Answer of Q. No. 4

For better understanding let consider two figures:



front = 0 rear = 4



front = n-4 rear = 0

Tanishq Chauhan
21C3184
CS-B

Chauhan
Date
Page 15

In both the algorithms, condition for queue full in ADD Q and the condition for queue empty in DELETEQ are the same.

In the case of ADDQ, however, when front = rear there is one space free i.e. Q(rear), since the first element in the queue is not at Q(front) but is one position clockwise from this point.

However, if we insert an item here, then we will not be able to distinguish between the cases full and empty, since ~~the~~ this insertion would leave front == rear.

Method for Storing n element :-

For storing n elements rather than n-1 we should avoid signal queue full. One way to use all n positions would be to use another variable tag to distinguish between the two situation.

If tag = 0 then we print the message queue is empty.

Tanishq Chauhan
2103184
CS-B

Chauhan

Date
Page 16

procedure ADDQ (item, Q, n, front, rear)

//insert item in the circular queue stored in Q(0:n-1); rear points to the last item and front is one position counterclockwise from the first item in Q//

```
rear ← (rear + 1) mod n   //advance rear clockwise//
if front = rear then call QUEUE-FULL
Q(rear) ← item   //insert new item//
end ADDQ
```

procedure DELETEQ (item, Q, n, front, rear)

//removes the front element of the queue Q(0:n-1)//

```
if front = rear then call QUEUE-EMPTY
front ← (front + 1) mod n   //advance front clockwise//
item ← Q(front)   //set item to front of queue//
end DELETEQ
```

Tanishq Chauhan
21C3184
CS-B

Q.5. What are balanced trees ? Why are they require ? Take an example of each rotation to explain the process.

Answer of Q. No. 5.

Balanced Trees

- A balanced tree which is also known as Height Balanced Tree. It is defined as binary tree in which the difference between the height of the left sub-tree and right sub-tree is not more than 1.
- The height of a tree is the number of edges on the longest path between the root node and the leaf node.

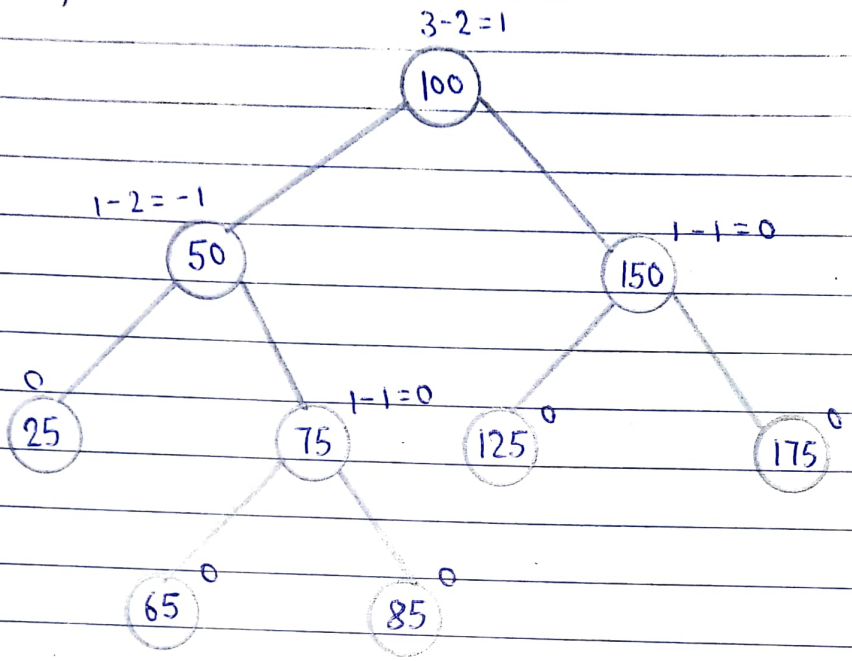
Balance Factor = height (left subtree) - Hight (right subtree)

- Tree is said to be balanced if balance factor of each node is in between -1 to 1, otherwise the tree will be unbalanced and need to be balanced.
- If balance factor of any node is 1, it means that the left sub-tree is one level higher than the right sub-tree.
- If balance factor of any node is 0, it means left subtree and right sub-tree contain equal height.
- If balance factor of any node is -1, it means that the left sub-tree is one level lower than the right sub-tree.

Tanishq Chauhan
21C3184
CS-B

Chauhan
Date
Page 18

For Example :-



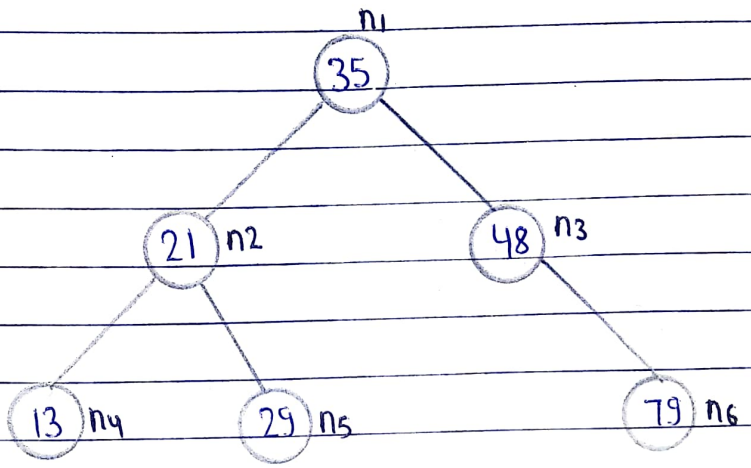
Balance Factor of 100 = 1
Balance Factor of 50 = -1
Balance Factor of 25 = 0
Balance Factor of 75 = 0
Balance Factor of 65 = 0
Balance Factor of 85 = 0
Balance Factor of 150 = 0
Balance Factor of 125 = 0
Balance Factor of 175 = 0

Requirement of Balanced Tree :-

Let's understand the need for a balanced binary tree through an example :-

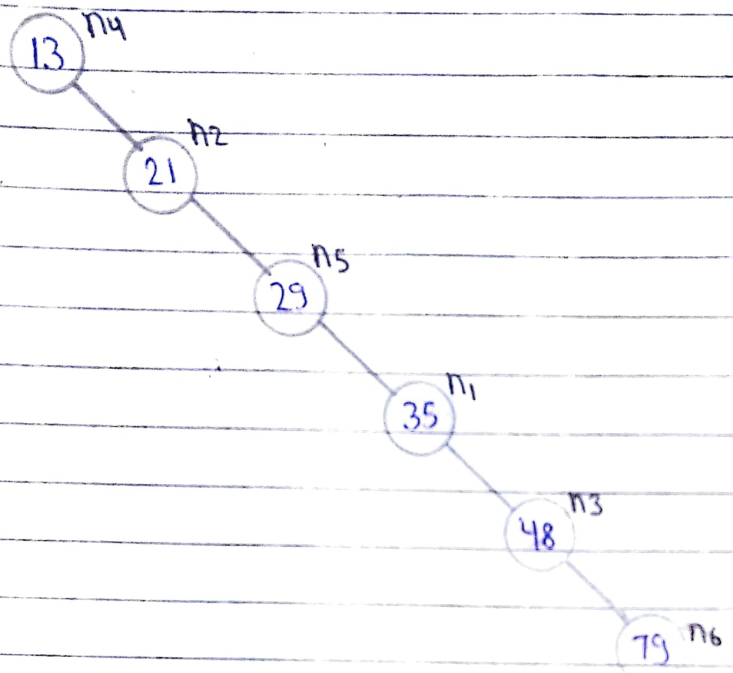
Tanishq Chauhan
21C3184
CS-B

Chauhan
Date
Page 19



- The above tree is a binary search tree because all the left sub-tree nodes are smaller than its parent node and all the right subtree nodes are greater than the parent node.
- Suppose we want to find the value 79 in the above tree.
- First, we compare the value of node n_1 with 79.
- Since the value of 79 is not equal to 35 and it is greater than 35 so we move to the node n_3, i.e., 48.
- Since the value of 79 is not equal to 48 and 79 is greater than 48, so we move to the right child of 48.
- The value of the right child of node 48 is 79 which is equal to the value to be searched.
- The number of hops required to search an element 79 is 2 and the maximum number of hops required to search any element is 2.
- The average case to search an element is $O(\log n)$.

Tanishq Chauhan
21C3184
CS-B



- The above tree is also a binary search tree ~~because~~ but in this scenario if we want to find 79, the number of hops required is 5.
- It is the worst-case scenario, O(n).
So it is required to use Hight-Balanced Tree.

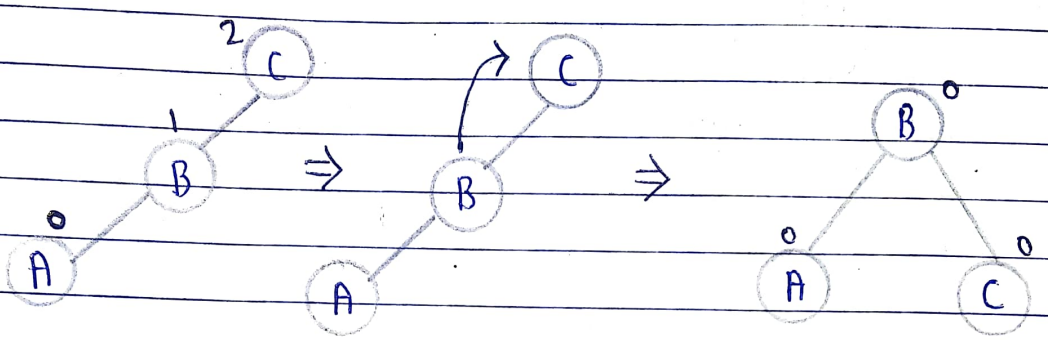
Rotations in Height Balanced Tree

We perform rotations only in case of balance factor is other than -1, 0, 1.
There are basically 4 types of rotation :-

(i) LL Rotation
(ii) RR Rotation
(iii) LR Rotation
(iv) RL Rotation

Tanishq Chauhan
21C3184
CS-B

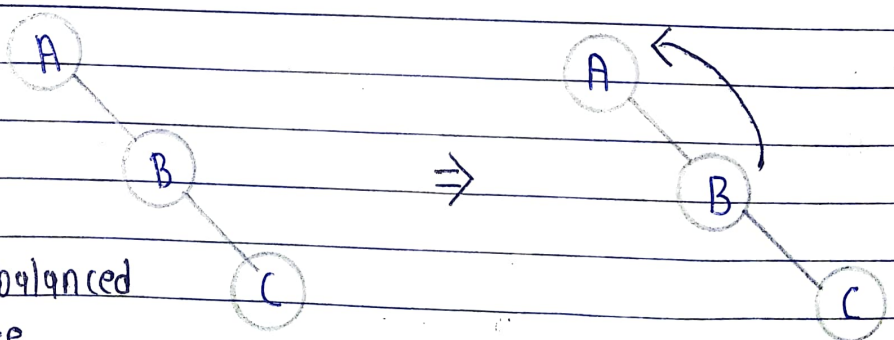
(i) LL Rotation :- When BST becomes unbalanced, due to a node is inserted into the left sub-tree of the left subtree, then we perform LL rotation. It is a clockwise rotation.



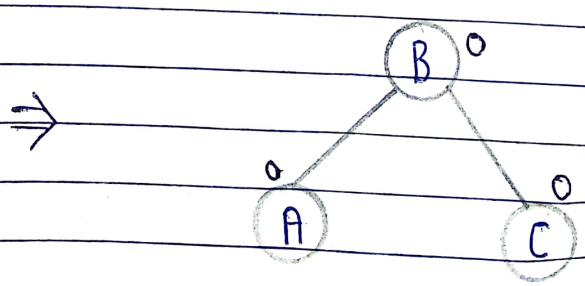
Left-Unbalanced Tree

Balanced Tree

(ii) RR Rotation :- When BST becomes unbalanced, due to a node is inserted into the right sub-tree of the right-subtree, then we perform RR rotation. It is anti-clockwise rotation.



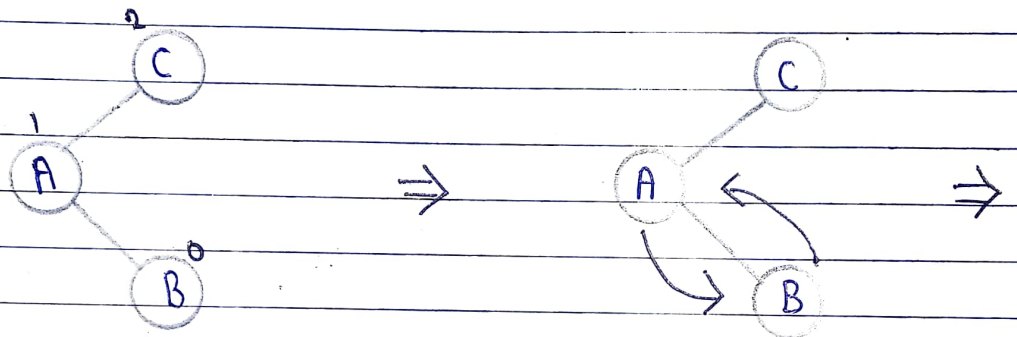
Right Unbalanced Tree

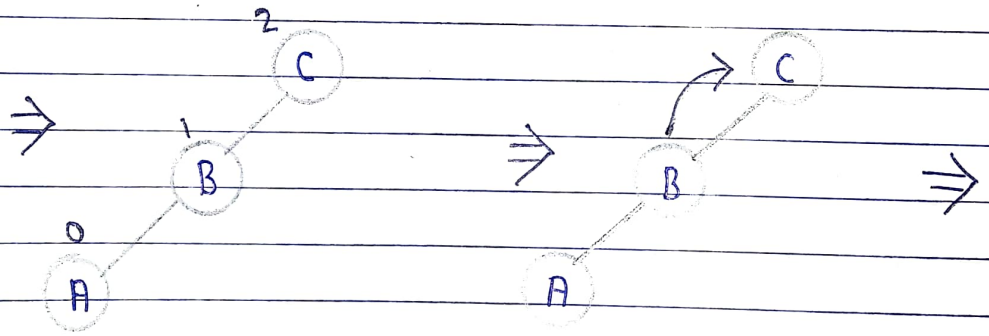


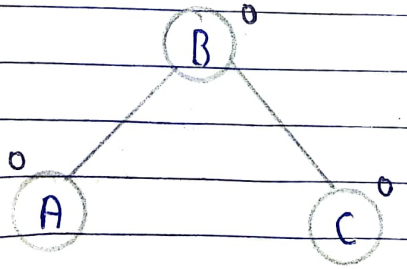
Balanced Tree

Tanishq Chauhan
21C3184
CS-B

(iii) LR Rotation :- Double rotation are bit tougher than single rotation. LR rotation = RR rotation + LL rotation, first RR rotation is performed on subtree and then LL rotation is performed on full tree, by full tree we mean the first nodes from the path of inserted node whose balance factor is other than -1, 0, or 1.



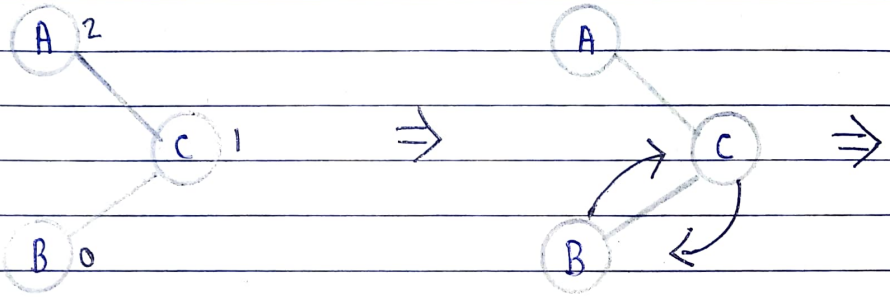


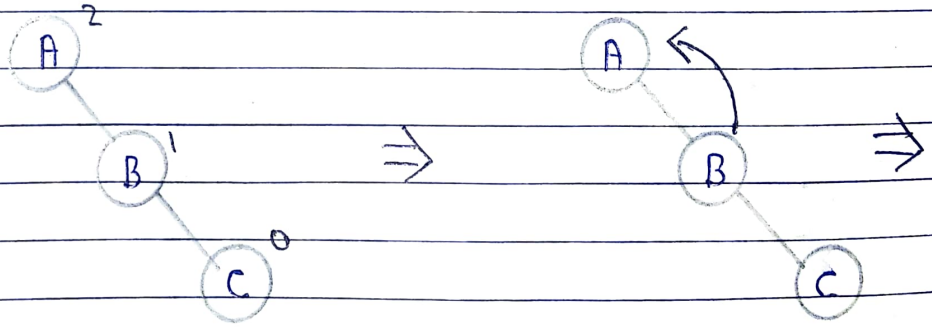


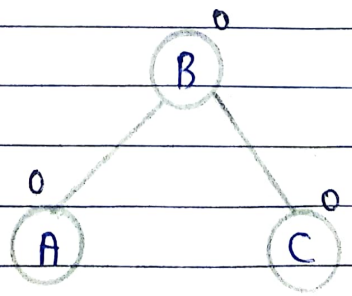
Balanced Tree

Tanishq Chauhan
21C3184
CS-B

(iv) RL Rotation :- As already discussed, that double rotations are bit tougher than single rotation which has already explained above. RL rotation = LL rotation + RR rotation, i-e first LL rotation is performed on subtree and then RR rotation is performed on full tree, by full tree we mean the first node from the path of ~~intsr~~ inserted node whose balance factor is other than -1, 0 or 1.







Balanced Tree

Tanishq Chauhan
2103184
CS-B

Chauhan

Date
Page 24

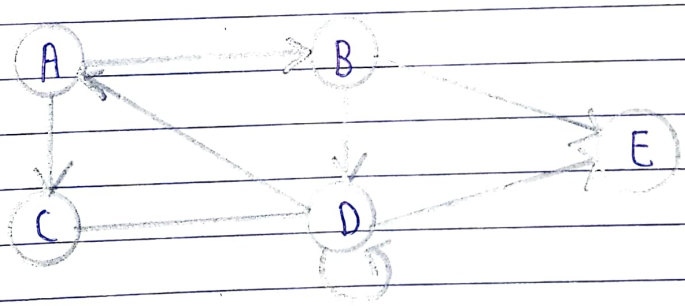
Q.6. What is directed graph ? How would you implement it using adjacency graph and list. Take an example to explain.

Answer of Q. No. 6

Directed Graph

A directed graph which is also known as digraph in which each edge of the graph is associated with some direction and the traversing can be done only in it specified direction.

Example of the directed graph :-



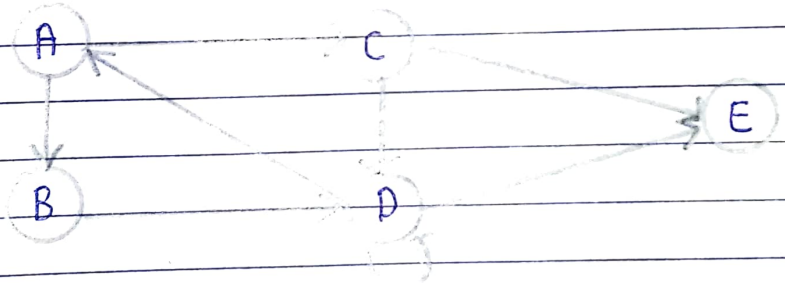
There are 2 ways to implement directed graph :-

(i) Adjacency Matrix
(ii) Adjacency List

Tanisha Chauhan
2103184
CS-B

(i) Adjacency Matrix :-

- It is a sequential representation.
- It is used to represent which nodes are adjacent to each other i.e. is there any edge connecting nodes to graph.
- In this representation, we have to construct a n*n matrix. If there is any edge from vertex i to vertex j, then the corresponding element $a^{ij} = 1$, otherwise $a^{ij} = 0$.



	A	B	C	D	E
A	0	1	1	0	0
B	0	0	0	1	0
C	0	0	0	1	1
D	1	0	0	1	0
E	0	0	0	0	0

Tanishq Chauhan
2103184
CS-B

(ii) Adjacency List

- It is a linked representation.
- In this representation, for each vertex in graph, we maintain the list of its neighbours. It means every vertex of the graph contains list of its adjacent vertices.
- We have an array of vertices which is indexed by vertex number, the corresponding array elements points to a singly linked list of neighbours of v.