EMERGENCY SERVICE SYSTEM

A MINI-PROJECT REPORT

of

BACHELOR OF TECHNOLOGY

in

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

at

DAYANANDA SAGAR UNIVERSITY

SCHOOL OF ENGINEERING, BANGALORE-560068

SEMESTER

Course Code: 16CS209

ANALYSIS AND DESIGN OF ALGORITHMS

DAYANANDA SAGAR UNIVERSITY

CERTIFICATE

This is to certify that the Analysis and Design of Algorithms Mini-Project report entitled **"Emergency Service System"** being submitted by Ananth Desai (ENG18CS0034) to Department of Computer Science and Engineering, School of Engineering, Dayananda Sagar University, Bangalore, for the 4th semester B.Tech C.S.E of this university during the academic year 2019-2020.

Date:_____ _____

Signature of the Faculty in Charge

_____

Signature of the Chairman

DAYANANDA SAGAR UNIVERSITY

CERTIFICATE

This is to certify that the Analysis and Design of Algorithms Mini-Project report entitled **"Emergency Service System"** being submitted by Ashish Sreenivas (ENG18CS0048) to Department of Computer Science and Engineering, School of Engineering, Dayananda Sagar University, Bangalore, for the 4th semester B.Tech C.S.E of this university during the academic year 2019-2020.

Date:_____ _____

Signature of the Faculty in Charge

_____

Signature of the Chairman

DAYANANDA SAGAR UNIVERSITY

CERTIFICATE

This is to certify that the Analysis and Design of Algorithms Mini-Project report entitled **"Emergency Service System"** being submitted by B Evans Nikith Royan (ENG18CS0054) to Department of Computer Science and Engineering, School of Engineering, Dayananda Sagar University, Bangalore, for the 4th semester B.Tech C.S.E of this university during the academic year 2019-2020.

Date:_____ _____

Signature of the Faculty in Charge

_____

Signature of the Chairman

DAYANANDA SAGAR UNIVERSITY

CERTIFICATE

This is to certify that the Analysis and Design of Algorithms Mini-Project report entitled **"Emergency Service System"** being submitted by Burhan Baig (ENG18CS0061) to Department of Computer Science and Engineering, School of Engineering, Dayananda Sagar University, Bangalore, for the 4th semester B.Tech C.S.E of this university during the academic year 2019-2020.

Date:_____ _____

Signature of the Faculty in Charge

_____

Signature of the Chairman

ACKNOWLEDGEMENT

We are pleased to acknowledge **Bindu Madavi K.P, Assistant Professor,** Department of Computer Science & Engineering for her invaluable guidance, support, motivation, and patience during the course of this mini-project work.

We extend our sincere thanks to **Dr. Banga M.K**, **Chairman,** Department of Computer Science & Engineering who continuously helped us throughout the project, and without his guidance, this project would have been an uphill task.

We have received a great deal of guidance and co-operation from our friends and we wish to thank one and all that have directly or indirectly helped us in the successful completion of this mini-project work.

Team Members

Ananth Desai

Ashish Sreenivas

B Evans Nikith Royan

Burhan Baig

Abstract

Organ transplantation in India comes with a varied number of unwanted discrepancies. Lack of efficiently centralized metadata to generate potential matches in real-time as well as transport issues and traffic congestion pose a major threat to successful transplants in a diverse country like India. The problem also exists in fire emergencies and crime-related emergencies.

The main aspect of approaching the problem is to represent the map as a two-dimensional matrix with rows and columns being the node/location numbers. The corresponding values of a row and a column represent the distance between the two nodes. Each node is an object of a class that holds the information about the type and is assigned a number.

Dijkstra's algorithm is used to find the shortest path if it exists between two nodes on the map. This algorithm operates on the 2-D matrix or "Graph" and the shortest path and the path distance is given to the user as output.

TABLE OF CONTENTS

Chapter 1

INTRODUCTION

1.1 Problem Statement:

Given a grid consisting of locations as nodes, find the shortest path and the path distance between any two given nodes.

1.2 Objectives of the project:

The shortest path and the path distance between two locations are to be calculated given the distances between all the nodes of the graph.

Chapter 2

SYSTEM REQUIREMENTS

2.1 Functional Requirements

The following functions are used in the project:

- ***get_graph()*** : input the adjacency matrix form the user.
- ***put_graph()*** : Print the adjacency matrix to the user.
- ***show_info()*** : Print the number of different types of nodes present and display them to the user.
- ***input_graph()*** : Input the number of nodes and their types.
- ***increase_graph()*** : Increase the memory of the matrix by adding a row and a column.
- ***add_graph()*** : Input the details of the node to be added and call increase_graph().
- ***Dijkstra()*** : Find the closest source given the destination node.
- ***put_path()*** : Prints the shortest path and the path distance to the user.
- ***Emergency()*** : Input the source node from the user and call Dijkstra().
- ***delete_node()*** : removes the specified node from the grid.
- ***choice_entry()*** : input the operation to be performed by the program.

2.2 Software and Hardware Requirements:

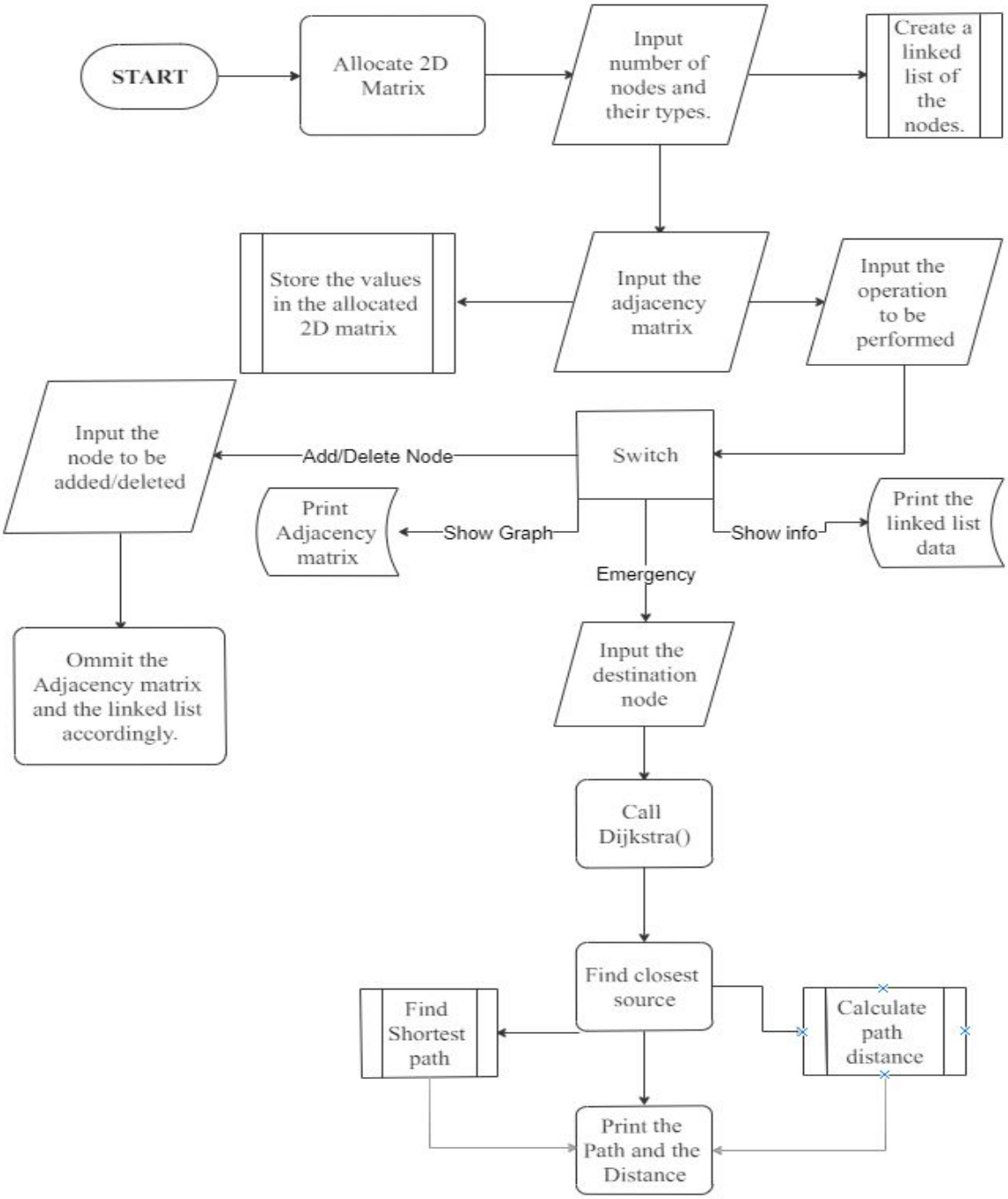
- Software requirements:
  - Code Blocks
  - XTerm

- Hardware requirements:
  - Laptop / PC
  - Keyboard
  - Mouse

Chapter 3

System Design

3.1 Architecture/Data Flow Diagrams:



3.2 Modules

- Input Module:
  - Graph input module
  - Linked list module
  - Choice entry

- Process Module:
  - Data allocation module
  - Algorithm module
    - Path determination module
    - Distance calculation module
  - Memory alteration module

- Output Module:
  - Graph output module
  - Info output module
  - Path - Distance output

Chapter 4

System Implementation

4.1 Module Description

- ***Graph input module***: Uses the data allocation module to allocate data for the 2D matrix and stores the graph elements input by the user.

- ***Linked list module***: Uses the data allocation module to allocate data to the class objects. Input the nodes from the user and create a linked list.

- ***Choice entry module***: Input the choice of operation from the user.

- ***Data Allocation module***: Allocate the appropriate memory space dynamically. Eg. The graph uses a dynamically allocated 2D matrix whereas the nodes are objects of a class.

- ***Path Determination module***: Given the destination node, this module finds the shortest path to the appropriate source node, i.e., if the required service is a medical emergency, then the source node will be one of the hospitals on the grid.

- ***Distance calculation module***: Given the destination node it calculates the distances to all the appropriate source nodes and selects the one having the least travel distance.

- ***Data alteration module***: When selected the add/delete node operation, this module reallocates the memory of the graph and the linked list appropriately. It then calls the *Input modules* to input the data into the added memory space.

- ***Graph output module***: This module prints the adjacency matrix to the user.

- ***Info output module:*** This module prints the number of nodes, corresponding node types, and the number of nodes of each type on the graph.

- ***Path - Distance output:*** This module prints the shortest path between the source node and the destination node calculated in the *Path Determination* and *Distance calculation* modules of the program.

4.2 Code

```
#include<iostream>
#include<stdlib.h>

#include<string>

#include<iomanip>

#include<time.h>

#include<ctime>



#define MAX 9999



using namespace std;



int **graph;

int police_count = 0, med_count = 0, fire_count = 0,
org_count = 0, home_count = 0;         //COUNT
```

```
typedef class Node{             // Main class of the
node : Represents the type of location


    public:

        int number;             // Node number

        char type;              // Type of node

        char bl_type;           // Blood group

        Node *next;             // Next node




        Node(int num, char typ){        //Constructor
initializing the values of the node
            number = num;

            type = typ;

            next = NULL;

        }

        Node(){

            number = 0;

            type = '$';

        }
```

```
        int display_number(){
            cout<<number;
            return 0;
        }
        int display_type(){
            cout<<type;
            return 0;
        }
        int display(){
            display_number();
            cout<<"\t\t";
            display_type();
            cout<<endl;
            return 0;
        }
};

Node* start = NULL;
Node* last = NULL;
```

```
int get_graph(int dimension){                // INPUT
ADJACENCY MATRIX



   // Dynamic memory allocation of 2d array;

   graph = (int**)malloc(dimension * sizeof(int*));



   for(int i=0;i<dimension;i++)

       graph[i] = (int*)malloc(dimension *
sizeof(int));



   cout<<"Enter the distance between the following
locations   (9999 for nodes not connected): "<<endl;



   for(int i=0;i<dimension;i++)

       for(int j=i+1;j<dimension;j++){

           cout<<"Node "<<i+1<<" to Node "<<j+1<<" : ";

               cin>>graph[i][j];

           graph[j][i]=graph[i][j];
```

```
        }
    for(int i=0, j=0; i<dimension, j<dimension; i++,j++)
        graph[i][j] = 0;


    return 0;
}


int put_graph(){                    //OUTPUT ADJACENCY
MATRIX
    int dimension = last->number;
    Node* temp=start;
    cout<<"Node\t";
    for(int i=0;i<dimension;i++){
        cout<<temp->type<<"\t";
        temp=temp->next;
    }
    temp=start;
    cout<<endl<<endl;
    for(int i=0;i<dimension;i++){
```

```
        cout<<temp->type<<"\t";
        for(int j=0;j<dimension;j++)
            if(graph[i][j]!=9999)
                cout<<graph[i][j]<<"\t";
            else
                cout<<"-"<<"\t";
        cout<<endl;
        temp=temp->next;
    }
    temp = NULL;
    delete temp;
    return 0;
}


int count_update(char node_type){           // Updating
the number of service points available


    switch(node_type){
        case 'P':
```

```
        case 'p': police_count++;
                  break;
        case 'm':
        case 'M': med_count++;
                  break;
        case 'f':
        case 'F': fire_count++;
                  break;
        case 'h':
        case 'H': home_count++;
                  break;


        case 'o':
        case 'O': org_count++;
                  break;


        default: cout<<"Not a valid type. Enter
again."<<endl;
                  return 0;
```

```
    }

    return 1;

}



int show_info(){            /*  DISPLAY THE GRAPH INFO
*/

    Node* temp = start;

    cout<<endl<<endl<<"Node\tType"<<endl;

    for(int i = 0;i<last->number, temp!=NULL; i++){

        cout<<i+1<<"\t"<<temp->type<<endl;

        temp=temp->next;

    }

    cout<<"Number of Police
Stations\t:\t"<<police_count<<endl;

    cout<<"Number of Hospitals\t\t:\t"<<med_count<<endl;

    cout<<"Number of Fire
Stations\t\t:\t"<<fire_count<<endl;

    cout<<"Number of Houses\t\t:\t"<<home_count<<endl;

    cout<<endl;

    return 0;

}
```

```
int input_graph(){                          //Input initial
graph
   int num_of_nodes;
   char node_type;
   cout<<"Enter the number of nodes: ";
       cin>>num_of_nodes;
   for(int i=0;i<num_of_nodes;i++){
       goto again;
       again:
       cout<<"Enter the type of node "<<i+1<<": ";
           cin>>node_type;
           int n = count_update(node_type);
           if(n==0)
               goto again;
       Node *node =new Node(i+1,node_type);
      if(start==NULL){
           start=last=node;
       }
```

```
        else{
            last->next=node;
            last=last->next;
        }
    }
    get_graph(num_of_nodes);              //Input initial
adjacency matrix
    return 0;
}



//INCREASE THE SIZE OF THE GRAPH
int increase_graph(){                     //UPDATE THE
GRAPH WITH INCREASED SIZE
    int temp_graph[last->number][last->number];
    for(int i=0;i<last->number;i++)
        for(int j=0;j<last->number;j++)
            temp_graph[i][j] = graph[i][j];


    delete graph;
```

```
    // Dynamic memory allocation of 2d array;

   graph = (int**)malloc((last->number+1)*
sizeof(int*));



   for(int i=0;i<last->number+1;i++)

       graph[i] = (int*)malloc((last->number+1) *
sizeof(int));



   for(int i=0;i<last->number;i++)

       for(int j=0;j<last->number;j++)

           graph[i][j]=temp_graph[i][j];



   return 0;

}



int add_graph(){              // INPUT THE ELEMENTS OF
THE UPDATED GRAPH

   increase_graph();

   char typ;
```

```
again1:

   cout<<"Enter the type of node "<<last->number+1<<" :
";

       cin>>typ;

   int n = count_update(typ);

   if(n==0)

       goto again1;

   Node* node = new Node(last->number+1,typ);

   last->next = node;

   last = last->next;

   cout<<"Enter the distance between the following
locations   (9999 for nodes not connected): "<<endl;

   for(int i=0;i<last->number-1;i++){

       cout<<"Node "<<last->number<<" to Node "<<i+1<<"
: ";

           cin>>graph[last->number-1][i];

           graph[i][last->number-1] =
graph[last->number-1][i];

   }

   graph[last->number-1][last->number-1] = 0;

   return 0;

}
```

```
//DISPLAY THE SHORTEST PATH

void put_path(int path[], int dest_node){

   if(path[dest_node]==-1)

       return;



   put_path(path, path[dest_node]);

       cout<<" <- "<<dest_node+1;

}



int chooseVertex(int distance[], int visited_array[]){
/*  CHOOSES THE NEXT MIN WEIGHT VERTEX   */

   int min_node;

   int  min_dist = MAX;

   for(int i=0;i<last->number;i++)

       if(visited_array[i]==0 && distance[i] <=
min_dist){

           min_dist = distance[i];

           min_node = i;
```

```
    }

    return min_node;

}



int chooseService(int distance[], int path[] , char
node_type, int node_req){

    Node* temp = start;

    int dist[last->number],min_node,k=0, min_dist = MAX;

    for(int i=0;i<last->number, temp!=NULL; i++){

        if(((int)temp->type==(int)node_type) ||
((int)temp->type==(int)node_type+32)){      // CHOOSE
THE APPROPRIATE SERVICE FROM ALL THE SHORTEST

            dist[k] = distance[i];

            if(dist[k]<min_dist){

                min_dist = dist[k];

                min_node = temp->number;

            }

            k++;

        }

        temp=temp->next;

    }
```

```
   cout<<endl<<"The fastest service is given by node
"<<min_node<<" in the following route:
"<<endl<<node_req;

       put_path(path,k-1);

       cout<<"\b"<<min_node;

   return min_dist;

}
```

```
int Dijkstra(int node_req, char node_type){

   int distance[last->number];     //ARRAY WHICH HOLDS
THE DISTANCE TO ALL THE NODES OF THE SAME TYPE

   int visited_array[last->number];        // ARRAY
WHICH HOLD THE VISITED NODES

   int path[last->number];     //HOLDS THE PATH TO THE
SHORTEST ROUTE



   float start_time = clock();




   for(int i=0;i<last->number;i++){        //INITIALIZE
THE ARRAY VALUES: DIST TO INFINITY AND VISITED ALL TO
ZERO
```

```
        distance[i] = MAX;

        visited_array[i] = 0;

        path[i]=-1;

    }

      path[0]=-1;      //SOURCE NODE

    distance[node_req-1] = 0;        // SET SOURCE DIST
ZERO



    for(int i=0;i<last->number-1;i++){

        int k = chooseVertex(distance, visited_array);



        visited_array[k] = 1;



        for(int j=0;j<last->number;j++)

            if(!visited_array[j] && graph[k][j]  &&
distance[j]>( distance[k] + graph[k][j] )){

                path[j] = k;

                distance[j] = distance[k] + graph[k][j];

            }

    }
```

```
   float end_time = clock();

   float time_taken = (start_time +
end_time)/CLOCKS_PER_SEC;



   cout<<endl<<endl;

   distance[0] = chooseService(distance, path,
node_type, node_req);

   cout<<endl<<" and it is "<<distance[0]<<" units away
from your location."<<endl;



   cout<<"Start time: "<<start_time<<endl<<"End time:
"<<end_time<<endl<<"Time taken to produce result:
"<<time_taken<<endl<<endl;

   return 0;

}



int Emergency(char node_type){

   int node_req;

   cout<<endl<<"Enter the node at which service is
required: ";

       cin>>node_req;

   if(node_req>last->number)
```

```
        cout<<"Location not present on the graph. Please
try again."<<endl;
    else{
        Dijkstra(node_req,node_type);
    }
    return 0;
}


int Organ_Emergency(){
    Node *temp = start;
    int node_req;
again2:
    cout<<"Enter the source hospital from the following
list of hospitals: "<<endl;
    cout<<"Type\tNode"<<endl;
    while(temp!=NULL){
        if(temp->type=='M'|| temp->type=='m')
            cout<<temp->type<<"\t"<<temp->number<<endl;
            temp=temp->next;
    }
```

```
    cin>>node_req;

    temp=start;

    while(temp!=NULL){

        if(temp->number==node_req){

            Dijkstra(node_req,'M');

            break;

        }

        else

            temp=temp->next;

    }

    if(temp==NULL){

        cout<<"Source is not a hospital! Please enter
again."<<endl;

        goto again2;

    }

    temp==NULL;

    delete temp;

    return 0;

}
```

```
int delete_node(int node_num){

   int u=0,v=0;

   int temp_graph[last->number][last->number];

   for(int i=0;i<last->number;i++)

       for(int j=0;j<last->number;j++)

           temp_graph[i][j] = graph[i][j];



   delete graph;



    // Dynamic memory allocation of 2d array;

   graph = (int**)malloc((last->number-1)*
sizeof(int*));



   for(int i=0;i<last->number-1;i++)

       graph[i] = (int*)malloc((last->number-1) *
sizeof(int));



   for(int i=0;i<last->number;i++){

        v=0;
```

```
        for(int j=0;j<last->number;j++){
            if(i==node_num-1){
                i++;
            }
            if(j==node_num-1){
                j++;
            }
                graph[u][v] = temp_graph[i][j];
                v++;
        }
        u++;
    }
    return 0;
}


int remove_graph(){
    int node_num;
    cout<<"Enter the node you want to remove: ";
        cin>>node_num;
```

```
   delete_node(node_num);          // UPDATES THE GRAPH
AFTER DELETING THE ROW AND COLUMN

   Node* temp2 = start;            // UPDATION OF THE
LINKED LIST

   Node* temp1;

   temp1 = temp2->next;

   if(temp2->number==node_num){        //STARING
ELEMENT IS THE DELETED ELEMENT CONDITION

       start = temp1;

       temp2->next = NULL;

       delete temp2;

   }

   else{

       while(temp1!=NULL){

           if(temp1->number==node_num){

               temp2->next = temp1->next;

               temp1->next = NULL;

               delete temp1;

               break;

           }

           temp1 = temp1->next;
```

```
            temp2 = temp2->next;

        }

    }

    temp1 = start;

    while(temp1!=NULL){

        if(temp1->number>node_num){

            temp1->number--;

        }

        temp1 = temp1->next;

    }

    cout<<"\n"<<"Node "<<node_num<<" deleted
successfully!!\n";

    return 0;

}



int choice_entry(){

    int n;



    cout<<endl<<endl;
```

```
   cout<<"1. Codes:"<<endl<<"\t Police emergency: \t
100"<<endl<<"\t Medical emergency: \t 108"<<endl<<"\t
Fire emergency: \t 104"<<endl<<"\t Organ Transplant: \t
110"<<endl;

   cout<<"2. Show the present graph."<<endl;

   cout<<"3. Add an element to the present
graph"<<endl;

   cout<<"4. Delete a node from the graph."<<endl;

   cout<<"5. Show Graph info."<<endl;

   cout<<"6. Quit"<<endl<<endl;

   cout<<"Enter your choice/code: ";

         cin>>n;

   return n;

}



int welcome(){

   for(int i=0;i<73;i++)

       cout<<"*";

   cout<<endl;

   cout<<"*"<<"\t\t"<<"WELCOME TO EMERGENCY SERVICE
PROVISION SYSTEM"<<"\t\t"<<"*"<<endl;

   for(int i=0;i<73;i++)
```

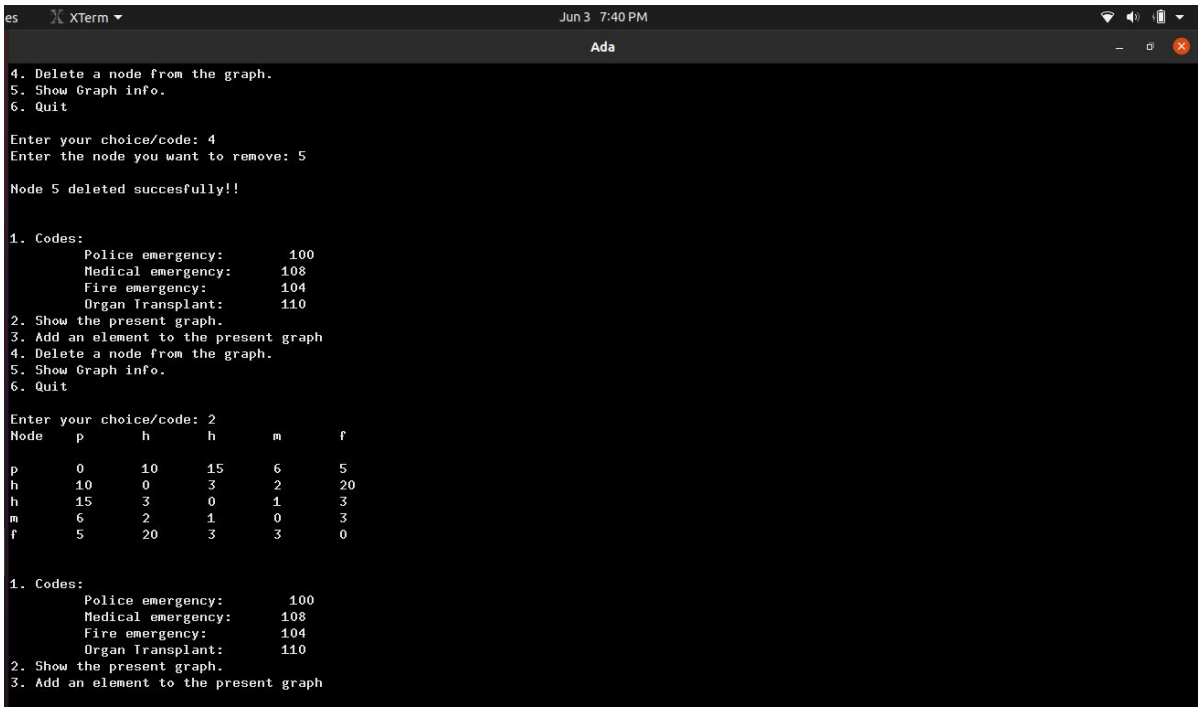
```
        cout<<"*";
    cout<<endl<<endl;
    return 0;
}



int main(){
    welcome();
    input_graph();
    int choice;
    do{
        choice = choice_entry();
        switch(choice){
            case 2: put_graph();
                    break;
            case 3: add_graph();
                    break;
            case 4: remove_graph();
```

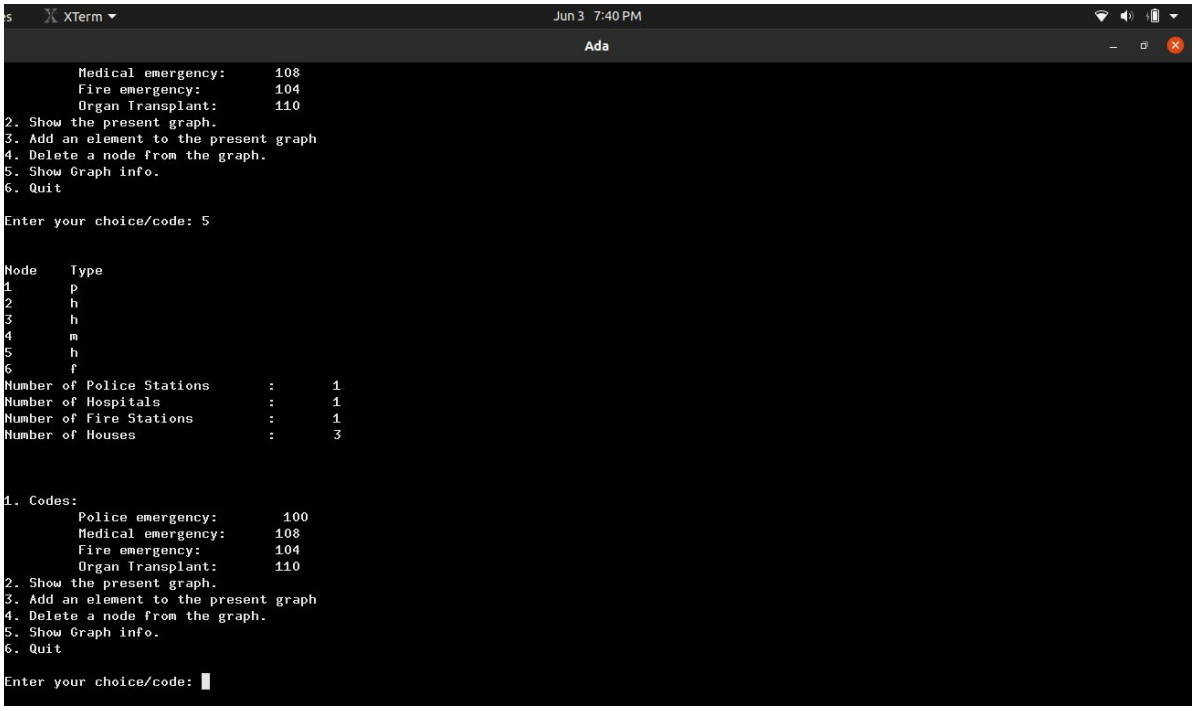
```
                break;
            case 5: show_info();
                break;
            case 6: exit(0);
                break;
            case 100:  if(police_count>0)
                        Emergency('P');
                    else
                        cout<<"No Police stations on
the grid.";
                    break;
            case 108: if(med_count>0)
                        Emergency('M');
                    else
                        cout<<"No Hospitals on the
grid.";
                    break;
            case 104: if(fire_count>0)
                        Emergency('F');
                    else
```

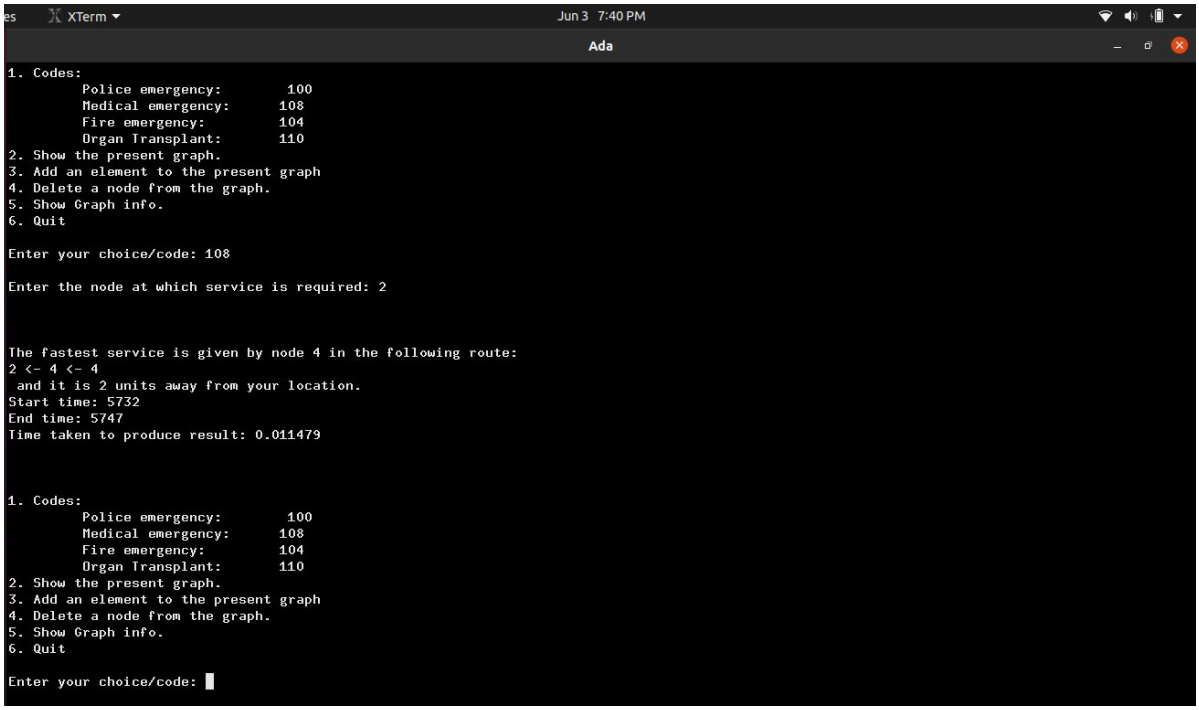
```
                                cout<<"No Fire Stations on
the grid.";
                            break;
                case 110: if(med_count>1)
                                Organ_Emergency();
                            else
                                cout<<"Only one/none
hospital available on the grid.";
                            break;
                default: cout<<"Not a Valid choice!"<<endl;
                            break;
            }
    }while(choice!=6);
    return 0;
}
```

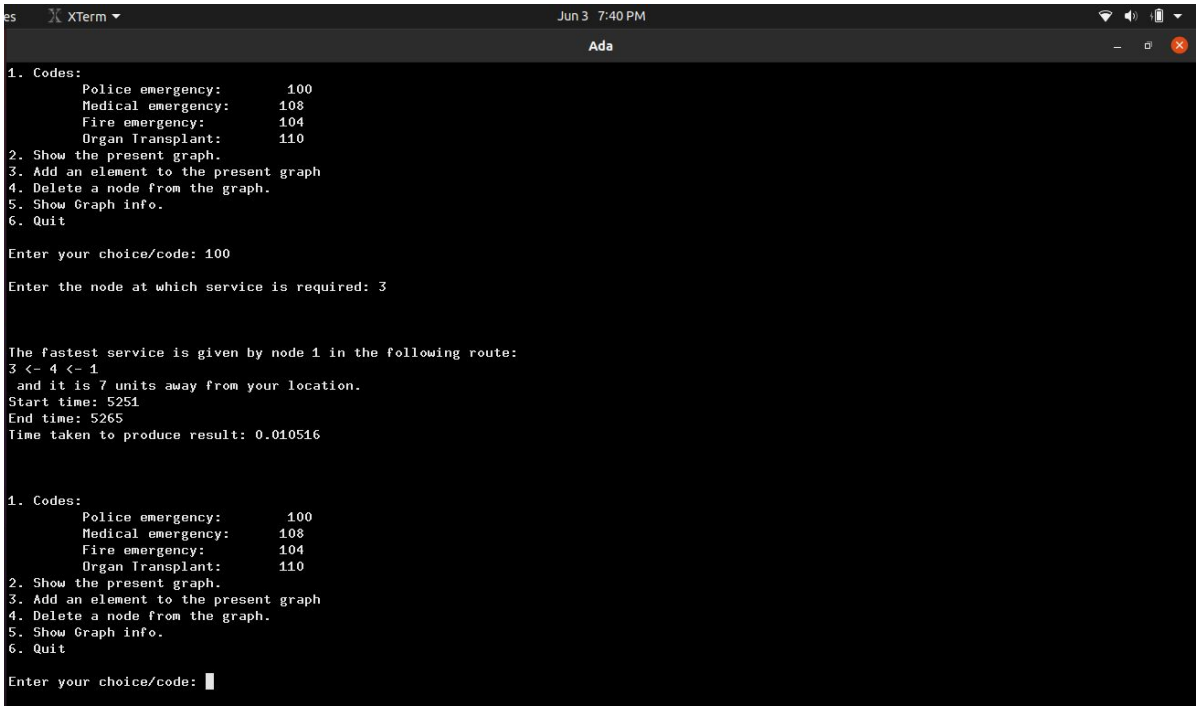
Chapter 5

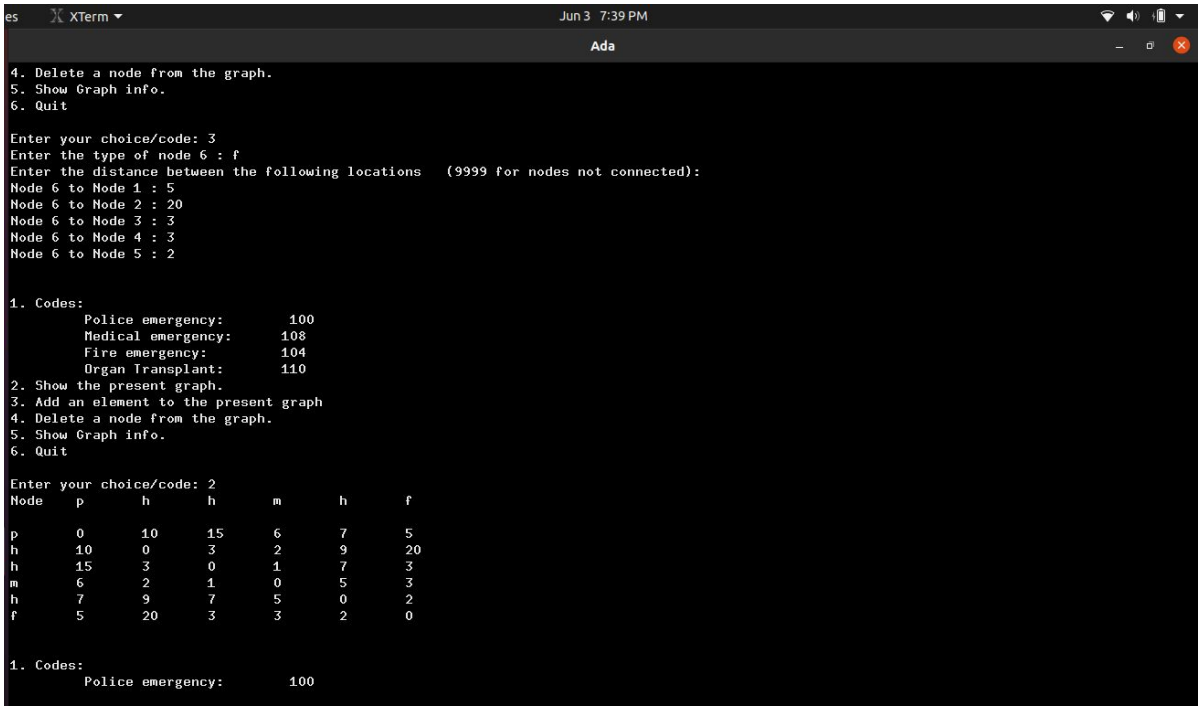
Output

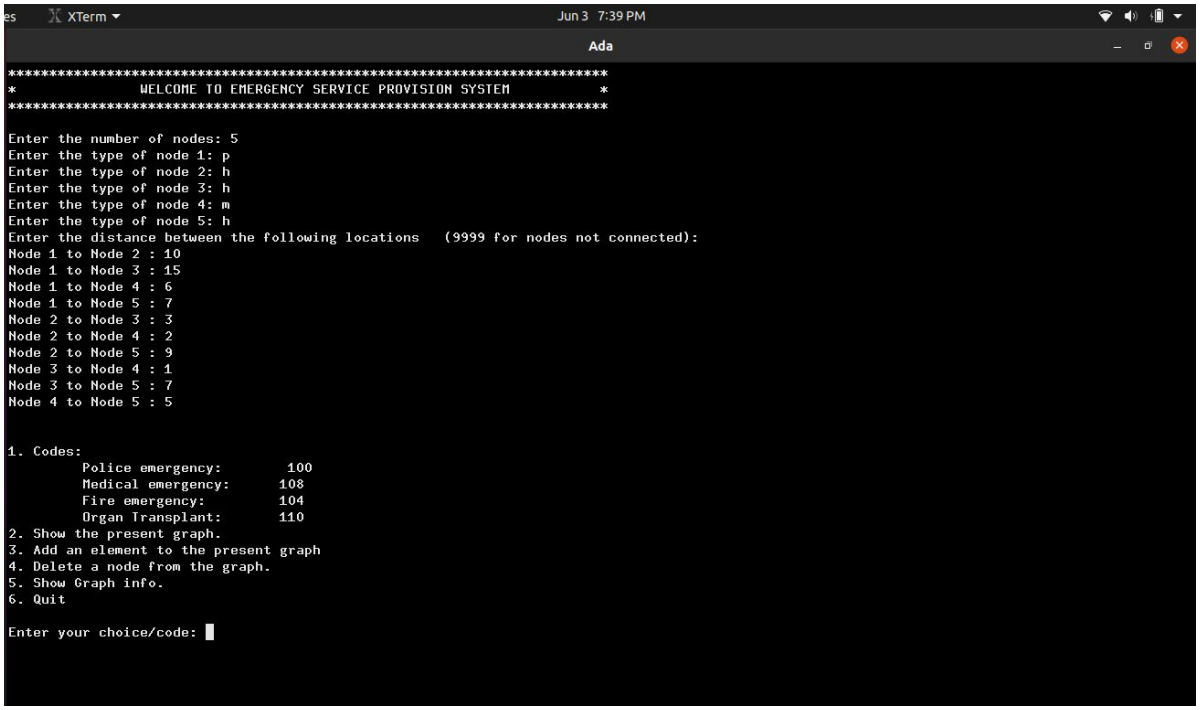





es
XTerm
Jun 3 7:40 PM
Ada
1. Codes:
Police emergency: 100
Medical emergency: 108
Fire emergency: 104
Organ Transplant: 110
2. Show the present graph.
3. Add an element to the present graph
4. Delete a node from the graph.
5. Show Graph info.
6. Quit
Enter your choice/code: 108
Enter the node at which service is required: 2
The fastest service is given by node 4 in the following route:
2 <- 4 <- 4
and it is 2 units away from your location.
Start time: 5732
End time: 5747
Time taken to produce result: 0.011479
1. Codes:
Police emergency: 100
Medical emergency: 108
Fire emergency: 104
Organ Transplant: 110
2. Show the present graph.
3. Add an element to the present graph
4. Delete a node from the graph.
5. Show Graph info.
6. Quit
Enter your choice/code:


es
XTerm
Jun 3 7:40 PM
Ada
1. Codes:
Police emergency: 100
Medical emergency: 108
Fire emergency: 104
Organ Transplant: 110
2. Show the present graph.
3. Add an element to the present graph
4. Delete a node from the graph.
5. Show Graph info.
6. Quit
Enter your choice/code: 100
Enter the node at which service is required: 3
The fastest service is given by node 1 in the following route:
3 <- 4 <- 1
and it is 7 units away from your location.
Start time: 5251
End time: 5265
Time taken to produce result: 0.010516
1. Codes:
Police emergency: 100
Medical emergency: 108
Fire emergency: 104
Organ Transplant: 110
2. Show the present graph.
3. Add an element to the present graph
4. Delete a node from the graph.
5. Show Graph info.
6. Quit
Enter your choice/code:


es
XTerm
Jun 3 7:39 PM
Ada
4. Delete a node from the graph.
5. Show Graph info.
6. Quit
Enter your choice/code: 3
Enter the type of node 6 : f
Enter the distance between the following locations (9999 for nodes not connected):
Node 6 to Node 1 : 5
Node 6 to Node 2 : 20
Node 6 to Node 3 : 3
Node 6 to Node 4 : 3
Node 6 to Node 5 : 2
1. Codes:
Police emergency: 100
Medical emergency: 108
Fire emergency: 104
Organ Transplant: 110
2. Show the present graph.
3. Add an element to the present graph
4. Delete a node from the graph.
5. Show Graph info.
6. Quit
Enter your choice/code: 2
Node p h h m h f
p 0 10 15 6 7 5
h 10 0 3 2 9 20
h 15 3 0 1 7 3
m 6 2 1 0 5 3
h 7 9 7 5 0 2
f 5 20 3 3 2 0
1. Codes:
Police emergency: 100


es
XTerm
Jun 3 7:39 PM
Ada
****
* WELCOME TO EMERGENCY SERVICE PROVISION SYSTEM *
****
Enter the number of nodes: 5
Enter the type of node 1: p
Enter the type of node 2: h
Enter the type of node 3: h
Enter the type of node 4: m
Enter the type of node 5: h
Enter the distance between the following locations (9999 for nodes not connected):
Node 1 to Node 2 : 10
Node 1 to Node 3 : 15
Node 1 to Node 4 : 6
Node 1 to Node 5 : 7
Node 2 to Node 3 : 3
Node 2 to Node 4 : 2
Node 2 to Node 5 : 9
Node 3 to Node 4 : 1
Node 3 to Node 5 : 7
Node 4 to Node 5 : 5
1. Codes:
Police emergency: 100
Medical emergency: 108
Fire emergency: 104
Organ Transplant: 110
2. Show the present graph.
3. Add an element to the present graph
4. Delete a node from the graph.
5. Show Graph info.
6. Quit
Enter your choice/code:

Chapter 6

Conclusion

The above algorithm runs in O(<number_of_nodes>^{2}). The program successfully implements Dijkstra's algorithm to find the shortest path between the nodes.

References

- Algorithms : by Robert Sedgewick and Kevin Wayne.

- Data Structure and Algorithms analysis, by Clifford A Shaffer.

- https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm