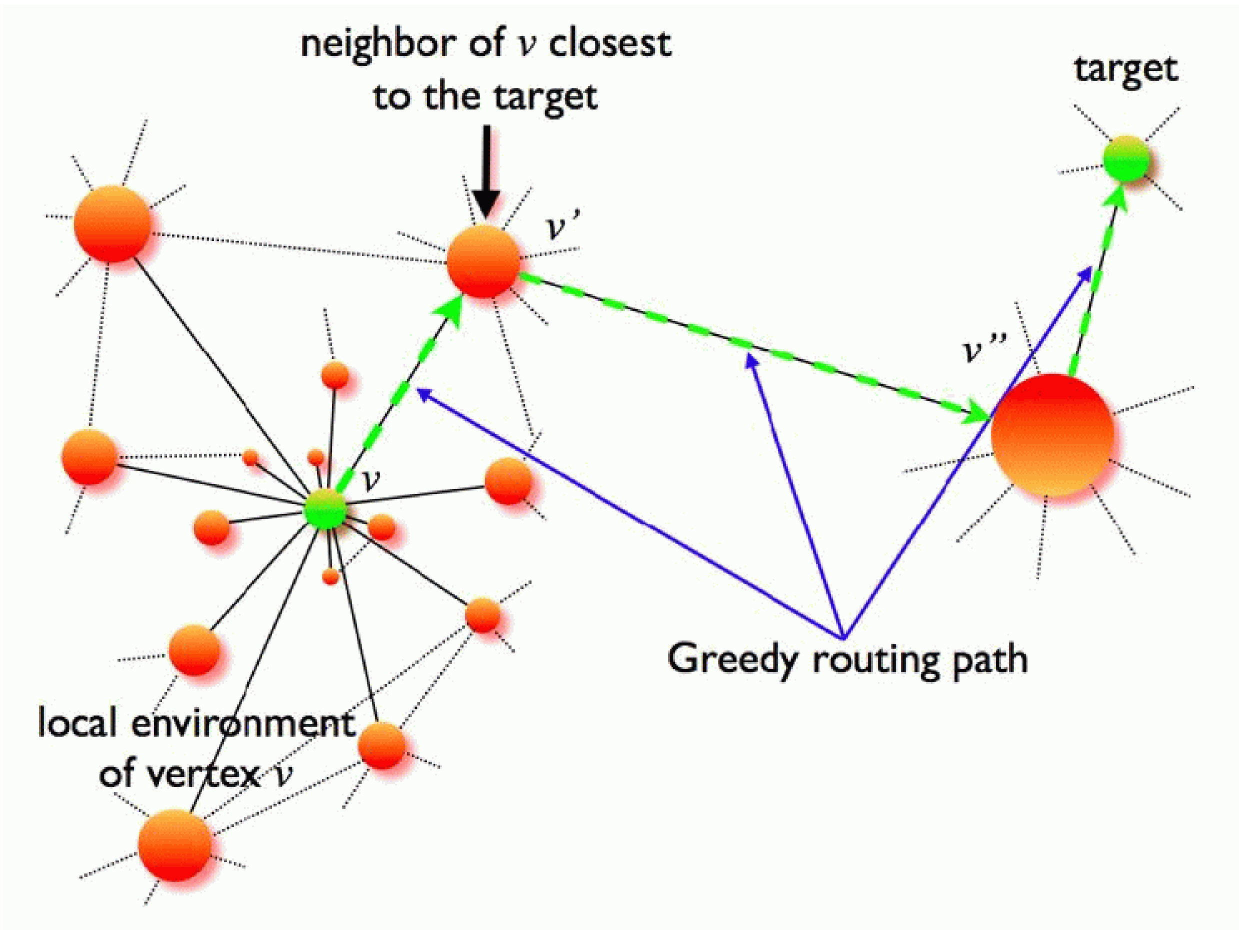
* Map Routing *



Team Members :-

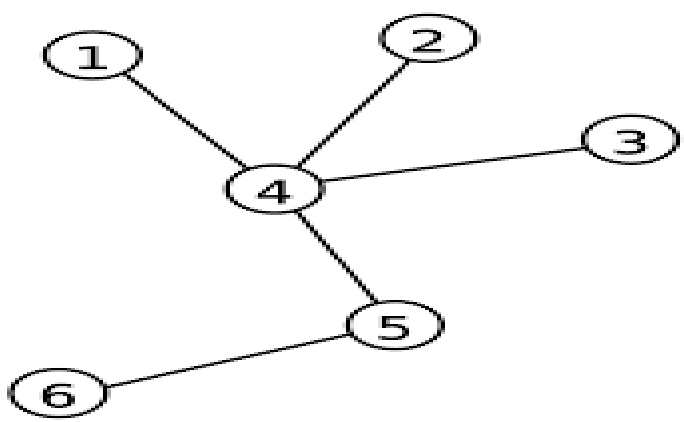
1. Mohamed Hassen Mohamed Radwan (Sec 14) .
2. Mohamed Khaled Gomaa (Sec 14) .
3. Ahmed Saad Eldeen Abdul Ghani (Sec 1) .

-3rd Year 2018-2019

* Map construction *

- Map was represented as Undirected Weighted Graph:-

Each Point represented as node and each road represented As edge and each weight represented as time to reach from Point to another point.



- Construct _Graph_Source_Code:- Complexity:- O(E).

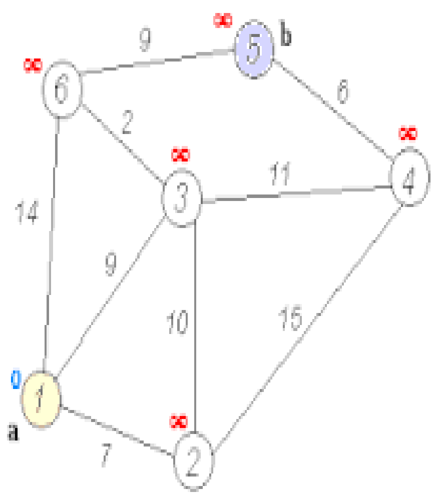
```
Console.WriteLine ("Enter Number of Intersection Points ?!..");
            int number_of_intersection = int.Parse(Console.ReadLine());
            for (int i = 0; i < number_of_intersection; i++)
            {
                Line = Console.ReadLine();
                spliter = Line.Split(' ');

                ID1 = int.Parse(spliter[0]);
                x1 = double.Parse(spliter[1]);
                y1 = double.Parse(spliter[2]);

                try { ID_getter[x1][y1] = ID1; }
                catch (Exception e){ ID_getter[x1] = new Dictionary<double, int>(); ID_getter[x1][y1]
= ID1; }

                Tuple<double, double> coordinate = new Tuple<double, double>(x1, y1);
                Coordinate_getter[ID1] = coordinate;

                all_nodes.Add(ID1);

            }
            Console.WriteLine("Enter Number of Road ?!..");
            int number_of_roads = int.Parse(Console.ReadLine());
            for (int i = 0; i < number_of_roads; i++)
            {
                Line = Console.ReadLine();
                spliter = Line.Split(' ');

                ID1 = int.Parse(spliter[0]);
                ID2 = int.Parse(spliter[1]);
                length = double.Parse(spliter[2]);
                volocity = double.Parse(spliter[3]);

                try { adj[ID1].Add(ID2); }
                catch(Exception e) { adj[ID1] = new List<int>(); adj[ID1].Add(ID2); }
                try { adj[ID2].Add(ID1); }
                catch (Exception e) { adj[ID2] = new List<int>(); adj[ID2].Add(ID1); }

                try { speed[ID1][ID2] = volocity; }
                catch(Exception e) { speed[ID1] = new Dictionary<int, double>(); speed[ID1][ID2] =
volocity; }
                try { speed[ID2][ID1] = volocity; }
                catch (Exception e) { speed[ID2] = new Dictionary<int, double>(); speed[ID2][ID1] =
volocity; }

                try { distance[ID1][ID2] = length; }
                catch (Exception e) { distance[ID1] = new Dictionary<int, double>();
distance[ID1][ID2] = length; }
                try { distance[ID2][ID1] = length; }
                catch (Exception e) { distance[ID2] = new Dictionary<int, double>();
distance[ID2][ID1] = length; }

            }
```

1. Read all Intersection Point that exists in a Map.
2. Read all Roads data and make for each intersection point adjacent list for all its neighbors and save both speed and length for each road in some dictionary.

* Dijkstra Algorithm *

- **Dijkstra Algorithm** is an Algorithm for finding Shortest Path Between -

1. One Source to Only One Destination.
2. One Source to all Destination Can Reached by This Node.
   USING PRIORITY_QUEUES.



Complexity for Algorithm: - **O(S E` log V`).**

- S: number of starting nodes
- V`: # of intersection points that are checked until reaching the destinations
- E`: # of roads that are checked until reaching the destinations

- **But we can minimize this complexity by decreasing its constant factor not its dominant factor.**
  i.e. (Minimizing # of Starting Nodes in our Algorithm).

Ex: - When Source is not Intersection Point and Destination is also not Intersection Point (HOW TO MINIMIZE COMPLEXITY)-?!

Solution: - Make # of starting nodes = min (# of intersection can i reach from Source on foot, # of intersection that from it can I reach to destination on foot).

-Sure I Can Reach form Node to Node On foot if Distance between this Nodes is Less than or Equal Radius that given.

- Dijkstra_Source_Code:-

```
static void dijkstra(int source)
        {
            shortest_path.Clear();
            parent.Clear();
            shortest_path[source] = 0;

            PriorityQueue pq = new PriorityQueue();
            Tuple<double, int> start = new Tuple<double, int>(0, source);
            pq.push(start);
            while(!pq.empty())
            {
                int cur = pq.top();
                pq.pop();
                for (int i = 0; i < adj[cur].Count; i++)
                {
                    int child = adj[cur][i];
                    double LocalMin;

                    if (child == source) continue;
                    try { LocalMin = shortest_path[child]; }
                    catch(Exception e) { shortest_path[child] = -1; }

                    double time = (double)distance[cur][child] / speed[cur][child];
                    if (shortest_path[child] == -1 || time +
shortest_path[cur]<shortest_path[child])
                    {
                        shortest_path[child] = time + shortest_path[cur];
                        parent[child] = cur;
                        Tuple<double, int> temp = new Tuple<double, int>(time +
shortest_path[cur], child);
                        pq.push(temp);
                    }

                }
            }
        }
```

- Greedy Solution.

- Greedy Choice:-For each step find Shortest Time to pass from it.

- ## Handling Cases.

1. **IF Source is intersection and destination is not.**
   (Dijkstra on Source and find node that from it can I reach to destination on foot with max R meters and with min time from Source.)
2. **IF Source is not intersection and destination is intersection.**
   (Dijkstra on Destination and find node that I can reach to it from Source on foot with max R meters and with min time to destination.)
3. **IF Source is intersection and destination is intersection.**
   (Dijkstra on Destination or Source and find shortest time directly.)
4. **IF Source is not intersection and destination is not intersection.**
   (Make # of starting nodes on Dijkstra algorithm = min (# of intersection can i reach from Source on foot, # of intersection that from it can I reach to destination on foot) and find shortest time form source to destination**).**

- ## Printing Path:-

  To print path we need to Backtrack for each node to its Parent.

Complexity: - O (V).

- ## Source Code:-

```
for (int j = 0; j < path.Count; j++)
{
    Console.Write(path[j]);
    Console.Write(" ");
}
Console.WriteLine();
```