Experiment No. 10

Problem Statement:
Read the marks obtained by students of second year in an online examination of particular subject. Find out maximum and minimum marks obtained in that subject. Use heap data structure. Analyze the algorithm.

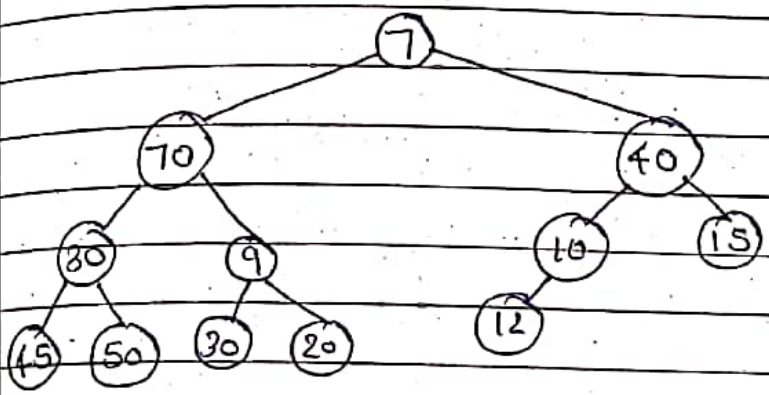
Objective:
To understand the basic concepts of Heap Data structure

Outcome: To implement the concept and basic of MaxHeap and MinHeap Data structure and its use in Data structure.

Theory
Concepts:
A double ended priority queue is a data structure that support the following operation
1. Inserting an element with an arbitrary key.
2. Deleting an element with largest key.
3. Deleting an element with smallest key.

Definition: A min-max heap is a complete binary tree such that if it is not empty. each element has a data member called key.

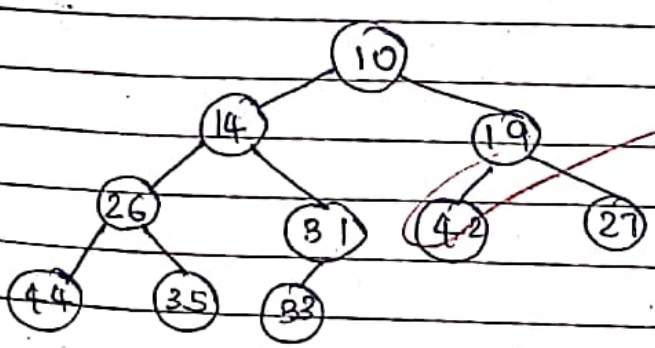
PAGE NO.
DATE:



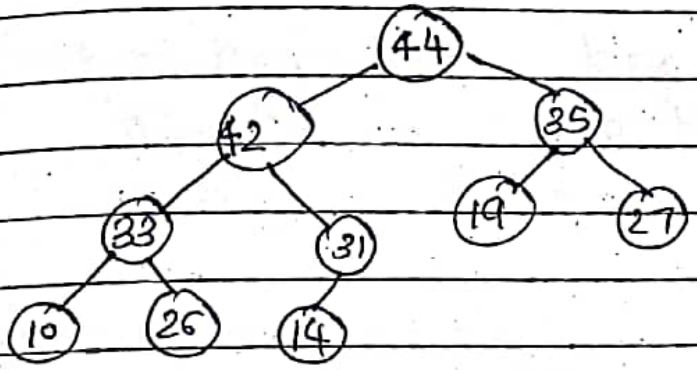
min
max
min
max

```
template <class KeyType>
class DEPQ{
public:
    virtual void Insert(const Element<KeyType>&)=0;
    virtual Element<KeyType>* DeleteMax(Element<KeyType>&)
    virtual Element<KeyType>* DeleteMin(Element<KeyType>&)=
};
```

MinHeap : where the value of the root node is less than or equal to either of its children



Max - Heap - where the value of the root node is greater than or equal to either of its children.



Maxheap Construction.

1- Create a new node at the end of heap.
2- Assign new value to the node
3- Compare the value of this child node with its parent
4- If value of parent is less than child, then swap.
5- Repeat step 3 & 4 until Heap property holds.

Delete:

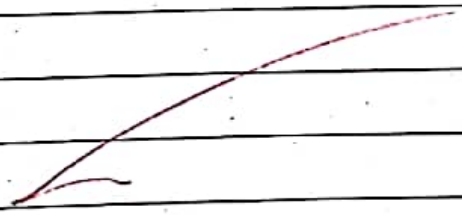
Delete the minimum element

1. Remove the root node and the node which is at the end of heap. Let it be x.
2. Reinsert key of x into the min-max heap
3. Reinsertion have 2 cases.
   1. If root has no children, then x can be inserted into the root.
   2. Suppose root has at least one child.
      1. x.key <= h[m].key : x must be inserted into the root.
      2. x.key > h[m].key and m is child of the root i.e. Since m is in max level, it has node ascendants.

3. x.key > h[m].key and m is grandchild of the root: So the element h[m] is moved to the root.

Algorithm	Average	Worstcase
Insert	$O(\log_2 n)$	$O(\log_2 n)$
Delete	$O(\log_2 n)$	$O(\log_2 n)$

Conclusion: We are able to implement Max/min heap concept in Data Structures.



PAGE NO.
DATE:

Experiment No. 11

Title:
Department maintains a student information. The file contains roll number, name and division and address. Allow user to add, delete information of studen. Display information of particular employee. If record of student does not exist an appropriate message is displayed.

Objective: To understand the concept and basic of sequential file and its use in Data structure.

Concepts:

File Organization:
File organization ensures that records are available for processing. It is used to determine an efficient file organization for each base relation.

For example, if we want to retrieve employee records in alphabetical order of name. Sorting the file by employee name is a good file organization. However if we want to retrieve all employees whose marks are in a certain range.

PAGE NO.
DATE:

Types of file organization

There are three types of organizing the file:

1. Sequential access file organization
2. Direct access file organization
3. Indexed sequential access file organization

Sequential access file organization

Advantages of sequential file:
- It is simple to program and easy to design
- Sequential file is best use if storage space.

Disadvantages of sequential file:
- Sequential file is time consuming process
- It has high data redundancy.
- Random searching is not possible.

Open a file:
The first operation generally performed on an object of one of these classes is to associate it to real life. This procedure is known as to open a file. An open file is represented within a program by a stream object.

PAGE NO.
DATE:

Special function to move file pointer within the file:

- seekg(): we can move input pointer to a specified location for reading using this function
- seekp(): we can move output pointer to a specified location for writing using this function
- tellg(): This function return the current position of input pointer.
- tellp(): This function return the current position of output pointer.

Origin can be any of the three values:

- ios::beg: Go to start: No matter how far into a file we have read by using this.
- ios::cur: Stay at current position: using this syntax, the file pointer will point to current position
- ios::end: Go to end of the file: using this syntax, the file pointer will point to end of the file. nded binary search tree.

Conclusion: we understand the concept and basic of sequential file and its use in Data structure.

PAGE NO.
DATE:

Experiment No. 12

Aim: Implementation of a direct access file- Insertion and deletion of a record from a direct access file.

Objective: to study direct file access

Theory:

Direct access file organization

- Direct access file is also known as random access or relative file organization
- In direct access file, all records are stored in direct access storage device (DASD), such as hard disk.
- This file organization is useful for immediate access to large amount of information.
- It is also called as hashing.

Advantages of direct access the organization

- In direct access file, sorting of the records are not required.
- It accesses the desired records immediately
- It updates several files quickly.
- It has better control over record allocation

Disadvantages of direct access file organization

- Direct access file does not provide back up facility
- It is expensive
- It has less storage space as compared to sequential file.

Algorithms:

1. Algorithm for create function

S1: open the file in the write mode.
S2: Read the no. of records N to be inserted to the file.
S3: Repeat the step 4 N no. of times
S4: Read the details of each student from the keyboard
S5: Close the file
S6: Return to the main function.

2. Algorithm for displaying all records.

S1: Open the specified file in read mode.
S2: If the file is unable to open or not found then display error message and go to step 4
S3: Scan all the student
S4: Close the file
S5: Return to the main function

PAGE NO.
DATE:

3. Algorithm for add a record.
S1: Open the file in the append mode, if the file specified is not found or unable
S2: Scan all the student details one by one from file until end of file is reached.
S3: Read the details of the form the keyboard and write the same to the file
S4: Close the file.
S5: Return the main function.

4. Algorithm the deleting a record
S1: Open the file in the append mode, if the file specified is not found or unable to open then display error message.
S2: Accept the roll no from the user to delete the record.
S3: Search for the roll no in file
S4: Close both files.
S5: Now, remove the old file & name the temporary file with name same as that of old file name.

Conclusion: hence studied the direct file insertion.

PAGE NO.
DATE:

Experiment No. 09

Aim : A dictionary stores keywords and its meanings. Provide facility for adding new keywords, deleting keywords, updating values of any entry. Provide facility to display whole data sorted in ascending/descending order. Also find how many maximum comparisons may require for finding any keyword.

Objective :
To understand the basic concepts and working of AVL Tree.

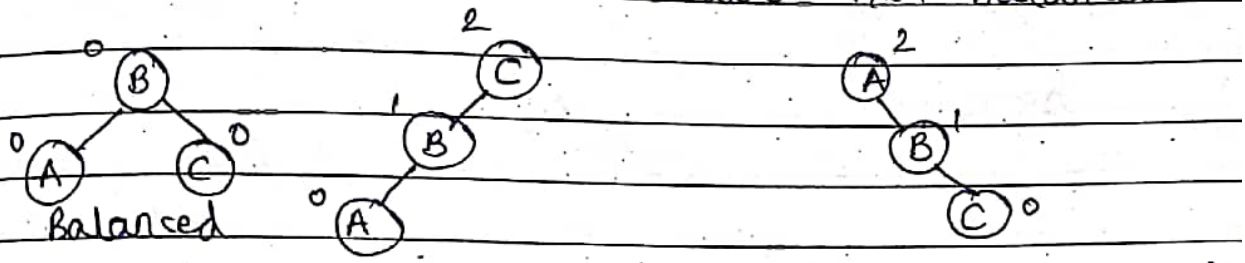
Theory :

What if the input to binary search tree comes in a sorted manner?
It will then look like this -
It is observed that BST's worst-case performance is closest to linear search algorithm that is O(n). In real-time data, we can not predict data pattern and their frequencies.

Named after their inventor Adelson, Velski & Landis, AVL trees are height balancing binary search tree. AVL tree checks the height of the left and the right sub-trees and assures that the difference is not more than 1.

PAGE NO.
DATE:

Here we see that the first tree is balanced and the next two trees are not balanced -



In the second tree, the left subtree of C has height 2 and the right subtree has height 0, so the difference is 2. In the third tree, the right subtree of A has height 2 and the left is missing, so it is 0, and the difference is 2 again. AVL tree permints difference to be only!

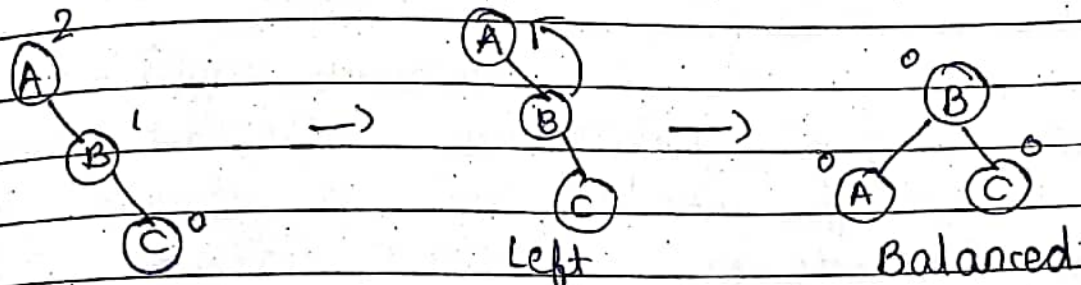
AVL Rotations

To balance itself, an AVL tree may perform the following four kinds of rotations -

- Left rotation
- Right rotation
- Left - Right rotation
- Right - left rotation

Left Rotation:

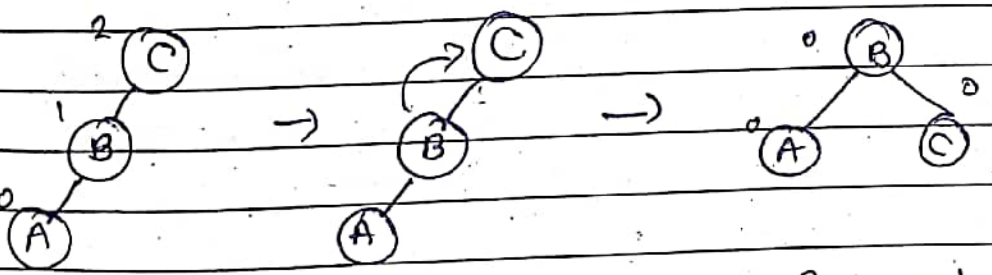
If a tree becomes unbalanced, when a node is inserted into the right subtree then we perform a single left rotation.



Right unbalanced tree

In our example, node A has become unbalanced as a node is inserted in the right subtree of A's right subtree.

Right rotation:
AVL tree may become unbalanced, if a node is inserted in the left subtree of the left subtree. The tree then needs a right rotation.



Left unbalanced tree

Right rotation

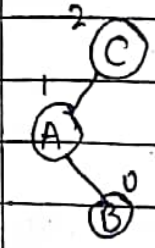
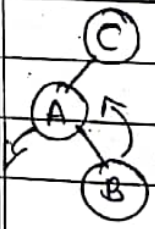
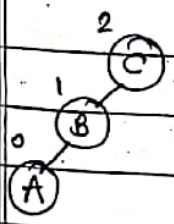
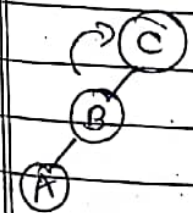
Balanced tree

As depicted, the unbalanced node becomes the right child of its left child by performing a right rotation.

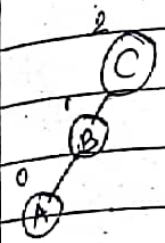
PAGE NO.
DATE:

Left - Right rotation

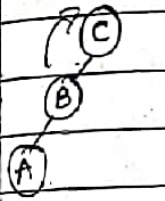
Double rotations are slightly complex version of already explained versions of rotations. To understand them better, we should take note of each action performed while rotation.

State	Action
	A node has been inserted into the right subtree of the left subtree. This makes C an unbalanced node. These scenarios cause AVL tree to perform left-right rotation.
	We first perform the left rotation on the left subtree of C. This makes A, the left subtree of B.
	Node C is still unbalanced, however now, it is because of the left-subtree of the left-subtree
	We shall now right-rotate the tree, making B the new root node of this subtree. C now becomes the right subtree of its own left subtree

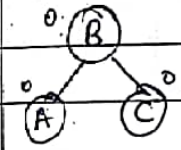
PAGE NO.
DATE:



Node C is still unbalanced, however now, it is because of the left-subtree of C



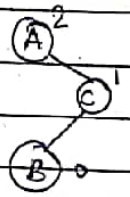
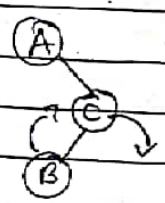
We shall now right-rotate the tree, making B the new root node of this subtree



The tree is now balanced.

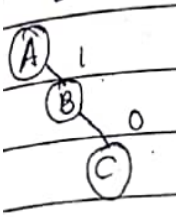
Right left rotation

The second type of double rotation is Right-Left Rotation. It is a combination of right rotation followed by left rotation.

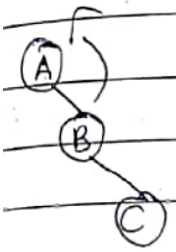
State	Action
	A node has been inserted into the left subtree of the right subtree.
	First we perform the right rotation along C node, making the C right subtree of its own left subtree B.

PAGE NO.
DATE:

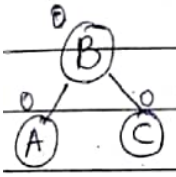
2



Node A is still unbalanced because of the right subtree of its right subtree and requires a left rotation.



A left rotation is performed by making B the new root node of the subtree. A becomes the left subtree of its right subtree B



The tree is now balanced

Conclusion: hence we have studied AVL tree in detail