TRAFFIC CONGESTION OPTIMIZATION USING GREEDY ALGORITHM

A MINI PROJECT REPORT

18CSC204J -Design and Analysis of AlgorithmsLaboratory

Submitted by

R.J AJITH SOWMYAN [RA2111003011410]

S. SURAJ [RA2111003011415]

Under the guidance of

Dr. P.SARANYA SURESH

Assistant Professor, Department of COMPUTING TECHNOLOGIES

In Partial Fulfillment of the Requirements for the Degree of

BACHELOR OF TECHNOLOGY

In

COMPUTER SCIENCE ENGINEERING

DEPARTMENT OF COMPUTING TECHNOLOGIES
COLLEGE OF ENGINEERING AND TECHNOLOGY
SRM INSTITUTE OF SCIENCE AND TECHNOLOGY
KATTANKULATHUR - 603 203

April 2023

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY
KATTANKULATHUR – 603 203

BONAFIDE CERTIFICATE

Certified that this is B.Tech mini project titled " **Traffic congestion optimization using greedy algorithm "**is the bonafide work of R.J AJITH SOWMYAN [RA2111003011410] and S.SURAJ [RA2111003011415] who carried out the project work under my supervision for **18CSC204J – DESIGN AND ANALYSIS OF ALGORITHMS LABARATORY .** Certified further, that do the best of my knowledge the work reported her in does not perform part of any other thesis or dissertation on the basis of which a degree or award was conferred on an earlier occasion for this or any other candidate.

DR. P.SARANYA SURESH
ASSISTANT PROFESSOR
Department of computing technologies

DR.PUSHPALATHA.M
Professor and head
Department of computing technologies

Problem: Traffic Congestion Optimization

PROBLEM STATEMENT:

You are given a map of a city with N intersections and M roads connecting them. Each road has a certain length and a certain traffic congestion level. You want to find the best route from your home to your office that minimizes the total travel time.

You can use Kruskal's algorithm to find the minimum spanning tree of the graph where the vertices are the intersections and the edges are the roads, weighted by their travel time. Alternatively, you can use Prim's algorithm directly on the original graph, by starting from your home and adding the edge with the smallest weight that connects to an unvisited vertex, until you reach your office. This will also give you the shortest path from your home to your office, but it may not be part of the minimum spanning tree of the graph.

About:

A Minimum Spanning Tree (MST) is a subset of the edges of a connected, edge-weighted undirected graph that connects all the vertices together, without any cycles and with the minimum possible total edge weight. That is, it is a spanning tree whose sum of edge weights is as small as possible.

Design Approaches:

- Kruskal's Algorithm
- Prim's Algorithm

Kruskal's Algorithm:

The main idea behind the Kruskal algorithm is to sort the edges based on their weight. After that, we start taking edges one by one based on the lower weight.

In case we take an edge, and it results in forming a cycle, then this edge isn't included in the MST (minimum spanning tree). Otherwise, the edge is included in the MST. Hence this is a **Greedy Algorithm**.

Algorithm steps:

- Sort the graph edges with respect to their weights.
- Start adding edges to the MST from the edge with the smallest weight until the edge of the largest weight.
- Only add edges which doesn't form a cycle, edges which connect only disconnected components.

Pseudocode:

```
MST_Kruskal (Edges, V, E):
 e = 0, i = 0
 sum = 0
 Sort (Edges)
 While (e<V-1):
   u = Edges[i].u
   v = Edges[i].v
   if (Adding edge {u, v} do not form cycle}:
     Print (Adding edge {u, v} to MST)
     sum+=Edges[i].weight
     e+=1
```

i+=1

CODE:

```
// Kruskal's Algorithm in C
#include <stdio.h>
#define MAX 30
typedef struct edge {
int u, v, w;
} edge;
typedef struct edge_list {
edge data[MAX];
int n;
} edge_list;
edge_list elist;
int Graph[MAX][MAX], n;
edge_list spanlist;
void kruskalAlgo();
int find(int belongs[], int vertexno);
void applyUnion(int belongs[], int c1, int c2);
void sort();
```

```
void print();

// Applying Krushkal Algo

void kruskalAlgo() {

int belongs[MAX], i, j, cno1, cno2;

elist.n = 0;

for (i = 1; i < n; i++)

for (j = 0; j < i; j++) {

if (Graph[i][j] != 0) {

elist.data[elist.n].u = i;

elist.data[elist.n].v = j;

elist.data[elist.n].w = Graph[i][j];

elist.n++;

}

}

sort();

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

belongs[i] = i;

spanlist.n = 0;

for (i = 0; i < elist.n; i++) {

cno1 = find(belongs, elist.data[i].u);

cno2 = find(belongs, elist.data[i].v);

if (cno1 != cno2) {
```

```
spanlist.data[spanlist.n] = elist.data[i];
spanlist.n = spanlist.n + 1;
applyUnion(belongs, cno1, cno2);
}
}
}
int find(int belongs[], int vertexno) {
return (belongs[vertexno]);
}
void applyUnion(int belongs[], int c1, int c2) {
int i;
for (i = 0; i < n; i++)
if (belongs[i] == c2)
belongs[i] = c1;
}
// Sorting algo
void sort() {
int i, j;
edge temp;
for (i = 1; i < elist.n; i++)
for (j = 0; j < elist.n - 1; j++)
if (elist.data[j].w > elist.data[j + 1].w) {
```

```
temp = elist.data[j];
elist.data[j] = elist.data[j + 1];
elist.data[j + 1] = temp;
}
}
// Printing the result
void print() {
int i, cost = 0;
for (i = 0; i < spanlist.n; i++) {
printf("\n%d - %d : %d", spanlist.data[i].u, spanlist.data[i].v, spanlist.data[i].w);
cost = cost + spanlist.data[i].w;
}
printf("\nSpanning tree cost: %d", cost);
}
int main() {
int i, j, total_cost;
n = 6;
Graph[0][0] = 0;
Graph[0][1] = 4;
Graph[0][2] = 4;
Graph[0][3] = 0;
Graph[0][4] = 0;
```

Graph[0][5] = 0;

Graph[0][6] = 0;

Graph[1][0] = 4;

Graph[1][1] = 0;

Graph[1][2] = 2;

Graph[1][3] = 0;

Graph[1][4] = 0;

Graph[1][5] = 0;

Graph[1][6] = 0;

Graph[2][0] = 4;

Graph[2][1] = 2;

Graph[2][2] = 0;

Graph[2][3] = 3;

Graph[2][4] = 4;

Graph[2][5] = 0;

Graph[2][6] = 0;

Graph[3][0] = 0;

Graph[3][1] = 0;

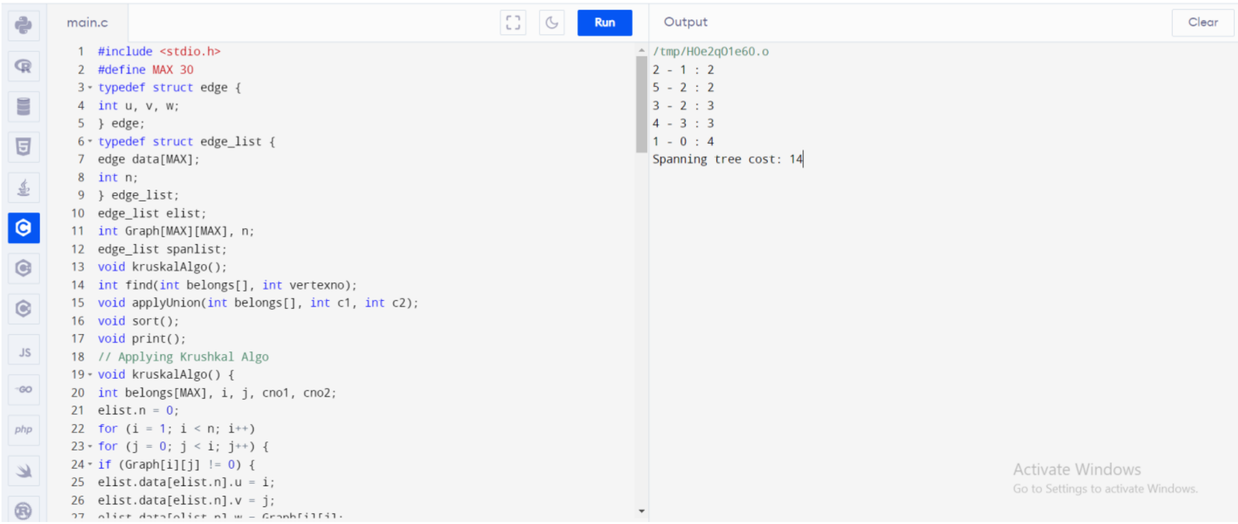
Graph[3][2] = 3;

Graph[3][3] = 0;

Graph[3][4] = 3;

Graph[3][5] = 0;

```
Graph[3][6] = 0;

Graph[4][0] = 0;

Graph[4][1] = 0;

Graph[4][2] = 4;

Graph[4][3] = 3;

Graph[4][4] = 0;

Graph[4][5] = 0;

Graph[4][6] = 0;

Graph[5][0] = 0;

Graph[5][1] = 0;

Graph[5][2] = 2;

Graph[5][3] = 0;

Graph[5][4] = 3;

Graph[5][5] = 0;

Graph[5][6] = 0;

kruskalAlgo();

print();

}
```



OUTPUT:

2 - 1 : 2

5 - 2 : 2

3 - 2 : 3

4 - 3 : 3

1 - 0 : 4

Spanning tree cost: 14

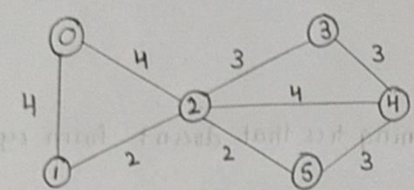
TIME COMPLEXITY:

In summary, Kruskal's algorithm requires-

- A worst-case time complexity of O (E log E).
- An average-case time complexity of O (E log E).
- A best-case time complexity of O (E log E).
- A space complexity of O (E+V).

IMPLEMENTATION OF KRUSKAL’S ALGORITHM:

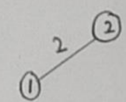
Example of Kruskal's Algorithm:



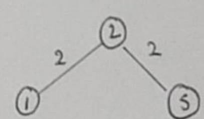
(i) sort the graph edges with respect to their weights

1 - 2 → 2
2 - 5 → 2
2 - 3 → 3
3 - 4 → 3
4 - 5 → 3
0 - 1 → 4
0 - 2 → 4
2 - 4 → 4

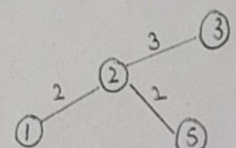
ii) choose the edge with the least weight, if they are more than 1, choose anyone



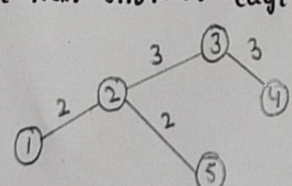
iii) choose the next shortest edge and add it

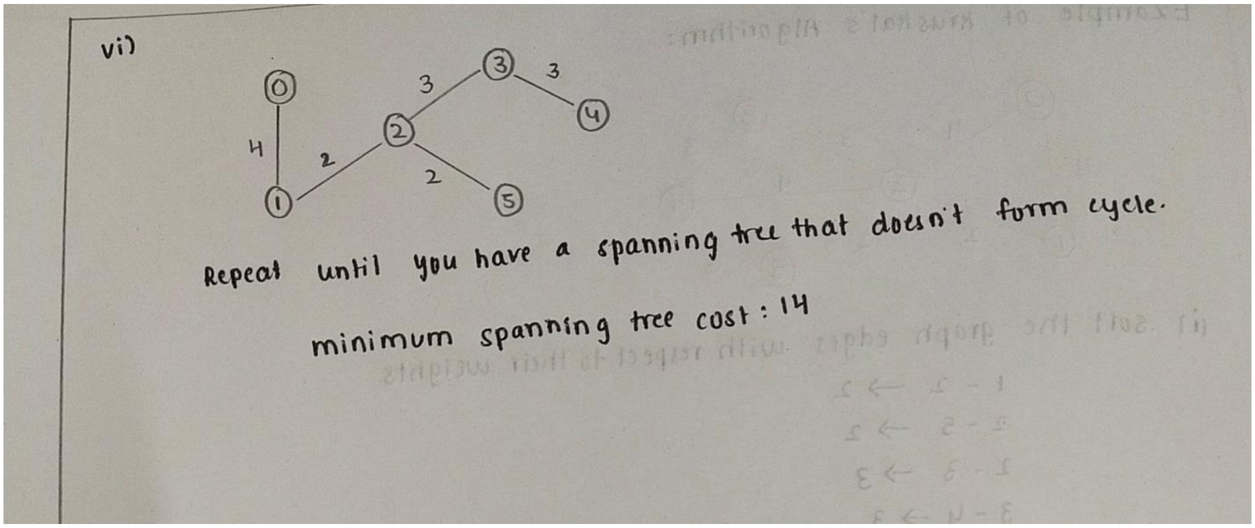


iv) choose the next shortest edge that doesn't create a cycle and add it



v) choose the next shortest edge that doesn't create a cycle and add it





PRIM'S Algorithm:

Prim's algorithm is another popular greedy algorithm used to find the minimum spanning tree of a connected, weighted, undirected graph. It works as follows: Choose an arbitrary vertex to start with and add it to the minimum spanning tree. Find the edge with the smallest weight that connects the minimum spanning tree to a vertex not yet in the minimum spanning tree.

Algorithm Steps:

- Prim's algorithm finds the subset of edges that includes every vertex of the graph such that the sum of the weights of the edges can be minimized.
- Prim's algorithm starts with the single node and explores all the adjacent nodes with all the connecting edges at every step.
- The edges with the minimal weights causing no cycles in the graph got selected.

Pseudocode:

Procedure prims

G-input graph

U-random vertex

V-vertices in graph G

begin

T= 0;

U = \{1\} while (U = V)

let (u, v) be the least cost edge such that u in U and v in V - U;

T=T cup \ {(u, v)\}

U=U cup \{v\}

end procedure

CODE:

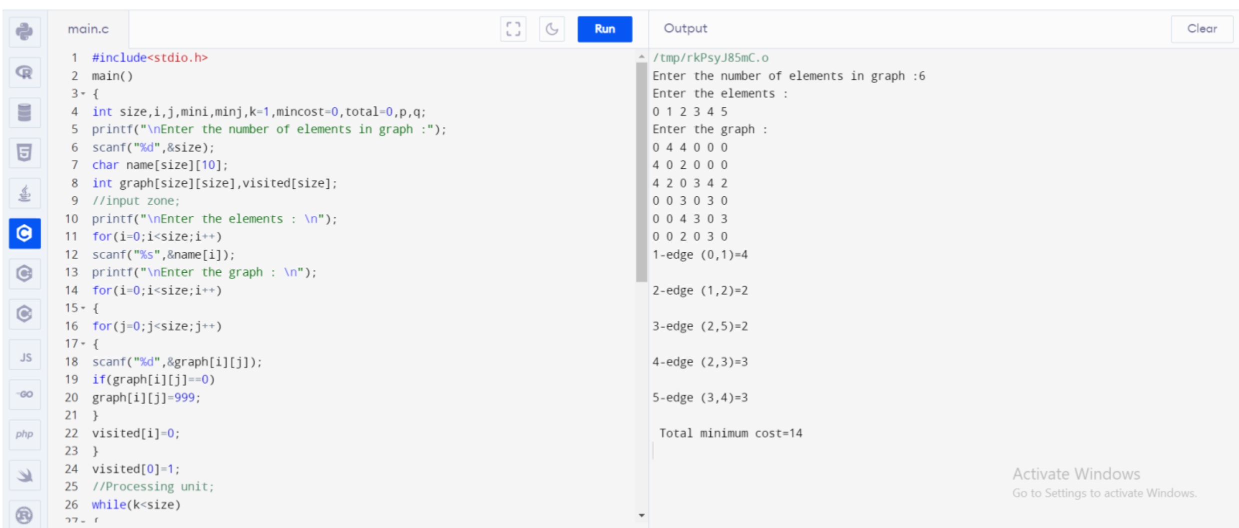
```
// Prim’s Algorithm in C

#include<stdio.h>

main()

{

int size,i,j,mini,minj,k=1,mincost=0,total=0,p,q;

printf("\nEnter the number of elements in graph :");

scanf("%d",&size);

char name[size][10];
```

```
int graph[size][size],visited[size];
//input zone;
printf("\nEnter the elements : \n");
for(i=0;i<size;i++)
scanf("%s",&name[i]);
printf("\nEnter the graph : \n");
for(i=0;i<size;i++)
{
for(j=0;j<size;j++)
{
scanf("%d",&graph[i][j]);
if(graph[i][j]==0)
graph[i][j]=999;
}
visited[i]=0;
}
visited[0]=1;
//Processing unit;
while(k<size)
{
```

```
   mini=0;
   minj=0;
for(i=0;i<size;i++)
{
   for(j=0;j<size;j++)
   {
      if(graph[i][j]<graph[mini][minj] && visited[i]!=0)
      {
         mini=i;
         minj=j;
         mincost=graph[i][j];

      }

   }
}
if(visited[mini]==0 || visited[minj]==0)
{
   printf("\n%d-edge
(%s,%s)=%d\n",k++,name[mini],name[minj],mincost);
```

```
    total+=mincost;
    visited[minj]=1;
}
graph[mini][minj]=999;
graph[minj][mini]=999;
}
printf("\n Total minimum cost=%d \n",total);
}
```

OUTPUT :



Enter the number of elements in graph: 6

Enter the elements:

0 1 2 3 4 5

Enter the graph:

0 4 4 0 0 0

4 0 2 0 0 0

4 2 0 3 4 2

0 0 3 0 3 0

0 0 4 3 0 3

0 0 2 0 3 0

1-edge (0, 1) =4

2-edge (1, 2) =2

3-edge (2, 5) =2

4-edge (2, 3) =3

5-edge (3, 4) =3

Total minimum cost=14

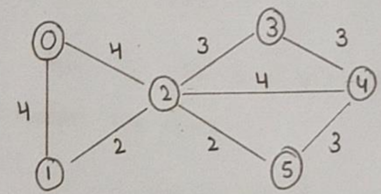
TIME COMPLEXITY:

In summary, Prims algorithm requires-

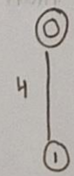
- A worst-case time complexity of O (V^2).
- An average-case time complexity of O (E + (V)log V).
- A best-case time complexity of O (V).
- A space complexity of O (E+V)

IMPLEMENTATION OF PRIM'S ALGORITHM:

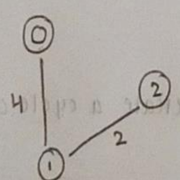
Example of prim's Algorithm:



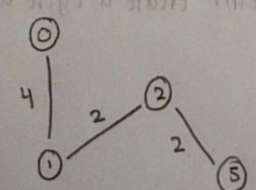
1. Start with node 0 and consider the cheapest weight to the edge from node



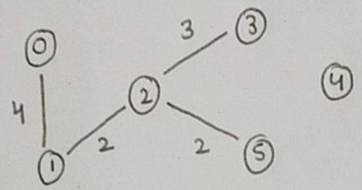
2. Add 1 and find the cheapest weight to the edge adjacent to that node



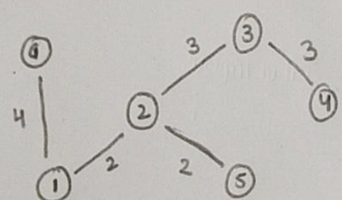
3. Now from node 2 and find the cheapest weight to the edge adjacent to that node and the graph should not form cycle



4. From the node (5, 4) the weight is 3 but from the node (2, 3) the weight is 3. but as 2 comes first we consider the node (2, 3)



5. consider the adjacent to the node which is having less weight.



this is the final minimum spanning tree which don't form any cycle between the node

minimum spanning tree cost = 14

Analysis:

The advantage of Prim’s algorithm is its complexity, which is better than Kruskal’s algorithm. Therefore, Prim’s algorithm is helpful when dealing with dense graphs that have lots of edges. However, Prim’s algorithm doesn’t allow us much control over the chosen edges when multiple edges with the same weight occur. The reason is that only the edges discovered so far are stored inside the queue, rather than all the edges like in Kruskal’s algorithm. Also, unlike Kruskal’s algorithm, Prim’s algorithm is a little harder to implement.

Conclusion:

The MST algorithm can be used to identify the most efficient routes for traffic flow by creating a tree that connects all the vertices in a graph with the minimum possible total weight. In the context of traffic congestion, this means that the algorithm can be used to identify the shortest and most direct routes for vehicles, which can help to reduce travel times and minimize traffic backups.

	KRUSKAL	**PRIM'S**
MULTIPLE MSTs	Offers a good control over the resulting MST	Controlling the MST might be a little harder
IMPLEMENTATION	Easier to implement	Harder to implement
REQUIREMENTS	Disjoint set	Priority queue
TIME COMPLEXITY	O(E.log(v))	O(E.log(v))

So hence Prims algorithm is more efficient and easier to find shortest path from your home to your office