Data Structures and Algorithms Project Report

Project Title- Covid-19 Vaccination System

Team Members	Enrolment No.	Batch
1) Ankur Sinha	19102107	A4
2) Gagan Gupta	19102097	A4
3) Raghav Khandelwal	19102101	A4
4) Yash Mathur	19102104	A4

INTRODUCTION

The project “Covid-19 Vaccination System” is a model made with the help of Data Structure and Algorithms. The project aims to ease the vaccination drive for common people. In this system, one can book a vaccination slot easily by uploading his/her details. The user will be prompted to provide certain details regarding his/her identity and regarding the phase of vaccine he/she is applying for once the user is registered successfully, the system will display all the information regarding the location of the vaccination centers present nearby user and the user can book his/her slot for the vaccine depending on the availability of the vaccine on the particular center which will also be displayed to the user at the time of booking. Once done with the selection of the vaccination center process the user will be registered for an automatic first come first serve vaccination process, the details of the same can be viewed in the waiting list. If the user enters any incorrect information, then he can cancel the appointment process and the very next person on the waiting list will be allotted the slot.

BACKGROUND

Features:

1. The covid vaccination project provides the user an easy way to book slots for their vaccination.

2. It gives them choice for selecting their desired vaccination centers, which becomes very convenient for the users.

3. All the fields are validated and do not take any invalid input.

4. Keeping the ease of use in mind the interface is kept simple and elegant.

5. Efficient and appropriate algorithms used in the project helps in speeding up the process without any hassles.

Novelty:

By accessing the COVID 19 vaccination system, you can easily book your COVID 19 vaccines slots. Also, you can do that by staying inside the safety of your homes and very easily online. It becomes easier to choose your nearest vaccination center for your vaccines but if you're unable to get your vaccine slot at that center, the system also suggests you the nearest center for your vaccines. This makes it efficient for the citizens to have their vaccines on time without any hassle.

APPLICATIONS

This Project uses the concept of the queue, vectors, Dijkshtra's Algorithm and basic concept of queue for good ease for slot booking and also it suggests any other vaccination centers nearby if you don't want the first one.

It helps us with a reliable system for easily and efficiently booking for a vaccination by the safety of your homes.

REQUIREMENT SPECIFICATION: HARDWARE /SOFTWARE (DATA STRUCTURES, ALGORITHMS)

Vectors:

Vectors are sequence containers which are the same as dynamic arrays with the ability to resize itself automatically when an element is inserted or deleted, with their storage being handled automatically by the container. The elements in the vector are placed in contiguous storage so that they can be accessed and traversed with the help of iterators. Vectors are very efficient accessing its elements (just like arrays) and relatively efficient adding or removing elements from its end.

Queue:

Queue is also an abstract data type or a linear data structure, just like stack data structure, in which the first element is inserted from one end called the REAR(also called tail), and the removal of the existing element takes place from the other end called as FRONT(also called head).

This makes the queue a FIFO(First in First Out) data structure, which means that element inserted first will be removed first.

Real-Life Explanation of Queue:
If you go to a ticket counter to buy movie tickets, and are first in the queue, then you will be the first one to get the tickets. Right? Same is the case with Queue data structure. Data inserted first, will leave the queue first.

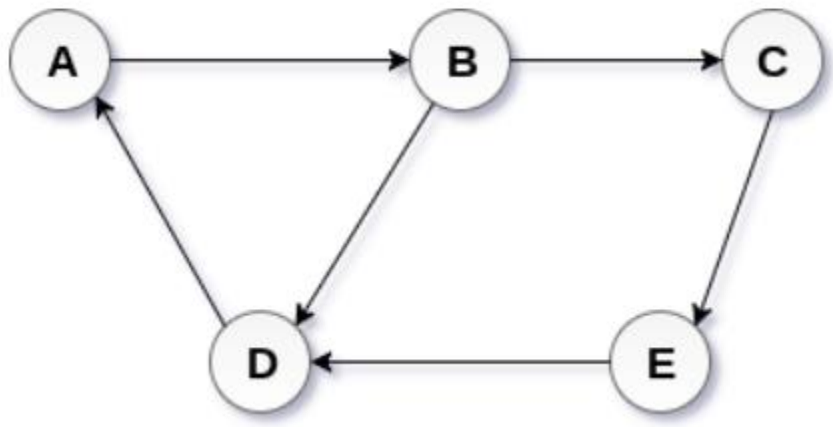
Queue Operations:
The process to add an element into the queue is called Enqueue and the process of removal of an element from the queue is called Dequeue.

Graphs:
A graph can be defined as a group of vertices and edges that are used to connect these vertices. A graph can be seen as a cyclic tree, where the vertices (Nodes) maintain any complex relationship among them instead of having parent-child relationship.
In other words a graph G can be defined as an ordered set G(V, E) where V(G) represents the set of vertices and E(G) represents the set of edges which are used to connect these vertices.

Types of Graphs:

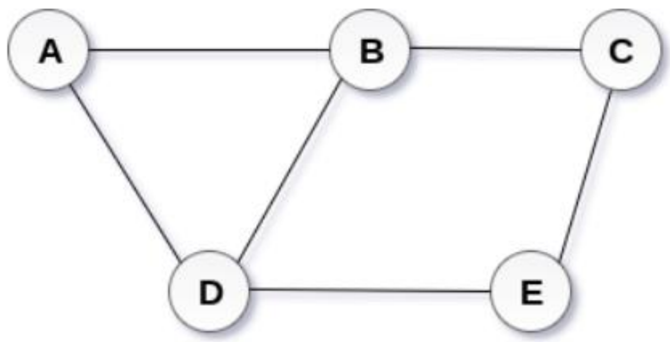
Classification based on edges-

1) Directed Graph- In a directed graph, edges form an ordered pair. Edges represent a specific path from some vertex A to another vertex B. Node A is called the initial node while node B is called the terminal node.



Directed Graph

2) Undirected Graph- In an undirected graph, edges are not associated with the directions with them. An undirected graph is shown in the above figure since its edges are not attached with any of the directions. If an edge exists between vertex A and B then the vertices can be traversed from B to A as well as A to B.

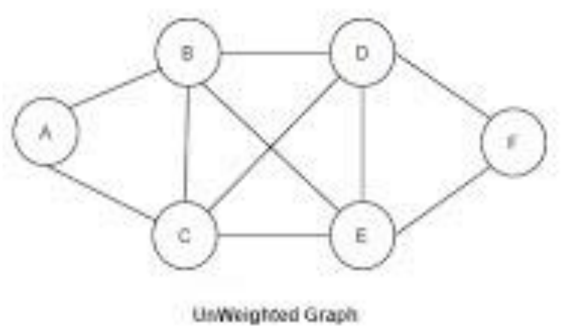


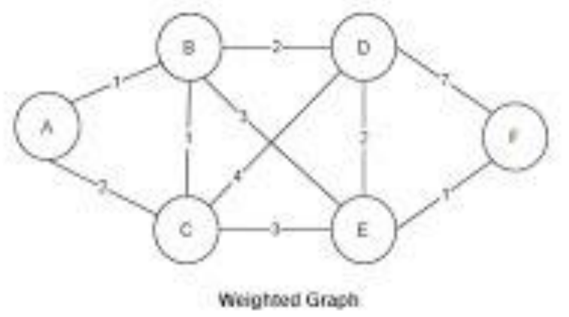
Undirected Graph

Classification based on weights-

1) Labeled Graph- Each of the edges is assigned a weight. (also known as weighted graph)

2) Unlabeled Graph- Each of the edges is not labelled with weight. (also known as unweighted graph)



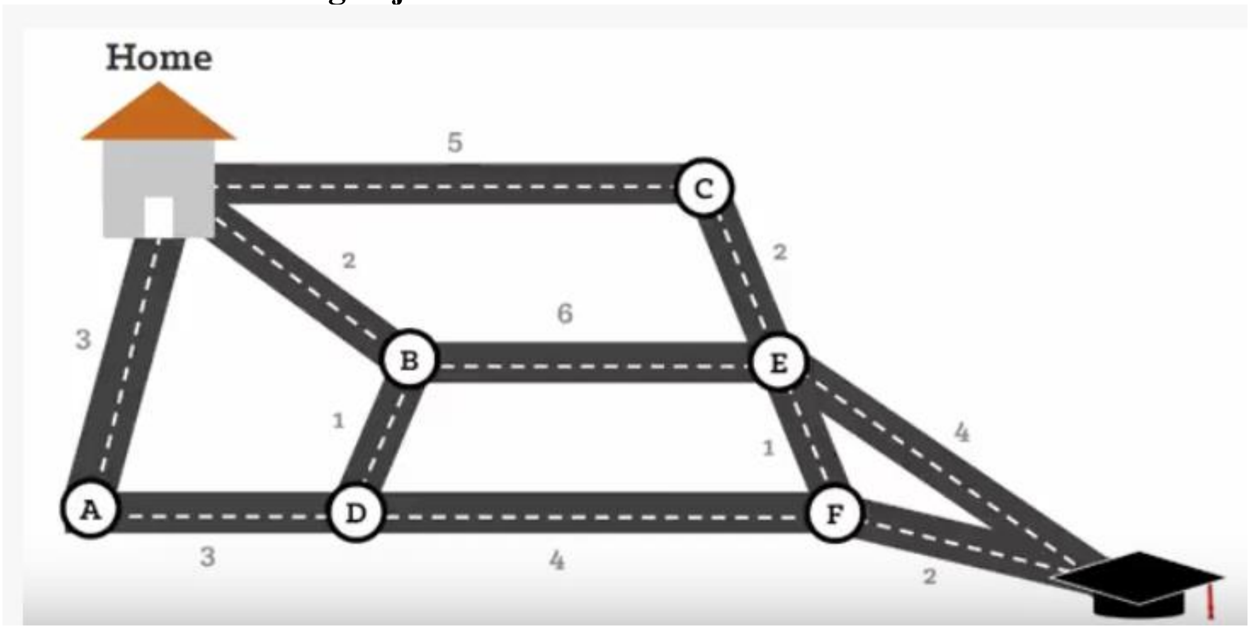


ALGORITHMS USED:

Dijkstra's Algorithm- Dijkstra's algorithm was conceived by computer scientist Edsger W. Dijkstra in 1956 and published three years later. It is used for finding the shortest paths between nodes in a graph, which may represent, for example, road networks. The algorithm creates a tree of shortest paths from the starting vertex, the source, to all other points in the graph.

The graph can either be directed or undirected. One stipulation to using the algorithm is that the graph needs to have a nonnegative weight on every edge.

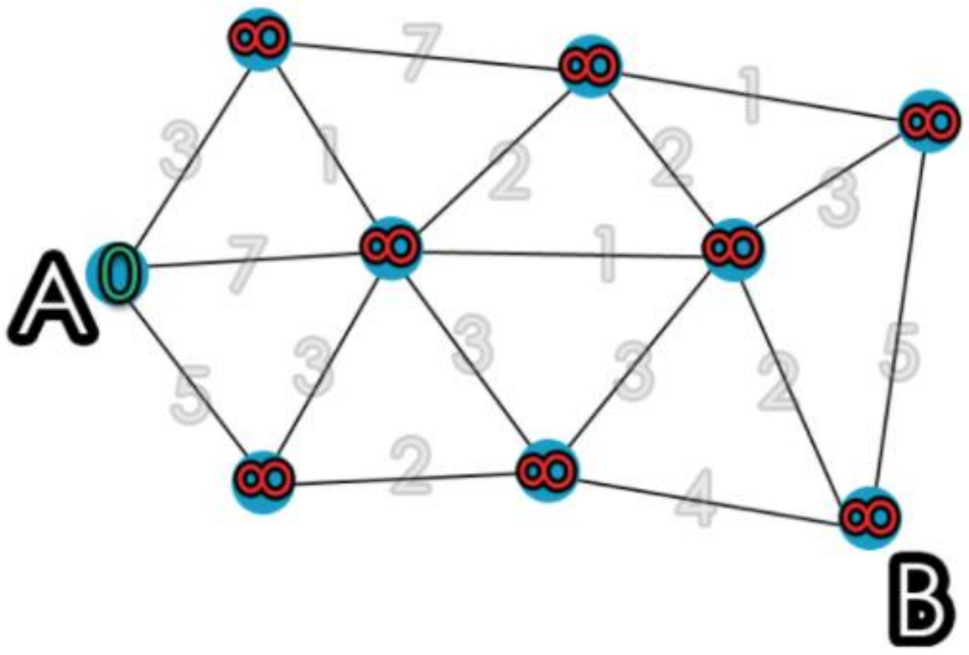
Walk-to-School using Dijkstra's



Weighted graph representing roads from school

Suppose a student wants to go from home to school in the shortest possible way. She knows some roads are heavily congested and difficult to use. In Dijkstra's algorithm, this means the edge has a large weight- the shortest path tree found by the algorithm will try to avoid edges with larger weights. If the student looks up directions using a map service, it is likely they may use Dijkstra's algorithm, as well as others.

A closer look at Dijkstra's



Dijkstra's algorithm finds a shortest path tree from a single source node, by building a set of nodes that have minimum distance from the source.

The graph has the following:

i) vertices or nodes, denoted in the algorithm by v or u.
ii) weighted edges that connect two nodes: (u,v) denotes an edge, w(u,v) denotes the weighted edge.

This is done by initializing three values:
=> dist, an array of distances from the source node s to each node in the graph, initialized the following way: dist(s) = 0; and for all other nodes v, dist(v) = ∞. This is done at the beginning because as the algorithm proceeds, the dist from the source to each node v in the graph will be recalculated and finalized when the shortest distance to v is found.
=> *Q*, a queue of all nodes in the graph. At the end of the algorithm's progress, Q will be empty.

=> *S*, an empty set, to indicate which nodes the algorithm has visited. At the end of the algorithm's run, *S* will contain all the nodes of the graph.

The algorithm proceeds as follows:

1) While Q is not empty, pop the node v, that is not already in S, from Q with the smallest dist (v). In the first run, source node s will be chosen because dist(s) was initialized to 0. In the next run, the next node with the smallest dist value is chosen.

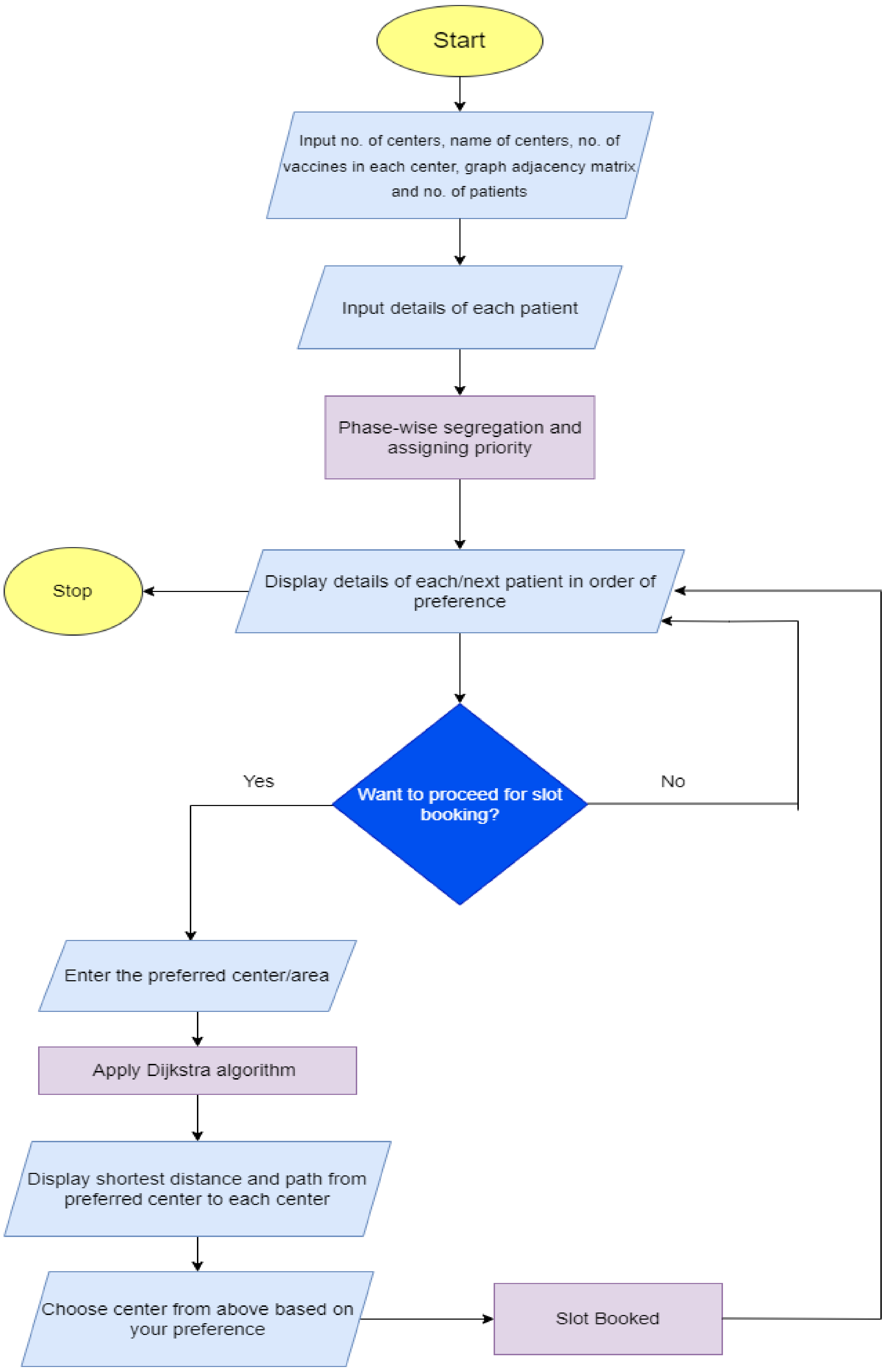
2) Add node v to S, to indicate that v has been visited.

3) Update dist values of adjacent nodes of the current node v as follows: for each new adjacent node u,

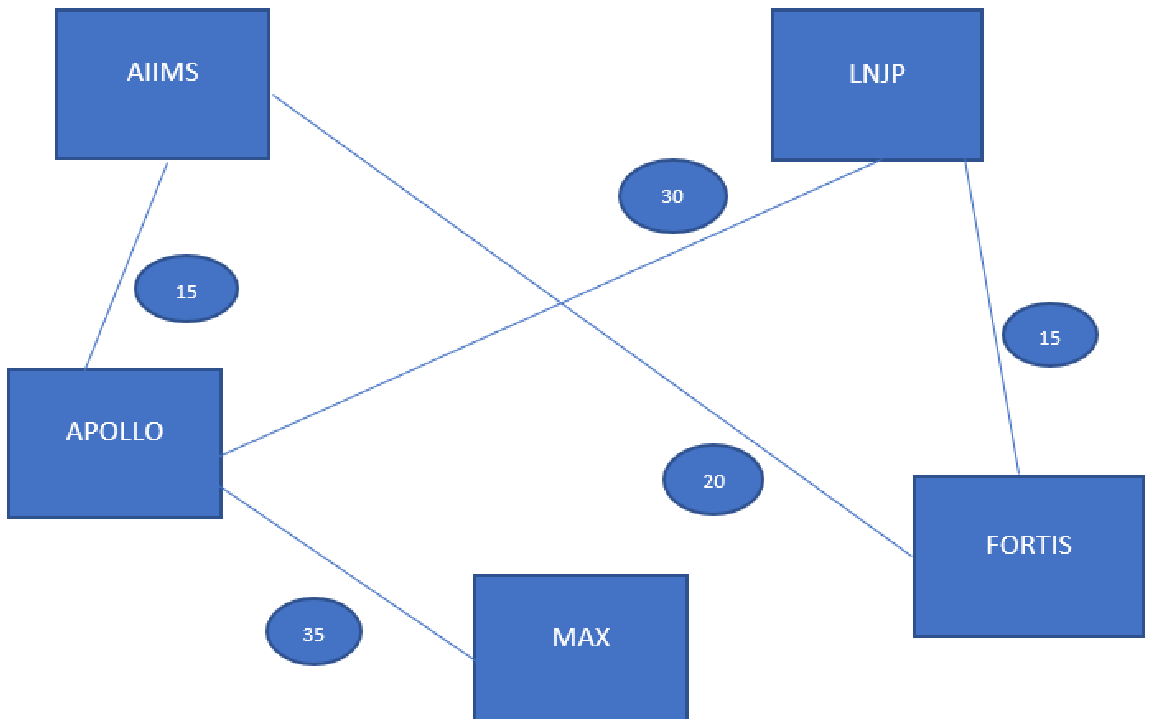
a. if dist (v) + weight(u,v) < dist (u), there is a new minimal distance found for u, so update dist (u) to the new minimal distance value;
b. otherwise, no updates are made to dist (u).

The algorithm has visited all nodes in the graph and found the smallest distance to each node. dist now contains the shortest path tree from source s.

FLOWCHART


Start
Input no. of centers, name of centers, no. of vaccines in each center, graph adjacency matrix and no. of patients
Input details of each patient
Phase-wise segregation and assigning priority
Display details of each/next patient in order of preference
Stop
Want to proceed for slot booking?
Yes
No
Enter the preferred center/area
Apply Dijkstra algorithm
Display shortest distance and path from preferred center to each center
Choose center from above based on your preference
Slot Booked

INPUT GRAPH



IMPLEMENTATION: CODE

```
#include <iostream>

#include<queue>

#include<vector>

#include<stack>

#include<string>

#include<cmath>

#include<map>

#include<bits/stdc++.h>
```

```
using namespace std;


class Patient{

public:

        int patient_id;

        string p_name;

        int p_contact;

        int P_condition;


        Patient(){

                cout<<"ENTER THE DETAILS OF THE
PATITENT"<<endl;

                cout<<"            ENTER Patient ID: ";

                cin>>patient_id;

                cout<<"\n          Enter Patient name: ";

                cin>>p_name;

                cout<<"\n          Enter Patient Contact number: ";

                cin>>p_contact;

                cout<<"NOTE:-";

                cout<<"Phase 1: For patients aged between 18 to 45"<<endl;

       cout<<"Phase 2: For patients aged between 45 to 60"<<endl;

       cout<<"Phase 3: For patients aged above 60"<<endl;

                cout<<"\n          Enter Patient Phase(1,2,3): ";

                cin>>P_condition;
```

```
}

    int getPatientCondition()
    {
            return P_condition;
    }

    int getPatientID(){
            return patient_id;
    }

    int getPatientContact(){
            return p_contact;
    }

    string getPatientName(){
            return p_name;
    }
    void getPatientDetails()
    {
            cout<<"\n\tPatient ID: "<<patient_id;
            cout<<"\n\tPatient Name: "<<p_name;
            cout<<"\n\tPatient Contact Number: "<<p_contact;
```

```
        }

};

#define MAX 10
#define INF 100000;

void dijk(int G[MAX][MAX], int n, int start, vector<string> centers,
vector<int> Vaccine)
{
        int cost[MAX][MAX], dist[MAX], visited[MAX], pred[MAX];
        int i, j, count, mindist, nextnode;

        for (i = 0; i < n; i++)
        {
                for (j = 0; j < n; j++)
                {
                        if (G[i][j] == 0)
                        {
                                cost[i][j] = INF;
                        }
                        else
                        {
                                cost[i][j] = G[i][j];
                }
        }
}
for (i = 0; i < n; i++)
{
        dist[i] = cost[start][i];
        pred[i] = start;
        visited[i] = 0;
}
dist[start] = 0;
visited[start] = 1;
count = 1;

while (count < n - 1)
{
        mindist = INF;
        for (i = 0; i < n; i++)
        {
                if (dist[i] < mindist && !visited[i])
                {
                        mindist = dist[i];
                        nextnode = i;
                }
        }
        visited[nextnode] = 1;

        for (i = 0; i < n; i++)
        {
            if (!visited[i])
            {
                if ((mindist + cost[nextnode][i]) < dist[i])
                {
                    dist[i] = mindist + cost[nextnode][i];
                    pred[i] = nextnode;
                }
            }
        }
        count++;
    }

    for (i = 0; i < n; i++)
    {
        int sum = 0;
        if (i != start)
        {
            cout << "\n        ----OUTPUT----\n";
```

```
                        cout << "The shortest distance between " << 
centers[start] << " and " << centers[i] << " is " << dist[i] << endl;

                        cout << "The path is " << centers[i] << "-(" << 
Vaccine[i] << " Vaccines) ";

                        sum = sum + Vaccine[i];

                        j = i;

                        do

                        {

                                j = pred[j];

                                cout << "<-" << centers[j] << "-(" << Vaccine[j] 
<< " Vaccines) ";

                                sum = sum + Vaccine[j];

                        } while (j != start);

                }

                cout << "\n";

        }

}


int getIndex(vector<string> centers, int n)

{

        string x;

        int index;

        cout << "\n\tEnter the preferred center/area-->  ";

        cin >> x;
        for (int i = 0; i != n; i++)
        {
                if (centers[i] == x)
                {
                        index = i;
                        break;
                }
        }
        return index;
}

void getGraph(int G[MAX][MAX], vector<string> centers, int n)
{
        for (int i = 0; i < n; i++)
        {

                for (int j = i; j < n; j++)
                {
                        if (i != j)
                        {
                                int X;
                                cout << "Weight/Distance of " << centers[i] << "
-> " << centers[j] << endl;
                                cin >> X;
}

                    G[i][j] = G[j][i] = X;
                }
                else
                {
                    G[i][j] = 0;
                }
            }
        }
}


void showq(queue<int> gq)
{
    queue<int> g = gq;
    while (!g.empty()) {
        cout << '\t' << g.front();
        g.pop();
    }
    cout << '\n';
}

void makePatientQueue(queue<int> q1,queue<int> q2,queue<int>
q3,queue<int> patientQueue)
{
        queue<int> g1 = q1;
        queue<int> g2 = q2;
        queue<int> g3 = q3;

        while (!g1.empty()) {
                patientQueue.push(g1.front());
       g1.pop();
    }
        while (!g2.empty()) {
                patientQueue.push(g2.front());
       g2.pop();
    }
        while (!g3.empty()) {
                patientQueue.push(g3.front());
       g3.pop();
    }
}

int main()
{
        system("CLS");
        int G[MAX][MAX], n, start;
        int i, j;
```

```
}



cout << "\n";

cout << "Enter the number of centers:" << endl;

cin >> n;

cout << "\n";


cout << "Enter the centers:" << endl;

vector<string> center(10);

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

{

        cin >> center[i];

}

cout << "\n";

cout << "Enter the number of vaccines:" << endl;

vector<int> vaccine(10);

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

{

        cout << center[i] << " -> ";

        cin >> vaccine[i];

}


cout << "\n";

cout << "Enter the Graph" << endl;
```

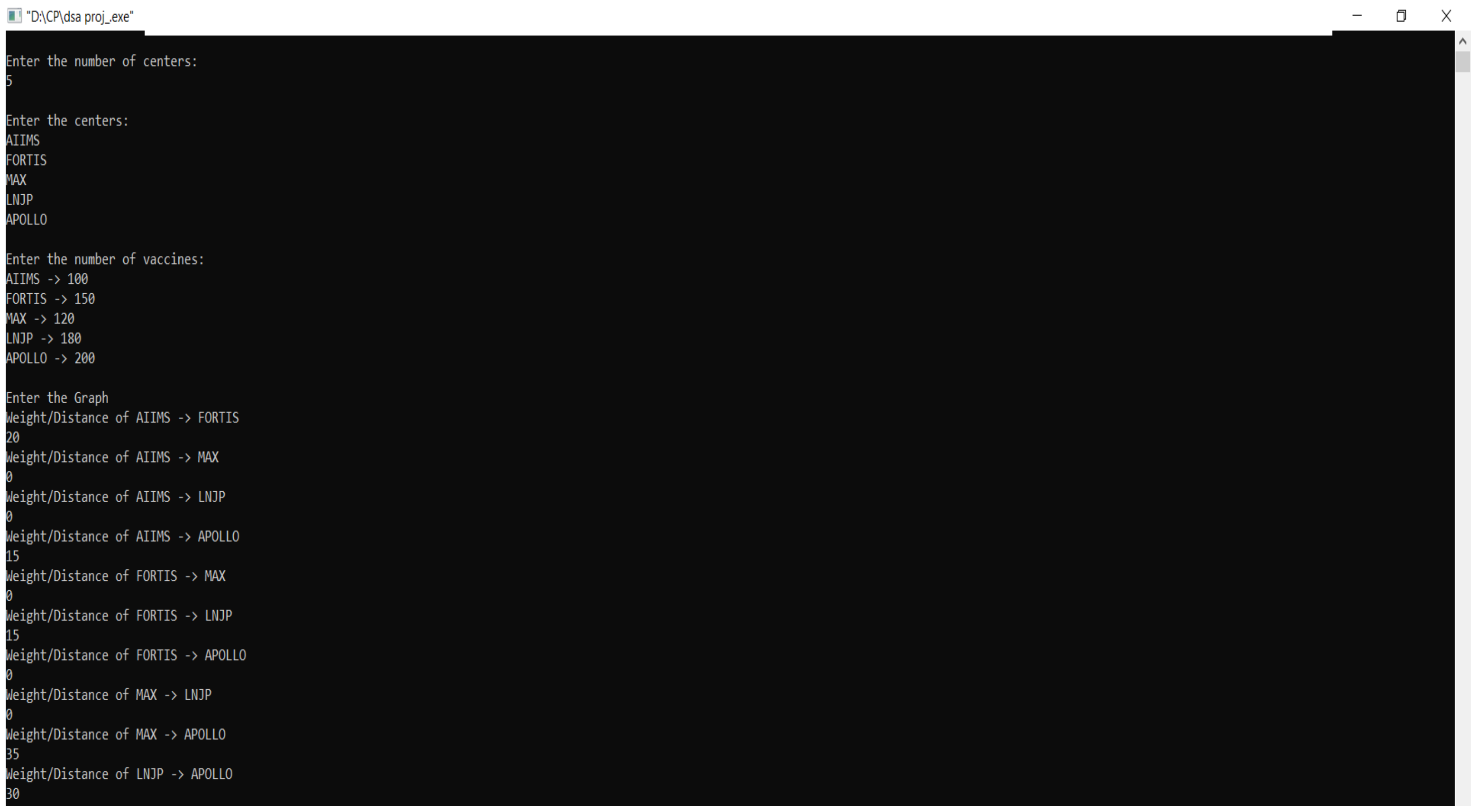
```
     getGraph(G, center, n);


     int m;
     cout<<"     ------------Enter Number Of Patients------------"<<endl;
     cin>>m;
     Patient *P[m];

for(int i=0;i<m;i++)
{
   P[i]=new Patient;
}


     queue<int> q1;
     queue<int> q2;
     queue<int> q3;
     queue<int> patientQueue;
     cout<<endl;
     for(int i=0;i<m;i++){
             if(P[i]->P_condition==3)
             {
                     int p=P[i]->patient_id;
                     q1.push(p);
             }
```

```
            else if(P[i]->P_condition==2){
                    int p=P[i]->patient_id;
                    q2.push(p);
            }
            else if(P[i]->P_condition==1){
                    int p=P[i]->patient_id;
                    q3.push(p);

            }
    }

    cout<<"\n  ---- OUTPUT----";
    cout << "\nThe Patient(s) in PHASE 3 : ";
showq(q1);
    cout << "\nThe Patient(s) in PHASE 2 : ";
showq(q2);
    cout << "\nThe Patient(s) in PHASE 1 : ";
showq(q3);
    cout<<"\n";
    queue<int> g1 = q1;
    queue<int> g2 = q2;
    queue<int> g3 = q3;
    while (!g1.empty()) {
            patientQueue.push(g1.front());
        g1.pop();
    }
        while (!g2.empty()) {
            patientQueue.push(g2.front());
        g2.pop();
    }
        while (!g3.empty()) {
            patientQueue.push(g3.front());
        g3.pop();
    }

        cout << "\nThe Patient(s)_id(s) assigned by priority phases(3,2,1)
are : ";
        showq(patientQueue);
        queue<int> patientIDS=patientQueue;
        while(!patientIDS.empty()){

            char choice;
            int PidQueue=patientIDS.front();
            for(int i=0;i<m;i++)
            {
                if(P[i]->patient_id==PidQueue)
                {
```

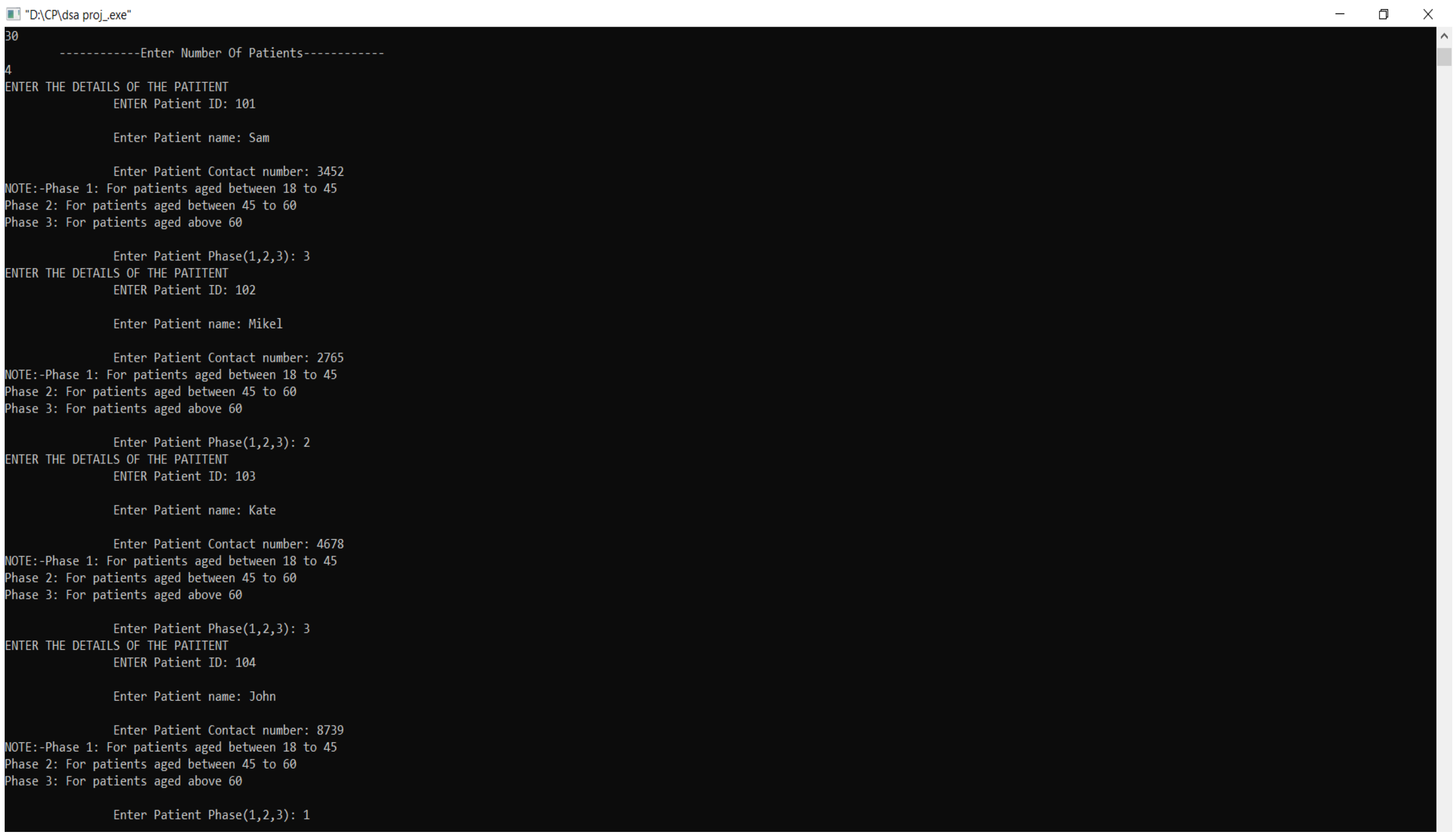
```
			P[i]->getPatientDetails();

			cout<<"\nWant to proceed?(Y/N): ";

			cin>>choice;

			if(choice=='Y'){

			start=getIndex(center,n);


			dijk(G, n, start, center, vaccine);

			string cc;

			cout<<"\n  Choose center from above based on 
your preference: ";

			cin>>cc;

			for(int j=0;j<n;j++)

			{

				if(cc==center[j])

				{

					vaccine[j]-=1;

					break;

				}

			}

			cout<<"\n  SLOT BOOKED  \n";

			cout<<"\nID: "<<P[i]->getPatientID()<<endl;

			cout<<"Patient Name: "<<P[i]-
>getPatientName()<<endl;

			cout<<"Patient Choosen Center: "<<cc<<endl;
```

```
                                cout<<"Patient Phase: "<<P[i]-
>getPatientCondition()<<endl;

                                cout<<"Patient Contact Number: "<<P[i]-
>getPatientContact()<<endl;

                                cout<<"\n______\n";

                                patientIDS.pop();

                                }

                                else{

                                        patientIDS.pop();

                                }

                        }

                }

                cout<<endl;

        }


        return 0;

}
```

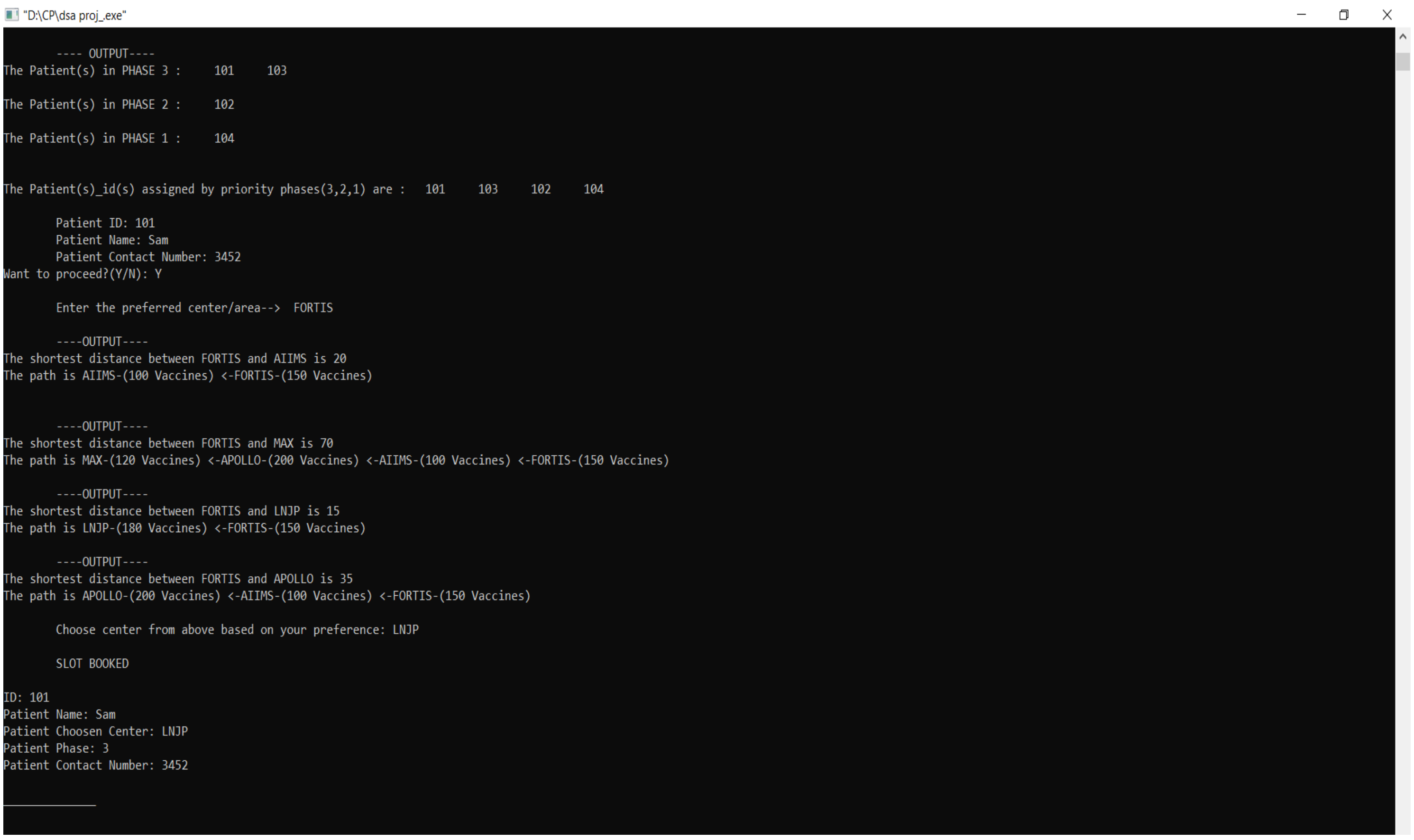
RESULTS AND OUTPUTS SCREENSHOTS



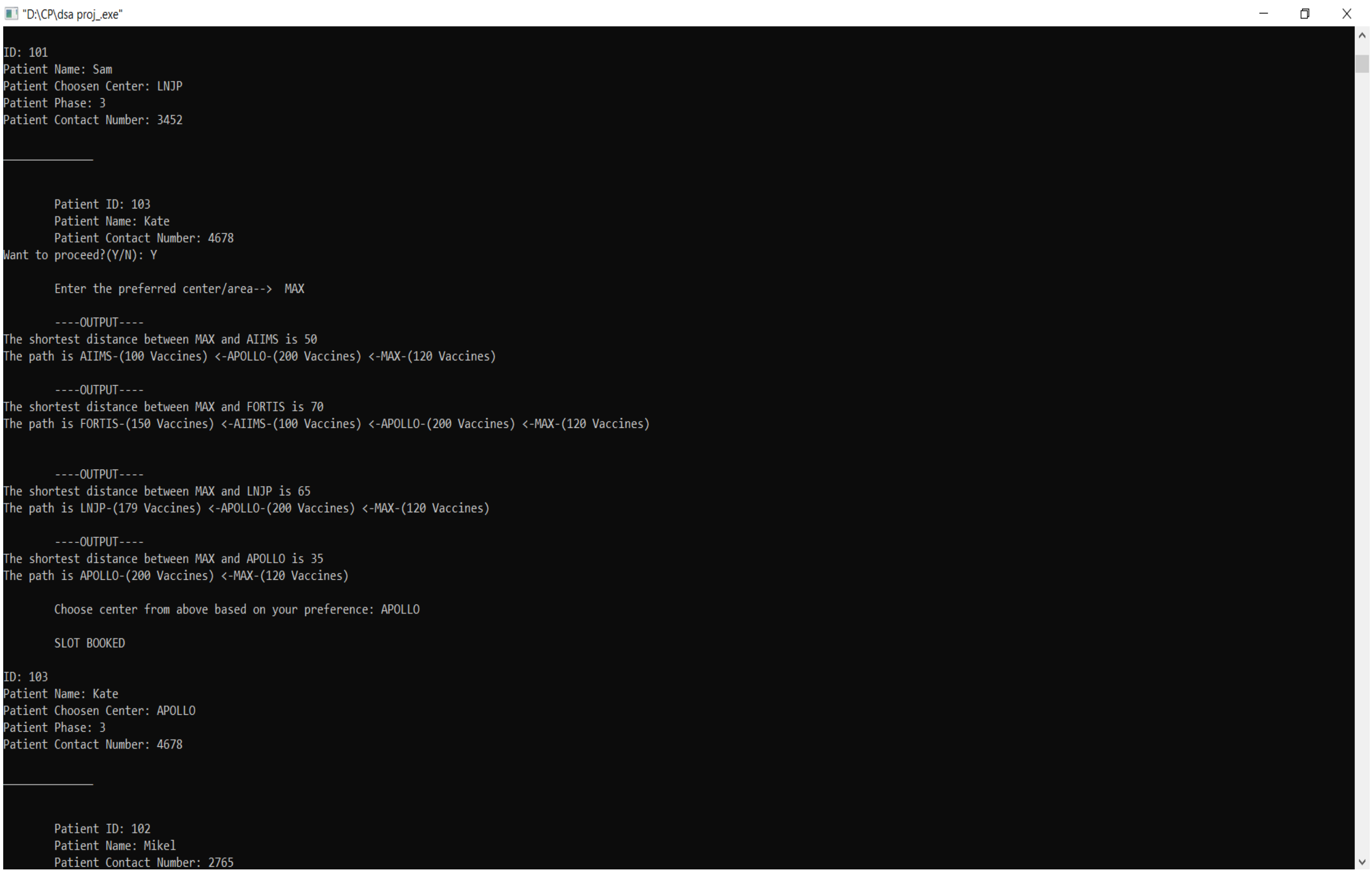
```
Enter the number of centers:
5

Enter the centers:
AIIMS
FORTIS
MAX
LNJP
APOLLO

Enter the number of vaccines:
AIIMS -> 100
FORTIS -> 150
MAX -> 120
LNJP -> 180
APOLLO -> 200

Enter the Graph
Weight/Distance of AIIMS -> FORTIS
20
Weight/Distance of AIIMS -> MAX
0
Weight/Distance of AIIMS -> LNJP
0
Weight/Distance of AIIMS -> APOLLO
15
Weight/Distance of FORTIS -> MAX
0
Weight/Distance of FORTIS -> LNJP
15
Weight/Distance of FORTIS -> APOLLO
0
Weight/Distance of MAX -> LNJP
0
Weight/Distance of MAX -> APOLLO
35
Weight/Distance of LNJP -> APOLLO
30
```



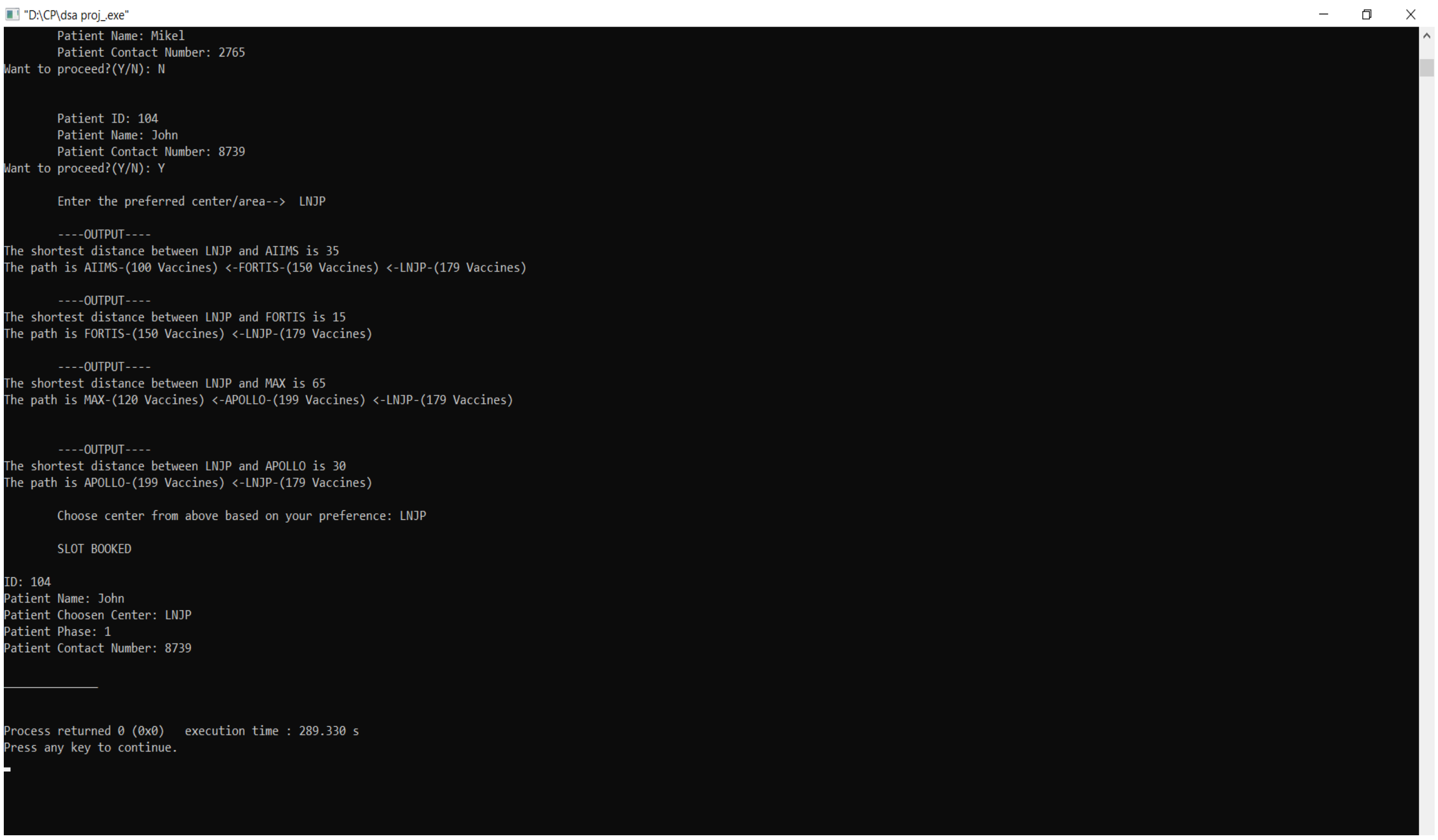
```
30
        ------------Enter Number Of Patients------------
4
ENTER THE DETAILS OF THE PATITENT
                ENTER Patient ID: 101

                Enter Patient name: Sam

                Enter Patient Contact number: 3452
NOTE:-Phase 1: For patients aged between 18 to 45
Phase 2: For patients aged between 45 to 60
Phase 3: For patients aged above 60

                Enter Patient Phase(1,2,3): 3
ENTER THE DETAILS OF THE PATITENT
                ENTER Patient ID: 102

                Enter Patient name: Mikel

                Enter Patient Contact number: 2765
NOTE:-Phase 1: For patients aged between 18 to 45
Phase 2: For patients aged between 45 to 60
Phase 3: For patients aged above 60

                Enter Patient Phase(1,2,3): 2
ENTER THE DETAILS OF THE PATITENT
                ENTER Patient ID: 103

                Enter Patient name: Kate

                Enter Patient Contact number: 4678
NOTE:-Phase 1: For patients aged between 18 to 45
Phase 2: For patients aged between 45 to 60
Phase 3: For patients aged above 60

                Enter Patient Phase(1,2,3): 3
ENTER THE DETAILS OF THE PATITENT
                ENTER Patient ID: 104

                Enter Patient name: John

                Enter Patient Contact number: 8739
NOTE:-Phase 1: For patients aged between 18 to 45
Phase 2: For patients aged between 45 to 60
Phase 3: For patients aged above 60

                Enter Patient Phase(1,2,3): 1
```



```
        ---- OUTPUT----
The Patient(s) in PHASE 3 :     101     103

The Patient(s) in PHASE 2 :     102

The Patient(s) in PHASE 1 :     104


The Patient(s)_id(s) assigned by priority phases(3,2,1) are :   101     103     102     104

        Patient ID: 101
        Patient Name: Sam
        Patient Contact Number: 3452
Want to proceed?(Y/N): Y

        Enter the preferred center/area-->  FORTIS

        ----OUTPUT----
The shortest distance between FORTIS and AIIMS is 20
The path is AIIMS-(100 Vaccines) <-FORTIS-(150 Vaccines)


        ----OUTPUT----
The shortest distance between FORTIS and MAX is 70
The path is MAX-(120 Vaccines) <-APOLLO-(200 Vaccines) <-AIIMS-(100 Vaccines) <-FORTIS-(150 Vaccines)

        ----OUTPUT----
The shortest distance between FORTIS and LNJP is 15
The path is LNJP-(180 Vaccines) <-FORTIS-(150 Vaccines)

        ----OUTPUT----
The shortest distance between FORTIS and APOLLO is 35
The path is APOLLO-(200 Vaccines) <-AIIMS-(100 Vaccines) <-FORTIS-(150 Vaccines)

        Choose center from above based on your preference: LNJP

        SLOT BOOKED

ID: 101
Patient Name: Sam
Patient Choosen Center: LNJP
Patient Phase: 3
Patient Contact Number: 3452

______________
```



```
ID: 101
Patient Name: Sam
Patient Choosen Center: LNJP
Patient Phase: 3
Patient Contact Number: 3452

______________


        Patient ID: 103
        Patient Name: Kate
        Patient Contact Number: 4678
Want to proceed?(Y/N): Y

        Enter the preferred center/area-->  MAX

        ----OUTPUT----
The shortest distance between MAX and AIIMS is 50
The path is AIIMS-(100 Vaccines) <-APOLLO-(200 Vaccines) <-MAX-(120 Vaccines)

        ----OUTPUT----
The shortest distance between MAX and FORTIS is 70
The path is FORTIS-(150 Vaccines) <-AIIMS-(100 Vaccines) <-APOLLO-(200 Vaccines) <-MAX-(120 Vaccines)


        ----OUTPUT----
The shortest distance between MAX and LNJP is 65
The path is LNJP-(179 Vaccines) <-APOLLO-(200 Vaccines) <-MAX-(120 Vaccines)

        ----OUTPUT----
The shortest distance between MAX and APOLLO is 35
The path is APOLLO-(200 Vaccines) <-MAX-(120 Vaccines)

        Choose center from above based on your preference: APOLLO

        SLOT BOOKED

ID: 103
Patient Name: Kate
Patient Choosen Center: APOLLO
Patient Phase: 3
Patient Contact Number: 4678

______________


        Patient ID: 102
        Patient Name: Mikel
        Patient Contact Number: 2765
```



"D:\CP\dsa proj_.exe"

```
        Patient Name: Mikel
        Patient Contact Number: 2765
Want to proceed?(Y/N): N


        Patient ID: 104
        Patient Name: John
        Patient Contact Number: 8739
Want to proceed?(Y/N): Y

        Enter the preferred center/area-->  LNJP

        ----OUTPUT----
The shortest distance between LNJP and AIIMS is 35
The path is AIIMS-(100 Vaccines) <-FORTIS-(150 Vaccines) <-LNJP-(179 Vaccines)

        ----OUTPUT----
The shortest distance between LNJP and FORTIS is 15
The path is FORTIS-(150 Vaccines) <-LNJP-(179 Vaccines)

        ----OUTPUT----
The shortest distance between LNJP and MAX is 65
The path is MAX-(120 Vaccines) <-APOLLO-(199 Vaccines) <-LNJP-(179 Vaccines)


        ----OUTPUT----
The shortest distance between LNJP and APOLLO is 30
The path is APOLLO-(199 Vaccines) <-LNJP-(179 Vaccines)

        Choose center from above based on your preference: LNJP

        SLOT BOOKED

ID: 104
Patient Name: John
Patient Choosen Center: LNJP
Patient Phase: 1
Patient Contact Number: 8739

______________


Process returned 0 (0x0)   execution time : 289.330 s
Press any key to continue.
```

FUTURE WORK:

As a future work, some additional work can be implemented and integrated into the code while making it more reliable and flexible for the users, for instance security, storage and its reliability.

We will be generating a unique code on each slot booking and when the person reaches the vaccination center he/ she will have to show the unique code at the center and then only vaccination will be processed further. We can even connect a database for storage purposes.

CONCLUSION:

In this report, an information systems development has been presented. It was emphasized on the basic steps, consequently taken during the projects development course as a particular attention was turned to the basic operative functions performed upon the data into the database.