1. No. this greedy strategy will not always find a shortest path from start to goal

Diagram, shape

Description automatically generated

When starting from start in the above graph, v will have the minimum weight with start, with the weight of 5 it will be selected in the path and the next time v will be taken as a starting vertex and the edge (v, goal) or weight 6 will be selected in the path. If such a pat is followed, then we have a weight of 5 + 6 = 11 which is >10. If we took the path from start to goal, we would have a weight of 10 which is less that 11 from the previous path. It is clear now that this algorithm will not always give the “greedy” results we want.

START

FUNCTION **minimumCostPath (int u, int destination, visited array [ ], graph G, bool prev\_edge)**

//check if we find the destination then further cost will be 0

if (u = destination)

return 0;

// mark the current node as visited

    visited[u] = 1

Initialize **ans to** Infinite value

// traverse through all the adjacent nodes

FOR all the adjacent vertex(node) of a vertex u

**If node** **is not visited then**

**If edge = red AND prev\_edge = true then**

**Do not continue this path**

**ELSE**

**continue to this path and calculate the cost of the further path**

CALL FUNCTION **minimumCostPath(node, destination, visited [ ], Graph, curr\_edge)**

And assign the value returned by this call to variable **cost**

IF **cost < INF then**

ans = Minimum of (ans, previous edge weight + current edge weight)

//Taking the minimum cost path

// unmarking the current node to make it available for other simple paths

​​​visited[u] = 0

END

Running time: O(E+V) e = edges and V = vertices

1. G(V.E); Graph, V-Vertices

E - Edges.

Void minDistance ( Graph \*G, int s)

priorityQueue \*pQ;

int v, w;

Enque (pQ, s);

for ( i <- o, i< g -> v; i++)

Distance [i] = INT\_MAX;

Distance[s] <- 0;

while (!isemplyQueue(pQ))

V= Deletemin ( PQ) ;

For (v to w)

d = Distance [v] + weight[v][w]

if (Distance[w] = INT\_MAX)

Distance [w] = d

enque (pQ, w}

path[w] = v

if (Distance[w] > d)

Distance [w] =d

update (pQ, w)

path[w] = v

Time Complexity: O(v2)

4.

Let G = (V, E) be the given graph, with | V| = n

    Start with a graph T = (V, phi) consisting of only the

    vertices of G and no edges;

    Arrange E in the order of increasing costs;

    for (i = 1, i to n - 1, i + +)

        Select the next smallest cost edge;

        if (the edge connects two different connected components)

        add the edge to T;