بسم الله الرحمن الرحیم

نام و نام خانوادگی: فائزه حیدری فرج

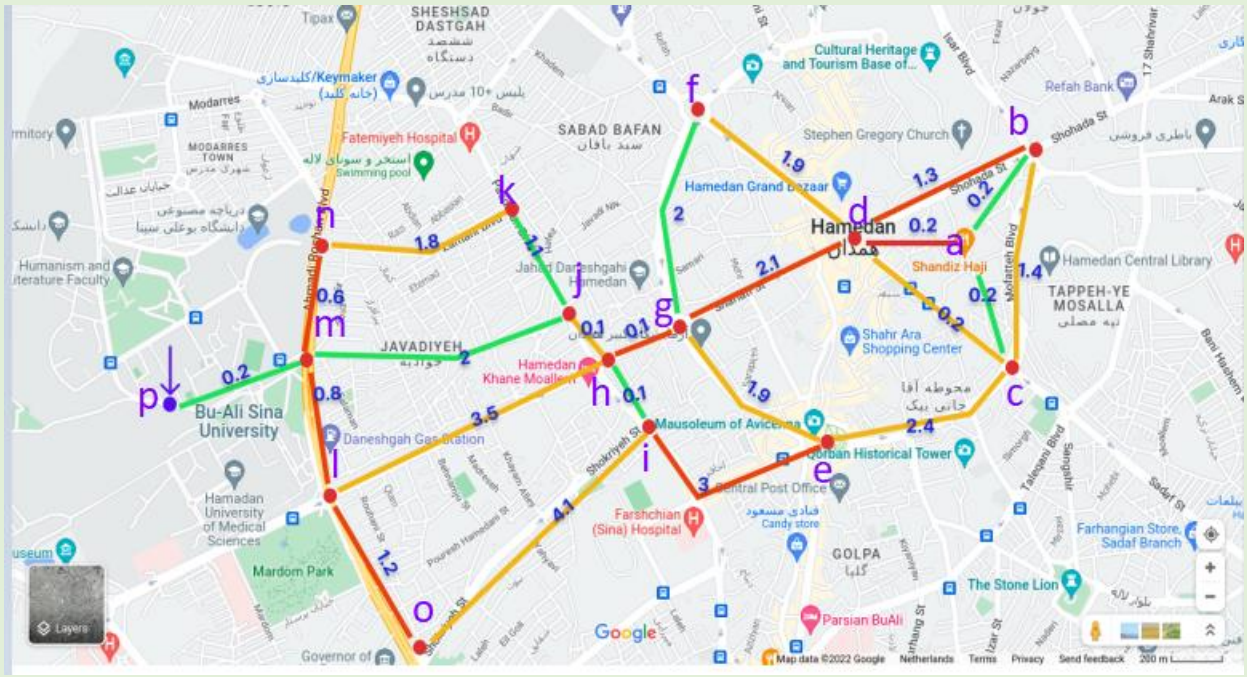
عنوان پروژه: مسیر یابی

نام استاد: سرکارخانم خدابنده لویی

تابستان ۱۴۰۱

شرح پروژه:

در این پروژه قرار است از رستوران شاندیز حاجی به خوابگاه غدیر دانشگاه بوعلی سینا بهترین مسیر را پیدا کنیم. مسیر داده شده یک گراف است. که نقاط آن را مطابق شکل زیر نام گذاری کرده ایم.



برنامه شامل سه کلاس به نام های Vertex , Graph , Edge است.

کلاس Edge :

```
class Edge
{
private:
    string destination;
    double cost;
public:
    Edge(string des, double cost)
    {
        this->destination = des;
        this->cost = cost;
    }
    string getDestination()
    {
        return this->destination;
    }
    double getCost()
    {
        return this->cost;
    }
    void setCost(double ct)
    {
        this->cost = ct;
    }
};
```

کلاس Vertex :

```
class Vertex
{
private:
    vector<Edge> edges;
    string name;
public:
    Vertex(vector<Edge> edg, string name)
    {
        this->name = name;
        for (int i = 0; i < edg.size(); i++)
        {
            edges.push_back(edg[i]);
        }
    }
    Vertex()
    {

    }
    void add_vertex(string name, vector<Edge> edg)
    {
        this->name = name;
        for (int i = 0; i < edg.size(); i++)
        {
            edges.push_back(edg[i]);
        }
    }
    void clear_vertex()
    {
        this->edges.clear();
        this->name = "";
    }
    string getVertexName()
    {
        return this->name;
    }
    vector<Edge> getEdges()
    {
        return edges;
    }

};
```

کلاس Graph :

تابع string shortest_path(string start, string finish) در کلاس Graph با گرفتن نود مبدا و نود مقصد کم هزینه ترین مسیر را می یابد .

```
class Graph
{
public:
    vector<Vertex> vertices;
    string shortest_path(string start, string finish)
    {
        vector<TempString>* previous = new vector<TempString>();
        vector<Edge>* distances = new vector<Edge>();
        vector<string>* nodes = new vector<string>();
        vector<string>* path = new vector<string>();

        string returnPath = "";

        for (int i = 0; i < vertices.size(); i++)
        {
            string x = vertices.at(i).getVertexName();
            if (x == start)
            {
                distances->push_back({ x,1 });
            }
            else
            {
                distances->push_back({ x,INT_MAX });
            }
            nodes->push_back(x);
        }
```

```
while (nodes->size() != 0)
{

    string smallest = nodes->at(0);
    nodes->erase(nodes->begin() + 0);
    if (smallest == finish)
    {
        path->clear();

        bool f = false;
        do
        {
            f = false;
            int l = 0;
            for (; l < previous->size(); l++)
            {
                f = false;
                if (previous->at(l).s1 == smallest)
                {
                    f = true;
                    break;
                }
            }
            if (f == false)
            {
                break;
            }
            path->push_back(smallest);
            smallest = previous->at(l).s2;

        } while (f);
        break;
    }

    bool f = false;
    for (int i = 0; i < distances->size(); i++)
    {
        if (distances->at(i).getDestination() == smallest)
        {
            if (distances->at(i).getCost() == INT_MAX)
```

```
        bool f = false;
        for (int i = 0; i < distances->size(); i++)
        {
            if (distances->at(i).getDestination() == smallest)
            {
                if (distances->at(i).getCost() == INT_MAX)
                {
                    f = true;
                    break;
                }
            }
        }
        if (f == true)
        {
            break;
        }

        int indexOfSmallest = -1;
        for (int i = 0; i < vertices.size(); i++)
        {
            if (vertices.at(i).getVertexName() == smallest)
            {
                indexOfSmallest = i;
                break;
            }
        }


        vector<Edge> neighbor = vertices.at(indexOfSmallest).getEdges();//

        for (int i = 0; i < neighbor.size(); i++)
        {
            int indexOFSmallestInDistances = -1;
            for (int j = 0; j < distances->size(); j++)
            {
                if (distances->at(i).getDestination() == smallest)
                {
                    indexOFSmallestInDistances = j;
                    break;
                }
```

```
                if (distances->at(i).getDestination() == smallest)
                {
                    indexOFSmallestInDistances = j;
                    break;
                }
            }
            if (indexOFSmallestInDistances == -1)
            {
                continue;
            }
            double alt = neighbor.at(i).getCost() + distances->at(indexOFSmallestInDistances).getCost();
            string neighborName = neighbor.at(i).getDestination();
            int indexOFNeighbor = -1;
            for (int j = 0; j < neighbor.size(); j++)
            {
                if (neighbor.at(j).getDestination() == neighborName)
                {
                    indexOFNeighbor = j;
                    break;
                }
            }
            if (indexOFNeighbor == -1)
            {
                continue;
            }
            if (alt < distances->at(indexOFNeighbor).getCost())
            {
                distances->at(indexOFNeighbor).setCost(alt);
                previous->push_back({ neighborName,smallest });
            }
        }
    }
    path->push_back(start);
    for (int l = 0; l < path->size(); l++)
    {
        returnPath += "-->" + path->at(l);
    }
    return returnPath;
}
};
```