PUNE INSTITUTE OF COMPUTER TECHNOLOGY
DEPARTMENT OF COMPUTER ENGINEERING
LAB MANUAL
ACADEMIC YEAR: 2018-2019

DEPARTMENT: COMPUTER ENGINEERING
CLASS: S.E. **SEMESTER:** II

SUBJECT: Advanced Data Structure Laboratory

INDEX OF LAB EXPERIMENTS

Expt. No.	Problem Statement	Revised on
1.	Given binary tree with n nodes, perform following operations on it: • Perform preorder and post order traversal • Create a mirror image of it • Find the height of tree • Copy this tree to another [operator=] • Count number of leaves, number of internal nodes. • Erase all nodes in a binary tree. (implement both recursive and non-recursive methods)	**14-12-2018**
2.	A Dictionary stores keywords & its meanings. Provide facility for adding new keywords, deleting keywords, updating values of any entry, assign a given tree into another tree (=). 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.	**14-12-2018**
3.	Create an inordered threaded binary tree and perform inorder and preorder traversals. Analyze time and space complexity of the algorithm	**14-12-2018**
4.	There are flight paths between cities. If there is a flight between city A and city B then there is an edge between the cities. The cost of the edge can be the time that flight takes to reach city B from A, or the amount of fuel used for the journey. Represent this as a graph. The node can be represented by airport name or name of the city. Use adjacency list representation of the graph or use adjacency matrix representation of the graph. Justify the storage representation used.(Operation to be performed adding and deleting edge, adding and deleting vertices, calculated in-degree and out-degree for directed graph and degree of a node in undirected graph, Use any traversal to traverse through the graph)	**14-12-2018**
5.	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	**14-12-2018**

	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. (solve using prims Or kruskal both)	
6.	Implement all the functions of a dictionary (ADT) **OR** directory of phonebook using hashing. Data: Set of (key, value) pairs, Keys are mapped to values, Keys must be comparable, Keys must be unique Standard Operations: Insert(key, value), Find(key), Delete(key) (use linear probing with replacement and without replacement)	**14-12-2018**
7.	The symbol table is generated by compiler. From this perspective, the symbol table is a set of name-attribute pairs. In a symbol table for a compiler, the name is an identifier, and the attributes might include an initial value and a list of lines that use the identifier. Perform the following operations on symbol table: (1) Determine if a particular name is in the table (2) Retrieve the attributes of that name (3) Modify the attributes of that name (4) Insert a new name and its attributes (5) Delete a name and its attributes (use chaining with replacement and without replacement)	**14-12-2018**
8.	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. OR A 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 Height balance tree and find the complexity for finding a keyword.	**14-12-2018**
9.	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,	**14-12-2018**

	h) Subset	
10.	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 employee. If record of student does not exist an appropriate message is displayed. If it is, then the system	**14-12-2018**

Prof. Pujashree Vidap **Prof. Dr. R. B Ingle**

Subject Coordinator **HOCD**

	packages: In this assignment design and use of customized interfaces and packages for a specific application are expected.	
12.	Write a program on template and exception handling in Java: in this assignment multiple templates are to be designed as a pattern and these patterns to be used to take decisions	**14-12-2018**
13.	Write a Java program for the implementation of different data structures using JAVA collection libraries (Standard toolkit library): at least 5 data structures are used to design a suitable application.	**14-12-2018**

	D

Assignment 1:

Aim: To write a program for Dictionary implementation using Binary Search Tree

Prerequisites:

Basic knowledge of linked list, searching

Linked list node deletion, addition, updating

Object oriented programming features

Learning Objectives:

To understand binary search tree implementation

Learning Outcomes

After successful completion of this assignment, students will be able to

- Implement graph using adjacency matrix or adjacency list.
- Create minimum cost spanning tree using Prim's or Kruskal's algorithm.

Concepts related Theory:

In computer science, binary search trees (BST), sometimes called ordered or sorted binary trees, are a particular type of containers: data structures that store "items" (such as numbers, names etc.) in memory. They allow fast lookup, addition and removal of items, and can be used to implement either dynamic sets of items, or lookup tables that allow finding an item by its key (e.g., finding the phone number of a person by name).

Binary search trees keep their keys in sorted order, so that lookup and other operations can use the principle of binary search: when looking for a key in a tree (or a place to insert a new key), they traverse the tree from root to leaf, making comparisons to keys stored in the nodes of the tree and deciding, based on the comparison, to continue searching in the left or right subtrees. On average, this means that each comparison allows the operations to skip about half of the tree, so that each lookup, insertion or deletion takes time proportional to the logarithm of the number of items stored in the tree. This is much better than the linear time required to find items by key in an (unsorted) array, but slower than the corresponding operations on hash tables.

A Binary Search Tree (BST) is a tree in which all the nodes follow the below-mentioned properties –

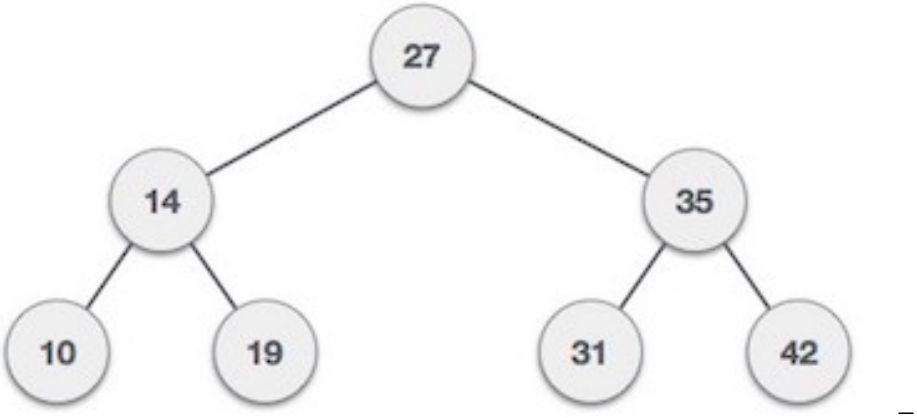
- The left sub-tree of a node has a key less than or equal to its parent node's key.
- The right sub-tree of a node has a key greater than to its parent node's key.

Thus, BST divides all its sub-trees into two segments;

Representation:

BST is a collection of nodes arranged in a way where they maintain BST properties. Each node has a key and an associated value. While searching, the desired key is compared to the keys in BST and if found, the associated value is retrieved.

Following is a pictorial representation of BST



We observe that the root node key (27) has all less-valued keys on the left sub-tree and the higher valued keys on the right sub-tree.

Basic Operations:

Following are the basic operations of a tree −

- Search − Searches an element in a tree.
- Insert − Inserts an element in a tree.
- Deletion − Delete an element in a tree.

Node:- Define a node having some data, references to its left and right child nodes.

Search Operation:

Whenever an element is to be searched, start searching from the root node. Then if the data is less than the key value, search for the element in the left subtree. Otherwise, search for the element in the right subtree. Follow the same algorithm for each node.

Algorithm:

```
struct node* search(int data){
  struct node *current = root;
  printf("Visiting elements: ");
```

```
    while(current->data != data){
      if(current != NULL) {
        printf("%d ",current->data);
        //go to left tree
        if(current->data > data){
          current = current->leftChild;
        }//else go to right tree
        else {
          current = current->rightChild;
        }
        //not found
        if(current == NULL){
          return NULL;
        }
      }
    }
    return current;
  }
```

Insert Operation:

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.

Algorithm:

```
void insert(int data) {
  struct node *tempNode = (struct node*) malloc(sizeof(struct node));
```

```
  struct node *current;
  struct node *parent;
  tempNode->data = data;
  tempNode->leftChild = NULL;
  tempNode->rightChild = NULL;
  //if tree is empty
  if(root == NULL) {
    root = tempNode;
  } else {
    current = root;
    parent = NULL;
    while(1) {
      parent = current;
      //go to left of the tree
      if(data < parent->data) {
        current = current->leftChild;
        //insert to the left
        if(current == NULL) {
          parent->leftChild = tempNode;
          return;
        }
      }//go to right of the tree
      else {
        current = current->rightChild;
        //insert to the right
```

```
        if(current == NULL) {
          parent->rightChild = tempNode;
          return;
        }
      }
    }
  }
}
```

Deletion:

There are three possible cases to consider:

- Deleting a node with no children: simply remove the node from the tree.
- Deleting a node with one child: remove the node and replace it with its child.
- Deleting a node with two children: call the node to be deleted N. Do not delete N. Instead, choose either its in-order successor node or its in-order predecessor node, R. Copy the value of R to N, then recursively call delete on the original R until reaching one of the first two cases. If you choose in-order successor of a node, as right sub tree is not NIL (Our present case is node has 2 children), then its in-order successor is node with least value in its right sub tree, which will have at a maximum of 1 sub tree, so deleting it would fall in one of the first 2 cases.

Algorithm:

```
bool BinarySearchTree::remove(int value) {
    if (root == NULL)
        return false;
   else {
        if (root->getValue() == value) {
            BSTNode auxRoot(0);
            auxRoot.setLeftChild(root);
```

```
            BSTNode* removedNode = root->remove(value, &auxRoot);
            root = auxRoot.getLeft();
            if (removedNode != NULL) {
                delete removedNode;
                return true;
            } else
                return false;
        } else {
            BSTNode* removedNode = root->remove(value, NULL);
            if (removedNode != NULL) {
                delete removedNode;
                return true;
            } else
                return false;
           }
    }
}
BSTNode* BSTNode::remove(int value, BSTNode *parent) {
    if (value < this->value) {
        if (left != NULL)
            return left->remove(value, this);
        else
            return NULL;
    } else if (value > this->value) {
        if (right != NULL)
            return right->remove(value, this);
        else
```

```
            return NULL;
        } else {
            if (left != NULL && right != NULL) {
                this->value = right->minValue();
                return right->remove(this->value, this);
            } else if (parent->left == this) {
                parent->left = (left != NULL) ? left : right;
                return this;
            } else if (parent->right == this) {
                parent->right = (left != NULL) ? left : right;
                return this;
            }
        }
    }
    int BSTNode::minValue() {
        if (left == NULL)
            return value;
        else
            return left->minValue();}
```

Threaded Binary Tree:

Threaded binary tree is a binary tree variant that allows fast traversal: given a pointer to a node in a threaded tree, it is possible to cheaply find its in-order successor (and/or predecessor).

Binary trees, including (but not limited to) binary search trees and their variants, can be used to store a set of items in a particular order. For example, a binary search tree assumes data items are somehow ordered and maintain this ordering as part of their insertion and deletion algorithms. One useful operation on such a tree is traversal: visiting the items in the order in which they are stored (which matches the underlying ordering in the case of BST).

Algorithm:

1. For the current node check whether it has a left child which is not there in the visited list. If it has then go to step-2 or else step-3.
2. Put that left child in the list of visited nodes and make it your current node in consideration. Go to step-6.
3. Print the node and If node has right child then go to step 4 else go to step 5.
4. Make right child as current node.
5. if there is a thread node then make it the current node.
6. if all nodes have been printed then END else go to step 1.

Review Questions:

1. What is binary search tree?
2. What are threaded binary tree?
3. How do you find the depth of a binary tree?
4. Explain pre-order and in-order tree traversal?
5. Define threaded binary tree. Explain its common uses.
6. Explain implementation of traversal of a binary tree.
7. Explain implementation of deletion from a binary tree.

ASSINGMENT NO.	3
TITLE	Threaded binary tree
PROBLEM STATEMENT /DEFINITION	Convert given binary tree into inordered and preordered threaded binary tree. Analyze time and space complexity of the algorithm.
OBJECTIVE	To understand construction of inordered and preordered Threaded binary tree from given binary tree.
OUTCOME	At the end of this assignment students will able to construct threaded binary tree and able to perform basic operations on Threaded binary tree.
S/W PACKAGES AND HARDWARE APPARATUS USED	• (64-bit)64-BIT Fedora 17 or latest 64-BIT Update of Equivalent Open source OS • Programming Tools (64-Bit) Latest Open source update of Eclipse Programming frame work, TC++, GTK++.
REFERENCES	E. Horowitz S. Sahani, D. Mehata, "Fundamentals of data structures in C++", Galgotia Book Source, New Delhi, 1995, ISBN: 1678298 Sartaj Sahani, —Data Structures, Algorithms andApplications in C++‖, Second Edition, University Press, ISBN:81-7371522 X.
INSTRUCTIONS FOR WRITING JOURNAL	1. Date 2. Assignment no. 3. Problem definition 4. Learning objective 5. Learning Outcome 6. Concepts related Theory 7. Algorithm

	8. Test cases 9. Conclusion/Analysis

Prerequisites:

Object oriented programming,features and basic concepts of data structures.

Concepts related Theory:

Binary Tree: is a special data structure used for data storage purposes. A binary tree has a special condition that each node can have a maximum of two children. A binary tree has the benefits of both an ordered array and a linked list as search is as quick as in a sorted array and insertion or deletion operation are as fast as in linked list. A binary tree is either empty (represented by a null pointer), or is made of a single node, where the left and right pointers each point to a binary tree.

In a binary search tree, there are many nodes that have an empty left child or empty right child or both. You can utilize these fields in such a way so that the empty left child of a node points to its inorder predecessor and empty right child of the node points to its inorder successor

Threaded Binary Tree: A binary tree is threaded by making all right child pointers that would normally be null point to the inorder successor of the node (if it exists), and all left child pointers that would normally be null point to the inorder predecessor of the node**.**

It is also possible to discover the parent of a node from a threaded binary tree, without explicit use of parent pointers or a stack. This can be useful where stack space is limited, or where a stack of parent pointers is unavailable.

One way threading:-A thread will appear in the right field of a node and will point to the next node in the inorder traversal.

Two way threading:- A thread will also appear in the left field of a node and will point to the preceding node in the inorder traversal.

Types of threaded binary tree

Single threaded: Each node is threaded towards either the in-order predecessor or successor (left or right) means all right null pointers will point to inorder successor or all left null pointers will point to inorder predecessor.

Double threaded: Each node is threaded towards both the in-order predecessor and successor (left and right) means all right null pointers will point to inorder successor AND all left null pointers will point to inorder predecessor.

Threaded binary search tree: Threaded binary search tree is BST in which all right pointers of node which point to NULL are changed and made to point to inorder successor current node (These are called as single threaded trees). In completely threaded tree (or double threaded trees), left pointer of node which points to NULL is made to point to inorder predecessor of current node if inorder predecessor exists.

Now, there is small thing needs to be taken care of. A right or left pointer can have now two meanings : One that it points to next real node, second it is pointing inorder successor or predecessor of node, that means it is creating a thread. To store this information, we added a bool in each node, which indicates whether pointer is real or thread.

Approach:

1. we can do the inorder traversal and store it in some queue. Do another inorder traversal and where ever you find a node whose right reference is NULL , take the front element from the queue and make it the right of the current node.

2.Now we will see the solution which will convert binary tree into threaded binary tree in one single traversal with no extra space required.

- Do the reverse inorder traversal, means visit right child first.
- In recursive call, pass additional parameter, the node visited previously.
- whenever you will find a node whose right pointer is NULL and previous visited node is not NULL then make the right of node points to previous visited node and mark the boolean right threaded as true.
- Important point is whenever making a recursive call to right subtree, do not change the previous visited not and when making a recursive call to left subtree then pass the actual previous visited node.

Algorithm:

```
public void convert(Node root){

inorder(root, null);

}

public void inorder(Node root, Node previous){

f(root==null){
```

```
return;
}else{
inorder(root.right, previous);
if(root.right==null && previous!=null){
root.right = previous;
root.rightThread=true;
}
inorder(root.left, root);
}
}
public void print(Node root){
//first go to most left node
Node current = leftMostNode(root);
//now travel using right pointers
while(current!=null){
System.out.print(" " + current.data);
//check if node has a right thread
if(current.rightThread)
current = current.right;
current = current.right;
else // else go to left most node in the right subtree
current = leftMostNode(current.right);
}
System.out.println()
}
```

```
public Node leftMostNode(Node node){

if(node==null){

return null;

}else{

while(node.left!=null)

node = node.left

}

return node;

}

}
```

Conclusion: After successfully completing this assignment, Students have learned construction of Threaded binary tree and various operations on Threaded binary tree.

Review Questions:

1. What is binary tree? Explain its uses.
2. How do you find the depth of a binary tree?
3. Explain pre-order and in-order tree traversal.
4. Define threaded binary tree. Explain its common uses.
5. Explain implementation of traversal of a binary tree.
6. Why do we need Threaded binary search trees ?
7. How to created a threaded BST?
8. How to convert a Given Binary Tree to Threaded Binary Tree?
9. What are the drawbacks of bi-threaded trees? Are single threaded trees enough to do traversals on the tree? Justify.
10. Did you require stacks to do get the output along with threads? Justify.
11. Justify that only single threads are required to traverse the tree efficiently.
12. If a tree is bi threaded for postorder traversal, and then is it enough to traverse it in in-order and post-order. Justify

ASSINGMENT NO.	**4**
TITLE	Adjacency list representation of the graph or use adjacency matrix representation of the graph.
PROBLEM STATEMENT /DEFINITION	There are flight paths between cities. If there is a flight between city A and city B then there is an edge between the cities. The cost of the edge can be the time that flight takes to reach city B from A, or the amount of fuel used for the journey. Represent this as a graph. The node can be represented by airport name or name of the city. Use adjacency list representation of the graph or use adjacency matrix representation of the graph. Justify the storage representation used.(Operation to be performed adding and deleting edge, adding and deleting vertices, calculated in-degree and out-degree for both directed and undirected graph)
OBJECTIVE	1. Define a graph (undirected and directed), a vertex/node, and an edge. 2. Given the figure of a graph, give its set of vertices and set of edges. 3. Given the set of vertices and set of edges of a graph, draw a figure to show the graph. 4. Given a graph, show its representations using an adjacency list and adjacency matrix. Also give the space required for each of those representations. 6. Given a DAG, show the steps in topologically sorting the vertices, and give the time complexity of the algorithm.
OUTCOME	The adjacency list easily find all the links that are directly connected to a particular vertex i.e. edge between the cities.
S/W PACKAGES AND HARDWARE APPARATUS USED	• (64-bit)64-BIT Fedora 17 or latest 64-BIT Update of Equivalent Open source OS • Programming Tools (64-Bit) Latest Open source update of Eclipse Programming frame work, TC++, GTK++.
REFERENCES	E. Horowitz S. Sahani, D. Mehata, "Fundamentals of data structures in C++", Galgotia Book Source, New Delhi, 1995, ISBN: 1678298 Sartaj Sahani, —Data Structures, Algorithms andApplications in C++‖, Second Edition, University Press, ISBN:81-7371522 X.

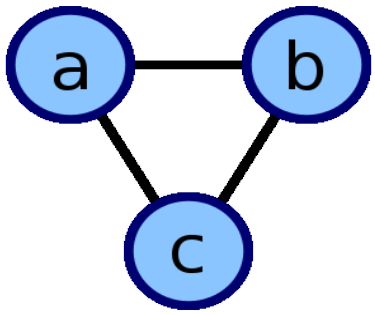
INSTRUCTIONS FOR WRITING JOURNAL	1. Date 2. Assignment no. 3. Problem definition 4. Learning objective 5. Learning Outcome 6. Concepts related Theory 7. Algorithm 8. Test cases 9. Conclusion/Analysis

Prerequisites:

Object oriented programming,features and basic concepts of data structures.

Concepts related Theory:

An adjacency list representation for a graph associates each vertex in the graph with the collection of its neighboring vertices or edges. There are many variations of this basic idea, differing in the details of how they implement the association between vertices and collections, in how they implement the collections, in whether they include both vertices and edges or only vertices as first class objects, and in what kinds of objects are used to represent the vertices and edges.



This undirected cyclic graph can be described by the three unordered lists {b, c}, {a, c}, {a, b}

The graph pictured above has this adjacency list representation:		
a	adjacent to	bc
b	adjacent to	ac
c	adjacent to	ab

OPERATION:

The main operation performed by the adjacency list data structure is to report a list of the neighbors of a given vertex. this can be performed in constant time per neighbor. In other words, the total time to report all of the neighbors of a vertex *v* is proportional to the degree of *v*.

It is also possible, but not as efficient, to use adjacency lists to test whether an edge exists or does not exist between two specified vertices. In an adjacency list in which the neighbors of each vertex are unsorted, testing for the existence of an edge may be performed in time proportional to the minimum degree of the two given vertices, by using a sequential search through the neighbors of this vertex. If the neighbors are represented as a sorted array, binary search may be used instead, taking time proportional to the logarithm of the degree.

Trade-offs:

The main alternative to the adjacency list is the adjacency matrix, a matrix whose rows and columns are indexed by vertices and whose cells contain a Boolean value that indicates whether an edge is present between the vertices corresponding to the row and column of the cell. For a sparse graph (one in which most pairs of vertices are not connected by edges) an adjacency list is significantly more space-efficient than an adjacency matrix (stored as an array): the space usage of the adjacency list is proportional to the number of edges and vertices in the graph, while for an adjacency matrix stored in this way the space is proportional to the square of the number of vertices. However, it is possible to store adjacency matrices more space-efficiently, matching the linear space usage of an adjacency list, by using a hash table indexed by pairs of vertices rather than an array.

The other significant difference between adjacency lists and adjacency matrices is in the efficiency of the operations they perform. In an adjacency list, the neighbors of each vertex may be listed efficiently, in time proportional to the degree of the vertex. In an adjacency matrix, this operation takes time proportional to the number of vertices in the graph, which may be significantly higher than the degree. On the other hand, the adjacency matrix allows testing whether two vertices are adjacent to each other in constant time; the adjacency list is slower to support this operation.

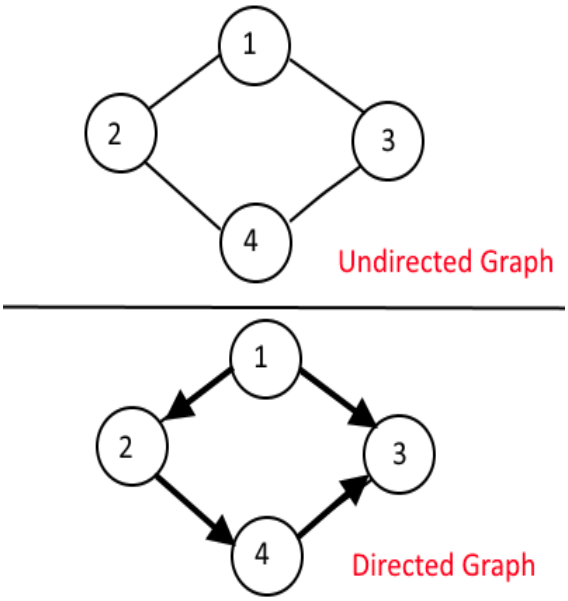
What is Graph ?

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.



Now how do we represent a Graph, There are two common ways to represent it:

- 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

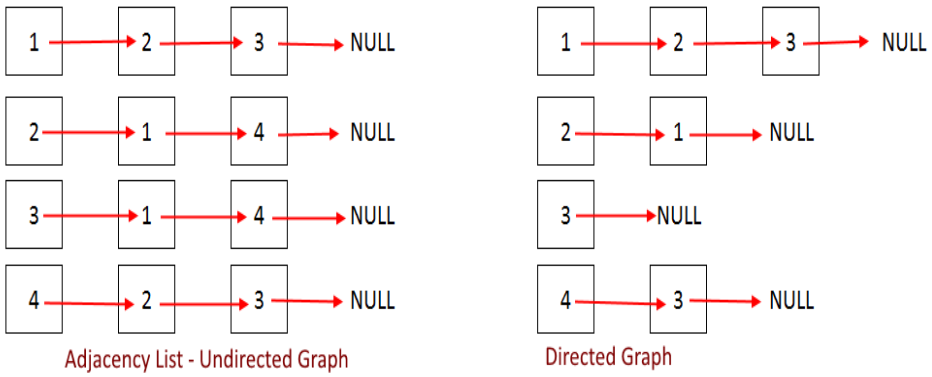
adjMaxtrix[i][j] = 1 when there is edge between Vertex i and Vertex j, else 0.

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.



Algorithm:

class Edge

{
private:
Vertex *source;
Vertex *destination;
int distance;
public:
Edge(Vertex *s, Vertex *d, int dist)
{
source = s;
destination = d;
distance = dist;
}

```
Vertex *getSource()
{
return source;
}
Vertex * getDestination()
{
return destination;
}
int getDistance()
{
return distance;
}
};
class Vertex
{
private : string city;
vector<Edge> edges;//vector edges created for edges
public:
Vertex(string name)

{
city = name;
}
void addEdge(Vertex *v, int dist)
{
Edge newEdge(this,v,dist);// creating object for edge
edges.push_back(newEdge);//creating adjusting list
}
void showEdge()
{
cout<<"From"<<city<<"to"<<endl;
for(int i=0; i<(int)edges.size();i++)
{
Edge
e = edges[i];
cout<<e.getDestination()
-
>getCity()<<"requires"<<e.getDistance()<<"hrs"<<endl;
}
cout<<endl;
}
```

```
string getCity()
{
return city;
}
vector<Edge> getEdges()
{
return edges;
}
};
class Graph
{
vector<Vertex*> v;
public:
Graph(){}
void insert(Vertex *val)
{
v.push_back(val);
}
void Display()
{
for(int i=0;i<(int)v.size();i++)
{
// v[i].showEdge();
v[i]
-
>showEdge();
}
}
};
int main()
{
Graph g;
// crea
ting verticex ot nodes for each city
Vertex v1 = Vertex("Mumbai");
Vertex v2 = Vertex("Pune");
Vertex v3 = Vertex("Kolkata");
Vertex v4 = Vertex("Delhi");
//creating pointers to nodes
Vertex *vptr1 = &v1;
Vertex *vptr2 = &v2;
```

```
Vertex *vptr3 = &v3;
Vertex *vptr4 = &v4;
//attaching the nodes by adding edges
v1.addEdge(vptr4,2);
v2.addEdge(vptr1,1);
v3.addEdge(vptr1,3);
v4.addEdge(vptr2,2);
v4.addEdge(vptr3,3);
//cretaing graph
g.insert(vptr1);
g.insert(vpt
r2);
g.insert(vptr3);
g.insert(vptr4);
cout<<"
\
n
\
t Displaying City Transport Map Using Adjacency List"<<endl;
g.Display();
return 1;
}
```

TEST CASES:

Check The is there adjacency list find all the edges that are directly connected to a cities or not.

Conclusion:

Thus we have studies adjacency list representation of the graph successfully for cities.

Review Questions:

1. What is Graph? Explain its uses.
2. Explain Adjacency List.
3. Explain Adjacency Matrix.
4. What is the difference between undirected and directed graph?
5. Explain Sparse graph.

D

Assignment 5

Aim: To write a program for Graph creation and find its minimum cost using Prim's or Kruskal's algorithm.

Prerequisites:

Basic knowledge of graph

Graph representation method (Adjacency matrix or Adjacency list)

Object oriented programming features

Learning Objectives

To understand concept of graph and minimum cost spanning tree.

To understand minimum cost spanning tree algorithms.

Learning Outcomes

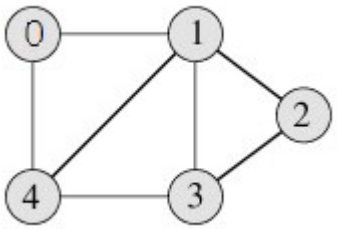
After successful completion of this assignment, students will be able to

- Implement graph using adjacency matrix or adjacency list.
- Create minimum cost spanning tree using Prim's or Kruskal's algorithm.

Concepts related Theory:

- **Representation of Graph**

Following is an example undirected graph with 5 vertices.



 Using Adjacency Matrix

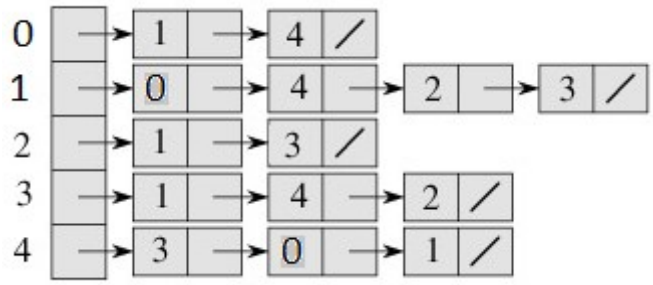
Adjacency Matrix is a 2D array of size V x V where V is the number of vertices in a graph. Let the 2D array be adj[][], a slot adj[i][j] = 1 indicates that there is an edge from vertex i to vertex j. Adjacency matrix for undirected graph is always symmetric. Adjacency Matrix is also used to represent weighted graphs. If adj[i][j] = w, then there is an edge from vertex i to vertex j with weight w.

The adjacency matrix for the above example graph is:

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

 Using Adjacency list

An array of linked lists is used. Size of the array is equal to number of vertices. Let the array be array[]. An entry array[i] represents the linked list of vertices adjacent to the ***i***th vertex. This representation can also be used to represent a weighted graph. The weights of edges can be stored in nodes of linked lists. Following is adjacency list representation of the above graph.



- **Minimum Spanning Tree:**

Give a graph G = (V, E), the minimum spanning tree (MST) is a weighted graph G' = (V, E') such that:

- E' Í E
- G' is connected
- G' has the minimum cost

A minimum spanning tree (MST) or minimum weight spanning tree is a subset of the edges of a connected, edge-weighted undirected graph that connects all the vertices together, without any cycles and with the minimum possible total edge weight. That is, it is a spanning tree whose sum of edge weights is as small as possible. More generally, any undirected graph (not necessarily connected) has a minimum spanning forest, which is a union of the minimum spanning trees for its connected components. There are quite a few use cases for minimum spanning trees. One example would be a telecommunications company which is trying to lay out cables in new neighborhood.

 Prim's Algorithm

Step 1: Select any vertex

Step 2: Select the shortest edge connected to that vertex.

Step 3: Select the shortest edge connected to any vertex already connected

Step 4: Repeat step 3 until all vertices have been connected

 Kruskal's Algorithm:

Step 1: Enter number of cities (vertices in graph).

Step 2: Enter the cost of connectivity between each pair of cities (edges in graph).

Step 3: Initialize cost_of_connectivity to 0.

Step 4: Sort all the edges in non-decreasing order of their cost.

Step 5: Pick the smallest cost edge

Step 6: Check if it forms a cycle with the already include edges in the minimum spanning tree

Step 7: If cycle is not formed, include this edge in MST, else discard it

Step 8: Add weight of the selected edge to the cost_of_connectivity.

Step 9: Repeat step 5 ,6, 7 until there are (v-1) edges in the graph.

Step 10: cost_of_connectivity will have minimum cost in the end

Algorithm:

- **Prim's Algorithm**

    // input: a graph G

    // output: E: a MST for G

1. Select a starting node, v
2. T ß {v} //the nodes in the MST
3. E ß {} //the edges in the MST
4. While not all nodes in G are in the T do

        Choose the edge v' in G − T such that there is a v in T:

        weight (v,v') is the minimum in

        {weight(u,w) : w in G − T and u in T}

        T ß T È {v'}

E ß E È {(v,v')}

5. return E

- **Kruskal's Algorithm:**

// input: a graph G with n nodes and m edges

// output: E: a MST for G

- EG[1..m] Sort the m edges in G in increasing weight order
- E {} //the edges in the MST
- i 1 //counter for EG
- While |E|<n-1 do

  if adding EG[i] to E does not add a cycle then

  E E {EG[i]}

  i i + 1
- return E

Conclusion: We have successfully calculated total minimum cost of graph using minimum spanning tree algorithm.

Review Questions:

1) What is minimum spanning tree?

2) What are the algorithms to find minimum spanning tree?

3) What is Time and space complexity of the algorithm used?

4) What is adjacency list and adjacency matrix?

5) Difference between adjacency list and adjacency matrix.?

6) Draw and compare graph using Prim's and Kruskal's algorithm?

7) Explain steps in kruskal's algorithm ?

8) Explain steps in prim's algorithm?

9) What are Applications of minimum spanning tree?

10) Explain number of edges in any Minimum Spanning Tree?

ASSINGMENT NO.	6
TITLE	The Dictionary ADT
PROBLEM STATEMENT /DEFINITION	Implement all the functions of a dictionary (ADT) using hashing. Data: Set of (key, value) pairs, Keys are mapped to values, Keys must be comparable, Keys must be unique Standard Operations: Insert(key, value), Find(key), Delete(key)
OBJECTIVE	To understand implementation of all the functions of a dictionary (ADT) and standard operations on Dictionary.
OUTCOME	At the end of this assignment students will able to perform standard operations on Dictionary ADT.
S/W PACKAGES AND HARDWARE APPARATUS USED	• (64-bit)64-BIT Fedora 17 or latest 64-BIT Update of Equivalent Open source OS • Programming Tools (64-Bit) Latest Open source update of Eclipse Programming frame work, TC++, GTK++.
REFERENCES	• E. Horowitz S. Sahani, D. Mehata, "Fundamentals of data structures in C++", Galgotia Book Source, New Delhi, 1995, ISBN: 1678298 • Sartaj Sahani, —Data Structures, Algorithms and Applications in C++‖, Second Edition, University Press, ISBN:81-7371522 X.
INSTRUCTIONS FOR WRITING JOURNAL	1. Date 2. Assignment no. 3. Problem definition 4. Learning objective 5. Learning Outcome 6. Concepts related Theory 7. Algorithm 8. Test cases 9. Conclusion/Analysis

Prerequisites:

- Basic knowledge of Dictionary and Hashing.
- Object oriented programming,features and basic concepts of data structures.

Concepts related Theory:

The Dictionary ADT: A dictionary is an ordered or unordered list of key-element pairs, where keys are used to locate elements in the list.

Dictionary is a data structure, which is generally an association of unique keys with some values. One may bind a value to a key, delete a key (and naturally an associated value) and look up for a value by the key. Values are not required to be unique.

Example: consider a data structure that stores bank accounts; it can be viewed as a dictionary, where account numbers serve as keys for identification of account objects.

A Dictionary (also known as Table or Map) can be implemented in various ways: using a list, binary search tree, hash table, etc.

In each case: the implementing data structure has to be able to hold key-data pairs and able to do insert, find, and delete operations paying attention to the key.

Hashing: Hashing is a method for directly referencing an element in a table by performing arithmetic transformations on keys into table addresses. This is carried out in two steps:

1. Computing the so-called hash function H: K -> A.
2. Collision resolution, which handles cases where two or more different keys hash to the same table address.

Implementation of Hash table:

Hash tables consist of two components: a *bucket array* and a *hash function*.

A hash table is a collection of items which are stored in such a way as to make it easy to find them later. Each position of the hash table, often called a slot, can hold an item and is named by an integer value starting at 0. For example, we will have a slot named 0, a slot named 1, a slot named 2, and so on.

Consider a dictionary, where keys are integers in the range [0, N-1]. Then, an array of size N can be used to represent the dictionary. Each entry in this array is thought of as a “bucket”. An element *e* with key *k* is inserted in A[k]. Bucket entries associated with keys not present in the dictionary contain a special NO_SUCH_KEY object. If the dictionary contains elements with the same key, then two or more different elements may be mapped to the same bucket of A. In this case, we say that a *collision* between these elements has occurred. One easy way to deal with collisions is to allow a sequence of elements with the same key, k, to be stored in A[k].

Assuming that an arbitrary element with key k satisfies queries findItem(k) and removeItem(k), these operations are now performed in O(1) time, while insertItem(k, e) needs only to find where on the existing list A[k] to insert the new item, e. The drawback of this is that the size of the bucket array is the size of the set from which key are drawn,which may be huge.

Algorithm:

HashNode Class Declaration:

```
class HashNode

{
public:
int key;
int value;
HashNode* next;
HashNode(int key, int value)
{
this->key = key;
this->value = value;
this->next = NULL;
}
};
```

Insertion:

```
void Insert(int key, int value)
{
int hash_val = HashFunc(key);
HashNode* prev = NULL;
HashNode* entry = htable[hash_val];
while (entry != NULL)
{
prev =entry;
entry = entry->next;
}
if (entry == NULL)
{
entry = new HashNode(key, value);
if (prev == NULL)
{
```

```
htable[hash_val] = entry;
}
else
{
prev->next = entry;
}
}
else
{
entry->value = value;
}
}
```

Deletion:

```
void Remove(int key)
{
int hash_val = HashFunc(key);
HashNode* entry = htable[hash_val];
HashNode* prev = NULL;
if (entry == NULL || entry->key != key)
{
cout<<"No Element found at key "<<key<<endl;
return;
}
while (entry->next != NULL)
{
prev = entry;
entry = entry->next;
}
if (prev != NULL)
{
prev->next = entry->next;
}
delete entry;
cout<<"Element Deleted"<<endl;
}
```

Search:

```
int Search(int key)
{
```

```
bool flag = false;
int hash_val = HashFunc(key);
HashNode* entry = htable[hash_val];
while (entry != NULL)
{
if (entry->key == key)
{
cout<<entry->value<<" ";
flag = true;
}
entry = entry->next;
}
if (!flag)
return -1;
}
};
```

Conclusion: After successfully completing this assignment, Students have learned implementation of Dictionary(ADT) using Hashing and various Standard operations on Dictionary ADT .

Review Questions:

- In what ways is a dictionary similar to an array? In what ways are they different?
- What does it mean to hash a value? What is a hash function?
- What is a perfect hash function?
- What is a collision of two values?
- What does it mean to probe for a free location in an open address hash table?
- What is the load factor for a hash table?
- Why do you not want the load factor to become too large?
- Can you come up with a perfect hash function for the names of the week? The names of the months? The names of the planets?

- How to define a good hash function?
- What is the best definition of a collision in a hash table?
- Describe in reasonable detail a way to implement the Dictionary ADT such that the **insertItem**, **findItem**, and **removeItem** methods would all run in $O(1)$ time, assuming that all of the keys associated with elements in the structure are integers in the range

ASSINGMENT NO.	7
TITLE	To write a program for implementation of symbol table. And perform various operations.
PROBLEM STATEMENT /DEFINITION	The symbol table is generated by compiler. From this perspective, the symbol table is a set of name-attribute pairs. In a symbol table for a compiler, the name is an identifier, and the attributes might include an initial value and a list of lines that use the identifier. Perform the following operations on symbol table: (1) Determine if a particular name is in the table (2) Retrieve the attributes of that name (3) Modify the attributes of that name (4) Insert a new name and its attributes (5) Delete a name and its attributes
OBJECTIVE	1) To understand concept of symbol table. 2) Why symbol table is needed.
OUTCOME	1) Use of symbol table 2) Various methods of implementing symbol table.
S/W PACKAGES AND HARDWARE APPARATUS USED	• 64-bit Open source Linux or its derivative. • Open Source C++ Programming tool like G++/GCC.
REFERENCES	• Data structures in C++ by Horowitz, Sahni.

INSTRUCTIONS FOR WRITING JOURNAL	1. Date 2. Assignment no. 3. Problem definition 4. Learning objective 5. Learning Outcome 6. Concepts related Theory 7. Algorithm 8. Test cases 10. Conclusion/Analysis

Assignment 7

- Aim: To write a program for implementation of symbol table. And perform various operations.

Prerequisites:

- Basic knowledge of array implementation
- Linked list implementation
- Basic knowledge of binary search tree
- Object oriented programming features

Learning Objectives

- To understand concept of symbol table and its use.
- To perform basic operations on symbol table.

Learning Outcomes

After successful completion of this assignment, students will be able to

- Implement symbol table.
- Become familiar with compiler working.

Concepts Related Theory:

Symbol Table:

In computer science, a symbol table is a data structure used by a language translator such as a compiler or interpreter, where each identifier (a.k.a. symbol) in a program's source code is associated with information relating to its declaration or appearance in the source.

A symbol table may only exist during the translation process, or it may be embedded in the output of that process, such as in an ABI object file for later exploitation. For example, it might be used during an interactive debugging session, or as a resource for formatting a diagnostic report during or after execution of a program. And used only in compilers mostly.

Symbol table is used to store information related to various entities like as function name, variable name, objects, classes, interfaces, etc. When identifiers are found, they will be entered into a symbol table, which will hold all relevant information about identifiers. Symbol table is type of data structure that captures scope information. One symbol table for each scope is used. It stores all entities in structured form at one place. By using symbol table it checks if variable is declared or not. It is also used for syntax checking.

A symbol table is simply a table which can be either linear or a hash table. It maintains an entry for each name in the format as <*symbol name*, *type*, *attribute*>. For example table has to store information about following variable declaration as *static int interest;* then it should store the entry such as <*interest, int, static*> The attribute clause contains the entries related to the name.

There are two types of symbol tables. Static symbol table and Dynamic symbol table.

Static symbol tables are tree tables. They are implemented when symbols are known in advance and no addition and deletion is allowed.

Dynamic symbol tables are used when symbol are not known in advance and insertion and deletion can be done any time.

Symbol Table Implementation Methods:

- Unordered Array Implementation
- Ordered Array Implementation
- Unordered Or Ordered List
- Binary Search Trees
- Balanced Binary Search Trees

- Hashing

Unordered Array Implementation:

It maintains arrays of keys and values. Instance variables are used to store data. Array *keys[]* holds the keys and *vals[]* holds the values, integer *N* holds the number of entries.

Ordered Array Implementation:

In ordered array implementation, keys are comparable to each other. Ordered array implementation for symbol table used because it provides ordered iteration. Binary search can speed up search.

Ordered or Unordered Linked-list Implementation:

Maintain a linked list with keys and values. Advantage of keeping linked list in order for comparable key, support ordered iterator and cuts search or insert time in half.

Binary Search Tree:

Keep data stored in parent to child format and sorted. Insertion, deletion, etc. operation can be done faster than other compared methods.

Balanced Binary Search Tree:

Space overhead is directly proportional to the number of items in the table. Insertion takes time compared to other methods of implementation.

Hash Table:

We can work faster in hashing methods. Hashing table method used into most compilers. Complexity is O(1) for hashing table.

Review Questions:

1. What method used into symbol table searching?
2. Which method used mostly in symbol table implementation?
3. How key and values are stored into symbol table?

4. Difference between symbol table and hash table?
5. What data is stored into symbol table?

ASSINGMENT NO.	10
TITLE	Sequential file to maintain the data.
PROBLEM STATEMENT /DEFINITION	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 employee. 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 main the data.
OBJECTIVE	1.Undestand how sequential files are organized. 2.Understand the processing limitations imposed by an unordered Sequential file 3..Be able to add, delete and update records in an ordered Sequential file
OUTCOME	At the end of this assignment students will able to contain student record in Sequntial file and also able to perform the add,delete operation successfully and system will be displays student records.
S/W PACKAGES AND HARDWARE APPARATUS USED	• (64-bit)64-BIT Fedora 17 or latest 64-BIT Update of Equivalent Open source OS • Programming Tools (64-Bit) Latest Open source update of Eclipse Programming frame work, TC++, GTK++.
REFERENCES	http://www.pembinatrails.ca/fortrichmondcollegiate/compsci/java/unit7/page3.htm
INSTRUCTIONS FOR WRITING JOURNAL	• Date • Assignment no. • Problem definition • Learning objective

	• Learning Outcome • Concepts related Theory • Algorithm 3. Test cases 9. Conclusion/Analysis

PREREQUISITES:

- Introduction to Sequential files
- java programming,features and basic concepts of data structures.
- Add,delete operations.

CONCEPT RELATED THEORY:

sequential file:

A file that contains records or other elements that are stored in a chronological order based on account number or some other identifying data. In order to locate the desired data, sequential files must be read starting at the beginning of the file. A sequential file may be stored on a sequential access device such as magnetic tape or on a direct access device such as magnetic disk. Contrast with random file.

A sequential data file is a straight text file, like the type of file you can create with a program like Notepad.

When you want to write code that uses a data file, you must first import the oi package of the

java language:

```
import java.io.*;
```

Next, you must create a File object that represents the data file you want to read data from or write data to:

```
File myFile = new File( "thedatafile.txt" );
```

Note: "thedatafile.txt" must be the name of a sequential file that exists in the same folder as your class file. Otherwise, you must include the complete path to the data file in the call to the File class' constructor.

Next, test to ensure that the file is present and that it contains data. This can be done using two methods of the File class: exists() and length:

```
if (myFile.exists() && myFile.length != 0)
{
    // the file exists and contains data ...
}
```

Now, you must create a BufferedReader object to access the data on the file. The BufferedReader class contains methods for getting data from a data file. The BufferedReader constructor requires a FileReader object that contains the reference to the data file you want to read.

```
BufferedReader in = new BufferedReader( new FileReader( myFile ) );
```

Since the FileReader object is just there to provide an interface between the BufferedReader object and the File object, we create it anonymously.

Now we're ready to start using the readLine() method of the BufferedReader class to read the data, one line at a time, from the file and store the values in variables:

```
String lastName = in.readLine();
String firstName = in.readLine();
String phoneNumber = in.readLine();
```

But the code can not just be written as I've described here. Since so many unexpected things can happen when your program tries to read a data file, and most of these unexpected things can crash a computer, Java forces you to write your code so that it catches errors when they occur.

To do this, you must use a try ... catch statement. The try clause in the try ... catch statement contains the code you want your program to do, while the catch clause contains code that handles errors if they occur.

```
try
{
```

```
    // code that reads data from the data file
}
catch (Exeption e)
{
    // code that deals with errors when they occur
}
```

So, written as a method, here's how you would read data from a sequential file:

```
private void readFile( String theFile )
{
    File myFile = new File( theFile );
    if (myFile.exists() && myFile.length() != 0)
        try
        {
            BufferedReader in = new BufferedReader(
                                new FileReader( myFile ) );
            String lastName, firstName, phoneNumber;
            firstName = in.readLine();
            lastName = in.readLine();
            phoneNumber = in.readLine();
        }
        catch( Exception e )
        {    System.out.println(e);    }
}
```

If your data file contains more than one record, you will likely store the contents of the file in an array or an ArrayList object. In this case, you will place the calls to the readLine() method in a loop that stops once a null value is read in (that is, there's nothing left to read on the file):

```
do
{
    firstName[numNames] = in.readLine();
    lastName[numNames] = in.readLine();
    phoneNumber[numNames] = in.readLine();
    if (phoneNumber[numNames] != null)
        numNames++;
}while phoneNumber[numNames] != null)
```

APPROACH:

- Understand the processing limitations imposed by an unordered *Sequential file*; Be able to *add*, *delete* and update *records* in an ordered *Sequential file*.

- *Sequential File* Organization - It is one of the simple *methods* of file organization in ... Here each file/*records* are stored one after the other in a sequential manner. ... Each time any *insert*/update/ *delete* transaction is performed, file is sorted.

ALGORITHM:

Processing Ordered Sequential Files:

An ordered Sequential file, is a file ordered upon some key field. The ordering of the records in the file makes it possible to process an ordered file in ways that are not available to us with unordered files. While it is not really possible to apply batch updates or deletes to an unordered file these are possible with ordered files.

ALGORITHM:

1. Inserting records into an ordered Sequential file.

2. Self Assessment questions.

(like In the program, no allowance has been made for when the key fields are equal. What would it mean if the keys were equal?

3. Add or Deleting records from an ordered Sequential file.

4. Updating records in an ordered Sequential file.

5. If record does not exist an appropriate message is displayed,If it is then display record.

CONCLUSION:

Thus we have successfully studied Sequential file to main the student data to add, delete information of student.

Review Questions:

1.What is the sequential file?

2.What is sequential data?

3.What is a sequential access storage device?

4.How to add record in sequential file?

5. How to add record in sequential file?

ASSINGMENT NO.	11
TITLE	Interfaces and packages
PROBLEM STATEMENT /DEFINITION	Write a Java program which will demonstrate a concept of Interfaces and packages: In this assignment design and use of customized interfaces and packages for a specific application are expected. Example: **Write a java program in which interface CommonList is define in package common with methods like add, remove, size, isempty, display. A class SLL in define in package specificList implements CommonList. Implements all methods given in an interface CommonList in SLL class.**
OBJECTIVE	• To understand and use of interfaces • To understand and use of packages
OUTCOME	• To be able to implement interfaces in java • To be able to use packages for specific application
S/W PACKAGES AND HARDWARE APPARATUS USED	• Operating Systems (64-Bit)64-BIT Fedora 17 or latest 64-BIT Update of Equivalent Open source OS or latest 64-BIT Version • Programming Tools (64-Bit) Latest Open source update of Eclipse Programming frame work, GTK+ • Programming language Java. • Editors like gedit, vi editor, etc.
REFERENCES	• The complete reference java seventh edition by Herbert Schildt • Java Programming: 24-Hour Trainer Second Edition by Yakov Fain
STEPS	Refer to steps below
INSTRUCTIONS FOR WRITING JOURNAL	1. Date 2. Assignment no. 3. Problem definition 4. Learning objective 5. Learning Outcome

	6. Concepts related Theory 7. Algorithm 8. Test cases 10. Conclusion/Analysis

Prerequisites:

- Knowledge of object oriented concepts.
- Basic knowledge of Java programming.

Concepts related Theory:

Interfaces:

An interface in java is a blueprint of a class. It has static constants and abstract methods. The interface in java is a mechanism to achieve abstraction. There can be only abstract methods in the java interface not method body. An interface declares (describes) methods but does not supply bodies for them. It is used to achieve abstraction and multiple inheritance in Java. Java Interface also represents IS-A relationship. It cannot be instantiated just like abstract class.

Example:

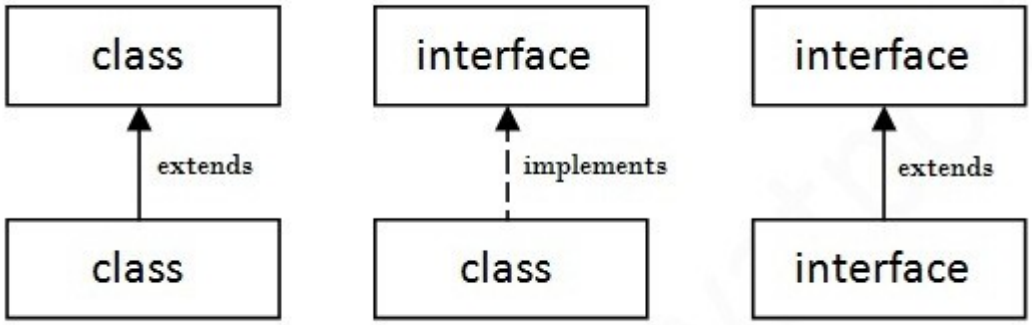
- Interface KeyListener {
  public void keyPressed(KeyEvent e);
  public void keyReleased(KeyEvent e);
  public void keyTyped(KeyEvent e);
  }

There are mainly three reasons to use interface. They are given below.

- It is used to achieve abstraction.
- By interface, we can support the functionality of multiple inheritance.
- It can be used to achieve loose coupling.

An interface is a reference type in Java. It is similar to class. It is a collection of abstract methods. A class implements an interface, thereby inheriting the abstract methods of the interface. Along with abstract methods, an interface may also contain constants, default methods, static methods, and nested types. Method bodies exist only for default methods and static methods. Writing an interface is similar to writing a class. But a class describes the attributes and behaviors of an object. And an interface contains behaviors that a class implements. Unless the class that implements the interface is abstract, all the methods of the interface need to be defined in the class.

Understanding relationship between classes and interfaces:



Interfaces have the following properties −

- An interface is implicitly abstract. You do not need to use the abstract keyword while declaring an interface.
- Each method in an interface is also implicitly abstract, so the abstract keyword is not needed.
- Methods in an interface are implicitly public.
- You cannot instantiate an interface
- An interface is like a *very* abstract class—*none* of its methods are defined
- An interface may also contain constants (final variables)

Packages:

A java package is a group of similar types of classes, interfaces and sub-packages. Package in java can be categorized in two form, built-in package and user-defined package. There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql etc.

Packages have following properties:

- They are containers for classes/interfaces to avoid name collision
- It stored in hierarchical manner and explicitly imported into new class using import statement.
- It provide both naming and visibility control mechanism (Access Protection)
- Using it classes/interfaces can easily maintained

A Package can be defined as a grouping of related types (classes, interfaces, enumerations and annotations) providing access protection and namespace management. Some of the existing packages in Java are java.lang that bundles the fundamental classes java.io that classes for input, output functions are bundled in this package. Programmers can define their own packages to bundle group of classes/interfaces, etc. It is a good practice to group related classes implemented by you so that a programmer can easily determine that the classes, interfaces, enumerations, and annotations are related. Since the package creates a new namespace there won't be any name conflicts with names in other packages. Using packages, it is easier to provide access control and it is also easier to locate the related classes.

Algorithm:

Simple interface and package example:

```
package pr;
interface printable
{void print();  }
Package aclass;
Import pr.*;
class A implements printable{
public void print(){System.out.println("Hello");}
public static void main(String args[]){
A obj = new A();
obj.print();  }  }
```

Example of multiple inheritance by using Interface:

```
Package pr;
interface Printable{
void print();  }
interface Showable{
void show();  }
A.java
Package aclass;
Import pr.*;
class A implements Printable,Showable{
public void print(){System.out.println("Hello");}
public void show(){System.out.println("Welcome");}
public static void main(String args[]){
A obj = new A();
obj.print();
obj.show();  }}
```

Conclusion:

Students are able to implement and use concepts of interface and packages in java

Review Questions:

4. What is mean by interface and package?
5. How to use interface with respect to multiple inheritance?

6. List any five common packages used in java?
7. What are the advantages of packages?
8. What are the properties of interface?

ASSINGMENT NO.	12
TITLE	Template design pattern and exception handling in Java
PROBLEM STATEMENT /DEFINITION	Write a program on template and exception handling in Java: in this assignment multiple templates are to be designed as a pattern and these patterns to be used to take decisions Example: **Write a program for implementation of stack ADT in java, Stack is abstract base class with template method consists of push and display. IntergerStack and CharStack are two concrete classes inherited from Stack which give specific implementation of these methods. Also handle stack full and stack empty condition as exceptions**
OBJECTIVE	• To understand and use the concept of template design pattern • To understand concept and importance of exception handling in java • To learn to use multiple templates as pattern to take decisions
OUTCOME	• To be able to implement multiple templates in java • To be able to implement exception handling in java
S/W PACKAGES AND HARDWARE APPARATUS USED	• Operating Systems (64-Bit)64-BIT Fedora 17 or latest 64-BIT Update of Equivalent Open source OS or latest 64-BIT Version • Programming Tools (64-Bit) Latest Open source update of Eclipse Programming frame work, GTK+ • Programming language Java. • Editors like gedit, vi editor, etc.
REFERENCES	• The complete reference Java seventh edition by Herbert Schildt • Java Programming: 24-Hour Trainer Second Edition by Yakov Fain
STEPS	Refer to steps below
INSTRUCTIONS FOR WRITING JOURNAL	1. Date

	2. Assignment no. 3. Problem definition 4. Learning objective 5. Learning Outcome 6. Concepts related Theory 7. Algorithm 8. Test cases 10. Conclusion/Analysis

Prerequisites:

- Knowledge of object oriented concepts.
- Basic knowledge of java programming.

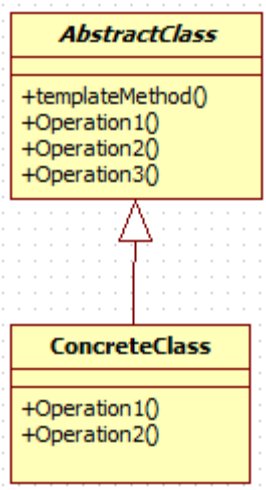
Concepts related Theory:

Template Method Design Pattern:

Design patterns are the best practices used by experienced object-oriented software developers. These design patterns are solutions to general problems that software developers faced during software development. There are total 23 design patterns. All these 23 design patterns are classified into Creational, Structural and Behavioral patterns. Among this behavioral patterns are specifically concerned with communication between objects.

Template method pattern is a behavioral design pattern which provides base method for algorithm, called template method which defers some of its steps to subclasses. So algorithm structure is same but some of its steps can be redefined by subclasses according to context.

Template means preset format like HTML templates which has fixed preset format. Similarly in template method pattern, a preset structure method called template method which consists of steps. These steps can be abstract method which will be implemented by its subclasses. In template method pattern, there is template method which defines set of steps and implementation of steps can be deferred to subclasses. Thus template method defines algorithm but exact steps can be defined in subclasses. We use template when you have a preset format or steps for algorithm but implementation of steps may vary or when you want to avoid code duplication, implementing common code in base class and variation in subclass.



So in above diagram, you can see we have defined template method with three steps i.e. operation1, operation2 and operation3 means,

public final void templateMethod(){

operation1( );

operation2( );

operation3( );

}

Suppose among them, opeation1 and operation2 are abstract steps, so these are implemented by ConcreteClass. As operation3 is common for all so it's defined in base class, so default implementation of that method is in abstract base class.

Components:

Abstract Class: It defines template method defining the structure of algorithm and it also defines abstract operations that will be implemented by subclasses to define steps of algorithm.

Concrete Class: It implements abstract operation of super class to carry out subclass specific steps of the algorithm and also overrides operation if default behavior is not required.

Exception Handling in java:

- An *exception* is an error condition that changes the normal flow of control in a program
- When something unexpected occurs
  - Ensure program detects the problem
  - Then program must do something about it
- Exceptions are runtime errors
- Exception handling gives us another opportunity to recover from the abnormality.
- It separates business logic and error handling code

In java, exception is an event that disrupts the normal flow of the program. It is an object which is thrown at runtime. The exception handling in java is one of the powerful mechanism to handle the runtime errors so that normal flow of the application can be maintained. Exception Handling

is a mechanism to handle runtime errors such as ClassNotFound, IO, SQL, Remote etc. The core advantage of exception handling is to maintain the normal flow of the application.

An exception (or exceptional event) is a problem that arises during the execution of a program. When an **Exception** occurs the normal flow of the program is disrupted and the program/ Application terminate abnormally, which is not recommended, therefore, these exceptions are to be handled.

An exception can occur for many different reasons. Following are some scenarios where an exception occurs.

9. A user has entered an invalid data.
10. A file that needs to be opened cannot be found.
11. A network connection has been lost in the middle of communications or the JVM has run out of memory.

There are mainly three types of exceptions:

- Checked Exception

  The classes that extend Throwable class except RuntimeException and Error are known as checked exceptions e.g. IOException, SQLException etc. Checked exceptions are checked at compile-time.

- Unchecked Exception

  The classes that extend RuntimeException are known as unchecked exceptions e.g. ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etc. Unchecked exceptions are not checked at compile-time rather they are checked at runtime.

- Error

  Error is irrecoverable e.g. OutOfMemoryError, VirtualMachineError, AssertionError etc.

There are 5 keywords used in java exception handling.

- try
- catch
- finally
- throw
- throws

Algorithm:

Template Method design pattern

Step 1: Define abstract class with template method consists of abstract methods and common methods

Step 2: Common implementations of individual steps are defined in the base class
Step 3: Override or implement specific steps to sub class
Step 4: Template method in super class should not be overridden so make it final

Example:

```
abstract class Generalization {
   public final void findSolution() {
      stepOne();
      stepTwo();
      stepThr();
      stepFor();
   }
   protected void stepOne() { //Generalization stepOne }
   abstract protected void stepTwo();
   abstract protected void stepThr();
   protected void stepFor() { //Generalization stepFor  }
}

class Specialization extends Generalization {
   protected void stepThr() {
      // Specific implementation of method for class Specialization
   }

   protected void stepTwo() { // Specific implementation of method for class
                        Specialization}
}

class Specialization1 extends Generalization {
   protected void stepThr() {
      // Specific implementation of method for class Specialization1
   }

 protected void stepTwo() { // Specific implementation of method for class
Specialization1 }
}
class TemplateMethodDemo {
   public static void main( String[] args ) {
      Generalization algorithm = new Specialization ();
      algorithm.findSolution();
algorithm=new Specialization1()
algorithm.findSolution();
}  }
```

Write a program for implementation of stack ADT in java , Stack is abstract base class with template method consists of push and display. IntergerStack and CharStack are two concrete classes inherited from Stack which give specific implementation of these methods. Also handle stack full and stack empty condition as exceptions

Conclusion:

Student will able unable to understand and implement template design pattern and exception handling in Java.

Review Questions:

1. What is the exception handling in java?
2. What is mean by template design pattern in java?
3. What are the types of exception?
4. What are the components of template?
5. Which are the keywords used in java exception handling?
6. What is use of design patterns?
7. What is template method?