DSA

Practicle Assignment 1.

Date:- / /
Page No.:-

A1-1

Title :- Telephone ~~datable~~ database using Hash Table.

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 comparisions required to find a set of telephone numbers.

Objective :- To understand the concept of Hashing and to understand to find record quickly using hash functions.

Theory :- Hash table is an efficient implementation of keyed array data structure. a structure sometimes known as a association array or map. Hash Table is a data structure which stores data in an associative manner. In a hash table, data is stored in an array format, where each data value has its own unique index value. Acess of data becomes very fast if we know the index of the desired data.

Thus, it is a data structure in which insertion and search operations are very fast irrespective of the size of the data. Hash table uses an array as a storage medium and uses hash technique to generate an index where an element is to be inserted or is to be located from.

Date:- / /
Page No.:-

- Hashing.
  - Hashing is a technique to convert a range of key values into a range of an array. We are are going to use modulo operator to get a range of key values.
  Example, keys ⇒ 3, 7, 15, 9, 2, 16, 21, 90.

Index	keys	
0	90	90 % 10 = 0
1	21	21 % 10 = 1
2	2	2 % 10 = 2
3	3	3 % 10 = 3
4	-	———
5	15	15 % 10 = 5
6	16	16 % 10 = 6
7	7	7 % 10 = 7
8	-	———
9	9	9 % 10 = 9

Hash Table

- linear Probing.
  - As we can see, it may happen that the hashing technique is used to create an already used index of array. In such a case, we can search the next empty location in the array by looking into the next cell until we find an empty cell. This technique is called as linear probing.

AI-2

Example. keys ⇒ 1, 2, 42, 4, 12, 14, 17, 13, 37.

Index	Keys	
0		
1	1	1 % 10 = 1 ✓
2	2	2 % 10 = 2 ✓
3	42	42 % 10 = 2, collision, 2+1 = 3 ✓ probing
4	4	4 % 10 = 4 ✓
5	12	12 % 10 = 2 collision, 2+1 = 3+1 = 4+1 = 5
6	14	14 % 10 = 4 collision 4+1 = 5+1 = 6 ✓
7	17	17 % 10 = 7 ✓
8	13	13 % 10 = 3 collision, 3+1 = 4+1 = 5+1 = 6 6+1 = 7+1 = 8 ✓
9	37	37 % 10 = 7 collision, 7+1 = 8+1 = 9 ✓

- Following are the basic primary operations of a hash table.
  1) Search - searches an element in a hash table.
  2) Insert - inserts an element in a hash table.
  3) Delete - Deletes an element from hash table.

- Algorithm.

1. Start.
2. Choose a hash function.
3. Choose an empty hash table.
4. Insert a new client.
5. Look up a client's telephone number.
6. a. Calculate the hash value, h, for the client's name using the same hash function as before.
   b. Check if the slot at index h is empty.

Date:- / /
Page No.:-

if it is, the client is not in the database.

c. if the slot at index h is occupied, traverse the linked list until the client with the matching name is found.
If the name is found, the client is in the database.

d. Once the maching client found, return their telephone number.

6. STOP.

Conclusion:- Using a hash function implementation to quickly look up a client's telephone number is an efficient and effective approach for a telephone book database of N clients.

Date:- / /
Page No.:-

A2-1

Title:- To create ADT that implement the set concept.

Problem Statement:- To create ADT that implement the 'set' concept. 1. Add (new element) - Place a value into set.
2. Remove (element)
3. Contains (element) Return value true if element is in collection.
4. Size () Return number of values in collection iterator.
5. Intersection of two sets
6. Union of two sets
7. Diffrence between two sets
8. Subset.

Objective:- To understand the concept of creation of ADT that implement the set concepts and its operations on set.

Theory:- Sets are a type of abstract data type that allows you to store a list of non-repeated values their name derives from the mathematical concept of finite sets.

Unlike an array, sets are unordered and unindexed. You can think about sets as a room full of people you know. They can move around the room, changing order, without altering the set of people in that room, and there are no duplicate people (unless you know someone who has cloned themselves).

These are the two properties of a set. The data is unordered and it is not duplicated.

Sets have the most impact in mathematical set theory. These the ones are used in many kinds of proof's, structures and abstract algebra creating relations from diffrent sets and codomains are also on important applications of sets.

In computer science, set theory is useful if you need to collect data and do not care about their multiplicity. or their order. As we've seen on this page, hash tables and sets are very related. In databases, especially for relational databases, sets are very useful. There are many commands that finds unions, intersections and diffrences of diffrent tables and set of data.

The set has four basic operations.

Function name	Provided Functionallity
insert ()	Adds it to the set
remove ()	Removes i from the set
size ()	Returns the size of the set
contains()	Returns whether or not the set contains.

Sometimes operations are implemented that allow interactions between the two sets.

A2-2

Function Name	Provided Functionality
Union (S,T)	Returns the union of sets S & T
intersection (S,T)	Returns the intersection of sets & S & T
diffrence (S,T)	Returns the diffrence of set s & T
subset (S,T)	Returns the whether or not sets is a sebset of set T.

Software requirement :- g++ / gcc compiler, 64 bit Fedora, eclipse IDE.

Algorithm :-

1. Start
2. Define a Set class with a private data member.
3. Define a constructor to initialize the array to fixed size and set the size variable to zero.
4. Define the add() method to add a new element to the set.
5. Define the remove() method.
6. Define the contains() method. - to check if the element exists or not.
7. Define the size() method - to return no. of elements
8. Define the iterator() method - to return an iterator
9. Define the intersection(set 1, set 2) method to return a new set that contains the elements that are in both set 1 and set 2.
10. Define the union() method to return a new set that contains all set elements from set 1 & set 2.

Date:- / /
Page No.:-

11. Define the diffrence (set 1, set 2) method to return a new set that contains the element that are in set 1 but not in set 2.
12. Define the subset (set 1, set 2) method to check if set 1 is a subset of set 2.
13. Stop.

- we can use the function required by calling it whenever required.

Conclusion :- In conclusion, the problem statement calls for the creation of an ADT that implement the "set" concept. A set is a collection of distinct elements with no particular order.

Date:- / /
Page No.:-

B1 - 1

Title:- To construct a tree and print the nodes.

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:- To understand the concept of tree datastructure and understand the features of object oriented programming.

Software requirement:- g++ / gcc compiler, 64-bit Fedora, eclipse IDE.

Theory:-

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 diffrent than the root has a unique parent node 'w'; each node with parent w is a child of 'w'.

- A tree is non-linear and a hierarchical data structure consisting of collection of nodes that each node of the tree stores a value and a list of references to other nodes.

Date:- / /
Page No.:-

Recursion in trees :-

- T is either empty or consists of a node r (root node) and a probably 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 represented most commonly as shown below. The circles are nodes and the edges are the links between them.

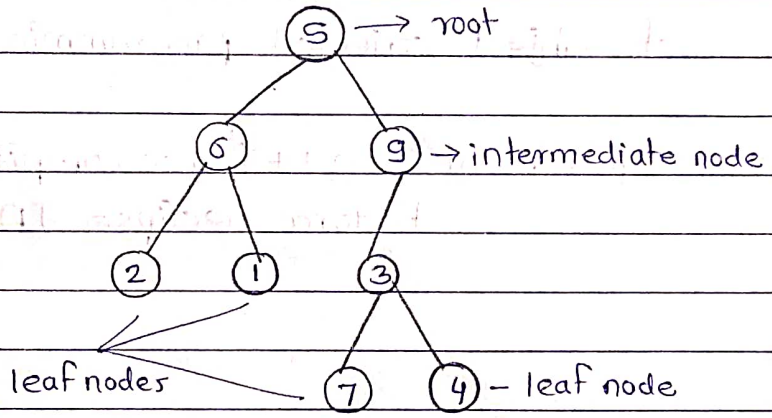


Fig. Tree data structure

- Trees are usually used to store and represent data in some heirarchical order. The data are stored in the nodes, from which the tree is considered of.
- A node may contain a value or a condition or represent a seperate data structure or a tree of its own.
- The topmost node in a tree is called the root node. It is the node where operations on the tree commonly begins.
- All the other nodes can be reached from it by following edges or links.

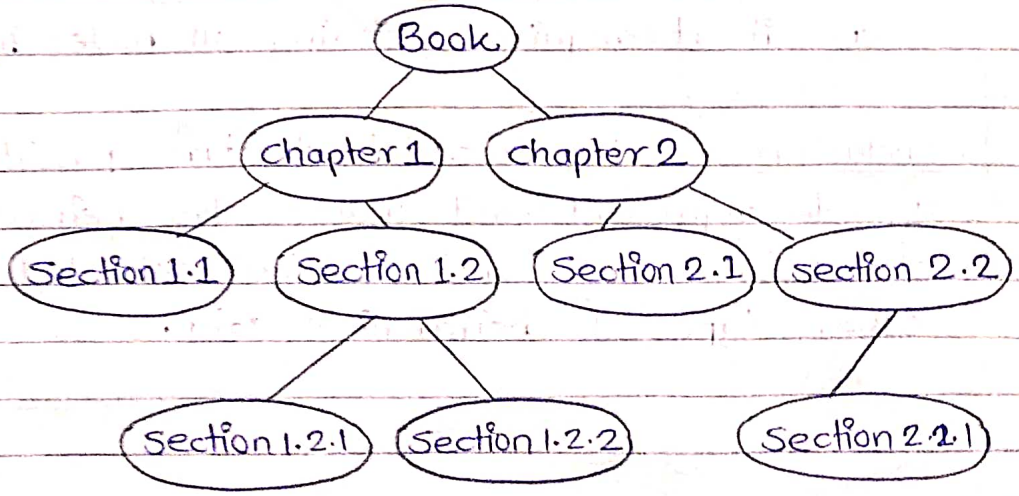
Date:- / /
Page No.:-

Important Terms.

- Following are the important terms with respect to tree data structure.

1. Path - refers to the sequence of nodes along the edges of the tree.
2. Root - The node at the top.
3. Parent - Any node except the root node has one edge upward to a node called parent.
4. Child - The node below a given node connected by its edge downward is called child
5. leaf - The node which does not have any child node is called the leaf node.
6. Subtree - Subtree represents the descendants of a node.
7. Traversing - means passing through node in a specific order.
8. levels - level of an order present the generation of node
9. Keys - represents a value of a node based on which an operation is carried out.

Example.



Date:- / /
Page No.:-

• Algorithm.

1. Start
2. Define an empty tree.
3. Define a node structure, for the tree to hold the data.
4. Read the book and parse it into chapters, sections and subsections.
5. Create a root node for the book and set its value to the title of the book.
6. For each chapter, create a child node of the root node and set its value to the chapter title.
7. For each section in chapter, create a child node.
8. For each subsections in section, create a child node.
9. Traverse the tree and print the value of each node.
10. Calculate the time and space complexity of algorithm.
11. Stop.

- The time complexity of the algorithm is O(n) where n is the total no. of nodes in the tree.
- The space complexity of the algorithm is also O(n) as it requires storing all nodes in memmory.

Conclusion :- Using tree data structure provides an efficient way to represent and analyze the heirarchical structure of a book, and can be easily extended to handle other types of heirarchical data.

Date:- / /
Page No.:-

B2-1

Title :- Binary Search Tree (BST) operations.

Problem Statement :- Beginning with an empty binary search tree, construct binary search tree by inserting the values in the order given. After constructiong a binary tree
i) Insert new node ii) Find number of nodes in longest path from the root iii) minimum data value found in the tree iv) change a tree so that the roles of the right and left pointers are swapped at every node v) search a value.

Objectives :- To construct and understand the concept the Binary Search Tree (BST) and its operations.

Software Requirements :- g++/gcc compiler, 64-bit Fedora, es eclipse IDE.

Theory :- A binary search tree is a binary tree where the value of each node is greater or equal to the values in its left subtree and less than or equal to the values in the right subtree. This property allows for efficient searching and insertion of values in the tree.
- when inserting values into a binary search tree, the values are compared to the value of the root node. If the value is less than the root node, it is inserted in the left subtree, otherwise is inserted in the right subtree.

This process is repeated recursively until a lead node is reached, at which point the new node is inserted as a child of the leaf node.

The order in which values are inserted into the tree can affect the height and the balance of the tree. In the worst case scenario, when the values are inserted in ascending or descending order, the resulting tree can become a degenerate tree, where all the nodes have only one child and the tree become essentially a linked list. To avoid this, it is important to insert values in a balanced order, such as randomly or by using a balanced binary search tree algorithm.

Example. Keys → 15, 3, 7, 9, 11, 18, 22, 17, 5.

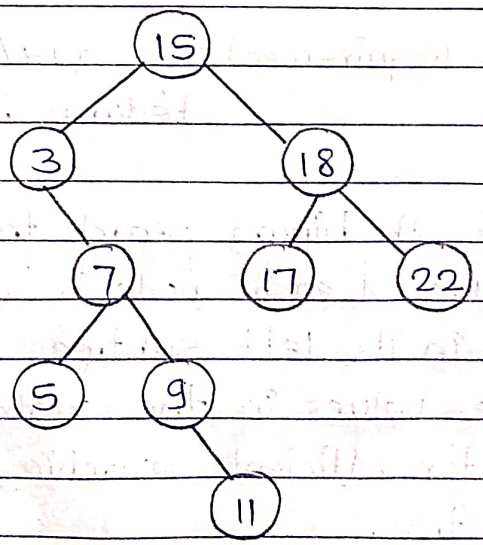


Fig. Binary Search Tree (BST).

Date:- / /
Page No.:-

B2-2

Algorithm :-

1. Start
2. Create an empty binary search tree.
3. Read the values to be inserted in the binary search tree in the order given.
4. For each value to be inserted, do the following.
   a. If the binary search tree is empty, create a new node with the value as its data, and make it the root node of the tree.
   b. If the binary search tree is not empty, compare the value to be inserted with the data of the current node.
   c. Create a new node with the value as its data, and insert it as a child node of the last visited leaf node based on the comparision in step b.
5. Repeat step 4 for all values to be inserted.
6. The binary search tree is now constructed.
7. Stop.

- This algorithm ensures that the binary search tree is always in the correct order, with values less than the current node's data on its left subtree and values greater than the current node's data on the right subtree.

- The time complexity of this algorithm is $O(\log n)$ in the average case, where n is the number of values to be inserted, and worst-case time complexity is $O(n^2)$.

Date:- / /
Page No.:-

Conclusion :-

Binary search trees are useful data structures for organizing and searching data, and their efficient construction and traversal make them a popular choice for many programming tasks.

Date:- / /
Page No.:-

B3-1

Title :- To convert given binary tree into threaded binary tree.

Problem Statement :- Convert given binary tree into threaded binary tree. Analyze time and space complexity of the algorithm.

Objectives :- To study the conversion of binary tree into Threaded Binary Tree.

Software Requirements :-

Theory :-

In a binary tree, a thread refers to a link between a node and its in-order successor or predecessor, which can be used to traverse the tree without using recursion or a stack. A threaded binary tree is a binary tree in which every node is threaded to its in-order successor predecessor, creating a more efficient traversal algorithm.

The problem statement requires converting a given binary tree into a threaded binary tree. This can be achieved by performing an in-order traversal of the binary tree and threading each node to its in-order successor as it is visited. The in-order traversal can be implemented recursively or iteratively using a stack, and the threading can be accomplished by adding links

Date:- / /
Page No.:-

between nodes as they are visited.

- The time complexity of this algorithm will be O(n), where n is number of codes in the binary tree.

- A threaded binary tree is a type of binary tree data structure where the empty left and right child pointers in a binary tree are replaced with threads that link nodes directly to their in-order predecessor or successor, thereby providing a way to traverse the tree without using recursion or a stack.

- Threaded binary trees can be ~~harmful~~ useful when space is a concern, as they can eliminate the need for a stack ~~using~~ during traversal. However they can be more complex to implement than standard binary trees.

There are two types of threaded binary trees.

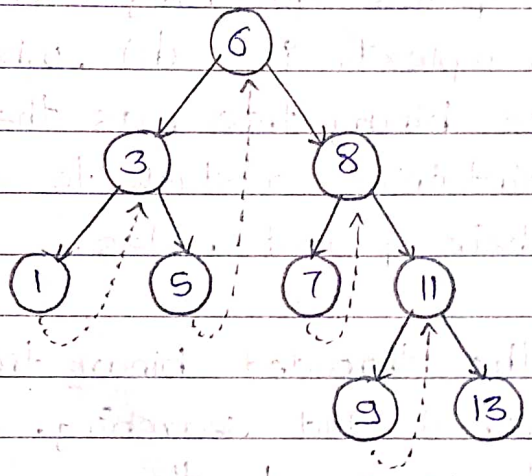
1. Single threaded : Where a NULL righ pointers is made to point to the inorder successor.

2. Double threaded : Where both left and right NULL pointers are made to point to inorder predessor and inorder successor respectively. The predessor threads are useful for reverse inorder traversal and postorder traversal.

The threads are also useful for fast accessing ancestors of nodes.

B3-2

Following diagram shows an example of Single Treaded Binary tree. The dotted lines represents threads.



A threaded binary tree.

Algorithm:-

1. Start.
2. Traverse the binary tree in-order (left, root, right), keeping track of the previously visited node.
3. For each node visited, check if its left child is null. If it is, set the left child to be the previously visited node and set the left thread flag to true.
4. If the previously visited node's right child is null, set the right child to be the current node and set the right thread flag to true.
5. Repeate steps 3 & 4 untill all nodes have been visited.

Date:- / /
Page No.:-

- The time complexity of this algorithm is O(n) where n is the number of nodes in the binary tree, as it requires traversing each node once.
- The space complexity is O(h), where h is the height of the binary tree, as the algorithm requires maintaining a stack to store the visited nodes and their parent nodes.

Conclusion :- The threaded binary tree allows for efficient traversal and searching, as the threads provide direct access to the predeccessor and successor nodes without the need for recursion or a stack.

Date:- / /
Page No.:-

C1-1

Title :- Representing Flight Paths Between Cities as a Graph.

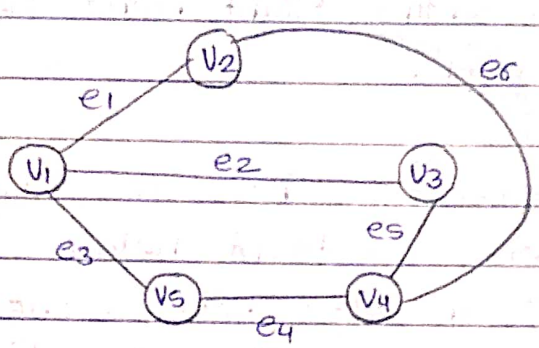
Problem Statement :- There are Flight paths between cities. If there is a flight between city A and city B then there is an edge between the cities. The cost of the edge can be the time that flight take to reach city B from A, or the amount of fuel used for the journey. Represent this as a graph. The node can be represented by airport name or name of the city. Use adjancency list representation of the graph or use adjancency matrix representation of the graph. Check whether the graph is connected or not. Justify the storage representation used.

Objectives :- To use Graph data structure for representing Flights paths between cities.

Theory :- Graphs are the most general data structure. They are also commonly used data structure.

Graph :- A non-linear data structure constisting of nodes and links between nodes. An undirected graph is a set of links between the nodes. Each node is known as vertex. Each link is known as edge. and each edge connects two vertices.
The order of the two connected vertices is important.
- An undirected graph is a finite set of vertices together with a finite set of edges. Both sets might be empty, which is called the empty graph.

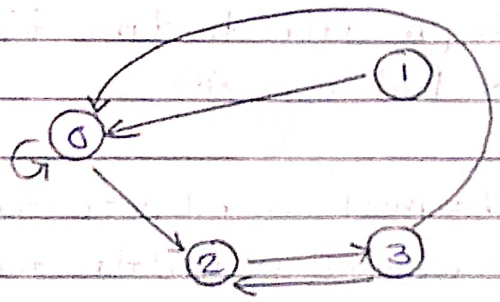
Example.



Graph implementation :- diffrent kinds of graphs require diffrent kinds of implementation, but the ~~Functi~~ fundamental concept of all graph implementation are similar.

Representation of graph

1) Adjancency Matrix.



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

Adjancey matrix

- An adjacency matrix is a square grid of the true/false or 0/1 values that represent the edges of a graph.
- If the graph contains n vertices, then the grid contains n rows and n columns.
- For two vertex numbers i and j, the component at row (i) and column (j) is true if there is an edge

CI-2

from vertex i to vertex j ; otherwise, the component is false.

- The adjacency matrix for an undirected graph is always symmetric.
- Adjacency matrix is also used to represent weighted graphs if adj[i][i] = w, then there is an edge from vertex i to vertex j with wight w.

Algorithm:-

1. Start
2. Initialize an nxn matrix (adjacency matrix), filled with zeros.
3. For each edge (i, j) in the list of edges, set adjacency_matrix[i][j] to 1 to represent the presence of the edge from vertex i to vertex j.
4. For each flight path, add an entry to the matrix for each city involved in the flight.
5. If the cost of the edge is the distance between two cities, store the distance in the corresponding entry in the matrix
6. Stop.

- The storage representation used depends on the size and sparsity of the graph. For small and dense graphs an adjacency matrix representation may be more efficient in terms of space and time complexity.

Date:- / /
Page No.:-

- The time complexity for both representations is O(edges + vertices)
- The space complexity for the adjacency list and matrix representation is O(E+V) and $O(V^2)$ respectively.
- Therefore the adjacency list representation is more memmory efficient for sparse graphs.

Conclusion:- The Graph data structure for representing flights path between cities is more efficient using representation as adjacency matrix for small size of the graph.

Date:- / /
Page No.:-

C2-1

Title :- Minimum Cost Spanning Tree Algorithm for Connecting Multiple offices.

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 diffrent amounts of money to connect diffrent 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 structure.

Objectives :- To find a minimum cost spanning tree that connects all the offices together.

Software Requirement :-

Theory :- This problem can be solved using the minimum spanning tree algorithm, which is graph theory algorithm that finds a tree that connects all the vertices of a graph and has the minimum possible total edge weight.

To apply the minimum spanning tree algorithm, we can represent the network of offices and phone line connections as an undirected graph, where each office is a vertex and each phone line is an edge with a height representing the cost of the connection.

Date:- / /
Page No.:-

We can use an adjacency matrix or an adjacency list to represent the graph. An adjacency matrix is a two dimensional array where the rows and colums represent the vertices, and the entries represent the edges weights. An adjacency list is a collection of linked lists, where each vertex has a list of its adjacent vertices and their edges weights.

Once we have the graph representation, we can apply the minimum spanning tree algorithm. There are two commonly used algorithms for finding the minimum spanning tree.

1) Kruskal's algorithm
2) Prim's algorithm.

Kruskal's algorithm sorts the edges in ascending order by weight, and then adds each edge to the minimum spanning tree as long as it does not create a cycle. This algorithm can be implemented using a disjoint-set data structure to efficiently detect cycles and keep track of which vertices are connected.

Prim's algorithm starts with an arbitrary vertex and greedily adds the cheapest edge that connects a vertex not already in the tree to the tree. This algorithm can be implemented using priority queue to efficiently find the cheapest edge to add to the tree at each step.

Date:- / /
Page No.:-

C2.2

The problem of finding the minimum spanning tree is a classic problem in graph theory, and it has numerous applications in various fields, including computer networks, transportation, and biology.

To apply the minimum spanning tree algorithm to the problem of leasing phone lines, we can represent the network of offices and phone line connections as an undirected graph, where each office is a vertex and each phone line is an edge with a weight representing the cost of the connection.

Once we have represented the graph, we can apply the minimum spanning tree algorithm to find the set of the phone lines that connects all the offices with a minimum total cost. Both Kruskal's and Prim's algorithm's have a time complexity of O(e log e), where 'e' is no. of edges in the graph.

Algorithm :-

- Here we can use the Kruskal's algorithm, which is a greedy algorithm that finds a minimum spanning tree for a connected weighted graph.
- Here we need a data structure to store the edges and another data structure to keep track of the connected components of the graph.

Date:- / /
Page No.:-

Here is the algorithm in pseudocode.

1. Sort all the edges in increasing order of their weights
2. Initialize an empty set of edges and an empty union-find data structure with one set for each vertex.
3. For each edge in the sorted list of edges.
   a. If the edge connects two different sets in the union data structure, add it to the set of edges and merge the sets.
4. Return the set of edges.

The time complexity is O(e log e), where e is the number of edges in the graph, due to cost of sorting the edges and performing the union-find operations.
The space complexity is O(V+E), where V is the number of the number of vertices, for storing the edges and the union-find data structure.

Conclusion :- The problem of connecting multiple offices with a minimum total cost can be solved by representing the graph using an appropriate data structure such as an adjacency matrix or an adjacency list, and the applying a suitable MST algorithm.