Title
Automated transport management system (for educational institutes)

Name: Maryam Khan SRN: PES2UG21CS283 Name: Mudundi Manasvi Varma SRN: PES2UG21CS305 Name: Pooja Satheesh SRN: PES2UG21CS374 Name: Preethi M SRN: PES2UG21CS396

Source Code:

- login.cpp

```
#include<iostream>
#include<string>
#include<fstream>
#include<sstream>
#include<bits/stdc++.h>
#include"bidirectional_dijkstra.h"
#include"seating.h"

using namespace std;

//Default constructor(using initializer list)
Dijkstra::Dijkstra():destination("kormangala"){} //setting kormangala as default destination

//Parametrized constructor(dynamic constructor)
Dijkstra::Dijkstra(string des,int vertices)
{
    //mapping to get string names as integers to use in indexing and vice versa
mapping={{"_pesuniversity", 0}, {"bannarghatta", 1}, {"churchstreet", 2}, {"diarycircle", 3}, {"hosur", 4}, {"indiranagar", 5}, {"jayadevahospital", 6}, {"kormangala", 7}, {"mgroad", 8}, {"silkboard", 9}, {"ulsoor", 10}};
mapping_rev={{0, "_pesuniversity"}, {1, "bannarghatta"}, {2, "churchstreet"}, {3, "diarycircle"}, {4, "hosur"}, {5, "indiranagar"}, {6, "jayadevahospital"}, {7, "kormangala"}, {8, "mgroad"}, {9, "silkboard"}, {10, "ulsoor"}};

    source="_pesuniversity";
    destination=des;
    no_vertices=vertices;

    graph=(Graph*)calloc(1,sizeof(Graph));
    graph->nodes=(Node**)calloc(no_vertices,sizeof(Node*));

    forward_queue.push_back({0,mapping[source]});
    backward_queue.push_back({0,mapping[destination]});
```

```
    //distance to reach node from source
    forward_distance=new int[no_vertices]{};
    //distance to reach node from destination
    backward_distance=new int[no_vertices]{};
    //gives the parent node in the route
    forward_path=new int[no_vertices]{};
    backward_path=new int[no_vertices]{};
    //to show node is already selected
    forward_selected=new int[no_vertices]{};
    backward_selected=new int[no_vertices]{};

    for(int i=0;i<no_vertices;i++)
    {
        forward_distance[i]=99999;
        backward_distance[i]=99999;
        forward_path[i]=-1;
        backward_path[i]=-1;
    }

    forward_distance[mapping[source]]=0;
    backward_distance[mapping[destination]]=0;
    forward_path[mapping[source]]=mapping[source];
    backward_path[mapping[destination]]=mapping[destination];

    count=0;
    meeting_point=0; //point where the forward and backward routing meet, algorithm stops when a meeting point can be found
}

//Convert input from csv(routes.csv) into the graph and node structures void Dijkstra::take_input()
{
    fstream fin;
    string line,name,no_neigh,neighbours,neighbour,distance;
    fin.open("routes.csv",ios::in);
    getline(fin,line); //discard the first line(has headings)

    for(int i=0;i<no_vertices;i++)
    {
        graph->nodes[i]=(Node*)calloc(1,sizeof(Node));
        graph->nodes[i]->id=i;

        //get each row
        getline(fin,line);

        //split to get info separated by ','
        stringstream s1(line);
        getline(s1,name,',');
        getline(s1,no_neigh,',');
        getline(s1,neighbours,',');
        stringstream s2(neighbours);

        graph->nodes[i]->no_neigh=stoi(no_neigh); //words split from file are of string type, converting it to int
        graph->nodes[i]->neighbours=(Edge**)calloc(graph->nodes[i]->no_neigh,sizeof(Edge*));
```

```
            for(int j=0;j<graph->nodes[i]->no_neigh;j++)
            {
                getline(s2,neighbour,';');
                stringstream s3(neighbour);
                getline(s3,name,':');
                getline(s3,distance,':');
                graph->nodes[i]->neighbours[j]=(Edge*)calloc(1,sizeof(Edge));
                graph->nodes[i]->neighbours[j]->id=mapping[name];
                graph->nodes[i]->neighbours[j]->weight=stoi(distance);
            }
        }
        fin.close();
}

//print graph(debugging and reference purposes)
void Dijkstra::print_graph()
{
        for(int i=0;i<no_vertices;i++)
        {
            cout<<"id "<<i<<endl;
            for(int j=0;j<graph->nodes[i]->no_neigh;j++)
            {
                cout<<"id "<<graph->nodes[i]->neighbours[j]->id<<" weight
"<<graph-  >nodes[i]->neighbours[j]->weight<<endl;
            }
        }
}

//relaxes edge from forward
void Dijkstra::relax_edge_f(int &u,int &v)
{
        int v_=graph->nodes[u]->neighbours[v]->id; //get id of neighbour from neighbour
count id
        count+=1;
        if(forward_distance[v_]>forward_distance[u]+graph->nodes[u]->neighbours[v]
-  >weight)
        {
            forward_distance[v_]=forward_distance[u]+graph->nodes[u]->neighbours[v]
-  >weight;
            forward_path[graph->nodes[u]->neighbours[v]->id]=u;
            forward_queue.push_back({forward_distance[v_],v_});
        }
}

//relaxes edge from backward
void Dijkstra::relax_edge_b(int &u,int &v)
{
        int v_=graph->nodes[u]->neighbours[v]->id; //get id of neighbour from neighbour
count id
        count+=1;
        if(backward_distance[v_]>backward_distance[u]+graph->nodes[u]->neighbours[v]
-  >weight)
        {
            backward_distance[v_]=backward_distance[u]+graph->nodes[u]->neighbours[v]-
>weight;
            backward_path[graph->nodes[u]->neighbours[v]->id]=u;
            backward_queue.push_back({backward_distance[v_],v_});
```

```
        }
    }

//print distance and other information about route taken
int Dijkstra::print_distance()
{
    int node;
    int dist=forward_distance[meeting_point]+backward_distance[meeting_point];

    //lambda function to calculate eta
    auto eta = [](int d,double s){return d/s;};

    cout<<"Meeting Point "<<meeting_point<<endl;
    cout<<"Distance from "<<source<<" to "<<destination<<" = "<<dist<<"
    Km"<<endl; cout<<"Estimated Time to reach = "<<eta(dist,30.0)<<" hrs"<<endl;
    cout<<"Route:"<<endl;
    list <int> path;
    node=meeting_point;

    //store node in path from front for forward
    while(forward_path[node]!=node)
    {
        path.push_front(node);
        node=forward_path[node];
    }
    node=meeting_point;

    //store node in path from back for backward
    while(backward_path[node]!=node)
    {
        node=backward_path[node];
        path.push_back(node);
    }
    path.push_front(mapping[source]);
    //print places in route
    for(auto ptr=path.begin();ptr!=path.end();ptr++)
        cout<<"-> "<<mapping_rev[int(*ptr)]<<endl;
    cout<<endl;
    cout<<"Count of updates: "<<count<<endl;

    return dist; //returns distance to reach destination
}

//implementation of bidirectional dijkstra
void Dijkstra::dijkstra()
{
    int q_count=1;
    int u;
    Queue pop;
    while(q_count<no_vertices)
    {
        while(1)
        {
            make_heap(forward_queue.begin(),forward_queue.end()); //form a heap
from the forward queue
            pop=forward_queue.front(); //gets the min node
            pop_heap(forward_queue.begin(),forward_queue.end()); //sends the min
```

```
node to end
            forward_queue.pop_back(); //pops end node
            u=pop.node_id;
            if(forward_selected[u]==1) //if min is already selected choose
                another continue;
            else
                break;
        }
        forward_selected[u]=1; //add into selected
        if(backward_selected[u]==1) //if already in backward then meeting point
found
        {
            meeting_point=u;
            break;
        }
        else
        {
            for(int j=0;j<graph->nodes[u]->no_neigh;j++)
            {
                if(forward_selected[graph->nodes[u]->neighbours[j]->id]==0)
                {
                    relax_edge_f(u,j);
                }
            }
        }

        while(1)
        {
            make_heap(backward_queue.begin(),backward_queue.end());
            pop=backward_queue.front();
            pop_heap(backward_queue.begin(),backward_queue.end());
            backward_queue.pop_back();
            u=pop.node_id;
            if(backward_selected[u]==1)
                continue;
            else
                break;
        }
        backward_selected[u]=1;
        if(forward_selected[u]==1)
        {
            meeting_point=u;
            break;
        }
        else
        {
            for(int j=0;j<graph->nodes[u]->no_neigh;j++)
            {
                if(backward_selected[graph->nodes[u]->neighbours[j]->id]==0)
                {
                    relax_edge_b(u,j);
                }
            }
        }
    }
}
```

```
//destructor
Dijkstra::~Dijkstra()
{
    delete[] forward_distance;
    delete[] backward_distance;
    delete[] forward_path;
    delete[] backward_path;
    delete[] forward_selected;
    delete[] backward_selected;
}

//template function to calculate fees
template <typename T> T mycalc(T initial, T distance)
{
    return initial+max(0,distance-3)*500; //initial fees for 3km, add 500 for every
additional km
}

//override the virtual function
void Seat::file_open()
{
    ifstream file("buses.csv"); //creating a file object for reading (file
    handler) if(!file) //checks if file opening was successful
    {
        cout<<"error"<<endl;
    }

    string line; //string that stores every line of csv file read
    while(getline(file,line)) //accessing every line of csv file until the end of
file
    {
        vector<string>row; //creating a vector to store a row of csv file
        stringstream ss(line); //retrieve a series of value from the line
        read string each;

        while(getline(ss,each,',')) //retrieving each value within the
        row {
            row.push_back(each); //pushing the value of each cell into the row
vector
        }
        data.push_back(row); //row vector pushed into data vector
    }
    file.close();//file handler closed
}

//function that writes into the file
void Seat::file_write()
{
    ofstream file("buses.csv",ofstream::out);
    for(auto& row:data) //accessing row in the 2d data vector
    {
        for(auto& each:row)
        {
            file<<each<<','; //writing into the file value by value with ',' as
delimiter
        }
```

```
            file<<'\n'; //moving to next line while storing
        }
        file.close();
}
void Seat::retrieve_seats()
{
    int check;
    for(auto& row:data) //accessing row by row
    {
        if(row[0]==dest) //if the dest requested is same as row header
        {
            int c=stoi(row[1]); //the capacity of bus retrieved
            int o=stoi(row[2]); //the number of seats occupied
            if(c==o) //if they are equal, the capacity would exceed
            {
                cout<<"AT FULL CAPACITY!"<<endl;
                cout<<"Please choose another destination."<<endl;
                break;
            }
            else //when the capacity is not exceeded
            {
                cout<<"Seats are available"<<endl;
                cout<<"Successfully booked...\nSeats now available:"<<c-o-1<<endl;
string o=to_string(stoi(row[2])+1); //converting the occupied into  int,
incrementing it and then converting back to string
                row[2]=o; //rewriting the data, to new number of occupied
            }
        }
    }
    file_write(); //calling of write function to update seats
booked }


//paramterized constructor
Routes::Routes(int id)
{
    this->id=id;
    int amount= mycalc<int>(1000,students[id].distance); //calculate fees by
sending a initial ammount, and the total distance calculated by using
Dijkstra  students[id].fee=amount;
    students[id].time=t;
    Seat obj(students[id].dest);
    obj.file_open();
    obj.retrieve_seats();
}

//user defined namespace for password
namespace space
{
    //check if password is in accepted format using exception
    handling  int check(string p)
    {
        try
        {
            if(p.size()<8)
            {
                throw(p);
```

```
            }
        }
        catch(string p)
        {
            cout<<"Bad password: Password should be greater than 8
characters"<<endl;
            return 0;
        }

        //check if a upper, lower and special character is in password
        int sc=0,lc=0,up=0;
        for (int x:p)
        {
            if(x>=33 && x<=47 || x==64)
            {
                sc++;
            }
            if(x>=65 && x<=90)
            {
                up++;
            }
            if(x>=97 && x<=122)
            {
                lc++;
            }
        }

        if (sc>0 && lc>0 && up>0)
        {
            cout<<"Good password"<<endl;
            return 1;
        }
        else
        {
            cout<<"Bad password: Include atleast one lower case, one upper case and
one special character"<<endl;
            return 0;
        }
    }
}

using namespace space;

// store values from login.csv into Student class variables
void create()
{
    string info[4];
    ifstream file;
    file.open("login.csv");

    string line,word;
    getline(file, line);
    int y=1;
    while(getline(file, line))
    {
        stringstream str(line);
        int x=0;
```

```
            while(getline(str, word, ','))
            {
                info[x]=word;
                x++;
            }
            students[y].name=info[0];
            students[y].id=stoi(info[2]);
            students[y].password=info[1];
            students[y].email=info[3];
            y++;
        }
}


void choose_destination(int id)
{
    int choice ;
    cout<<"\n------------CHOOSE DESTINATION------------\n\n";
    cout<<"||-------------1.MG
    ROAD-----------------------------||"<<endl;
    cout<<"||-------------2.CHURCH
    STREET--------------------------||"<<endl;
    cout<<"||-------------3.ULSOOR-------------------------------||"<<endl
    ;
    cout<<"||-------------4.INDIRANAGAR---------------------------||"<<endl
    ;
    cout<<"||-------------5.KORMANGALA----------------------------||"<<endl
    ;
    cout<<"||-------------6.HOSUR---------------------------------||"<<endl
    ;                                                cout<<"||-------------7.DIARY
    CIRCLE--------------------------||"<<endl;
    cout<<"||-------------8.JAYADEVA------------------------------||"<<endl
    ;                                                 cout<<"||-------------9.SILK
    BOARD---------------------------||"<<endl;
    cout<<"||-------------10.BANNARGHATTA
    ROAD--------------------||"<<endl;  cin>>choice;
    string
arr[10]={"mgroad","churchstreet","ulsoor","indiranagar","kormangala","hosur","diar
y  circle","jayadevahospital","silkboard","bannarghatta"}; //array to convert user
choice(int) to actual name
    string destination=arr[choice-1];

    int no_vertices=11;

    //call the functions to use implemented routing algorithm
    Dijkstra obj(destination,no_vertices);
    obj.take_input();
    //obj.print_graph();
    obj.dijkstra();
    students[id].distance=obj.print_distance();

    students[id].dest=destination;
    Routes r(id);
    cout<<"Total fees = "<<students[id].fee<<endl;
}

//overloaded function to check id and password match
```

```
void checkpass(int id,string pass)
{
    cout<<"In ID checkbox"<<endl;
    if (students[id].password==pass)
    {
        cout<<"logged in successfuly"<<endl;
        choose_destination( id);
    }
    else
    {
        cout<<"incorrect password"<<endl;
    }
}
//overloaded function to check emailid and password match
void checkpass(string e,string pass)
{
    cout<<"In email checkbox"<<endl;
    int flag=0;
    for (int x=1;x<5;x++)
    {
        if (students[x].email==e)
        {
            if (students[x].password==pass)
            {
                cout<<"logged in successfuly"<<endl;
                choose_destination(x);
                return;
            }
            else
            {
                cout<<"incorrect password"<<endl;
                return;
            }
        }
    }
    cout<<"Email ID does not exist"<<endl;
}

//starting page for user login
void login()
{
    string u;string p;
    cout<<"\nEnter user id or email id:"<<endl;
    cin >>u;
    cout<<"Enter password:"<<endl;
    cin>>p;
    int res=check(p); //check if password is valid
    if (res==1)
    {
        if (u.find('@')!=string::npos)
        {
            checkpass(u,p);
        }
        else
        {
            checkpass(stoi(u),p);
        }
```

```
    }
    else
    {
        cout<<"wrong format of password"<<endl;
    }
}

int main()
{
    create();
    login();
    return 0;
}
```

Header Files:

- bidirectional_dijkstra.h

```
/*
Topics covered:
- operator overloading(to make inbuilt heap function as min heap instead of its
default as max heap)
- insertion and extraction operators
- polymorphism
- structures
- vectors, maps, lists, heaps, arrays, iterators
- dynamic constructor, parametrized constructor, initializer
lists - DMA, pointers, reference variables
- class scope operator, access specifiers, virtual destructor
- file manipulation
*/

using namespace std;

struct Edge
{
    int id;
    int weight;
};
struct Node
{
    int id;
    int no_neigh;
    Edge** neighbours;
};
struct Graph
{
    Node** nodes;
};
struct Queue
{
    int distance;
    int node_id;

    bool operator<(const Queue& other)const{
        return distance>other.distance;
    }
};
```

```
class Dijkstra
{
    private:
        map<string,int> mapping;
        map<int,string> mapping_rev;
        string source;
        string destination;
        int no_vertices;
        Graph* graph;
        vector <Queue> forward_queue;
        vector <Queue> backward_queue;
        int* forward_distance;
        int* backward_distance;
        int* forward_path;
        int* backward_path;
        int* forward_selected;
        int* backward_selected;
        int count;
        int meeting_point;
    public:
        Dijkstra();
        Dijkstra(string, int);
        void take_input();
        void print_graph();
        void relax_edge_f(int&, int&);
        void relax_edge_b(int&, int&);
        int print_distance();
        void dijkstra();
        virtual ~Dijkstra();
};
```

- seating.h

```
/*
TOPICS COVERED:
--> Range For: iterating vectors
--> Pure Virtual Function: for opening and closing csv
files --> Abstraction: The class File
--> Template: to calculate fees
--> Friend function: choosing a destination
*/


using namespace std;

class Routes; //forward declaration
class Files
{
    public:
     //pure virtual function for opening csv files
     virtual void file_open()=0;
```

```
    //pure virtual function for writing into csv files
    virtual void file_write()=0;
};

class Seat:public Files
{
    string dest; //the destination of the uset
    vector<vector<string>> data; //2d vector of data inside of csv file
    public:
        //parameterised constructor which initialises the
        destination Seat(string d):dest(d){};
        //function that opens files
        void file_open();

        //function that writes into the file
        void file_write();

        void retrieve_seats();

        friend class Routes;
};

template <typename T> T mycalc(T initial, T
distance); class Student{
    public:
       string name;
       string password;
       int id;
       string email;
       string dest;
       int distance;
       double fee;
       int time;
       friend void choose_destination(int id);
};
Student students[10];
class Routes:public Student
{
    int id;
    double t = 10.0;
    public:
        Routes(int id);
    friend void choose_destination(int id);
};
```

CSV Files:

- buses.csv

```
destination,capacity,booked,
indiranagar,15,11,
bannarghatta,20,20,
kormangala,30,16,
silkboard,10,10,
churchstreet,10,3,
mgroad,10,9,
dairycircle,20,18,
```

```
hosur,30,10,
jayadevahospital,10,10,
ulsoor,10,5,
```

- routes.csv

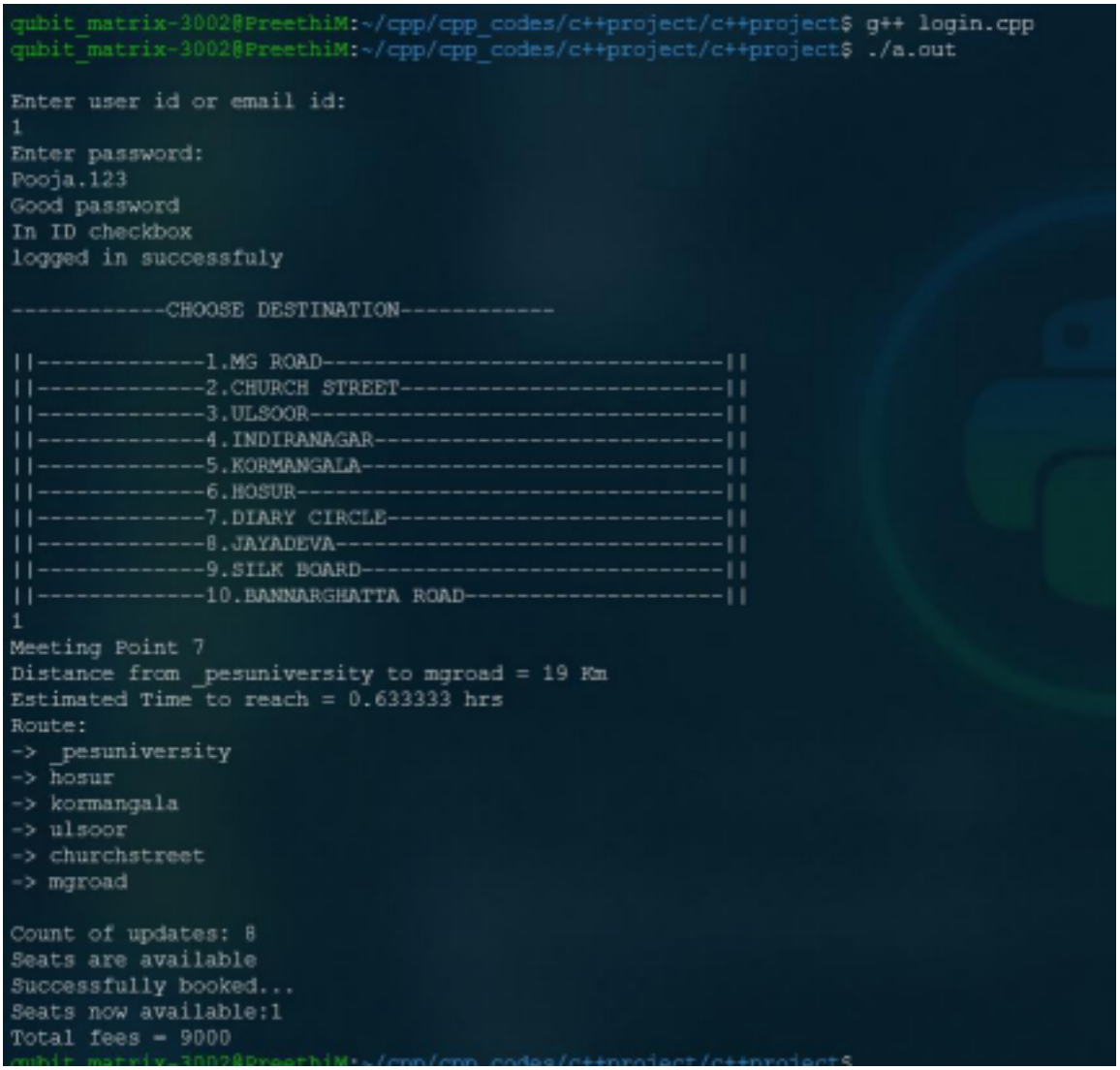
```
node_name,no_of_neighbours,neighbourlist
_pesuniversity,1,hosur:1
bannerghatta,2,jayadevahospital:4;silkboard:9
churchstreet,2,mgroad:1;ulsoor:10
dairycircle,3,hosur:1;jayadevahospital:2;kormangala:5
hosur,3,_pesuniversity:1;dairycircle:1;kormangala:1
indiranagar,1,ulsoor:9
jayadevahospital,4,bannerghatta:4;dairycircle:2;kormangala:1,silkboard:
3 kormangala,4,dairycircle:5;hosur:1;jayadevahospital:1;ulsoor:6
mgroad,1,churchstreet:1
silkboard,2,bannerghatta:9;jayadevahospital:3
ulsoor,3,kormangala:6;indiranagar:9;churchstreet:10
```

- login.csv

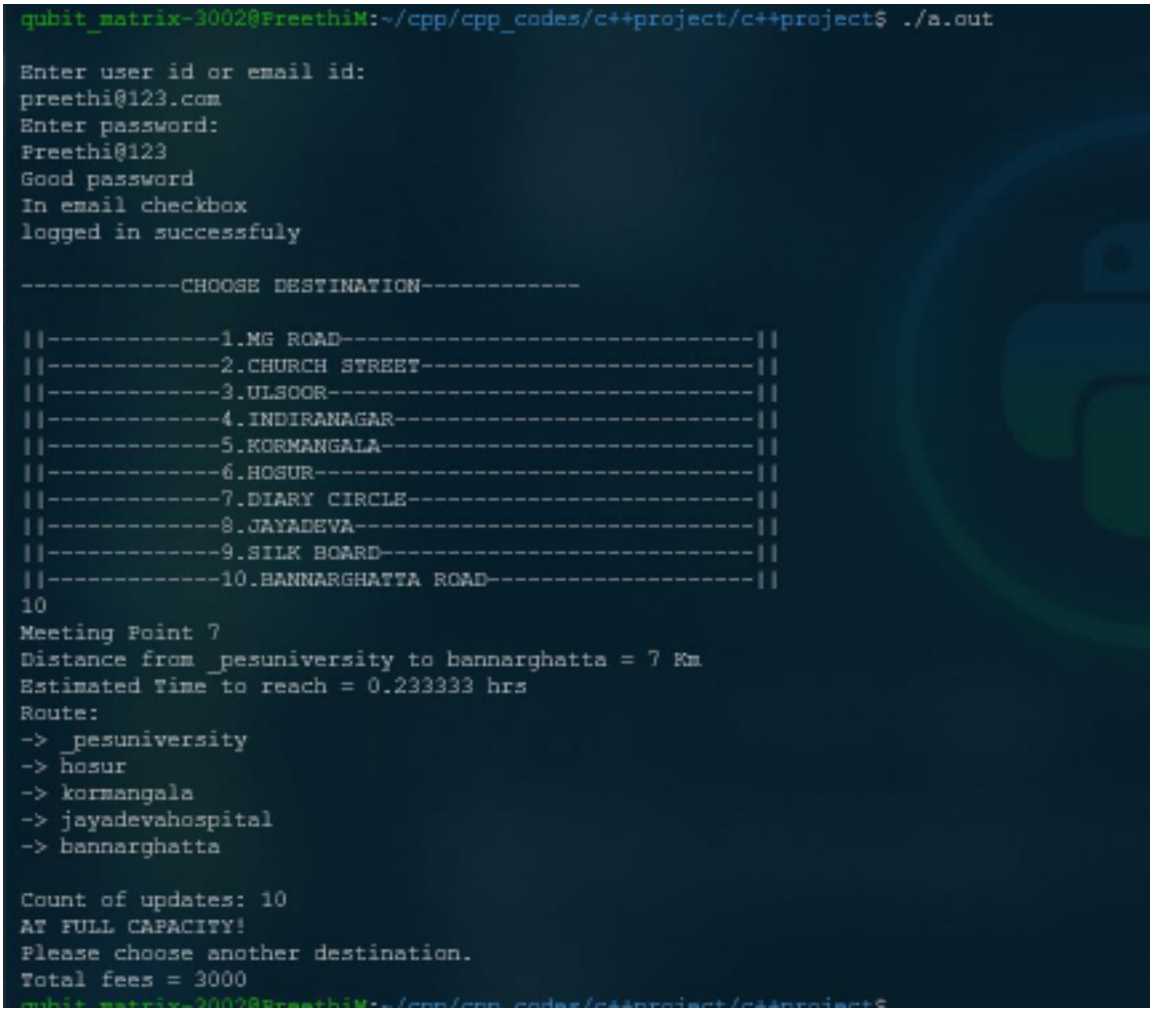
```
test,0,id,email
pooja,Pooja.123,1,pooja@123.com
preethi,Preethi@123,2,preethi@123.com
maryam,Maryam@123,3,maryam@123.com
manasvi,Manasvi#123,4,manasvi@123.com
```

Output Screenshots:

1. Logging in using user-id with valid and correct password, destination has available seats.



2. Logging in using email-id with valid and correct password, destination with capacity already full.



3. Logging in using email-id, valid and wrong password.

4.Logging in using email-id, invalid password

```
qubit_matrix-3002@PreethiM:~/cpp/cpp_codes/c++project/c++project$ ./a.out

Enter user id or email id:
manasvi@123.com
Enter password:
manasvi123
Bad password: Include atleast one lower case, one upper case and one special character
wrong format of password
qubit_matrix-3002@PreethiM:~/cpp/cpp_codes/c++project/c++project$ ./a.out

Enter user id or email id:
manasvi@123.com
Enter password:
Manasv1
Bad password: Password should be greater than 8 characters
wrong format of password
```

5. Logging in using non-existing email-id

```
qubit_matrix-3002@PreethiM:~/cpp/cpp_codes/c++project/c++project$ ./a.out

Enter user id or email id:
manasvi@1.com
Enter password:
Manasvi@213
Good password
In email checkbox
Email ID does not exist
qubit_matrix-3002@PreethiM:~/cpp/cpp_codes/c++project/c++project$
```