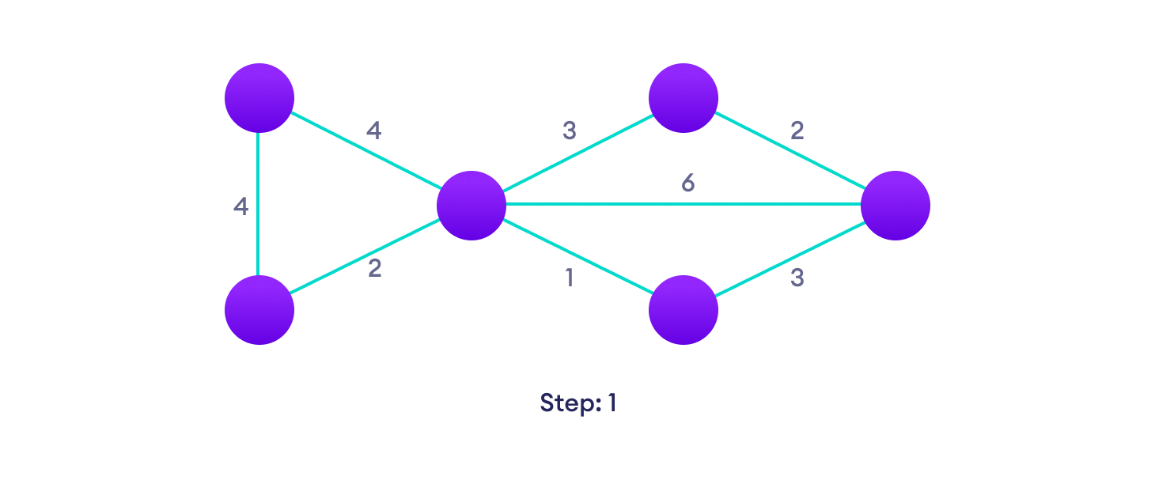
**TASK 9**

**Greedy Technique – Dijkstra’s algorithm**

It was Raj's first day at work place. His manager Anu asked him to meet every other branch and to deliver the goods. If there are n branches, then find the shortest distance from his branch to all other branches. Example:



**Output:**

Distance from source to 1: 3

Distance from source to 2: 1

Distance from source to 3: 2

Distance from source to 4: 4

Distance from source to 5: 4

**Test Case 1:** Show that the Dijkstra’s Algorithm cannot work for a weighted connected graph with negative weights.

**Test Case 2:** Analyze the correctness of Dijkstra’s algorithm for graphs with positive weights.

**Aim:**

To create a C program to find shortest path using dijkstra’s algorithm

**Algorithm**:

Step 1 : Create a set shortPath to store vertices that come in the way of the shortest path tree.

Step 2 : Initialize all distance values as INFINITE and assign distance values as 0 for source vertex so that it is picked first.

Step 3 : Loop until all vertices of the graph are in the shortPath.

Step 4 : Take a new vertex that is not visited and is nearest.

Step 5 : Add this vertex to shortPath.

Step 6 : For all adjacent vertices of this vertex update distances. Now check every adjacent vertex of V, if sum of distance of u and weight of edge is less the update it

**Program:**

#include <stdio.h>

#define INFINITY 9999

#define MAX 10

void Dijkstra(int Graph[MAX][MAX], int n, int start);

int hasNegativeEdge(int Graph[MAX][MAX], int n)

{

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

{

for (int j = 0; j < n; j++)

{

if (Graph[i][j] < 0)

{

return 1;

}

}

}

return 0;

}

void Dijkstra(int Graph[MAX][MAX], int n, int start)

{

if (hasNegativeEdge(Graph, n))

{

printf("Not applicable\n");

return;

}

int cost[MAX][MAX], distance[MAX], pred[MAX];

int visited[MAX], count, mindistance, nextnode, i, j;

// Creating cost matrix

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

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

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

cost[i][j] = INFINITY;

else

cost[i][j] = Graph[i][j];

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

{

distance[i] = cost[start][i];

pred[i] = start;

visited[i] = 0;

}

distance[start] = 0;

visited[start] = 1;

count = 1;

while (count < n - 1)

{

mindistance = INFINITY;

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

if (distance[i] < mindistance && !visited[i])

{

mindistance = distance[i];

nextnode = i;

}

visited[nextnode] = 1;

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

if (!visited[i])

if (mindistance + cost[nextnode][i] < distance[i])

{

distance[i] = mindistance + cost[nextnode][i];

pred[i] = nextnode;

}

count++;

}

// Printing the distance

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

if (i != start) {

printf("\nDistance from source to %d: %d", i, distance[i]);

}

}

int main() {

int Graph[MAX][MAX], i, j, n, u;

n = 7;

// Modified Graph with a Negative Weight (-2 at Graph[2][4])

int GraphData[MAX][MAX] = {

{0, 0, 1, 2, 0, 0, 0},

{0, 0, 2, 0, 0, 3, 0},

{1, 2, 0, 1, 3, 0, 0},

{2, 0, 1, 0, 0, 0, 1},

{0, 0, 3, 0, 0, 2, 0},

{0, 3, 0, 0, 2, 0, 1},

{0, 0, 0, 1, 0, 1, 0}

};

// Copy graph data

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

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

Graph[i][j] = GraphData[i][j];

u = 0;

Dijkstra(Graph, n, u);

return 0;

}

**Output:**

Distance from source to 1: 3

Distance from source to 2: 1

Distance from source to 3: 2

Distance from source to 4: 4

Distance from source to 5: 4

Distance from source to 6: 3

**Testcase 1:**

Enter no. of vertices:3

Enter the adjacency matrix:

0 1 -1

1 0 1

-1 1 0

Enter the starting node:0

Distance of 1 = 0

Path = 1 <-2 <-0

Distance of 2 = -1

Path = 2 <-0

Dijkstra's algorithm fails in graphs with negative weights because:

* It relies on a greedy approach, which can choose paths with negative weights incorrectly.
* It assumes convergence, but negative-weight cycles can lead to indefinite decrease in distances.
* It cannot maintain the invariant of the shortest path in the presence of negative weights.
* Its optimality proof doesn't hold with negative weights, potentially leading to suboptimal or incorrect paths.

**Test case :2**

Enter no. of vertices:4

Enter the adjacency matrix:

0 1 2 0

1 0 7 0

2 7 0 3

0 0 3 0

Enter the starting node:0

Distance of node1=1

Path=1<-0

Distance of node2=2

Path=2<-0

Distance of node3=5

Path=3<-2<-0

**Result:**

Thus the dijikstra’s algorithm was executed successfully.