Modern Education Society’s Wadia College of Engineering, Pune

210256: DATA STRUCTURES and ALGORITHM LABORATORY (2019 C0URSE)

NAME OF STUDENT:	**CLASS:**
SEMESTER/YEAR:	**ROLL NO:**
DATE OF PERFORMANCE:	**DATE OF SUBMISSION:**
EXAMINED BY:	**EXPERIMENT NO: 01**

TITLE: Create hash table for telephone book database and search data in it.

AIM/PROBLEM STATEMENT: Consider telephone book database of N clients. Make use of a hash table implementation to quickly look up client’s telephone number. Make use of two collision handling techniques and compare them using number of comparisons required to find a set of telephone numbers.

OBJECTIVES:

1. To understand structure of hash table.
2. To understand searching and collision handling concept in hashing.

OUTCOMES:

1. To apply appropriate advanced data structure and efficient algorithms to approach the problems of various domain
2. To use effective and efficient data structures in solving various Computer Engineering domain problems.

PRE-REQUISITES:
1. Knowledge of python programming
2. Basic knowledge of hashing
3. Basic knowledge of searching in hashing.

THEORY: Expected by students

QUESTIONS:

1. Explain different hashing functions with example.
2. Describe Extensible hashing for the given input keys: 1,10,7,8,15,16

Modern Education Society's Wadia College of Engineering, Pune

210256: DATA STRUCTURES and ALGORITHM LABORATORY (2019 C0URSE)

NAME OF STUDENT:	**CLASS:**
SEMESTER/YEAR:	**ROLL NO:**
DATE OF PERFORMANCE:	**DATE OF SUBMISSION:**
EXAMINED BY:	**EXPERIMENT NO: 02**

TITLE: PERFORM VARIOUS OPERATIONS ON SET

AIM/PROBLEM STATEMENT:

To create ADT that implements the "set" concept.

a. Add (new Element) -Place a value into the set
b. Remove (element) Remove the value
c. Contains (element) Return true if element is in collection
d. Size () Return number of values in collection Iterator () Return an iterator used to loop over collection
e. Intersection of two sets
f. Union of two sets
g. Difference between two sets
h. Subset

OBJECTIVES:

1. To understand structure of SET.
2. To understand how Create, Display and perform various operations on set.

OUTCOMES:

1. Understand the ADT/libraries to implement Set operations.
2. Analyze the algorithm design techniques for set concept.

PRE-REQUISITES:

1. Knowledge of python programming.
2. Knowledge of STL, set operations.

THEORY: Expected by students

QUESTIONS:

1. Define ADT of SET?
2. Explain iterator in C++ STL?

Modern Education Society's Wadia College of Engineering, Pune

210256: DATA STRUCTURES and ALGORITHM LABORATORY (2019 C0URSE)

NAME OF STUDENT:	**CLASS:**
SEMESTER/YEAR:	**ROLL NO:**
DATE OF PERFORMANCE:	**DATE OF SUBMISSION:**
EXAMINED BY:	**EXPERIMENT NO: 03**

TITLE: Tree data structure

AIM/PROBLEM STATEMENT: A book consists of chapters, chapters consist of sections and sections consist of subsections. Construct a tree and print the nodes. Find the time and space requirements of your method.

OBJECTIVES:

1. To understand concept of tree data structure
2. To understand concept & features of object oriented programming.

Learning Outcome:

1. Define class for structures using Object Oriented features.

2. Analyze tree data structure

THEORY:

Introduction to Tree:

A tree T is a set of nodes storing elements such that the nodes have a parent-child relationship that satisfies the following

• if T is not empty, T has a special tree called the root that has no parent

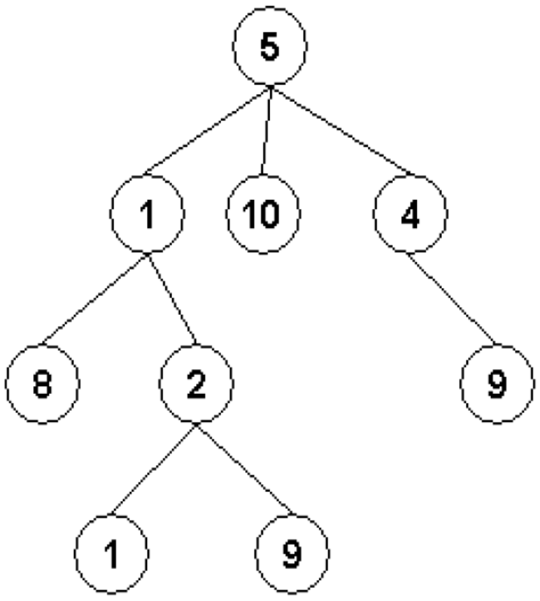
• each node v of T different than the root has a unique parent node w; each node with parent w is a child of w

Recursive definition

• T is either empty

• or consists of a node r (the root) and a possibly empty set of trees whose roots are the children of r

Tree is a widely-used data structure that emulates a tree structure with a set of linked nodes. The tree graphically is represented as below



The circles are the nodes and the edges are the links between them. Trees are usually used to store and represent data in some hierarchical order. The data are stored in the nodes, from which the tree is consisted of.

A subtree is a portion of a tree data structure that can be viewed as a complete tree in itself. Any node in a tree T, together with all the nodes below his height, that are reachable from the node, comprise a subtree of T.

Important Terms

Following are the important terms with respect to tree.

Path − Path refers to the sequence of nodes along the edges of a tree.

Root − The node at the top of the tree is called root. There is only one root per tree and one path from the root node to any node.

Parent − Any node except the root node has one edge upward to a node called parent.

Child − The node below a given node connected by its edge downward is called its child node.

Leaf − The node which does not have any child node is called the leaf node.

Subtree − Subtree represents the descendants of a node.

Visiting − Visiting refers to checking the value of a node when control is on the node.

Traversing − Traversing means passing through nodes in a specific order.

Levels − Level of a node represents the generation of a node. If the root node is at level 0, then its next child node is at level 1, its grandchild is at level 2, and so on.
Keys - Key represents a value of a node based on which a search operation is to be carried out for a node.

There are two basic types of trees. In an unordered tree, a tree is a tree in a purely structural sense — that is to say, given a node, there is no order for the children of that node. A tree on which an order is imposed — for example, by assigning different natural numbers to each child of each node — is called an ordered tree, and data structures built on them are called ordered tree data structures. Ordered trees are by far the most common form of tree data structure. Binary search trees are one kind of ordered tree.

Advantages of trees

Trees are so useful and frequently used, because they have some very serious advantages:

- Trees reflect structural relationships in the data
- Trees are used to represent hierarchies
- Trees provide an efficient insertion and searching
- Trees are very flexible data, allowing to move subtrees around with minumum effort

QUESTIONS:

1. What is class, object and data structure?
2. What is tree data structure?
3. Explain different types of tree?

Modern Education Society’s Wadia College of Engineering, Pune

210256: DATA STRUCTURES and ALGORITHM LABORATORY (2019 C0URSE)

NAME OF STUDENT:	**CLASS:**
SEMESTER/YEAR:	**ROLL NO:**
DATE OF PERFORMANCE:	**DATE OF SUBMISSION:**
EXAMINED BY:	**EXPERIMENT NO: 04**

TITLE: To illustrate the various Binary Tree functions.

AIM/PROBLEM STATEMENT: For given expression eg. a-b*c-d/e+f construct inorder sequence and traverse it using postorder traversal (non-recursive) and delete the entire tree.

OBJECTIVES:

1. To understand concept of Tree & Binary Tree.

2. To understand concept & features of object oriented programming.

OUTCOMES:

1. To analyze the working of various Tree operations.

THEORY:

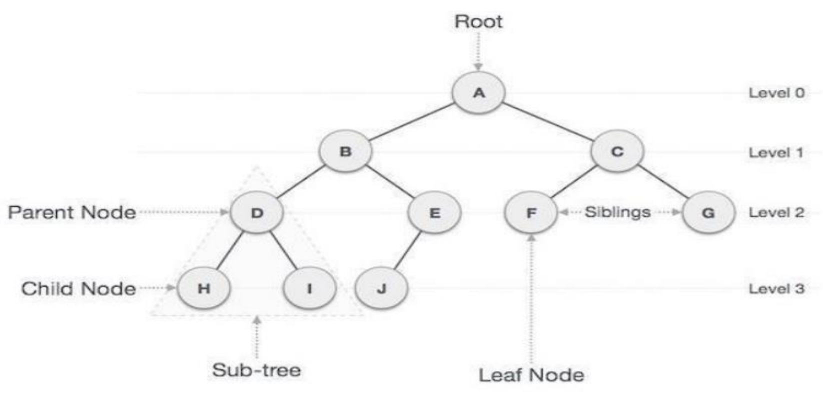
Tree

Tree represents the nodes connected by edges also a class of graphs that is acyclic is termed as trees. Let us now discuss an important class of graphs called trees and its associated terminology. Trees are useful in describing any structure that involves hierarchy. Familiar examples of such structures are family trees, the hierarchy of positions in an organization, and so on.

Binary Tree

A binary tree is made of nodes, where each node contains a "left" reference, a "right" reference, and a data element. The topmost node in the tree is called the root.

Every node (excluding a root) in a tree is connected by a directed edge from exactly one other node. This node is called a parent. On the other hand, each node can be connected to arbitrary number of nodes, called children. Nodes with no children are called leaves, or external nodes. Nodes which not leaves are called internal nodes. Nodes with the same parent are called siblings.



Insert Operation

The very first insertion creates the tree. Afterwards, whenever an element is to be inserted, first locate its proper location. Start searching from the root node, then if the data is less than the key value, search for the empty location in the left subtree and insert the data. Otherwise, search for the empty location in the right subtree and insert the data.

Traversals

A traversal is a process that visits all the nodes in the tree. Since a tree is a nonlinear data structure, there is no unique traversal. We will consider several traversal algorithms with we group in the following two kinds: Depth-first traversal , Breadth-first traversal

There are 3 different types of depth-first traversals:

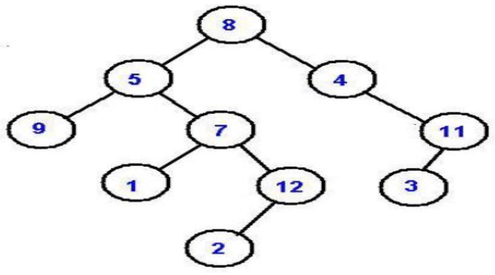
PreOrder traversal - visit the parent first and then left and right children

InOrder traversal - visit the left child, then the parent and the right child

PostOrder traversal - visit left child, then the right child and then the parent

There is only one kind of breadth-first traversal--the level order traversal. This traversal visits nodes by levels from top to bottom and from left to right.

As an example, consider the following tree and its four traversals:



PreOrder - 8, 5, 9, 7, 1, 12, 2, 4, 11, 3

InOrder - 9, 5, 1, 7, 2, 12, 8, 4, 3, 11

PostOrder - 9, 1, 2, 12, 7, 5, 3, 11, 4, 8

LevelOrder - 8, 5, 4, 9, 7, 11, 1, 12, 3, 2

QUESTIONS:

1. Consider the polynomial 5y 2 − 3y + 2.

(a) Write the polynomial as an expression tree that obeys the usual ordering of operations. Hint: to clarify the ordering for yourself, first write the expression with brackets indicating the order of operations. (b) Write the polynomial as postfix expression

2. Given a full binary expression tree consisting of basic binary operators (+ , – ,*, /) and some integers, Your task is to evaluate the expression tree.

```
        +
      /   \
     *     -
    / \   /  \
   5   4 100  20
```

3. (a) Draw the binary tree whose in-order traversal is DBEAFC and whose pre-order traversal is ABDECF.

(b) What is the post-order traversal of this tree?

(c) Draw all binary search trees of height 2 that can be made from all the letters ABCDEF, assuming the natural ordering.

210256: DATA STRUCTURES and ALGORITHM LABORATORY (2019 C0URSE)

NAME OF STUDENT:	**CLASS:**
SEMESTER/YEAR:	**ROLL NO:**
DATE OF PERFORMANCE:	**DATE OF SUBMISSION:**
EXAMINED BY:	**EXPERIMENT NO: 05**

TITLE: Creation of Dictionary using BST.

AIM/PROBLEM STATEMENT: Dictionary stores keywords & 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. Use Binary Search Tree for implementation.

OBJECTIVES:

1. To understand structure of binary search tree
2. To perform various objectives of dictionary using binary search tree

OUTCOMES:

1. To apply the binary search tree for dictionary operations
2. To analyze the

PRE-REQUISITES:

1. Knowledge of C++ programming
2. Basic knowledge of Dictionary ADT
3. Basic knowledge of searching in binary search tree.

THEORY:

Binary Search Tree, is a node-based binary tree data structure which has the following properties:

- The left subtree of a node contains only nodes with keys lesser than the node’s key.
- The right subtree of a node contains only nodes with keys greater than the node’s key.

- The left and right subtree each must also be a binary search tree.
- There must be no duplicate nodes

The above properties of Binary Search Tree provide an ordering among keys so that the operations like search, minimum and maximum can be done fast. If there is no ordering, then we may have to compare every key to search a given key.

Searching a key:

To search a given key in Binary Search Tree, we first compare it with root, if the key is present at root, we return root. If key is greater than root's key, we recur for right subtree of root node. Otherwise, we recur for left subtree.

A simple implementation for the Dictionary ADT can be based on sorted or unsorted lists. When implementing the dictionary with an unsorted list, inserting a new record into the dictionary can be performed quickly by putting it at the end of the list. However, searching an unsorted list for a particular record requires $\Theta(n)$ time in the average case. For a large database, this is probably much too slow. Alternatively, the records can be stored in a sorted list. If the list is implemented using a linked list, then no speedup to the search operation will result from storing the records in sorted order. On the other hand, if we use a sorted array-based list to implement the dictionary, then binary search can be used to find a record in only Θ (log n) time. However, insertion will now require $\Theta(n)$ time on average because, once the proper location for the new record in the sorted list has been found, many records might be shifted to make room for the new record.
The way to organize a collection of records so that inserting records and searching for records can both be done quickly, is by using binary search tree (BST). The advantage of using the BST is that all major operations (insert, search, and remove) are $\Theta(\log n)$ in the average case. Of course, if the tree is badly balanced, then the cost can be as bad as $\Theta(n)$

QUESTIONS:

1. Discuss about various operations of binary search tree.

2. What is Dictionary ADT and its operations?

Modern Education Society's Wadia College of Engineering, Pune

210256: DATA STRUCTURES and ALGORITHM LABORATORY (2019 C0URSE)

NAME OF STUDENT:	**CLASS:**
SEMESTER/YEAR:	**ROLL NO:**
DATE OF PERFORMANCE:	**DATE OF SUBMISSION:**
EXAMINED BY:	**EXPERIMENT NO: 06**

TITLE: REPRESENTATION OF ADJACENCY LIST AND MATRIX OF A GRAPH

AIM/PROBLEM STATEMENT: Represent a given graph using adjacency matrix/ list to perform DFS and using list to perform BFS. Use the map of the area around the college as the graph. Identify the prominent land marks as nodes and perform DFS and BFS on that

OBJECTIVES:

1. To understand graph data structure using adjacency matrix/ list
2. Able to perform graph traversal DFS using adjacency matrix with the help of stack 3.
3. Able to perform graph traversal BFS using adjacency list with the help of Queue

OUTCOMES:

1. Apply and analyze linear data structures to solve non-linear data structure problems.

PRE-REQUISITE:

1. Basic Knowledge of 2D array
2. Basic Knowledge of Linked list, Stack and Queue

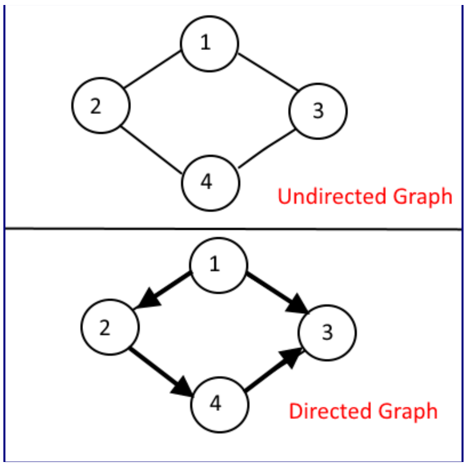
THEORY:

$$G = (V,E)$$

Graph is a collection of nodes or vertices (V) and edges(E) between them. We can traverse these nodes using the edges. These edges might be weighted or non-weighted.

There can be two kinds of Graphs

- Un-directed Graph – when you can traverse either direction between two nodes.
- Directed Graph – when you can traverse only in the specified direction between two nodes.

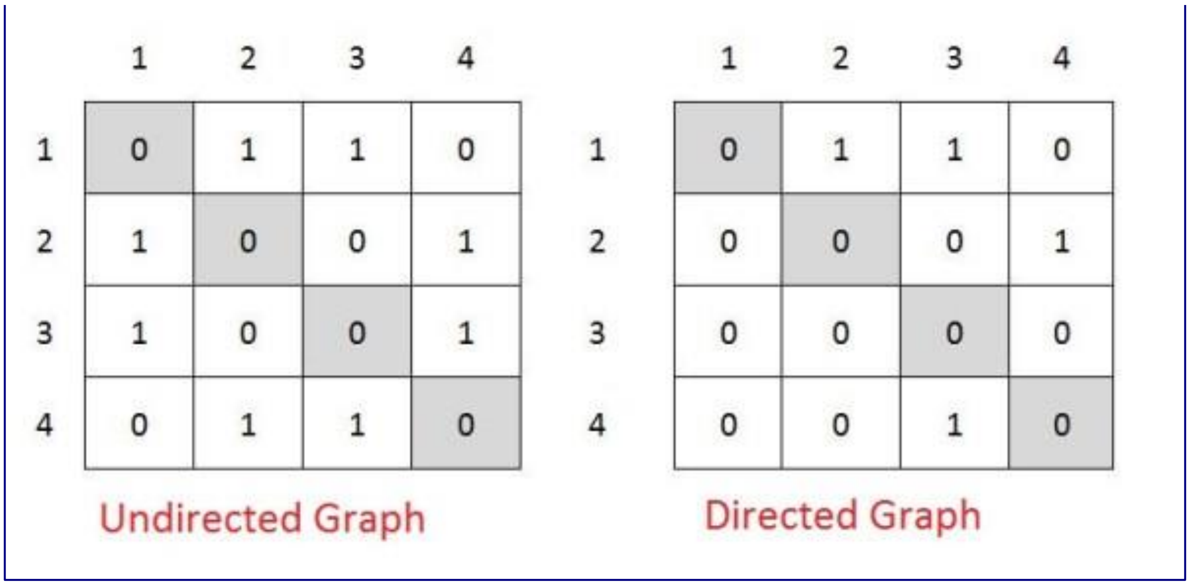


There are two common ways to represent a Graph:

- Adjacency Matrix
- Adjacency List

Adjacency Matrix:

Adjacency Matrix is 2-Dimensional Array which has the size VxV, where V are the number of vertices in the graph. See the example below, the Adjacency matrix for the graph shown above.



	1	2	3	4
1	0	1	1	0
2	1	0	0	1
3	1	0	0	1
4	0	1	1	0

Undirected Graph

	1	2	3	4
1	0	1	1	0
2	0	0	0	1
3	0	0	0	0
4	0	0	1	0

Directed Graph

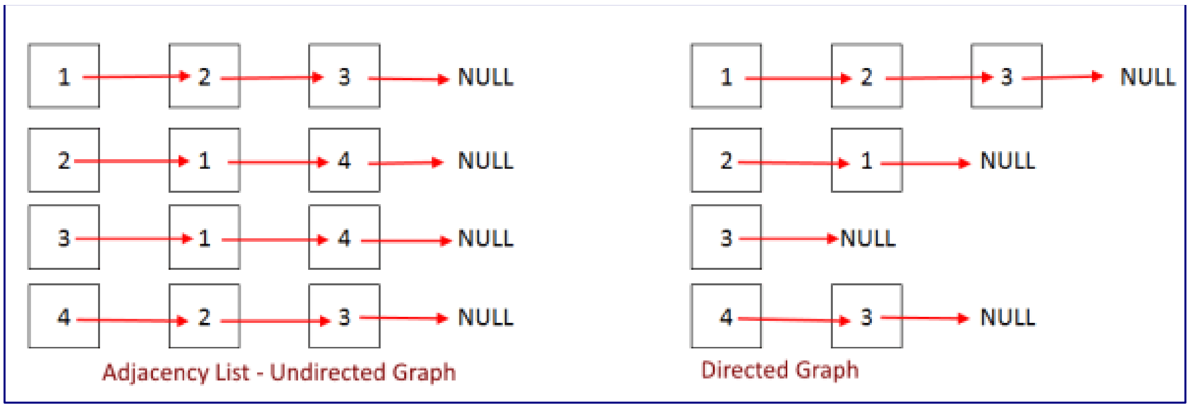
adjMaxtrix[i][j] = 1 when there is edge between Vertex i and Vertex j, else 0.

It’s easy to implement because removing and adding an edge takes only O(1) time.

But the drawback is that it takes O(V^2) space even though there are very less edges in the graph.

Adjacency List:

Adjacency List is the Array[] of Linked List, where array size is same as number of Vertices in the graph. Every Vertex has a Linked List. Each Node in this Linked list represents the reference to the other vertices which share an edge with the current vertex. The weights can also be stored in the Linked List Node.



The code may look complex since everything is being implemented from the scratch like linked list and so on. For better understanding, read more articles for easier implementations (Adjacency Matrix and Adjacency List)

QUESTIONS:

1) List the applications of graph?

2) Given an undirected graph G with V vertices and E edges, What is the sum of the degrees of all vertices?

Modern Education Society's Wadia College of Engineering, Pune

210256: DATA STRUCTURES and ALGORITHM LABORATORY (2019 C0URSE)

NAME OF STUDENT:	**CLASS:**
SEMESTER/YEAR:	**ROLL NO:**
DATE OF PERFORMANCE:	**DATE OF SUBMISSION:**
EXAMINED BY:	**EXPERIMENT NO: 07**

TITLE: PRIM'S ALGORITHM

AIM/PROBLEM STATEMENT: You have a business with several offices; you want to lease phone lines to connect them up with each other; and the phone company charges different amounts of money to connect different pairs of cities. You want a set of lines that connects all your offices with a minimum total cost. Solve the problem by suggesting appropriate data structures.

OBJECTIVES:

1. To understand minimum spanning tree of a Graph

2. To understand how Prim‘s algorithm works for Minimum Spanning Tree (MST)

OUTCOMES:

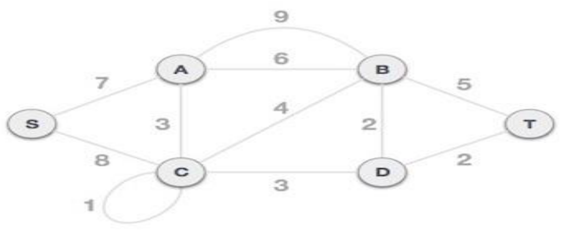
1. To use effective and efficient data structures in solving various Computer Engineering design problems.

PRE-REQUISITE:

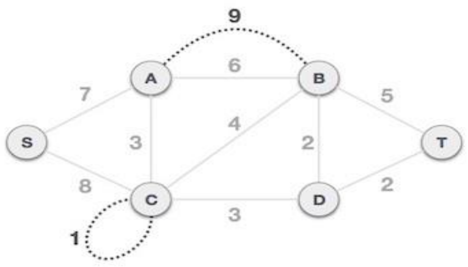
1. Knowledge of C++ programming
2. Knowledge of 2D-Array.

THEORY:

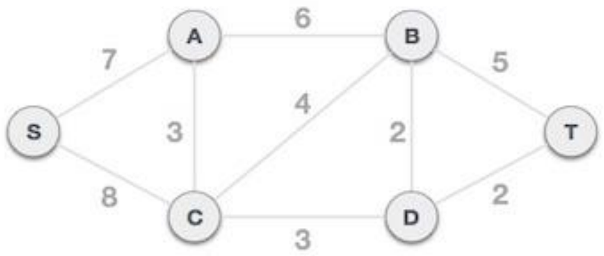
Prim's algorithm to find minimum cost spanning tree (as Kruskal's algorithm) uses the greedy approach. Prim's algorithm shares a similarity with the shortest path first algorithms.
Prim's algorithm, in contrast with Kruskal's algorithm, treats the nodes as a single tree and keeps on adding new nodes to the spanning tree from the given graph.
To contrast with Kruskal's algorithm and to understand Prim's algorithm better, we shall use the same example –



Step 1 - Remove all loops and parallel edges

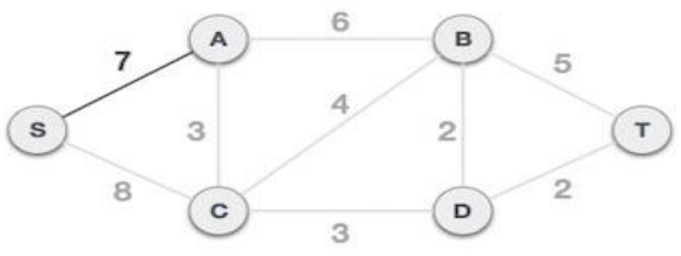


Remove all loops and parallel edges from the given graph. In case of parallel edges, keep the one which has the least cost associated and remove all others.

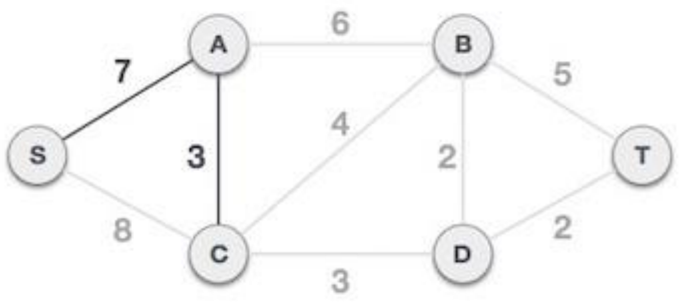


Step 2 - Choose any arbitrary node as root node
In this case, we choose **S** node as the root node of Prim's spanning tree. This node is arbitrarily chosen, so any node can be the root node. One may wonder why any video can be a root node. So, the answer is, in the spanning tree all the nodes of a graph are included and because it is connected then there must be at least one edge, which will join it to the rest of the tree.

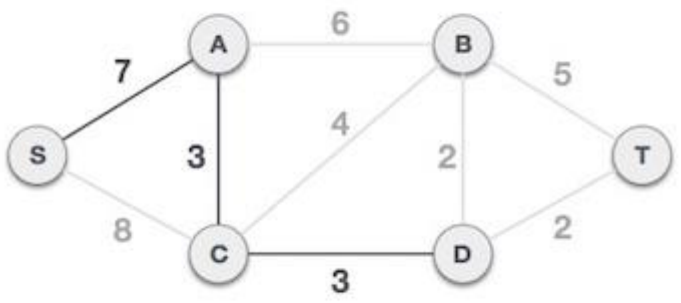
Step 3 - Check outgoing edges and select the one with less cost
After choosing the root node **S**, we see that S, A and S, C are two edges with weight 7 and 8, respectively. We choose the edge S, A as it is lesser than the other.



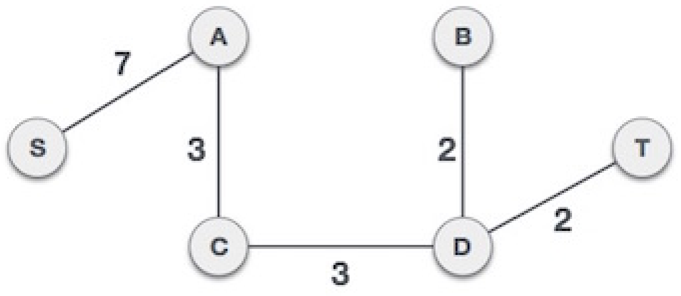
Now, the tree S-7-A is treated as one node and we check for all edges going out from it. We select the one which has the lowest cost and include it in the tree.



After this step, S-7-A-3-C tree is formed. Now we'll again treat it as a node and will check all the edges again. However, we will choose only the least cost edge. In this case, C-3-D is the new edge, which is less than other edges' cost 8, 6, 4, etc.



After adding node **D** to the spanning tree, we now have two edges going out of it having the same cost, i.e., D-2-T and D-2-B. Thus, we can add either one. But the next step will again yield edge 2 as the least cost. Hence, we are showing a spanning tree with both edges included.



The output spanning tree of the same graph using two different algorithms is same.

QUESTIONS:

1. Suppose we have an undirected graph with weights that can be either positive or negative. Do Prim's and Kruskal's algorithm produce an MST for such a graph?
2. Can a graph have more than one spanning tree?
3. State the difference between Prims and Kruskal's Algorithm.

Modern Education Society's Wadia College of Engineering, Pune

210256: DATA STRUCTURES and ALGORITHM LABORATORY (2019 C0URSE)

NAME OF STUDENT:	**CLASS:**
SEMESTER/YEAR:	**ROLL NO:**
DATE OF PERFORMANCE:	**DATE OF SUBMISSION:**
EXAMINED BY:	**EXPERIMENT NO: 08**

TITLE: Implementation of Optimal Binary Search Tree (OBST)

AIM/PROBLEM STATEMENT: Given sequence k = k1 <k2 < … < kn of n sorted keys, with a search probability pi for each key ki. Build the Binary search tree that has the least search cost given the access probability for each key ?

OBJECTIVES:

1. To understand concept of OBST.
2. To understand concept & features like extended binary search tree..

OUTCOMES:

1. Define class for Extended binary search tree using Object Oriented features.
2. Analyze working of functions.

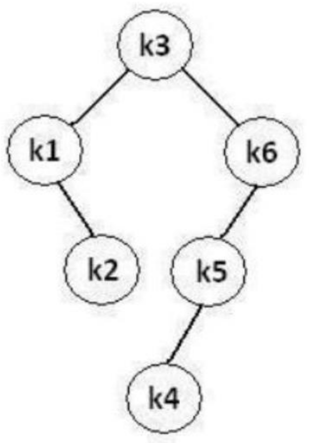
PRE-REQUISITES:

1. Knowledge of C++ programming.
2. Knowledge of OBST

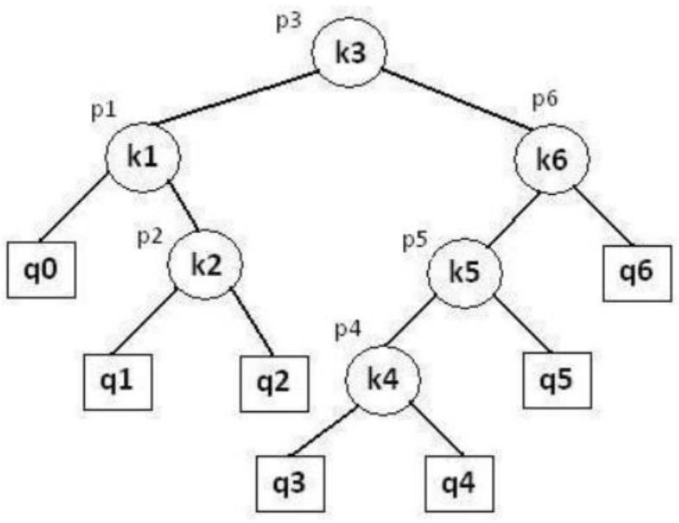
THEORY:

An optimal binary search tree is a binary search tree for which the nodes are arranged on levels such that the tree cost is minimum. For the purpose of a better presentation of optimal binary search trees, we will consider "extended binary search trees", which have the keys stored at their internal nodes. Suppose "n" keys k1, k2, … kn are stored at the internal nodes of a binary search tree. It is assumed that the keys are given in sorted order, so that k1< k2 < … < kn.

An extended binary search tree is obtained from the binary search tree by adding successor nodes to each of its terminal nodes as indicated in the following figure by squares:



Binary search tree



Extended binary search tree

In the extended tree:

 The squares represent terminal nodes. These terminal nodes represent unsuccessful searches of the tree for key values. The searches did not end successfully, that is, because they represent key values that are not actually stored in the tree;

 The round nodes represent internal nodes; these are the actual keys stored in the tree;

 Assuming that the relative frequency with which each key value is accessed is known, weights can be assigned to each node of the extended tree (p1 … p6). They represent the relative frequencies of searches terminating at each node, that is, they mark the successful searches.

 If the user searches a particular key in the tree, 2 cases can occur:

 1 – the key is found, so the corresponding weight „p" is incremented;

 2 – the key is not found, so the corresponding „q" value is incremented.

GENERALIZATION:

The terminal node in the extended tree that is the left successor of k1 can be interpreted as representing all key values that are not stored and are less than k1. Similarly, the terminal node in the extended tree that is the right successor of kn, represents all key values not stored in the tree that are greater than kn. The terminal node that is successes between ki and ki-1 in an inorder traversal represent all key values not stored that lie between ki and ki - 1.

COMPLEXITY ANALYSIS:

The algorithm requires O (n2) time and O (n2) storage. Therefore, as n increases it will run out of storage even before it runs out of time. The storage needed can be reduced by almost half by implementing the two-dimensional arrays as one-dimensional arrays.

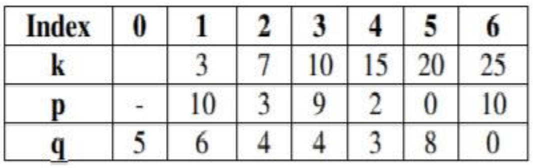
QUESTIONS:

1. Find the optimal binary search tree for N = 6, having keys k1 … k6 and weights p1 = 10, p2 = 3, p3 = 9, p4 = 2, p5 = 0, p6 = 10; q0 = 5, q1 = 6, q2 = 4, q3 = 4, q4 = 3, q5 = 8, q6 = 0. The following figure shows the arrays as they would appear after the initialization and their final disposition.

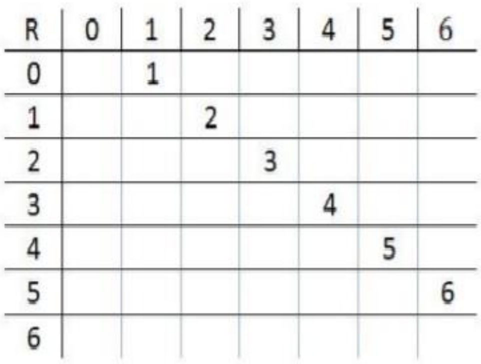
W indicates weighted indicates

C indicates cost

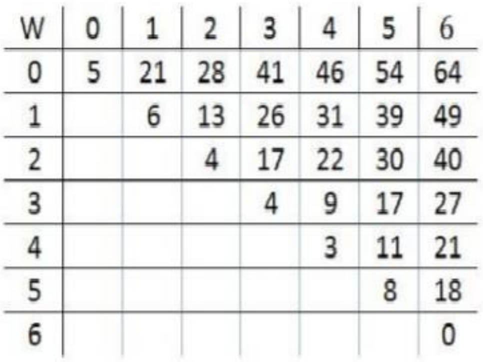
R indicates root



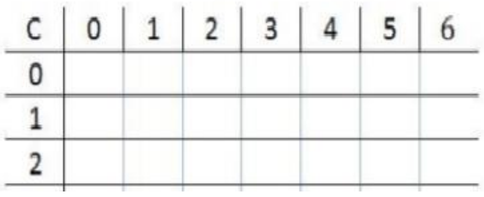
Index	0	1	2	3	4	5	6
k		3	7	10	15	20	25
p	-	10	3	9	2	0	10
q	5	6	4	4	3	8	0



R	0	1	2	3	4	5	6
0		1					
1			2				
2				3			
3					4		
4						5	
5							6
6							



W	0	1	2	3	4	5	6
0	5	21	28	41	46	54	64
1		6	13	26	31	39	49
2			4	17	22	30	40
3				4	9	17	27
4					3	11	21
5						8	18
6							0



C	0	1	2	3	4	5	6
0							
1							
2							

Modern Education Society's Wadia College of Engineering, Pune

210256: DATA STRUCTURES and ALGORITHM LABORATORY (2019 C0URSE)

NAME OF STUDENT:	**CLASS:**
SEMESTER/YEAR:	**ROLL NO:**
DATE OF PERFORMANCE:	**DATE OF SUBMISSION:**
EXAMINED BY:	**EXPERIMENT NO: 09**

TITLE: Implementation of Adelson, Velskii, and Landi (AVL) tree

AIM/PROBLEM STATEMENT: Dictionary stores keywords & 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 any keyword. Use Height balance tree and find the complexity for finding a keyword.

OBJECTIVES:

1. To understand tree data structure.
2. To understand practical implementation and usage of non-linear data structures to solving problems of AVL Tree.

OUTCOMES:

1. Apply and analyze non-linear data structures to solve real world complex problems.

PRE-REQUISITES:

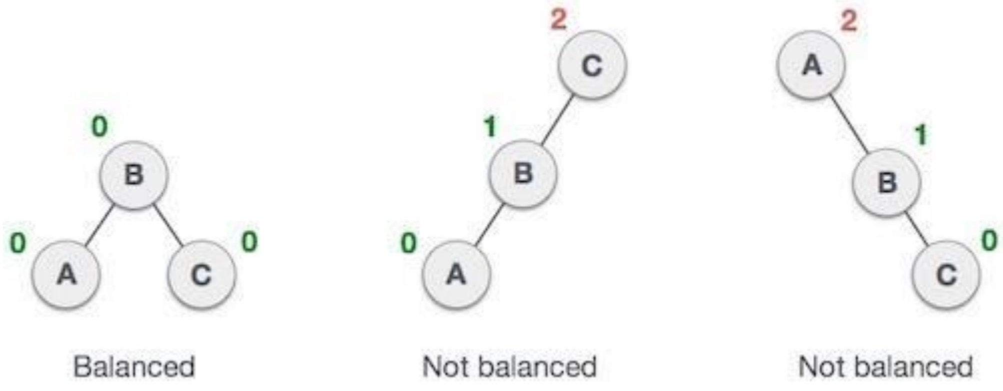
1. Knowledge of C++ programming.
2. Knowledge of AVL tree

THEORY:

It is observed that BST's worst-case performance is closest to linear search algorithms, that is O(n). In real-time data, we cannot predict data pattern and their frequencies. So, a need arises to balance out the existing BST.

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. This difference is called the **Balance Factor**.

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 permits difference (balance factor) to be only 1.

BalanceFactor = height(left-sutree) − height(right-sutree)

If the difference in the height of left and right sub-trees is more than 1, the tree is balanced using some rotation techniques.

AVL Rotations

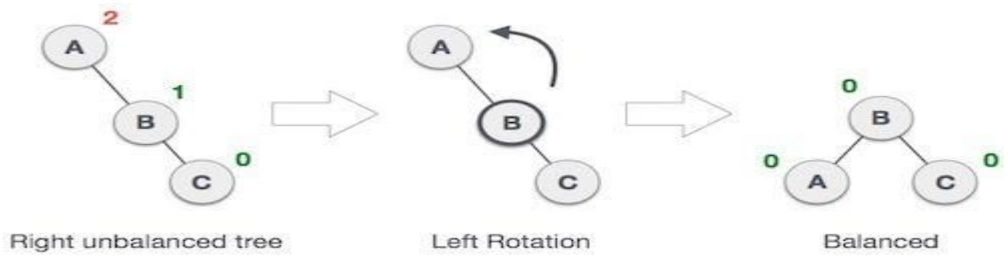
To balance itself, an AVL tree may perform the following four kinds of rotations −

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

The first two rotations are single rotations and the next two rotations are double rotations. To have an unbalanced tree, we at least need a tree of height 2. With this simple tree, let's understand them one by one.

Left Rotation

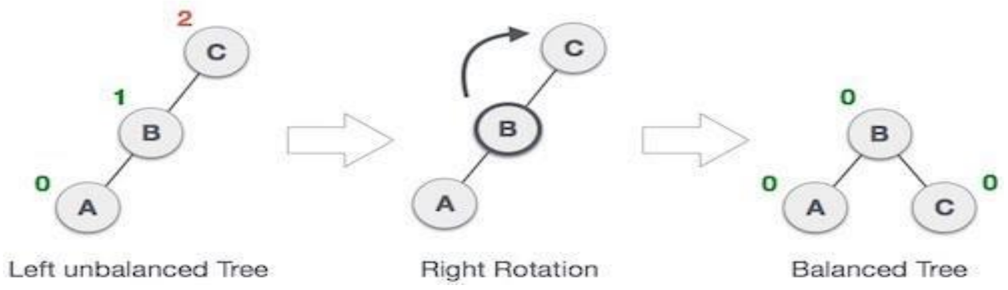
If a tree becomes unbalanced, when a node is inserted into the right subtree of the right subtree, then we perform a single left rotation –



In our example, node **A** has become unbalanced as a node is inserted in the right subtree of A's right subtree. We perform the left rotation by making **A** the left-subtree of B.

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.

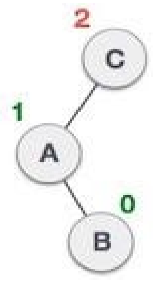
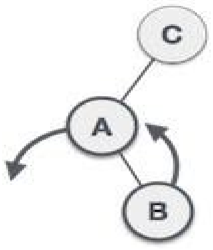
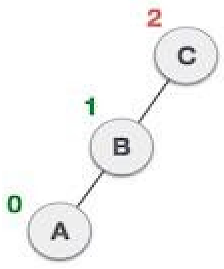
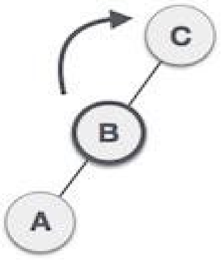
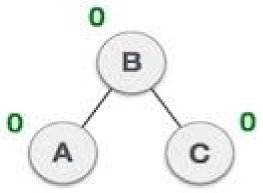


As depicted, the unbalanced node becomes the right child of its left child by performing a right rotation.

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. Let's first check how to perform Left-Right rotation. A left-right rotation is a combination of left rotation followed by right rotation.

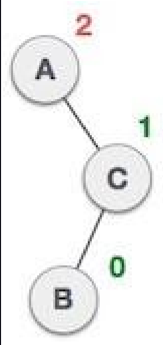
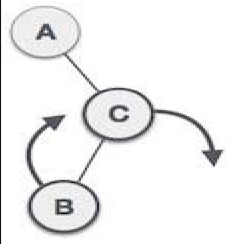
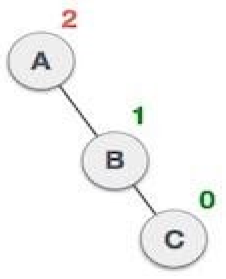
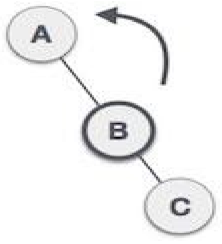
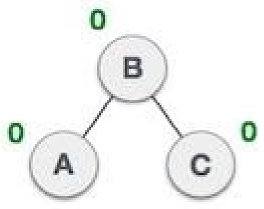
State	**Action**

	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.
	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. This makes **A**, an unbalanced node with balance factor 2.
	First, we perform the right rotation along **C** node, making **C** the right subtree of its own left subtree **B**. Now, **B** becomes the right subtree of **A**.
	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.

QUESTIONS:

1. What is an AVL tree. Explain with the help of example. what are the applications of AVL tree.

2. What is the difference between OBST, Huffman’s tree and AVL tree

3. Explain Single rotation and Double rotation with example?

4.Write down the time and space complexity of AVL Tree.

Modern Education Society's Wadia College of Engineering, Pune

210256: DATA STRUCTURES and ALGORITHM LABORATORY (2019 C0URSE)

NAME OF STUDENT:	**CLASS:**
SEMESTER/YEAR:	**ROLL NO:**
DATE OF PERFORMANCE:	**DATE OF SUBMISSION:**
EXAMINED BY:	**EXPERIMENT NO: 10**

TITLE: PRIORITY QUEUE

AIM/PROBLEM STATEMENT: Consider a scenario for hospital to cater services to different kind of patients as Serious (top priority), non-serious (Medium Priority), and General checkup (Least Priority). Implement priority queue to cater services to the patients.

OBJECTIVES:

1. To understand priority queue data structure.
2. To understand practical implementation and usage of queue linear data structures

OUTCOMES:

1. Apply and analyze appropriate data structure to solve the real time problems using priority queue

PRE-REQUISITE:

1. Knowledge of C++ programming
2. Knowledge of 2D-Array, structures

THEORY:

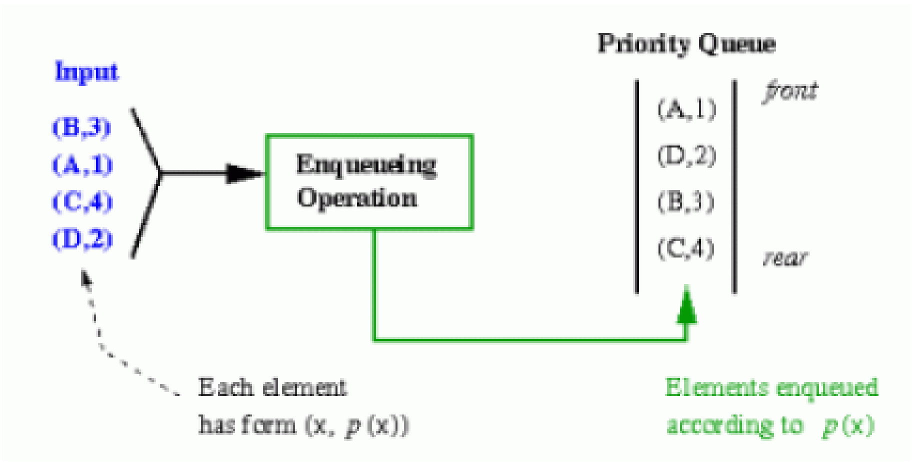
A Queue is a linear structure which follows a particular order in which the operations are performed. The order is First In First Out (FIFO). The difference between stacks and queues is in removing. In a stack we remove the item the most recently added; in a queue, we remove the item the least recently added.

A queue in which we are able to insert and remove items from any position based on some property (such as priority of the task to be processed) is often referred as priority queue. Fig 1 represents a priority queue of jobs waiting to use a computer.

Priority Queue is an extension of queue with following properties.

- Every item has a priority associated with it.
- An element with high priority is dequeued before an element with low priority.
- If two elements have the same priority, they are served according to their order in the queue.

In the below priority queue example is given:



A typical priority queue supports following operations.
insert(item, priority): Inserts an item with given priority.
getHighestPriority(): Returns the highest priority item.
deleteHighestPriority(): Removes the highest priority item.

How to implement priority queue?

Using Array: A simple implementation is to use array of following structure. insert() operation can be implemented by adding an item at end of array in O(1) time.
getHighestPriority() operation can be implemented by linearly searching the highest priority item in array. This operation takes O(n) time.

deleteHighestPriority() operation can be implemented by first linearly searching an item, then removing the item by moving all subsequent items one position back.

Using Heaps:

Heap is generally preferred for priority queue implementation because heaps provide better performance compared arrays or linked list.

In a Binary Heap, getHighestPriority() can be implemented in O(1) time, insert() can be implemented in O(Logn) time and deleteHighestPriority() can also be implemented in O(Logn) time.With Fibonacci heap, insert() and getHighestPriority() can be implemented in O(1) amortized time and deleteHighestPriority() can be implemented in O(Logn) amortized time.

Applications of Priority Queue:

1) CPU Scheduling

2) Graph algorithms like Dijkstra’s shortest path algorithm, Prim’s Minimum Spanning Tree, etc

3) All queue applications where priority is involved

QUESTIONS:

1. Define Ascending and Descending Priority queue.
2. Describe various applications of Priority Queue.
3. Write short note on ADT of Priority queue

Modern Education Society's Wadia College of Engineering, Pune

210256: DATA STRUCTURES and ALGORITHM LABORATORY (2019 C0URSE)

NAME OF STUDENT:	**CLASS:**
SEMESTER/YEAR:	**ROLL NO:**
DATE OF PERFORMANCE:	**DATE OF SUBMISSION:**
EXAMINED BY:	**EXPERIMENT NO: 11**

TITLE: Create sequential file and maintain data as specified

AIM/PROBLEM STATEMENT: Department maintains a student information. The file contains roll number, name, division, and address. Allow user to add delete information of student. Display information of particular student. If record of student does not exist an appropriate message is displayed. If it is, then the system displays the student details. Use sequential file to maintain the data.

OBJECTIVES:

1. To understand file handling.
2. To understand working of sequential file.

OUTCOMES:

1. To apply appropriate file handling techniques on given data
2. To use sequential file.

PRE-REQUISITE:

1. Knowledge of C++ programming
2. Basic knowledge of sequential file

THEORY:

File is a collection of records related to each other. The file size is limited by the size of memory and storage medium.

There are two important features of file:

1. **File Activity:** File activity specifies percent of actual records which proceed in a single run.

2. **File Volatility:** File volatility addresses the properties of record changes. It helps to increase the efficiency of disk design than tape.

File Organization

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

Types of File Organization-

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

1. **Sequential access file organization**

- Storing and sorting in contiguous block within files on tape or disk is called as **sequential access file organization**.
- In sequential access file organization, all records are stored in a sequential order. The records are arranged in the ascending or descending order of a key field.
- Sequential file search starts from the beginning of the file and the records can be added at the end of the file.
- In sequential file, it is not possible to add a record in the middle of the file without rewriting the file.

This method can be implemented in two ways:

Pile File Method:

- It is a quite simple method. In this method, we store the record in a sequence, i.e., one after another. Here, the record will be inserted in the order in which they are inserted into tables.
- In case of updating or deleting of any record, the record will be searched in the memory blocks. When it is found, then it will be marked for deleting, and the new record is inserted.

Sorted File Method:

- In this method, the new record is always inserted at the file's end, and then it will sort the sequence in ascending or descending order. Sorting of records is based on any primary key or any other key.
- In the case of modification of any record, it will update the record and then sort the file, and lastly, the updated record is placed in the right place.

2. 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. The records are randomly placed throughout the file.
- The records does not need to be in sequence because they are updated directly and rewritten back in the same location.
- This file organization is useful for immediate access to large amount of information. It is used in accessing large databases.
- It is also called as hashing.

3. Indexed sequential access file organization

- Indexed sequential access file combines both sequential file and direct access file organization.
- In indexed sequential access file, records are stored randomly on a direct access device such as magnetic disk by a primary key.
- This file has multiple keys. These keys can be alphanumeric in which the records are ordered is called primary key.
- The data can be access either sequentially or randomly using the index. The index is stored in a file and read into memory when the file is opened.

Primitive Operations on Sequential files:

• **Open**—This opens the file and sets the file pointer to immediately before the first record

• **Read-next**—This returns the next record to the user. If no record is present, then EOF condition will be set.

• **Close**—This closes the file and terminates the access to the file.

• **Write-next**—File pointers are set to next of last record and write the record to the file.

• **EOF**—If EOF condition occurs, it returns true, otherwise it returns false.

• **Search**—Search for the record with a given key.

• **Update**—Current record is written at the same position with updated values.

QUESTIONS:

1. Explain direct sequential file.
2. Explain advantage and disadvantages of the sequential file
3. Explain advantages and disadvantages of direct access method.

Modern Education Society's Wadia College of Engineering, Pune

210256: DATA STRUCTURES and ALGORITHM LABORATORY (2019 C0URSE)

NAME OF STUDENT:	**CLASS:**
SEMESTER/YEAR:	**ROLL NO:**
DATE OF PERFORMANCE:	**DATE OF SUBMISSION:**
EXAMINED BY:	**EXPERIMENT NO: 12**

TITLE: Create index sequential file and maintain data as specified

AIM/PROBLEM STATEMENT: Company maintains employee information as employee ID, name, designation, and salary. Allow user to add delete information of particular employee. If employee does not exist an appropriate message is displayed. If it is, then system displays the employee details. Use index sequential file to maintain the data

OBJECTIVES:

1. To understand file handling.
2. To understand working of index sequential file.

OUTCOMES:

1. To apply appropriate file handling techniques on given data
2. To use index sequential file.

PRE-REQUISITE:

1. Knowledge of C++ programming
2. Basic knowledge of sequential file

THEORY:

File is a collection of records related to each other. The file size is limited by the size of memory and storage medium.

There are two important features of file:

1. **File Activity:** File activity specifies percent of actual records which proceed in a single run.
2. **File Volatility:** File volatility addresses the properties of record changes. It helps to increase the efficiency of disk design than tape.

File Organization

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

Types of File Organization-

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

1. **Sequential access file organization**

- Storing and sorting in contiguous block within files on tape or disk is called as **sequential access file organization**.
- In sequential access file organization, all records are stored in a sequential order. The records are arranged in the ascending or descending order of a key field.
- Sequential file search starts from the beginning of the file and the records can be added at the end of the file.
- In sequential file, it is not possible to add a record in the middle of the file without rewriting the file.

This method can be implemented in two ways:

Pile File Method:

- It is a quite simple method. In this method, we store the record in a sequence, i.e., one after another. Here, the record will be inserted in the order in which they are inserted into tables.
- In case of updating or deleting of any record, the record will be searched in the memory blocks. When it is found, then it will be marked for deleting, and the new record is inserted.

Sorted File Method:

- In this method, the new record is always inserted at the file's end, and then it will sort the sequence in ascending or descending order. Sorting of records is based on any primary key or any other key.
- In the case of modification of any record, it will update the record and then sort the file, and lastly, the updated record is placed in the right place.

2. 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. The records are randomly placed throughout the file.
- The records does not need to be in sequence because they are updated directly and rewritten back in the same location.
- This file organization is useful for immediate access to large amount of information. It is used in accessing large databases.
- It is also called as hashing.

3. Indexed sequential access file organization

- Indexed sequential access file combines both sequential file and direct access file organization.
- In indexed sequential access file, records are stored randomly on a direct access device such as magnetic disk by a primary key.
- This file has multiple keys. These keys can be alphanumeric in which the records are ordered is called primary key.
- The data can be access either sequentially or randomly using the index. The index is stored in a file and read into memory when the file is opened.

Working:

Each record of a file has a key field which uniquely identifies that record.

- ✓ An index consists of keys and addresses (physical disc locations).
- ✓ An indexed sequential file is a sequential file (i.e. sorted into order of a key field) which has an index.
- ✓ A full index to a file is one in which there is an entry for every record.
- ✓ Indexed sequential files are important for applications where data needs to be accessed ----- >>> **sequentially** or **randomly** using the index
- ✓ An indexed sequential file allows fast access to a specific record.

  Example: A company may store details about its employees as an indexed sequential file. Sometimes the file is accessed....

  -- >> sequentially. For example, when the whole of the file is processed to produce payslips at the end of the month.

  -- >> randomly. Maybe an employee changes address, or a female employee gets married and changes her surname.

Primitive Operations on Index Sequential files:

• **Write (add, store):** User provides a new key and record, IS file inserts the new record and key.

• **Sequential Access (read next):** IS file returns the next record (in key order)

• **Random access (random read, fetch):** User provides key, IS file returns the record or "not there"

• **Rewrite (replace):** User provides an existing key and a new record, IS file replaces existing record with new**.**

• **Delete:** User provides an existing key, IS file deletes existing record

QUESTIONS:

1. Explain direct sequential file.
2. Explain advantage and disadvantages of the index sequential file