



## **Ambulance Management System**

**Submitted By:**

**UMAIR AHMED, 2020-CS-3**

**KASHIR SAEED, 2020-CS-27**

**M. FARRUKH HAIDER, 2020-CS-45**

**Submitted To:**

**Mr. Samyan Qayyum Wahla**

**For Fulfillment of:**

**CS 261 Data Structure and Algorithm**

**Department of Computer Science**

**University of Engineering and Technology, Lahore**

## Contents

Abstract.....	6
Introduction .....	7
Background.....	7
Gitlab Link.....	7
Gitlab Track .....	7
Date of Last commit .....	7
Project Overview.....	8
Acronyms and Abbreviations.....	8
Operational Details .....	8
Employee .....	8
Add .....	8
Update .....	8
Delete .....	8
BUS.....	9
Add .....	9
Update .....	9
Delete .....	9
Hospital .....	9
Add .....	9
Update .....	9
Delete .....	9
Attendance .....	9
Employee.....	9
Buses.....	9
Prepare Shift: .....	9
Update Shift:.....	10
Call from Patient:.....	10
Project Flow.....	11
Data Structures .....	11

Stack .....	11
Code .....	12
Linkedlist .....	13
Code .....	13
Queue .....	15
Code .....	16
Tree.....	16
Code .....	17
Graph .....	24
Code .....	25
Object Oriented Model .....	28
Employee Class .....	28
Bus Class .....	28
Rescue Group Class .....	28
Hospital Class .....	28
Rescue Case Class .....	28
ECF Class .....	29
Tree and Node Class:.....	29
Queue and queue node class: .....	29
Stack and stack Node class:.....	29
Linklist and its node class: .....	29
UML-Diagram:.....	30
Wireframes .....	31
Employee Information Window: .....	31
Update Employee Window .....	32
Delete Employee Window.....	33
Hospital Registration Window .....	34
Update Hospital Window .....	35
Remove Hospital Window.....	36
Add Bus Window .....	37
Update Bus Window .....	38
Delete Bus Window .....	39
Attendance Sheet.....	40

CTWO Window.....	41
ECF Window .....	42
File Management .....	43
Data Management .....	45
Tasks Division According To Features .....	45
Testing.....	45
Collaboration.....	45
Integration:.....	46
Limitations.....	46
Future Goals .....	46
Conclusion .....	46

## **Acknowledgements**

By the will of Almighty Allah , we were able to complete our project before the deadline. We as a team, worked day in and night for the fulfillment of this project. Each member of the group participated to his fullest abilities. Moreover , few of our friends helped us with different problems which we as a team were unable to solve. Abdul Ahad and Musawir Ahmed helped us With the repository problems in the GitLab and GitHub. Mubashar Ahmed helped us with Dijkstra's Algorithm and with the efforts of our teacher Sir Samyan we were able to accomplish this . Without his help we wouldn't be able to complete our project with this efficiency.

## **Abstract**

The main objective of our project was to digitalize the ambulance service of Pakistan and make it more efficient. We tried to modify every sector which is involved in the process so that our product yields maximum efficiency. As soon as the call is received to the nearest office of rescue 1122, the operator shall direct the nearest group consisting of the compoder, driver and the available ambulance towards the location. Moreover the operator has to direct the group towards the nearest hospital for catering sensitive patients as early as possible. The ambulances should be parked in a queue to avoid any delay. The operator shall be able to track each group and their timing of arrival and departure will be recorded. Moreover, the group will be directed towards the shortest path to any respective hospital from the place of its destination. Hospital's management should entertain any critical patient which is rescued and they should inform the rescue workers regarding the availability of beds.

## Introduction

Ambulance management system will help us to manage all the ambulance services provided by the Health Care department. This system will keep the record of all the ambulances which are available for the service. It will also manage the record of all the hospitals of the city and the employees which are available for the job. This system can provide the service of ambulances at any corner of the city. The main object of this project is to provide the general health facilities to the local public. Our project is able to add the new employees and new buses. It is also able to register the hospitals to know the number of beds available in the region. Our ambulance management system works in three shifts. When new shift starts, initially, the attendance of all the drivers, co-drivers and ambulances is taken which are in the working condition. After that we have made the groups of the present staff. Each group consists of a driver, co-driver and an ambulance. As the call of the patient is received, our system checks either some group is available in order to rescue the patient or not. If yes, then our project checks the availability of the beds from the hospitals which are currently registered. If some bed is also available, then a rescue group is dispatched. This rescue group rescue's the patient and delivers it to the current available hospital. At the same time, an ECF is also generated which contains patient's id and the destination address where the patient will be delivered. Lastly, our project provides a solid platform to all the patients for their treatment so that they can be cured as soon as possible.

## Background

Our project is an organized system which is very disciplined in performing its duties. With the help of this project, our society has achieved various goals. As the call of the patient is received, than the patient is rescued and is delivered to the current available hospital. Our project is also very useful for the rescue group. As the call of the patient is received, this system indicates the shortest path from the rescue 1122 office to the patient's address. This shortest path is very useful for the rescue group so that they can rescue the patient as soon as possible. It is also very useful for the patient because he/she is cured in a minimum amount of time.

## Gitlab Link

<https://gitlab.com/umairahmedpaki7/dsa2021q46>

## Gitlab Track

Name	Commits
Umair Ahmed	25
Kashir Saeed	20
Muhammad Farrukh Haider	5

Table1: Repository track

## Date of Last commit

Friday, December 17, 2021

## Project Overview

Project Overview	Detail
Project Name	Ambulance Management System
Project Type	Desktop Application
Project Language	C#

Table2: Project overview

## Acronyms and Abbreviations

Acronym/Abbreviation	Meaning
CTWO	Computer Wireless Operator
WO	Wireless Operator
PTA	Pakistan Telecommunication Authority
GPS	Global Positioning System
ECF	Emergency Control Form
FIFO	First In First Out

Table3: Acronyms

## Operational Details

### Employee

#### Add

Admin of the application will be able to add the new employee. The attribute of the new employee are:

- Name
- Contact Number
- Permanent Address
- Gender
- Status
- Shift
- Father Name
- Current Address
- CNIC
- City
- Post
- ID

Admin will get the data and automatically the objects of the employee class will be created and saved.

#### Update

Admin will also be able to update the record of the employee. Firstly, he will search the employee whose data he wants to update. Then, he will update the attributes of the employee.

#### Delete

Admin will also be able to delete the employee from the application. In the same way, as in the update he first searches the employee and then deletes it.



## **BUS**

### **Add**

Admin of the application will be able to add the new bus. The attribute of the new bus are:

- Name
- Status
- Model
- Number

Admin will get the data and automatically the objects of the BUS class will be created and added in the tree.

### **Update**

Admin will also be able to update the record of the bus. Firstly, he will search the bus whose data he wants to update. Then, he will update the attributes of the bus.

### **Delete**

Admin will also be able to delete the bus from the application. In the same way, as in the update he first searches the bus and then deletes it.

## **Hospital**

### **Add**

Admin of the application will be able to register the hospital. The attributes of the hospital are:

- Name
- Address
- Total Beds
- Occupied Beds

Admin will get the data and automatically the objects of the Hospital class will be created and saved.

### **Update**

The incharge of the hospital emergency ward will be able to update the record of the hospital ie: update the number of occupied beds in the ward.

### **Delete**

Admin of the application will be able to delete the hospital. Firstly, the desired hospital will be searched and then it will be deleted.

## **Attendance**

### **Employee**

When it is going to be the time for any shift we first take the attendance of all the employees and place the present employees in the different queue according to the job type i.e Driver and Compoder.

### **Buses**

In this operation we check the status of the buses whether the bus is in working condition or not. If the bus is in working condition then, we place it in the queue.

## **Prepare Shift:**

After the operation of the attendance we prepare the shift group in order to rescue the people. Each working group has following attributes

- Group Id                      • Driver
- Compoder                    • Bus

When all the groups are done then we place them in the form of the queue and the buses are also parked in the same sequence of the queue in the active position.

### Update Shift:

When it is going to be the time of a new shift, the admin will update the shift and load the record of the hospitals in the stack that are available for receiving the patient.

### Requeue the dispatched Group:

The dispatched group is dequeued from the queue once it gets the call to attend to a patient . As soon as it returns back , it is again enqueued in the queue and is ready to be launched again .

### Call from Patient:

Now moving towards the main operation of the application, when the caller calls the rescue number, it is first directed to the CTWO. He will check the authentication of the emergency and fill the ECF form then direct the call to the WO. The WO will get the location of the caller by the assistance of PTA and calculate the shortest path to reach the required destination. This task will be done by the assistance of the GPS. In the meanwhile, he will check the availability of the bed in the stack of the hospitals. Then, the rescue group is dequeue from the queue of the shift group and the information is given to the current group i.e pickup location, address of the hospital and the shortest path to reach the pickup point and the hospital. After dispatching the patient to the hospital, the rescue group will give the complete information to the WO relating to the ECF form. Then, WO reports to the CTWO that the patient is successfully rescued and ECF form will be uploaded on the web server.

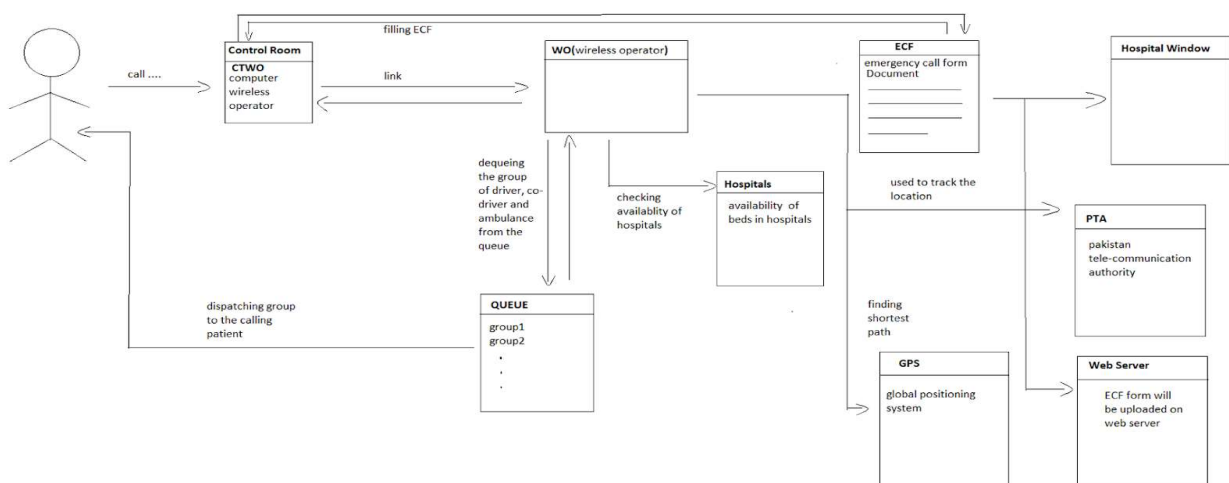


Figure1 : Flow of rescue operation

## Project Flow

In this project, we will be able to add new employees and new buses. We are also able to register the hospitals to know the number of beds available in the region. When a new shift starts, our first role is to call the attendance that will tell us the number of drivers, co-drivers and buses that are in working condition. Then, we will make the groups of all the present staff and each group contains one Driver, one Co-Driver and one Bus. When we receive the call, we take some basic information from the caller i.e the point of arrival and the current situation of the patient. We will also automatically get the current location of the caller by the assistance of the PTA. After getting this information we will check whether a group is available for the rescue. If yes, then we search the availability of the bed from the registered hospitals. If bed is also available, then we assign the current group to that patient. After that we will find the shortest path to reach the patient by the assistance of GPS. Then, the rescue group will receive the patient and dispatch him to the current available hospital. At the end we will upload the ECF form on the rescue web server so that the warden of the patient can search and trace their patient on the website.

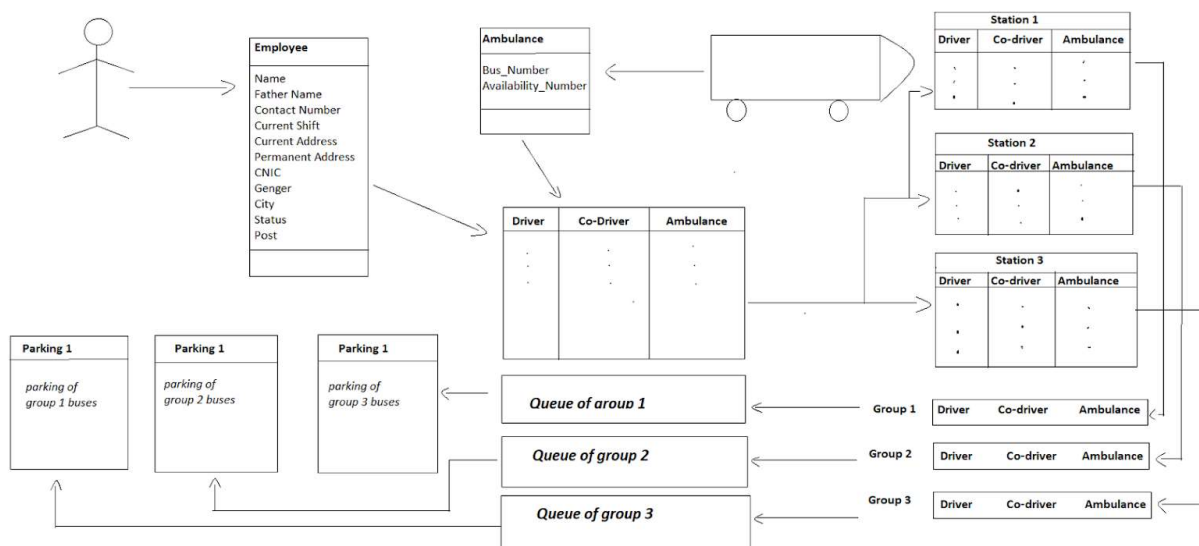


Figure2 : Flow of project

## Data Structures

### Stack

The stack is an abstract data type in the data structures. The stack data type follows the **LIFO** principle or in other words we can say that it follows the **Last in First out** principle. Basically, the stack contains two main operations ie: **push** and **pop**. The push operation is used to push the data into the stack. Where, as the pop operation deletes the data from the stack. The pop operation deletes the most recent element which is pushed into the stack. If we have to use the most recently stored data in a frequent manner, then at that stage stack is used. In our project, the data of all the hospitals is managed with the help of stack. If any hospital has number of available beds greater than zero that particular hospital is pushed into the stack. So, the stack contains all the hospitals

whose number of available beds are greater than zero. As the call of the patient is received then the number of available beds of the nearest hospital is checked. If it is greater than zero then a rescue group is dispatched and total number of available beds is decremented by one. As the number of available beds become zero then that particular hospital is popped from the stack.

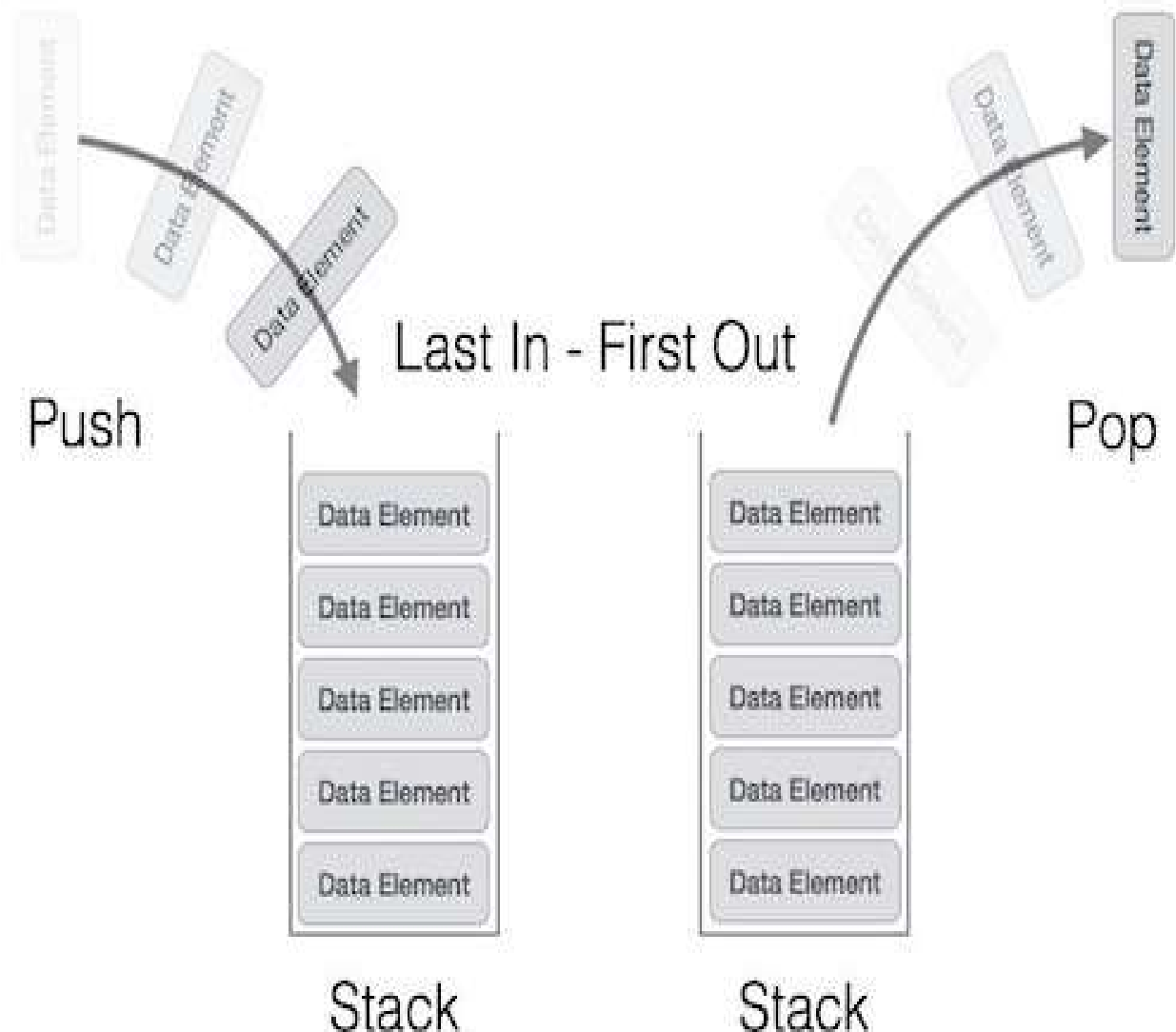


Fig 3: stack visualization

**Code**

```
class Stack
{
    public StackNode head;
    public void push(StackNode key)
    {
        if (this.head == null)
        {
            this.head = key;
            this.head.next = null;
        }
    }
}
```

```

    }
    else
    {
        StackNode temp = this.head;
        this.head = key;
        this.head.next = temp;
    }
}
public void pop()
{
    if (this.head != null)
    {
        this.head = this.head.next;
    }
}
public StackNode top()
{
    return this.head;
}
}

```

### Linkedlist

It is a type of data structure in which all the data is stored in a linear manner and this data structure is completely different from the arrays. Although, the data in linklist is stored linearly but it is not stored in a consecutive manner. Linklist contains the data in the form of nodes. Every node consists of a value and the next node but in the memory all of these nodes are placed in a random manner. Linklists are useful because its **insertion** and **deletion** operations can be used in a much efficient manner. It is also very useful in implementing the abstract data type ie: stack , queue etc. In our project, two types of linklists are used ie: singly linklist and doubly linklist. The singly linklist contains the data of buses or in other words, we can say that the data of all the buses is managed with the help of linklist. The doubly linklist contains the data of all the hospitals. Although, we can also use array for this purpose. But, its one big disadvantage is that in case of arrays, the consecutive memory locations are allocated. If our memory contains free space but it is not available in a consecutive manner, so in that case, linklist is used but arrays can't be used.

### Code

```

class LinkedList
{
    public BusNode head;
    public LinkedList()
    {
        this.head = null;
    }
    public void insert(BusNode obj)

```

```

    {
        if(head==null)
        {
            head = obj;
        }
        else
        {
            obj.next = head;
            head = obj;
        }
    }
}
public BusNode search(string numberPlate)
{
    BusNode n=this.head;
    while(n!=null)
    {
        if(n.busObject.busNumberPlate==numberPlate)
        {
            return n;
        }
        n = n.next;
    }
    return null;
}
public BusNode delete(string numberPlate)
{
    BusNode n = head;
    if(n==null)
    {
        return null;
    }
    while(n.next!=null)
    {
        if(n.next.busObject.busNumberPlate==numberPlate)
        {
            n.next = n.next.next;
            return n.next;
        }
        n = n.next;
    }
    return null;
}
}

```

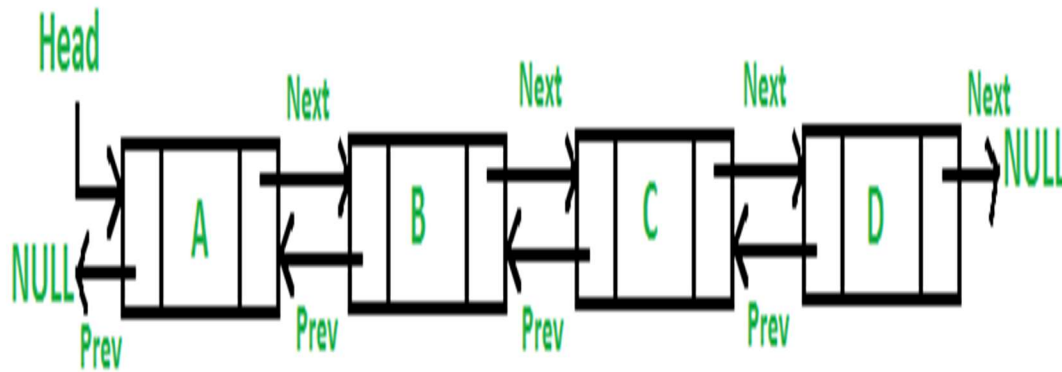


Fig 4: Doubly Linked List

### Queue

The queue is also an abstract data type which has various advantages over other data types in the data structures. Basically, the queue follows FIFO principle. In other words, we can say that queue follows the **First in First out** principle. The main operations of the queue are **enqueue** and **dequeue**. In the queue, the data is added into the queue from the one side and the data is removed from the other side. In our project, queue contains the data of all the rescue groups. After taking the attendance of all the employees and ambulances in a shift wise manner, the rescue groups are made. All the rescue groups which are in the working condition, are enqueued into the queue. As the call of the patient is received, than one of these rescue groups is dispatched and that particular group is dequeued.

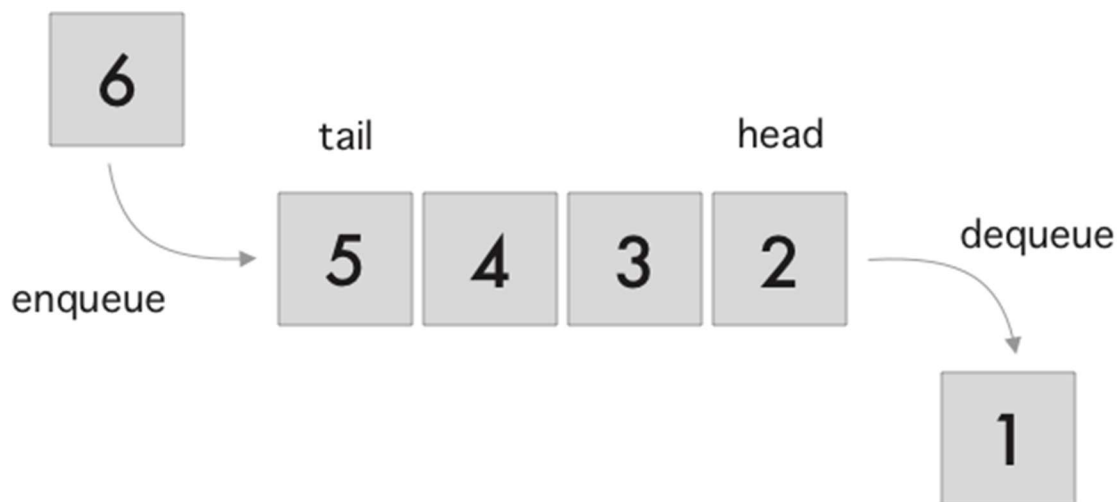


Fig 5: Queue

**Code**

```

class Queue
{
    public RescueGroup head;
    public RescueGroup tail;
    public Queue()
    {
        this.head=this.tail = null;
    }

    public void Enqueue(RescueGroup x)
    {
        if(this.tail!=null)
        {
            tail.next = x;
            tail = tail.next;
        }
        else
        {
            this.tail = x;
            this.head = x;
        }
    }
    public RescueGroup Dequeue()
    {
        if (this.head != null)
        {
            RescueGroup point = this.head;
            head = point.next;
            if(head==null)
            {
                head = tail = null;
            }
            return point;
        }
        return null;
    }
}

```

**Tree**

The tree is also an abstract data type which is very useful in storing a huge amount of data. There are various types of trees which are used in data structures. For example some of them are AVL trees, B-trees, Red-Black trees etc. In the tree, the data is stored in the form of nodes. Every node has a parent node and a child node and the node which is inserted firstly into the tree is the root of the tree. The tree data structure has also some useful operations ie: **insertion, deletion**. These



operations are applied on the data which is stored in a tree in a much efficient manner and most important operation of the tree data structure is the **traversal** which traverses the whole tree. In our project, the data of all the employees is stored in the form of tree. We have used **Red-Black trees** in this system. The main reason for using red black trees is the balancing factor. As some of the employees are inserted into the tree, so as a result, some ids are allocated to the employees. These ids are generated in a special sequence of increasing order. As we are inserting the employees into the trees on the basis of ids, so, in this case, red black tree was the best choice because its time complexity is  $O(n \log n)$  in the current scenario.

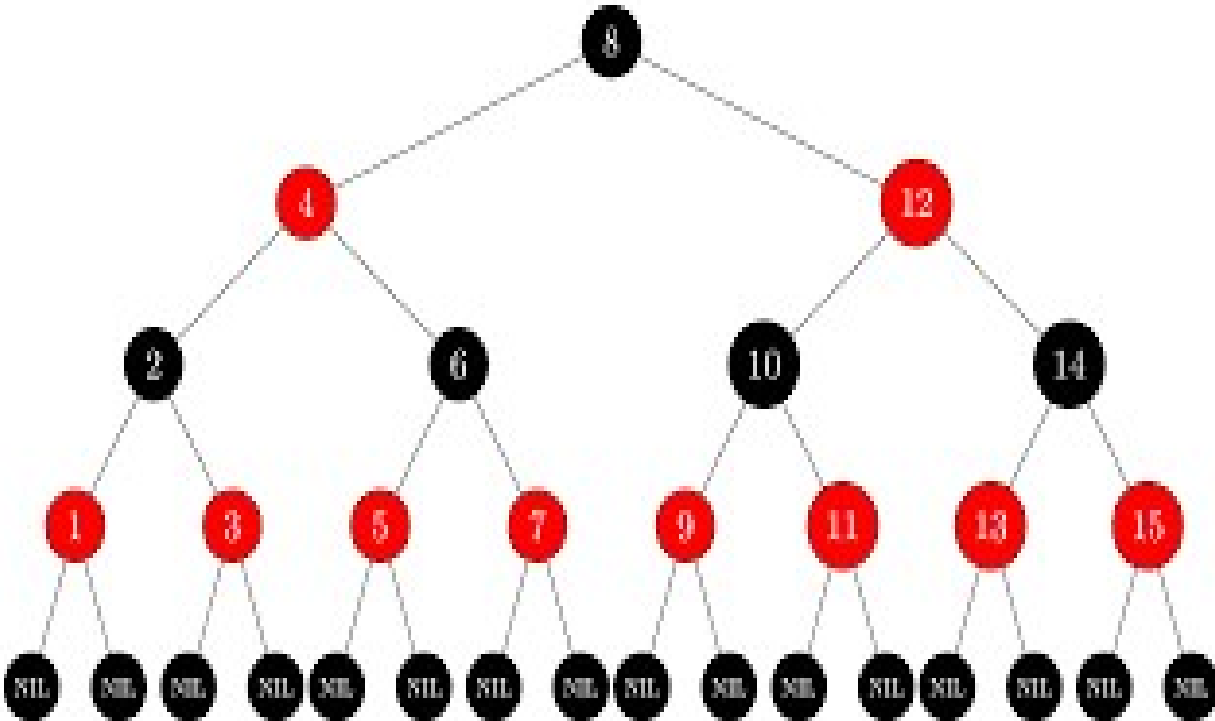


Fig 6: Tree

## Code

```

class Tree
{
    public Node head = null;
    public Tree(Node head)
    {
        this.head = head;
    }
    void LeftRotate(Node x)
    {
        Node y = x.rightNode;
        x.rightNode = y.leftNode;
        if (y.leftNode != null)
        {

```

```

        y.leftNode.parent = x;
    }
    y.parent = x.parent;
    if (x.parent == null)
        this.head = y;
    else if (x == x.parent.leftNode)
        x.parent.leftNode = y;
    else
    {
        x.parent.rightNode = y;
    }
    y.leftNode = x;
    x.parent = y;
}
void RightRotate(Node x)
{
    Node y = x.leftNode;
    x.leftNode = y.rightNode;
    if (y.rightNode != null)
    {
        y.rightNode.parent = x;
    }
    y.parent = x.parent;
    if (x.parent == null)
        this.head = y;
    else if (x == x.parent.rightNode)
        x.parent.rightNode = y;
    else
    {
        x.parent.leftNode = y;
    }
    y.rightNode = x;
    x.parent = y;
}
void InsertFixup(Node k)
{
    while (k.parent.color == "Red")
    {
        if (k.parent == k.parent.parent.rightNode)
        {
            Node u = k.parent.parent.leftNode;
            if (u.color == "Red")
            {
                u.color = "Black";
                k.parent.color = "Black";
                k.parent.parent.color = "Red";
            }
        }
    }
}

```

```

        k = k.parent.parent;
    }
    else
    {
        if (k == k.parent.leftNode)
        {
            k = k.parent;
            this.RightRotate(k);
        }
        k.parent.color = "Black";
        k.parent.parent.color = "Red";
        this.LeftRotate(k.parent.parent);
    }
}
else
{
    Node u = k.parent.parent.rightNode;
    if (u.color == "Red")
    {
        k.parent.color = "Black";
        u.color = "Black";
        k.parent.parent.color = "Red";
        k = k.parent.parent;
    }
    else
    {
        if (k == k.parent.rightNode)
            k = k.parent;
        this.LeftRotate(k);
        k.parent.color = "Black";
        k.parent.parent.color = "Red";
        this.RightRotate(k.parent.parent);
    }
}
if (k == this.head)
{
    break;
}
}
this.head.color = "Black";
}
public void Delete(Node node,int key)
{
    Node x;
    Node z=null;
    while(node!=null)

```

```

{
    if(Int32.Parse(node.employeeObject.employeeID)==key)
    {
        z=node;
    }
    if(Int32.Parse(node.employeeObject.employeeID) <= key)
    {
        node = node.rightNode;
    }
    else
    {
        node = node.leftNode;
    }
}
if(z==null)
{
    return;
}
Node y = z;
if(z.leftNode==null)
{
    x = z.rightNode;
    this.Transplant(z, z.rightNode);
}
else
{
    if (z.rightNode == null)
    {
        x = z.leftNode;
        this.Transplant(z, z.leftNode);
    }
    else
    {
        y = this.Minimum(z.rightNode);
        x = y.rightNode;
        if(y.parent==z)
        {
            x.parent = y;
        }
        else
        {
            this.Transplant(y, y.rightNode);
            y.rightNode = z.rightNode;
            y.rightNode.parent = y;
        }
        this.Transplant(z, y);
    }
}

```

```

        y.leftNode = z.leftNode;
        y.leftNode.parent = y;
        y.color = z.color;
    }
}
if(y.color=="Black")
{
    this.DeleteFix(x);
}
}

public void Transplant(Node u, Node v)
{
    if(u.parent==null)
    {
        this.head = v;
    }
    else if(u==u.parent.leftNode)
    {
        u.parent.leftNode = v;
    }
    else
    {
        u.parent.rightNode = v;
    }
    v.parent = u.parent;
}

public Node Minimum(Node u)
{
    while(u.leftNode!=null)
    {
        u = u.leftNode;
    }
    return u;
}

public void DeleteFix(Node x)
{
    Node s;
    while(x!=this.head && x.color=="Black")
    {
        if(x==x.parent.leftNode)
        {
            s = x.parent.rightNode;
            if(s.color=="Red")

```

```

        {
            s.color = "Black";
            x.parent.color = "Red";
            this.LeftRotate(x.parent);
            s = x.parent.rightNode;
        }
        if (s.leftNode.color == "Black" &&
s.rightNode.color == "Black")
        {
            s.color = "Red";
            x = x.parent;
        }
        else
        {
            if (s.rightNode.color == "Black")
            {
                s.leftNode.color = "Black";
                s.color = "Red";
                this.RightRotate(s);
                s = x.parent.rightNode;
            }
            s.color = x.parent.color;
            x.parent.color = "Black";
            s.rightNode.color = "Black";
            this.LeftRotate(x.parent);
            x = this.head;
        }
    }
    else
    {
        s = x.parent.leftNode;
        if(s.color=="Red")
        {
            s.color = "Black";
            x.parent.color = "Red";
            this.RightRotate(x.parent);
            s = x.parent.leftNode;
        }
        if(s.rightNode.color=="Black" &&
s.rightNode.color=="Black")
        {
            s.color = "Red";
            x = x.parent;
        }
        else
        {

```

```

        if(s.leftNode.color=="Black")
        {
            s.rightNode.color = "Black";
            s.color = "Red";
            this.LeftRotate(s);
            s = x.parent.leftNode;
        }
        s.color = x.parent.color;
        x.parent.color = "Black";
        s.leftNode.color = "Black";
        this.RightRotate(x.parent);
        x = this.head;
    }
}
x.color = "Black";
}
public void employeeAddition(Node z)
{
    Node y = null;
    Node x = this.head;
    while (x != null)
    {
        y = x;
        if (Int32.Parse(z.employeeObject.employeeID) <
Int32.Parse(x.employeeObject.employeeID))
        {
            x = x.leftNode;
        }
        else
        {
            x = x.rightNode;
        }
    }
    z.parent = y;
    if (y == null)
    {
        this.head = z;
    }
    else if (Int32.Parse(z.employeeObject.employeeID) <
Int32.Parse(y.employeeObject.employeeID))
    {
        y.leftNode = z;
    }
    else

```

```

        {
            y.rightNode = z;
        }
        if (z.parent == null)
        {
            z.color = "Black";
            return;
        }
        if (z.parent.parent == null)
        {
            return;
        }
        InsertFixup(z);
    }
    public Node employeeSearch(Node head, int value)
    {
        while (Int32.Parse(head.employeeObject.employeeID) !=
value)
        {
            if (value <
Int32.Parse(head.employeeObject.employeeID))
            {
                head = head.leftNode;
            }
            else
            {
                head = head.rightNode;
            }
        }
        return head;
    }
}
}

```

## Graph

The graph is also an abstract data type which is used to store the data in a non-linear form. There are many types of graphs in the data structures ie: undirected graphs, directed graphs, finite graph, infinite graphs, null graphs, simple graphs, multigraphs etc. In the graphs, the data is stored in the form of nodes. In other words, we can say that there are many vertices in the graph and each vertex has some neighbours in the tree. In the graphs, every vertex makes an edge with its neighbouring vertices so, in this way the graphs are connected. We have used undirected graphs in our project. In our project, every vertex is represented by an alphabetical letter or in other words, we can say that in our system, every station is represented by an alphabetical letter. Every station has an edge with its neighbouring stations. We have calculated the shortest path with the help of



this undirected graph in order to facilitate the rescue group. With the help of this shortest path, the rescue group can rescue the patient as soon as possible. In order to calculate the shortest path, we have used **Dijkstra's Algorithm** in our system.

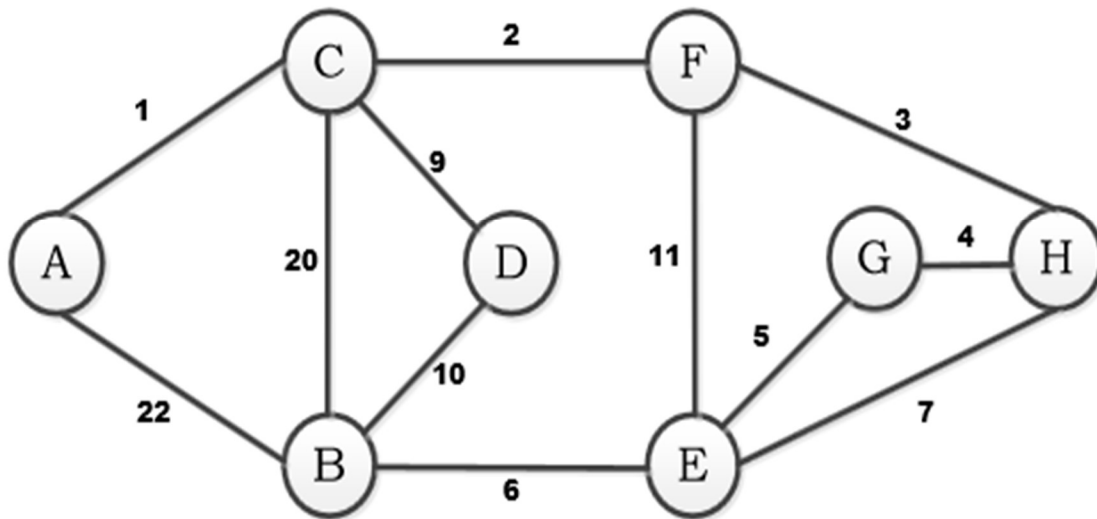


Fig 7: Graph

Code

```

class Graph
{
    public List<List<Vertex>> adjList;

    public Graph()
    {
        adjList = new List<List<Vertex>>();
        for(int i=0;i<=25;i++)
        {
            adjList.Add(new List<Vertex>());
        }
    }

    public void initializeVertex(string start)
    {
        foreach(List<Vertex> list in adjList)
        {
            if(list.Count>0)
            {
                Vertex v = list.First();
                if(v.name==start)
                {

```

```

        v.distance = 0;
        v.parent = null;
    }
    else
    {
        v.distance = Int32.MaxValue;
        v.parent = null;
    }
}
}

}

public void relax(Vertex u, Vertex v, int weight)
{
    if(v.distance > u.distance + weight)
    {
        v.distance = u.distance + weight;
        v.parent = u;
    }
}

public void Dijkstra(string start)
{
    initializeVertex(start);
    List<Vertex> queue = new List<Vertex>();
    List<Vertex> set = new List<Vertex>();
    foreach(List<Vertex> list in adjList)
    {
        if(list.Count > 0)
        {
            Vertex v = list.First();
            queue.Add(v);
        }
    }
    while(queue.Count > 0)
    {
        Vertex vertex = queue[0];
        foreach(Vertex v in queue)
        {
            if(v.distance < vertex.distance)
            {
                vertex = v;
            }
        }
        queue.Remove(vertex);
    }
}

```

```

        set.Add(vertex);
        int idx = (vertex.name[0]) % 97;
        foreach(Vertex v in adjList[idx])
        {
            int index = (v.name[0]) % 97;
            Vertex u = adjList[index][0];
            relax(vertex, u, u.weight);
        }
    }

}

public void AddEdge(string src, string des, int edgeWeight)
{
    int id1 = src[0] % 'a';
    int id2 = des[0] % 'a';
    bool sourceFound = false;
    bool destinationFound = false;
    foreach (List<Vertex> list in adjList)
    {
        if (list.Count > 0)
        {
            Vertex v = list.First();
            if (v.name == src)
            {
                sourceFound = true;
                Vertex temp = new Vertex(des);
                temp.weight = edgeWeight;
                list.Add(temp);
            }
            if (v.name == des)
            {
                destinationFound = true;
                Vertex temp = new Vertex(src);
                temp.weight = edgeWeight;
                list.Add(temp);
            }
        }
    }
    if(sourceFound=false)
    {
        Vertex sv = new Vertex(src);
        sv.weight = edgeWeight;
        Vertex ds = new Vertex(des);
        ds.weight = edgeWeight;
        adjList[id1].Add(sv);
        adjList[id1].Add(ds);
    }
}

```

```

    }
    if (destinationFound = false)
    {
        Vertex sv = new Vertex(src);
        sv.weight = edgeWeight;
        Vertex ds = new Vertex(des);
        ds.weight = edgeWeight;
        adjList[id2].Add(sv);
        adjList[id2].Add(ds);
    }
}

```

## Object Oriented Model

### Employee Class

This class contains the data related to the two types of employees ie: the driver and the compoder. This class has many relations with other classes. It has inheritance relation with Driver class and compoder class. It also has an aggregation relation with Bus class and composition relation with Rescue Group class. There is a separate Employee CRUD class which contains, all the CRUD operations.

### Bus Class

Basically, this class contains all the data related to the bus. It has an aggregation relationship with the employee class. This class also has a composition relation with the Rescue Group class. There is a separate Bus CRUD class which contains all the CRUD operations.

### Rescue Group Class

Basically, this class contains the objects of other classes. It contains an employee and an ambulance as an attribute. It has a composition relation with both Employee class and Bus class. This class contains several groups of employees and ambulances. Whenever some emergency call is received by the CTWO, the WO assigns some of the duties to one of these groups. This group rescues the patients and the patients are dispatch to the hospitals.

### Hospital Class

This class contains all the attributes of the hospitals. It has a separate Hospital CRUD class which contains all the CRUD operations ie: add, remove, search, update. These operations will be implemented on all the objects of the hospitals.

### Rescue Case Class

This class contains an object of ECF class, Bus class and Hospital class as an attribute. It also has an attribute of case ID. This ID contains the case number which was rescued by the rescue group.

### **ECF Class**

This class contains the information of the patient. For example, it contains all the personal information of the patient. It also contains the attribute of the pickup address from where the patients will be received and also destination address where the patients will be delivered.

### **Tree and Node Class:**

In our project, node class contains an employee object. This class also contains parent node, child node and the color attribute of the node. As we have implemented the red black tree in our project, so that's why we have there is a color attribute in node class. Also, there is a tree class in our project. This tree class contains the root of the tree. Also this class contains some methods which are the operations of the tree. This class contains an insertion method which inserts an employee into the red black tree. Similarly, there are also some other methods in the tree class ie: searching and deletion. Searching method searches the employees from the tree and the deletion method deletes the employees from the tree.

### **Queue and queue node class:**

Queue node class contains an object of rescue group class and it also contains an object of its own class. The object of its own class point to the next node of the queue. On the other hand, Queue class contains head of the queue and it also contains the tail of the queue. This class contains the two basic operations of the queue ie: enqueue and dequeue. The enqueue operation of the queue adds the rescue group object into the queue. On the other hand, dequeue operation of the queue removes a rescue group object from the queue.

### **Stack and stack Node class:**

The stack node class contains an object of hospital class and the objects of this stack node will be added into the stack. On the other hand, stack class contains the top of the stack. The top of the stack is also an object of stack node class. The stack class also contains the two main operation of the stack ie: push and pop. If number of available beds of the any hospital is greater than zero than that particular hospital is pushed into the stack. On the other hand, if number of available beds of any hospital is zero than that particular hospital is popped from the stack.

### **Linklist and its node class:**

Node class of linklist contains an object of hospital class and it also contains an object of its own class. This node points to the next node of the linklist. Also, there is a linklist class which contains an head of the linklist. The head of the linklist is also an object of the Linklist node class. This class contains the two major methods ie: insertion, searching. The insertion operation adds the hospital into the linklist. Similarly, the searching operation searches the hospital from the linklist. Our linklist class does not contains a deletion operation because we don't want to delete a hospital from the linllist.

## UML-Diagram:

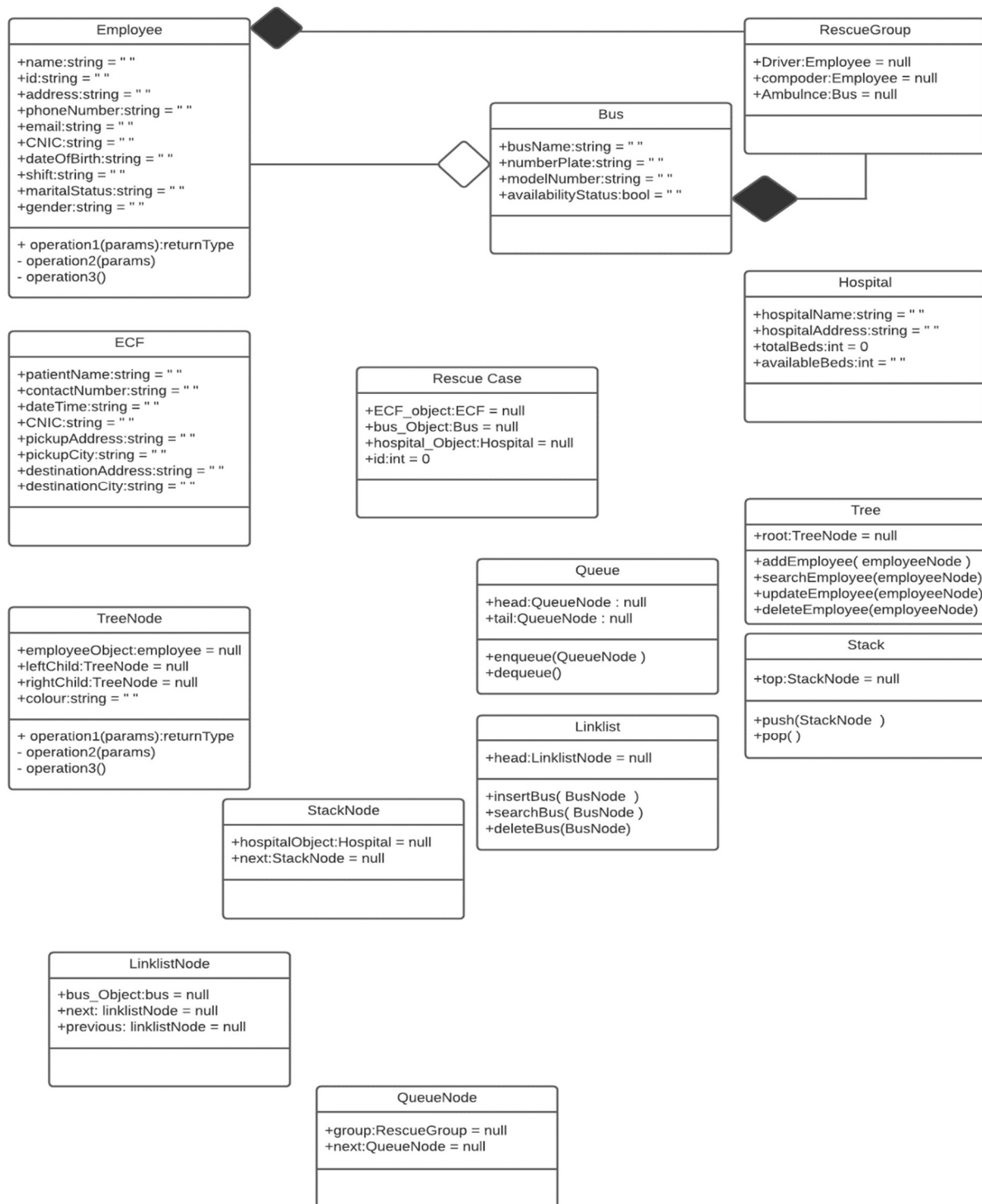


Fig 8: UML Diagram

## Wireframes

### Employee Information Window:

This window contains all the information of the employee. This form is useful for only a new employee. As the new employee is hired for this job, he has to provide all his personal information which is mentioned in the window. As the **Add Employee** button is pressed, then the data of the new employee is added in the tree.



### Employee Information

Employee Name	<input type="text"/>		
Employee ID	<input type="text" value="9"/>		
Employee Address	<input type="text"/>		
Phone Number	<input type="text"/>	Gender	<input type="radio"/> male <input type="radio"/> female
Email	<input type="text"/>		
CNIC	<input type="text"/>		
Date of birth	<input type="text"/>	Marital Status	<input type="text"/>
Job Type	<input type="text" value="v"/>	Shift	<input type="text" value="v"/>

Add Employee

## Update Employee Window

This window also contains the search panel at the top of the page. The admin will search the Employee which he wants to update. As the Employee is selected, then the rest of the information is automatically filled. Then, the admin will change the information which he wants to change. As the **Update Employee** button is pressed, then as a result the data of the employees in the tree is updated.



### Update Employee Information

Search Employee

*search the employee by id here*

search

Employee Name

Employee ID

Employee Address

Phone Number

Gender ☐ male ☐ female

Email

CNIC

Date of birth

Marital Status

Job Type


Shift

Update Employee



## Delete Employee Window

This window also contains a search panel. The admin searches the employees which he wants to delete. When he selects the employee, the rest of all the information regarding the employee will be automatically filled. The employees will be deleted from the tree as the **Delete employee button** is pressed.



Delete Employee

Search Employee

Employee Name

Employee ID

Employee Address

Phone Number  Gender ☐ male ☐ female

Email

CNIC

Date of birth  Marital Status

Job Type  Shift

## Hospital Registration Window

Whenever some new hospital will be built in some area, the admin will fill in all the information regarding that particular hospital which is mentioned in this window. As the **Register** button is pressed after filling all the information than, a new hospital will be added in their record.



## Hospital Registration

Name

Address

Total Bed

Available Bed

ID

Password

REGISTER

## Update Hospital Window

This window contains the search panel at the top of the page. The admin will search the hospital which he wants to update. As the hospital is selected, then the rest of the information is automatically filled. Then, the admin will change the information which he wants to change. As the **Update Hospital Data** button is pressed, then as a result the data of the hospital is updated.



## Update Hospital

Search

Name

Address

Total Bed

Available Bed

ID

Password

UPDATE

## Remove Hospital Window

In this window, there is a search panel. The admin will search the hospital from this panel which he wants to delete. As the hospital is selected, then all the next information will be autofilled. As the **Remove hospital** button is pressed, then the hospital is popped from the stack.



## Delete Hospital

Search

Name

Address

Total Bed

Available Bed

ID

Password

DELETE

## Add Bus Window

Whenever a new bus is available for this service, the admin of the bus fills all the information related to the bus which is mentioned in the above window. after filling this form, as the **Add Bus** button is pressed, then the object of this bus will be added in the queue.



## Add Bus

Bus Name

Bus Colour

Model Number

Number Plate

Add Bus

## Update Bus Window

This window also contains the search panel at the top of the page. The admin will search the bus which he wants to update. As the bus is selected, then the rest of the information is automatically filled. Then, the admin will change the information which he wants to change. As the **Update Bus** button is pressed, then as a result, the object of the bus is dequeued.



## Update Bus

Search Bus

Bus Name


Bus Colour

Model Number

Number Plate

## Delete Bus Window

This window also contains a search panel. The admin of all the buses searches for the bus which he wants to delete. When he selects the bus, the rest of all the information regarding the bus will be autofilled. The buses will be deleted from the queue as the **Delete bus** is pressed.



*Delete Bus*

Search Bus

Bus Name


Bus Colour

Model Number

Number Plate

## Attendance Sheet

This window contains all the information related to employees and buses. This window also contains two radio buttons at the end of each row. These radio buttons are used for marking the absence or presence of the employees or buses. In this way, attendance of the buses and employees can be taken.



### Attendance Sheet of Employees

	Employee Name	Employee ID	Employee Attendance	Bus Number	Bus Attendance
▶	b	12	<input type="checkbox"/>	c	<input type="checkbox"/>
	b	13	<input type="checkbox"/>	c	<input type="checkbox"/>
	b	14	<input type="checkbox"/>	c	<input type="checkbox"/>
	b	15	<input type="checkbox"/>	c	<input type="checkbox"/>
★			<input type="checkbox"/>		<input type="checkbox"/>



## CTWO Window

This is the main window of the project using this window CTWO will be able to perform the dictionary operation of the employee, bus, hospital and he can receive the call from the patient and suggest the rescue group to the case.



## ECF Window

This form is filled by the CTWO. As the information related to the patient is received, the CTWO will fill this form. This form contains all the information related to the patient. It also contains the information from where the patient is received and at which place the patient will be delivered. As the **Submit Form** button is pressed, the information of this form is stored in the record.



The image shows a software window titled "ECF" with a standard Windows-style title bar (minimize, maximize, close buttons). Inside the window, there is a logo on the top left and the text "ECF FORM" in red at the top center. Below the title, there are several input fields and labels:

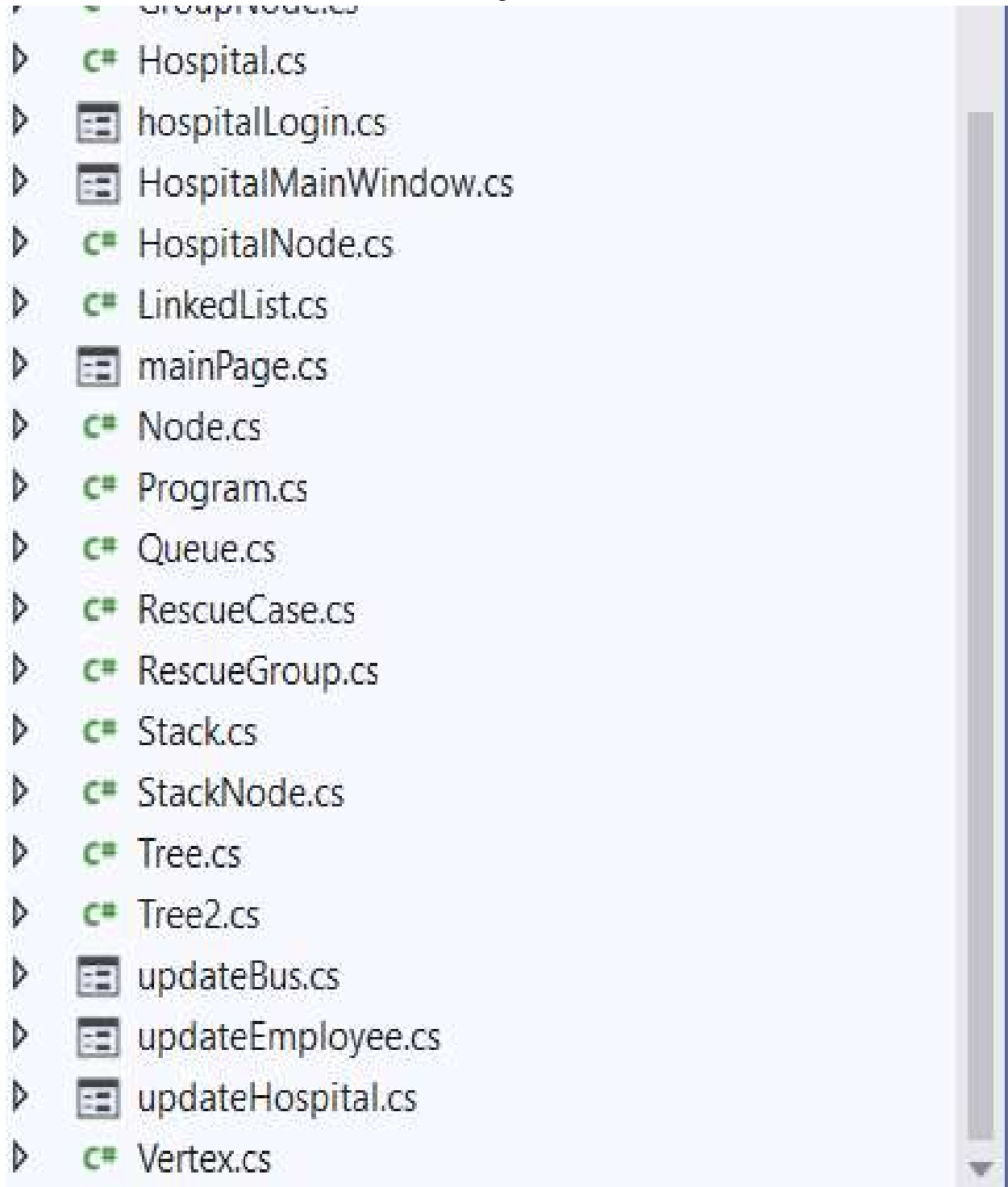
- ID:** label8
- Name:**
- Father Name:**
- Contact Number:**
- Date and Time of Request:**
- CNIC:**
- Pick Up Address:**
- Destination Address:**
- Driver:**
- Compoder:**
- Bus:**

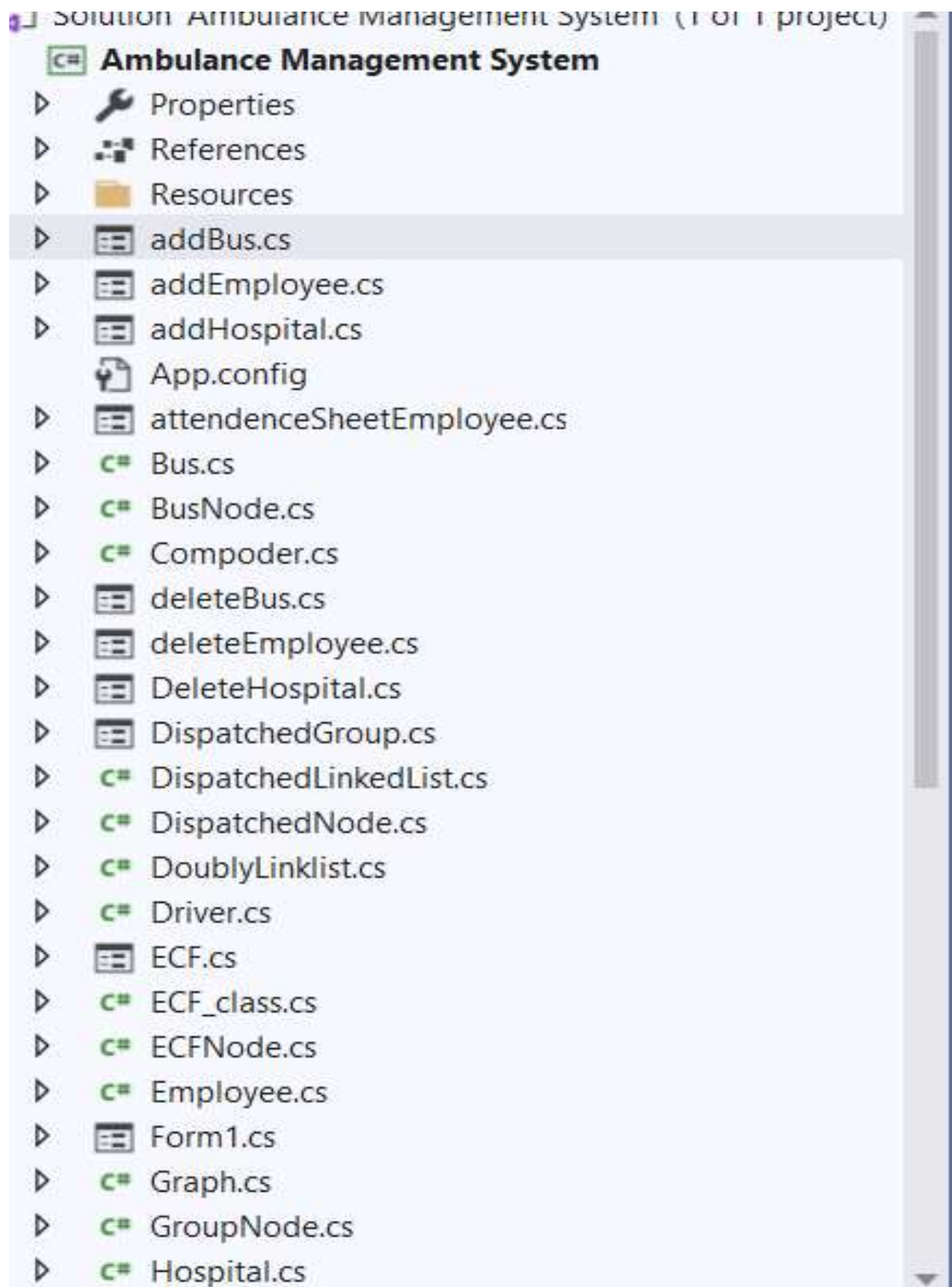
At the bottom center of the window is a green button labeled "SUBMIT".

## File Management

Here are the list of all the files that we used in our project. These screen shots from the IDE contains all the files i.e :

Files of all the classes that we used and the design files.





## Data Management

There were different ways to use the stored data but we chose text files to make comma separated data. We made comma separated text files and read the data of individual feature and later on showed the data to the user.

File Name	Number of entries
Employee Data	100
Bus Data	50
Hospital Data	20
ECF Data	10

Table4: Data Management

## Tasks Division According To Features

Name	Tasks Completed
Umair Ahmed	<ol style="list-style-type: none"> <li>1. Dictionary Operations Of Employee (RB Tree)</li> <li>2. Attendance Management System Of Employee (Queue)</li> </ol>
Kashir Saeed	Dictionary Operations Of Hospital
Muhammad Farrukh Haider	Shortest Path (Graph and Dijkstra)

Table4: List of Features

## Testing

We tested every feature which was added in the project by dry run and by debugging it on the Visual Studio. There were many testing problems incurred during the process. Sometimes it was syntax errors and majority of the time there were logical errors. Each group member tested using his approach so that when large amount of data was used, the system doesn't lag or stop irrespective of the reason which made it stop. Due to frequent testing, we were able to reach to the solution of certain problem which shows that testing is beneficial tool for checking the error.

## Collaboration

Before starting the project, we three members divided the workload. One of our group mates couldn't collaborate with other members due to a family function. Nevertheless, the other members distributed the workload equally and were able to meet deadlines. After a period of 4 days, all the three members were able to regroup and work together at the hostel of Umair Ahmed (Muhammad Bin Qasim room no 102). One of us was a day scholar and used to leave for his house at around 2 am midnight, thus depicting the dedication for the project. Umair Ahmed worked on the Red Black Tree and Stack. Queue and doubly linked list was implemented by Kashir Seed. Graph and a bit of Dijkstra was implemented by Farrukh Haider. Moreover, the three of us used to discuss every query irrespective of what was assigned and even resolved the problems in the code. The GUI and the report was done by each member thus managing the workload.

## Integration:

The integration of our project consists of two parts ie: the GUI of the project and the backend of the code. First of all, the group members had made on the GUI of the project. We did not use this approach of implementing all the data structures separately and then integrating it with the with the GUI. Instead of using this approach, we implemented the desired code which was relevant to the project on the respective click event. All the six data structures which we have used in our project are implemented in the same way. In this way, we have integrated our project.

## Limitations

Despite all our efforts, there were a couple of drawbacks of the project which can be resolved in the future.

- The dijkstra algorithm was implemented on self-based locations. The algorithm can be modified to the original locations which are present in different cities across the country.
- Currently we are not able to receive the call of the patient directly. We are using a button in the project which is depicting a call.

## Future Goals

In the coming years, this system can be modified significantly. We can use telecommunication systems which will enable effective communication between the members. Moreover, we can add bike drivers which will further help in assisting the patients in the crowded areas. We can add multiple algorithms in finding the shortest path which will enhance the overall efficiency of the system.

## Conclusion

We as group mates are satisfied with the working of the program. Every functionality is working properly as per the given requirements. Further we tried our best to guide each other and transfer the knowledge across the team. Through this we have learn a lot of things and one the best thing that we learn is the use of the gitlab.

## References

- Pseudo Codes from the book **Introduction to Algorithms**
- Basic Infrastructure of the project from the research paper
- Report Headings from the PDF **how-to-write-a-structured-project-report**

