Discipline Core

Data Structures

CS4002

Assignment 2

Sheikh Muhammed Tadeeb (AU19B1014)

❖ Problem Statement: -

- **Part 1:** Create a family hierarchical structure and print only the cousins from the family structure. Identify the data structure to be used and write the C++ program to output the cousins from the family.
- **Part 2:** A salesperson starts from the office (source point) and travels through multiple cities to sell his product. He can return to the office (source point) any time if all his products are sold. Write a C++ code to simulate the salesperson journey from office and traversing to multiple cities and returning to the source point.

❖ Tree Data Structure: -

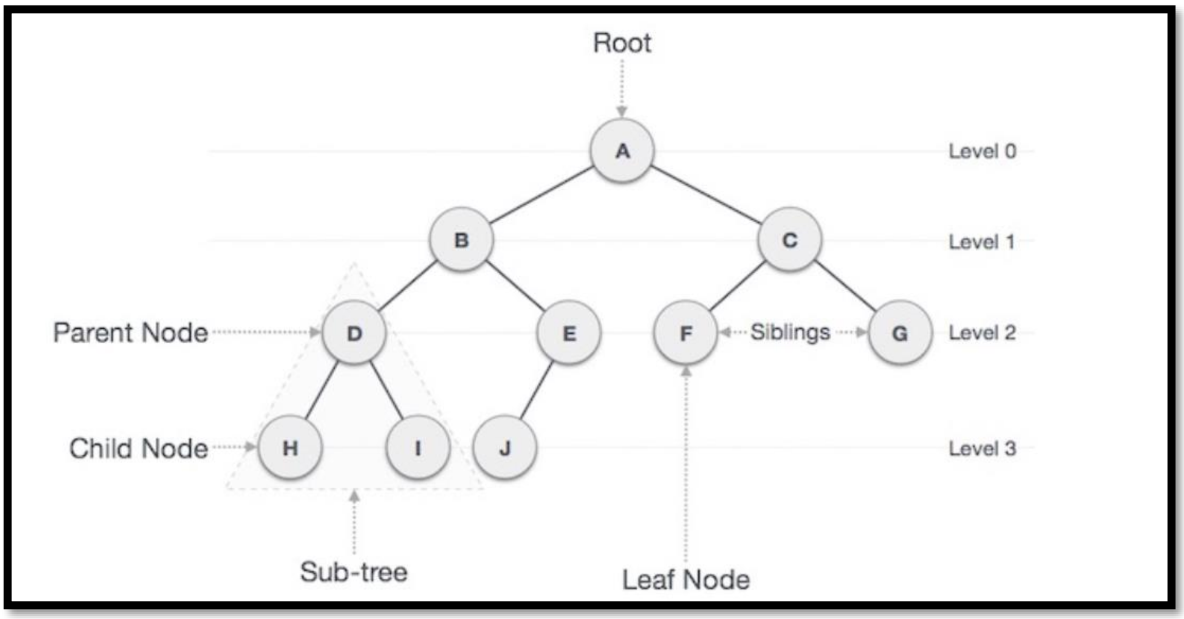
Tree represents the nodes connected by edges. We will discuss binary tree or binary search tree specifically.

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.

BST Basic Operations

The basic operations that can be performed on a binary search tree data structure, are the following −

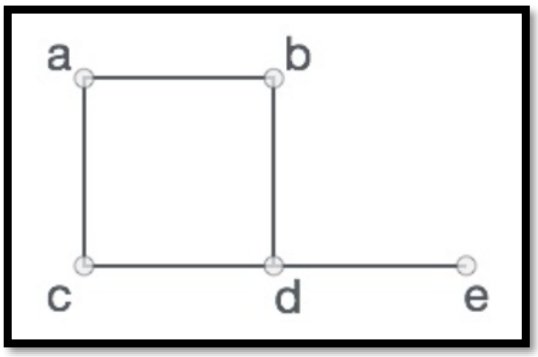
- **Insert** − Inserts an element in a tree/create a tree.
- **Search** − Searches an element in a tree.
- **Preorder Traversal** − Traverses a tree in a pre-order manner.
- **Inorder Traversal** − Traverses a tree in an in-order manner.
- **Postorder Traversal** − Traverses a tree in a post-order manner.



❖ Graph Data Structure: -

A graph is a pictorial representation of a set of objects where some pairs of objects are connected by links. The interconnected objects are represented by points termed as vertices, and the links that connect the vertices are called edges.

Formally, a graph is a pair of sets (V, E), where V is the set of vertices and E is the set of edges, connecting the pairs of vertices. Take a look at the following graph −



In the above graph,

V = {a, b, c, d, e}

E = {ab, ac, bd, cd, de}

Mathematical graphs can be represented in data structure. We can represent a graph using an array of vertices and a two-dimensional array of edges. Before we proceed further, let's familiarize ourselves with some important terms –

- **Vertex** − Each node of the graph is represented as a vertex. In the following example, the labelled circle represents vertices. Thus, A to G are vertices. We can represent them using an array as shown in the following image. Here A can be identified by index 0. B can be identified using index 1 and so on.
- **Edge** − Edge represents a path between two vertices or a line between two vertices. In the following example, the lines from A to B, B to C, and so on represents edges. We can use a two-dimensional array to represent an array as shown in the following image. Here AB can be represented as 1 at row 0, column 1, BC as 1 at row 1, column 2 and so on, keeping other combinations as 0.
- **Adjacency** − Two node or vertices are adjacent if they are connected to each other through an edge. In the following example, B is adjacent to A, C is adjacent to B, and so on.
- **Path** − Path represents a sequence of edges between the two vertices. In the following example, ABCD represents a path from A to D.

❖ C++ Code (Part-1) : -

```
#include <iostream>
#include <bits/stdc++.h>
#include <stdlib.h>
#include <queue>

using namespace std;

class Node
{
      public:
      string data;
   Node *left, *right;
};

class Binary1 : Node
{
      public:

             Node* createNode(string data);
             Node* Insert(Node* root,string data);
             int getLevel(Node *root, Node *node, int level);
             void printGivenLevel(Node* root, Node *node, int level) ;
             void printCousins(Node *root, Node *node) ;
             void inorder(Node *root);
             void preorder(Node *root);
             void postorder(Node *root);


             void call(){
                    cout<<endl<<endl<<endl<<"\t\t\t\t\tWelcome to the system which finds cousins";
                    string parent,lchild,rchild;

                    cout<<endl<<endl<<endl<<endl<<"\tEnter The root name of tree:- ";
                    cin>>parent;
                    Node *root = createNode(parent);
                    cout<<endl<<endl<<"\t\t\t         "<<root->data<<"\n" ;

                    cout<<endl<<endl<<"\tEnter The left child name of "<<parent<<" :- ";
                    cin>>lchild;
                 root->left = createNode(lchild);
```

```
                  cout<<endl<<endl<<"\t\t\t           "<<root->data<<"  \n\
                                 /                                        \n\
                              "<<root->left->data<<"                      \n";


                  cout<<endl<<endl<<"\tEnter The Right child name of "<<parent<<" :- ";
                     cin>>rchild;
                  root->right = createNode(rchild);
                  cout<<endl<<endl<<"\t\t\t           "<<root->data<<"  \n\
                                 /      \\                                \n\
                                "<<root->left->data<<"         "<<root->right->data<<"                \n";


                     cout<<endl<<endl<<endl<<"\tWho is the next parent now:- ";
                     cin>>parent;

                     if (parent == rchild){
                            cout<<endl<<endl<<"\tEnter The left child name of "<<parent<<" :- ";
                            cin>>lchild;
                            root->right->left = createNode(lchild);
                            cout<<endl<<endl<<"\t\t\t         "<<root->data<<"  \n\
                                 /      \\                                \n\
                                "<<root->left->data<<"         "<<root->right->data<<"                \n\
                                     /                                    \n\
                                    "<<root->right->left->data<<"         \n ";



                            cout<<endl<<endl<<"\tEnter The Right child name of "<<parent<<" :- ";
                            cin>>rchild;
                            root->right->right = createNode(rchild);
                            cout<<endl<<endl<<"\t\t\t         "<<root->data<<"  \n\
                              /      \\                                  \n\
                             "<<root->left->data<<"         "<<root->right->data<<"                         \n\
                                  /  \\                                      \
                                                    "<<root->right->left->data<<"    "<<root->right->right->data<<"                       \n ";


                    cout<<endl<<endl<<endl<<"\tWho is the next parent now:- ";
                    cin>>parent;

                    if (parent == lchild){
                        cout<<endl<<endl<<"\tEnter The Right child name of "<<parent<<" :- ";
                        cin>>rchild;
                        root->right->left->right = createNode(rchild);
                        cout<<endl<<endl<<"\t\t\t        "<<root->data<<"                                          \n\
                              /      \\                                  \n\
                             "<<root->left->data<<"         "<<root->right->data<<"                                          \n\
                                  /  \\                                      \
                                                    "<<root->right->left->data<<"     "<<root->right->right->data<<"                   \n\
                                 \\                                          \
                                                     "<<root->right->left->right->data<<"                                \n";
                    }
               }

               cout<<endl<<endl<<endl<<"\tWho is the next parent now:- ";
               cin>>parent;

               if(parent == root->left->data){
                        cout<<endl<<endl<<"\tEnter The left child name of "<<parent<<" :- ";
                        cin>>lchild;
                        root->left->left = createNode(lchild);
                        cout<<endl<<endl<<"\t\t\t        "<<root->data<<"                                          \n\
                                    /     \\                                \n\
                                   "<<root->left->data<<"          "<<root->right->data<<"                        \n\
                                /       /  \\                                \
                                              "<<root->left->left->data<<"           "<<root->right->left->data<<"    "<<root->right->right->data<<"                 \n\
                                         \\                                \
                                                        "<<root->right->left->right->data<<"                \n";


                cout<<endl<<endl<<"\tEnter The Right child name of "<<parent<<" :- ";
                cin>>rchild;
                root->left->right = createNode(rchild);
                cout<<endl<<endl<<"\t\t\t          "<<root->data<<"                                \n\
                                    /     \\                                \n\
                                   "<<root->left->data<<"          "<<root->right->data<<"                        \n\
                                /  \\          /  \\                          \
                                              "<<root->left->left->data<<"    "<<root->left->right->data<<"    "<<root->right->left->data<<"    "<<root->right->right->data<<"          \n\
                                         \\                                \
                                                        "<<root->right->left->right->data<<"                \n";

                cout<<endl<<endl<<endl<<"\tWho is the next parent now:- ";
                cin>>parent;

                if (parent == rchild){
                    cout<<endl<<endl<<"\tEnter The Right child name of "<<parent<<" :- ";
                    cin>>rchild;
                    root->left->right->right = createNode(rchild);
```

```
                                        cout<<endl<<endl<<"\t\t\t     "<<root->data<<"                                   \n\
                                       /      \\                     \n\
                                    "<<root->left->data<<"      "<<root->right->data<<"                  \n\
                                 /   \\          /   \\               \
                                                   "<<root->left->left->data<<"    "<<root->left->right->data<<"    "<<root->right->left->data<<"     "<<root->right->right->data<<"        \n\
                                        \\     \\                  \
                                                          "<<root->left->right->right->data<<"    "<<root->right->left->right->data<<"    \n";

                                 }
                          }

                          string cousins;

                          int i = 0;
                          while(i < 4){
                                 cout<<endl<<endl<<endl<<"\tNow whose cousin you wanna Print:- ";
                                 cin>>cousins  ;
                                 if(cousins == root->left->right->data || cousins == root->left->left->data){
                                        printCousins(root, root->left->right);
                                 }else if(cousins == root->right->left->data || cousins == root->right->right->data){
                                        printCousins(root, root->right->right);
                                 }else{
                                        cout<<endl<<endl<<"\tNo cousins found !";
                                 }
                                 i++;
                          }

                   string data;
                   cout<<"\n\n\n\n\n\n\tEnter the new data to be inserted:- ";
                   cin>>data;
                          Insert(root,data);

```

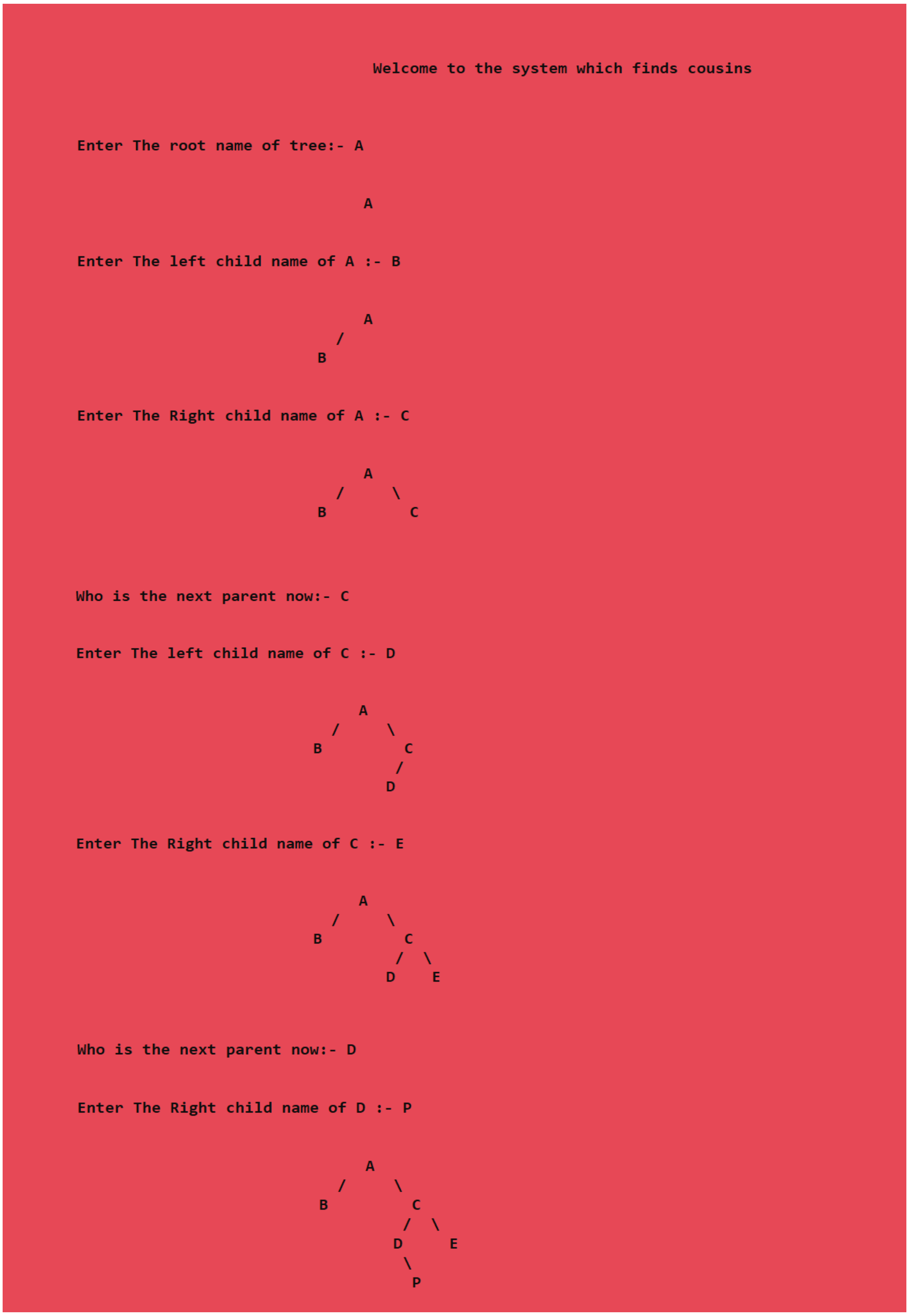
```
                cout<<endl<<endl<<"\t\t\t        "<<root->data<<"  \n\
                            /     \\          \n\
                        "<<root->left->data<<"          "<<root->right->data<<"          \n\
                      /  \\         /  \\        \
                                  "<<root->left->left->data<<"    "<<root->left->right->data<<"    "<<root->right->left->data<<"    "<<root->right->right->data<<"         \n\
                     /      \\     \\        \
                               " <<root->left->left->left->data<<"          "<<root->left->right->right->data<<"    "<<root->right->left->right->data<<"               \n";


          cout<<"\n\n\n\n\n\tEnter the new data to be inserted:- ";
          cin>>data;
                Insert(root,data);
                cout<<endl<<endl<<"\t\t\t        "<<root->data<<"  \n\
                            /     \\          \n\
                        "<<root->left->data<<"          "<<root->right->data<<"          \n\
                      /  \\         /  \\        \
                                  "<<root->left->left->data<<"    "<<root->left->right->data<<"    "<<root->right->left->data<<"    "<<root->right->right->data<<"         \n\
                    /      \\     \\     \\      \
                               " <<root->left->left->left->data<<"    "<<root->left->left->right->data<<"    "<<root->left->right->right->data<<"    "<<root->right->left->right->data<<"           \n";

              string y;
              i = 0;
              cout<<endl<<endl<<endl<<"\tDo you want to traverse the tree(y/n) :- ";
              cin>>y;
              if (y == "y" || y == "Y"){

                 while(i < 3){
```

```
                                        cout<<endl<<endl<<"\tWhich Traversal You wanna see (Inorder, Preorder, Postorder):- ";
                                  cin>>y;
                                        if (y == "Inorder" || y == "inorder"){
                                                cout<<"\n\t";
                                                inorder(root);
                                                cout<<"NULL";
                                    }else if(y == "Postorder" || y == "postorder"){
                                                cout<<"\n\t";
                                            postorder(root);
                                                cout<<"NULL";
                                    }else if(y == "Preorder" || y == "preorder"){
                                                cout<<"\n\t";
                                            preorder(root);
                                                cout<<"NULL";
                                    }
                                    i++;
                           }

                    }else{
                            cout<<endl<<endl<<"\tThank You !";
                    }

            }
    };

    Node* Binary1 :: createNode(string data){
            Node* newNode = new Node;

            if(!newNode){
                    cout<<endl<<endl<<"\t\t\t\tMemory Error";
                    return NULL;
            }
            newNode->data = data;
            newNode->left = newNode->right = NULL;
            return newNode;
    }

    Node* Binary1 :: Insert(Node* root, string data){
            if (root == NULL){
                            root = createNode(data);
                            return root;
                    }
                    queue<Node*> q;
                    q.push(root);
```

```cpp

                while(!q.empty()){
                    Node* temp = q.front();         // temp variable holding address of newNode created.
                    q.pop();                        // Emptying the queue.

                    if(temp->left != NULL){         // Checking Root newNodes left is NULL or not
                        q.push(temp->left);         // If not NULL insert left address of temp in queue.
                    }else{
                        temp->left = createNode(data); // If  NULL then in the left address of temp insert address of another newnode.
                        cout<<endl<<"\tData is inserted at "<<temp->data;
                        return root;
                    }


                    if(temp->right != NULL){        // Checking Root newNodes Right is NULL or not
                        q.push(temp->right);        // If not NULL insert Right address of temp in queue.
                    }else{
                        temp->right = createNode(data); // If  NULL then in the right address of temp insert address of another newnode.
                        cout<<endl<<"\tData is inserted at "<<temp->data;
                        return root;
                    }
                }

    }


    /* It returns level of the node if it is
    present in tree, otherwise returns 0.*/
    int Binary1 :: getLevel(Node *root, Node *node, int level)
    {

        // base cases
        if (root == NULL)
            return 0;
        if (root == node)
            return level;

        // If node is present in left subtree
        int downlevel = getLevel(root->left,
                                 node, level + 1);
        if (downlevel != 0)
            return downlevel;

        // If node is not present in left subtree
        return getLevel(root->right, node, level + 1);
    }

    /* Print nodes at a given level such that
    sibling of node is not printed if it exists */
    void Binary1 :: printGivenLevel(Node* root, Node *node, int level)
    {
        // Base cases
        if (root == NULL || level < 2)
            return;

        // If current node is parent of a node
        // with given level
        if (level == 2)
        {
            if (root->left == node || root->right == node)
                return;
            if (root->left)
                cout<<endl<<endl<<"\tThe Cousins are:- "<< root->left-
>data << " ";
            if (root->right)
                cout <<" "<< root->right->data;
        }

        // Recur for left and right subtrees
        else if (level > 2)
        {
            printGivenLevel(root->left, node, level - 1);
            printGivenLevel(root->right, node, level - 1);
        }
    }

    // This function prints cousins of a given node
    void Binary1 :: printCousins(Node *root, Node *node)
    {
        // Get level of given node
        int level = getLevel(root, node, 1);

        // Print nodes of given level.
        printGivenLevel(root, node, level);
    }

```

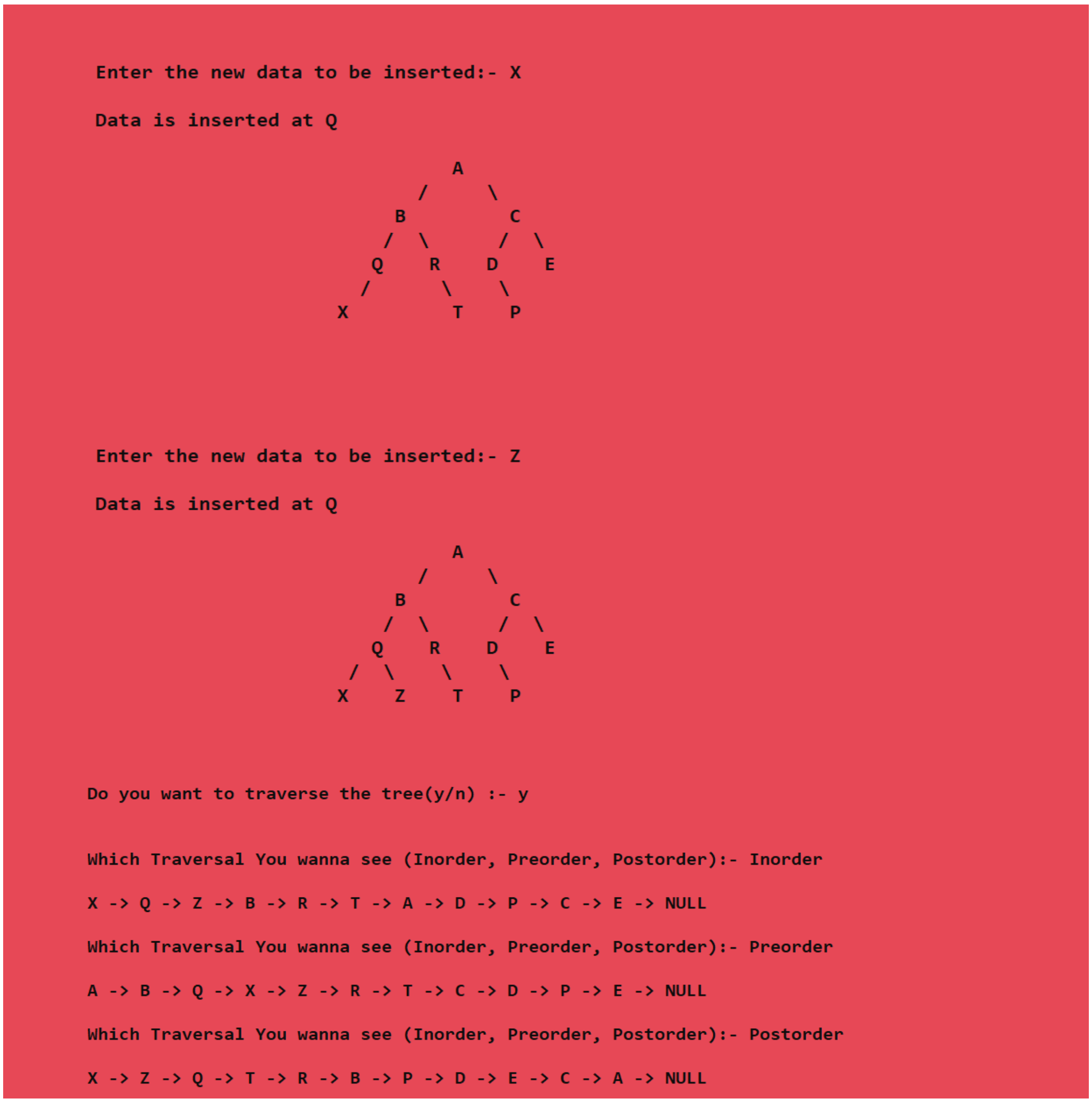
```
void Binary1 :: inorder(Node *root){
        if(root==NULL){
                return;
        }
        inorder(root->left);
        cout<<root->data<<" -> ";
        inorder(root->right);
}

void Binary1 :: preorder(Node *root){
        if(root==NULL){
                return;
        }
        cout<<root->data<<" -> ";
        preorder(root->left);
        preorder(root->right);
}

void Binary1 :: postorder(Node *root){
        if(root==NULL){
                return;
        }
        postorder(root->left);
        postorder(root->right);
        cout<<root->data<<" -> ";
}

// heirarchical Inheritance
class Binary2 : protected Binary1
{
        public:
                // Polymorphism
                void call(){
                        Binary1 obj;
                     obj.call();
                }
};

int main()
{
        system("Color C0");
        Binary2 ob;
        ob.call();
        return 0;
}
```

❖ Part-1 Code Output:



```
                                Welcome to the system which finds cousins


Enter The root name of tree:- A

                               A

Enter The left child name of A :- B

                               A
                            /
                          B

Enter The Right child name of A :- C

                               A
                            /     \
                          B         C


Who is the next parent now:- C

Enter The left child name of C :- D

                               A
                            /     \
                          B         C
                                   /
                                  D

Enter The Right child name of C :- E

                               A
                            /     \
                          B         C
                                   /  \
                                  D    E


Who is the next parent now:- D

Enter The Right child name of D :- P

                               A
                            /     \
                          B         C
                                   /  \
                                  D     E
                                   \
                                    P
```

```
Who is the next parent now:- B

Enter The left child name of B :- Q

                               A
                            /     \
                          B         C
                         /         /  \
                       Q          D    E
                                   \
                                    P

Enter The Right child name of B :- R

                                A
                             /     \
                           B         C
                          /  \      /  \
                         Q    R    D    E
                                    \
                                     P


Who is the next parent now:- R

Enter The Right child name of R :- T

                                A
                             /     \
                           B         C
                          /  \      /  \
                         Q    R    D    E
                               \    \
                                T    P


Now whose cousin you wanna Print:- Q

The Cousins are:- D  E

Now whose cousin you wanna Print:- D

The Cousins are:- Q  R

Now whose cousin you wanna Print:- E

The Cousins are:- Q  R

Now whose cousin you wanna Print:- P

No cousins found !
```



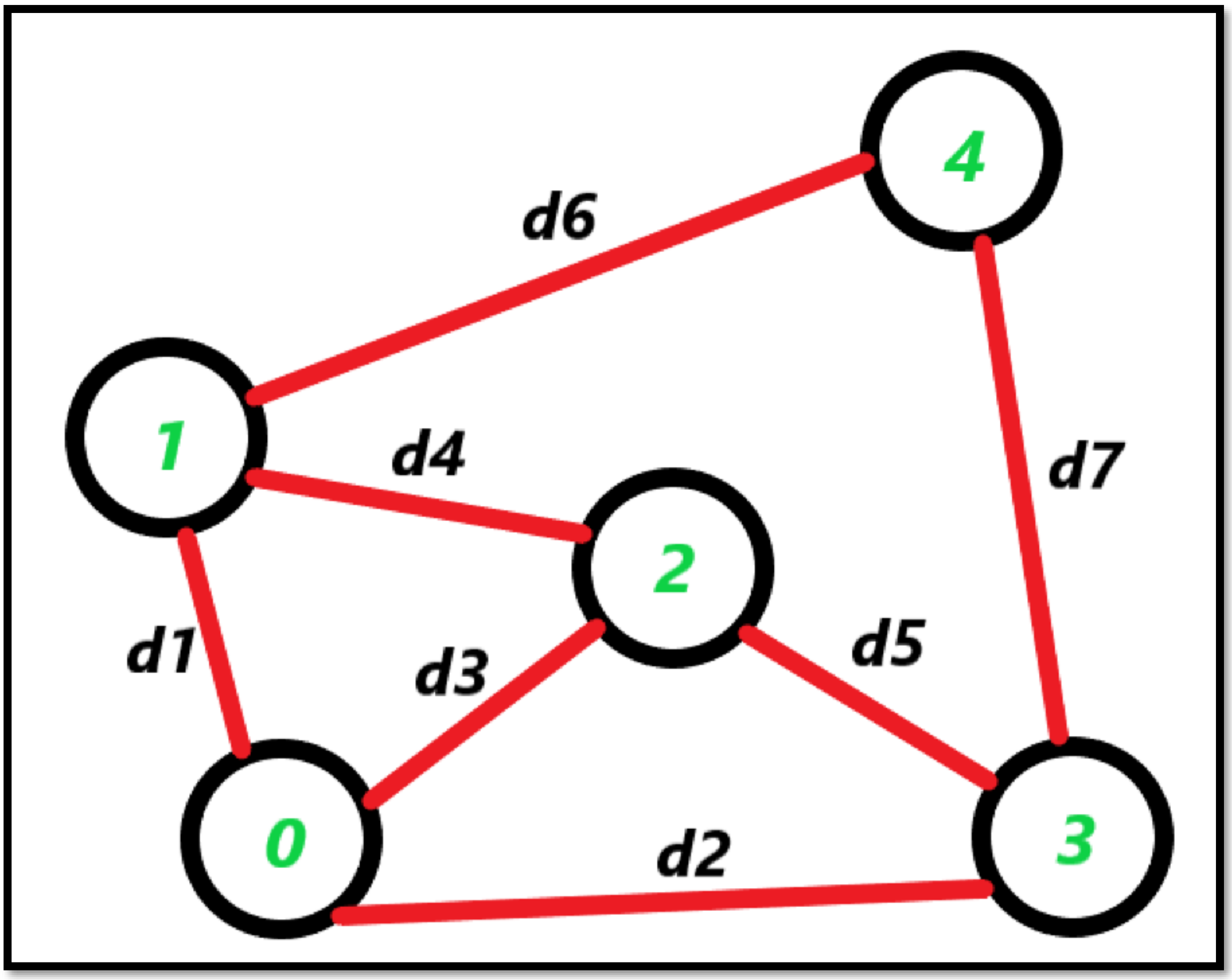
```
 Enter the new data to be inserted:- X

 Data is inserted at Q

                              A
                           /     \
                         B         C
                        /  \      /  \
                       Q    R    D    E
                      /      \    \
                    X         T    P



 Enter the new data to be inserted:- Z

 Data is inserted at Q

                              A
                           /     \
                         B         C
                        /  \      /  \
                       Q    R    D    E
                     /  \    \    \
                    X    Z    T    P


Do you want to traverse the tree(y/n) :- y

Which Traversal You wanna see (Inorder, Preorder, Postorder):- Inorder

X -> Q -> Z -> B -> R -> T -> A -> D -> P -> C -> E -> NULL

Which Traversal You wanna see (Inorder, Preorder, Postorder):- Preorder

A -> B -> Q -> X -> Z -> R -> T -> C -> D -> P -> E -> NULL

Which Traversal You wanna see (Inorder, Preorder, Postorder):- Postorder

X -> Z -> Q -> T -> R -> B -> P -> D -> E -> C -> A -> NULL
```

❖ C++ Code Part-2: -

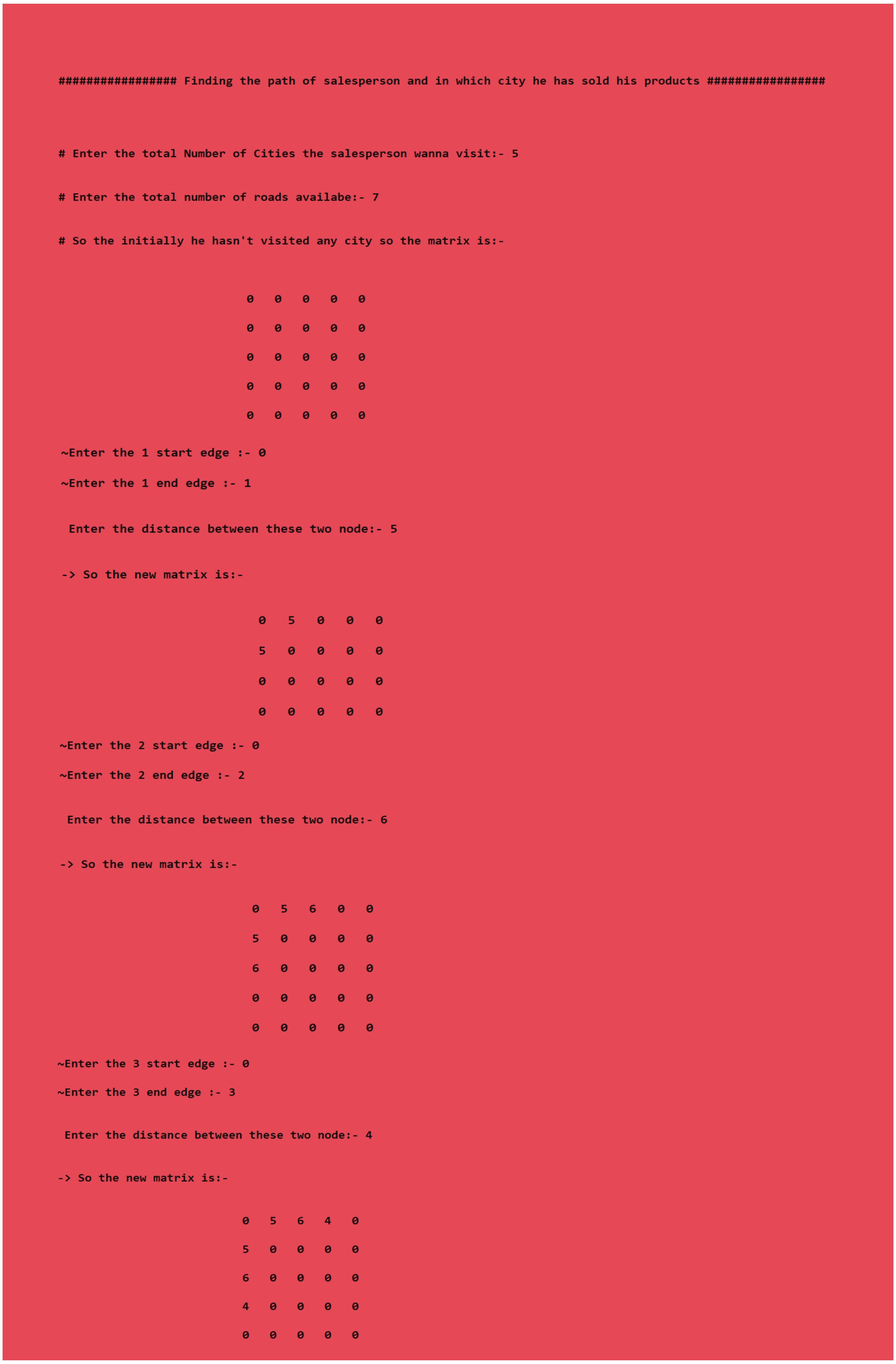
```cpp
#include <bits/stdc++.h>
using namespace std;

class Graph {

   public:
    // Number of vertex
    int v;
    // Number of edges
    int e;
    // Adjacency matrix
    int** adj;

      public:
          // To create the initial adjacency matrix
          void graph(int v);

          // Function to insert a new edge
          void addEdge(int start, int e);

          // Function to display the BFS traversal
          void BFS(int start);

              ~Graph(){};
};

class  Calling : Graph
{
      public:
              void Call(){
              int v,startedge, endedge,e,source;
              cout<<endl<<endl<<endl<<endl<<endl<<"\t################# Finding the path of salesperson and in which city he has sold his products #################";

                      cout<<endl<<endl<<endl<<endl<<endl<<"\tEnter the total Number of Cities the salesperson wanna visit:- ";
                  cin>>v;

                  cout<<endl<<endl<<"\tEnter the total number of roads availabe:- ";
                  cin>>e;

                  // Create the graph
                  cout<<endl<<endl<<"\tSo the initially he hasn't visited any city so the matrix is:- "<<endl;
                  Graph G;
                  G.graph(v);

                          for (int i = 0; i < e; i++){
                                  cout<<endl<<endl<<"\tEnter the "<<i+1<<" start edge :- ";  // 0 to 1
                                  cin>>startedge;
                                  cout<<endl<<"\tEnter the "<<i+1<<" end edge :- ";
                                  cin>>endedge;
                                  G.addEdge(startedge,endedge);
                          }
                      int n;
                      int i = 0;
                      cout<<endl<<endl<<endl<<"\tFor how many different starting point you want to check:- ";
                      cin>>n;
                      cout<<endl<<"\tGoing path: ";
                         while(i<n){
                            cout<<endl<<endl<<endl<<"\tEnter the source node or office of Salesperson:- ";
                                cin>>source;
                                cout<<"\t"<<endl<<"\t";
                                G.BFS(source);  // initial point 0
                                i++;
                            }


                    }
};

// Function to fill the empty adjacency matrix
void Graph :: graph(int v)
{
    this->v = v;
    adj = new int*[v];
    for (int row = 0; row < v; row++) {
        adj[row] = new int[v];
        for (int column = 0; column < v; column++) {
            adj[row][column] = 0;
        }
    }
    cout<<endl<<endl;
    for (int row = 0; row < v; row++) {
       cout<<endl<<"\t\t\t\t";
        for (int column = 0; column < v; column++) {
            cout<<"   "<<adj[row][column];
        }
        cout<<endl;
    }
}

// Function to add an edge to the graph
void Graph::addEdge(int start, int end)
{
    // Considering a bidirectional edge
    int dist;
    cout<<endl<<endl<<"\t Enter the distance between these two node:- ";
    cin>>dist;

```

```cpp
    adj[start][end] = dist;
    adj[end][start] = dist;

        cout<<endl<<endl<<"\tSo the new matrix is:- "<<endl<<endl;
        for (int row = 0; row < v; row++){
           cout<<endl<<"\t\t\t\t";
             for (int column = 0; column < v; column++) {
                 cout<<"    "<<adj[row][column];
             }
           cout<<endl;
           }

     }

     // Function to perform BFS on the graph
     void Graph::BFS(int start)
     {
        // Visited vector to so that
        // a vertex is not visited more than once
        // Initializing the vector to false as no
        // vertex is visited at the beginning
        vector<bool> visited(v, false);
        int total = 0;
        vector<int> q;
        q.push_back(start);

        // Set source as visited
        visited[start] = true;

        int vis;
        while (!q.empty()) {
            vis = q[0];

            // Print the current node
            cout << vis << " ";
            q.erase(q.begin());         // vis is empty now

            // For every adjacent vertex to the current vertex
            for (int i = 0; i < v; i++) {
                if (adj[vis][i] > 0 && (!visited[i])) {

                    // Push the adjacent node to the queue
                    total = total + adj[vis][i];
                    q.push_back(i);

                    // Set
                    visited[i] = true;
                }
            }
        }
        cout<<endl<<endl<<"\tReturning path: "<<vis<<" "<<start;
        cout<<endl<<endl<<"\tThe total distance traversed is:- "<<total;
    }

    // Driver code
    int main()
    {
          system("Color C0");
        Calling obj;
        obj.Call();
        return 0;
    }

```

❖ Part-2 Figure:



❖ Part-2 Code Output:



```
################# Finding the path of salesperson and in which city he has sold his products #################


# Enter the total Number of Cities the salesperson wanna visit:- 5

# Enter the total number of roads availabe:- 7

# So the initially he hasn't visited any city so the matrix is:-


                          0   0   0   0   0

                          0   0   0   0   0

                          0   0   0   0   0

                          0   0   0   0   0

                          0   0   0   0   0

~Enter the 1 start edge :- 0

~Enter the 1 end edge :- 1

 Enter the distance between these two node:- 5

-> So the new matrix is:-

                          0   5   0   0   0

                          5   0   0   0   0

                          0   0   0   0   0

                          0   0   0   0   0

~Enter the 2 start edge :- 0

~Enter the 2 end edge :- 2

 Enter the distance between these two node:- 6

-> So the new matrix is:-

                          0   5   6   0   0

                          5   0   0   0   0

                          6   0   0   0   0

                          0   0   0   0   0

                          0   0   0   0   0

~Enter the 3 start edge :- 0

~Enter the 3 end edge :- 3

 Enter the distance between these two node:- 4

-> So the new matrix is:-

                          0   5   6   4   0

                          5   0   0   0   0

                          6   0   0   0   0

                          4   0   0   0   0

                          0   0   0   0   0
```

```
~Enter the 4 start edge :- 1

~Enter the 4 end edge :- 2

 Enter the distance between these two node:- 8

-> So the new matrix is:-

                           0   5   6   4   0

                           5   0   8   0   0

                           6   8   0   0   0

                           4   0   0   0   0

                           0   0   0   0   0

~Enter the 5 start edge :- 2

~Enter the 5 end edge :- 3

 Enter the distance between these two node:- 2

-> So the new matrix is:-

                           0   5   6   4   0

                           5   0   8   0   0

                           6   8   0   2   0

                           4   0   2   0   0

                           0   0   0   0   0

~Enter the 6 start edge :- 1

~Enter the 6 end edge :- 4

 Enter the distance between these two node:- 9

-> So the new matrix is:-

                           0   5   6   4   0

                           5   0   8   0   9

                           6   8   0   2   0

                           4   0   2   0   0

                           0   9   0   0   0

~Enter the 7 start edge :- 3

~Enter the 7 end edge :- 4

 Enter the distance between these two node:- 7

-> So the new matrix is:-

                           0   5   6   4   0

                           5   0   8   0   9

                           6   8   0   2   0

                           4   0   2   0   7

                           0   9   0   7   0
```

```
# For how many different starting point you want to check:- 3


Enter the source node or office of Salesperson:- 0

Going path: 0 1 2 3 4

Returning path: 4 0

# The total distance traversed is:- 24

Enter the source node or office of Salesperson:- 3

Going path: 3 0 2 4 1

Returning path: 1 3

# The total distance traversed is:- 18

Enter the source node or office of Salesperson:- 4

Going path: 4 1 3 0 2

Returning path: 2 4

# The total distance traversed is:- 29
```

❖ Conclusion:

The following two conclusion came after doing the above project: -

1) The tree is a hierarchical and non-parametric data structure. It is used in data science to build predictive models as it can handle large amounts of data and can be validated statistically.

2) Graph theory is an exceptionally rich area for programmers and designers. Graphs can be used to solve some very complex problems, such as least cost routing, mapping, program analysis, and so on. Network devices, such as routers and switches, use graphs to calculate optimal routing for traffic. Note: - In the above project we tried to map the path of a salesperson.