/\*

**Dijkstra’s shortest path algorithm | Greedy Algorithm**

Given a graph and a source vertex in the graph, find shortest paths from source to all vertices in the given graph.

Dijkstra’s algorithm is very similar to Prim’s algorithm for minimum spanning tree. Like Prim’s MST, we generate a SPT (shortest path tree) with given source as root. We maintain two sets, one set contains vertices included in shortest path tree, other set includes vertices not yet included in shortest path tree. At every step of the algorithm, we find a vertex which is in the other set (set of not yet included) and has a minimum distance from the source.

Below are the detailed steps used in Dijkstra’s algorithm to find the shortest path from a single source vertex to all other vertices in the given graph.

**Algorithm**

1) Create a set sptSet (shortest path tree set) that keeps track of vertices included in shortest path tree, i.e., whose minimum distance from source is calculated and finalized. Initially, this set is empty.

2) Assign a distance value to all vertices in the input graph. Initialize all distance values as INFINITE. Assign distance value as 0 for the source vertex so that it is picked first.

3) While sptSet doesn’t include all vertices

….a) Pick a vertex u which is not there in sptSet and has minimum distance value.

….b) Include u to sptSet.

….c) Update distance value of all adjacent vertices of u. To update the distance values, iterate through all adjacent vertices. For every adjacent vertex v, if sum of distance value of u (from source) and weight of edge u-v, is less than the distance value of v, then update the distance value of v.



The set sptSet is initially empty and distances assigned to vertices are {0, INF, INF, INF, INF, INF, INF, INF} where INF indicates infinite. Now pick the vertex with minimum distance value. The vertex 0 is picked, include it in sptSet. So sptSet becomes {0}. After including 0 to sptSet, update distance values of its adjacent vertices. Adjacent vertices of 0 are 1 and 7. The distance values of 1 and 7 are updated as 4 and 8. Following subgraph shows vertices and their distance values, only the vertices with finite distance values are shown. The vertices included in SPT are shown in green colour.



Pick the vertex with minimum distance value and not already included in SPT (not in sptSET). Vertex 7 is picked. So sptSet now becomes {0, 1, 7}. Update the distance values of adjacent vertices of 7. The distance value of vertex 6 and 8 becomes finite (15 and 9 respectively).



Pick the vertex with minimum distance value and not already included in SPT (not in sptSET). Vertex 6 is picked. So sptSet now becomes {0, 1, 7, 6}. Update the distance values of adjacent vertices of 6. The distance value of vertex 5 and 8 are updated.



We repeat the above steps until sptSet doesn’t include all vertices of given graph. Finally, we get the following Shortest Path Tree (SPT).



How to implement the above algorithm?

We use a boolean array sptSet[] to represent the set of vertices included in SPT. If a value sptSet[v] is true, then vertex v is included in SPT, otherwise not. Array dist[] is used to store shortest distance values of all vertices.

**code:**

//https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/

// A C++ program for Dijkstra's single source shortest path algorithm for Undirected graph.

// The program is for adjacency matrix representation of the graph

#include<stdio.h>

#include<limits.h> /\* this header file deals with sizes of datatypes \*/

#include<stdbool.h>

//#define V 9 /\* for static input mentioned in main() \*/

int V;

void dijkstra(int [][V],int);

int minDistance(\_Bool [],int[]);

void printSolution(int[]);

int main()

{

int n,i,j;

printf("enter no of vertices : ");

scanf("%d",&V);

int graph[V][V];

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

{

for(j=i;j<V;j++)

{

if(i==j) { graph[i][j]=0; continue; } /\* self \*/

printf("enter weight for edge(%d,%d) :",i,j);

scanf("%d",&graph[i][j]);

graph[j][i]=graph[i][j];

}

}

// or

/\* int graph[V][V]={{0, 4, 0, 0, 0, 0, 0, 8, 0},{4, 0, 8, 0, 0, 0, 0, 11, 0},{0, 8, 0, 7, 0, 4, 0, 0, 2},{0, 0, 7, 0, 9, 14, 0, 0, 0},

{0, 0, 0, 9, 0, 10, 0, 0, 0},{0, 0, 4, 14, 10, 0, 2, 0, 0},{0, 0, 0, 0, 0, 2, 0, 1, 6},

{8, 11, 0, 0, 0, 0, 1, 0, 7},{0, 0, 2, 0, 0, 0, 6, 7, 0}};

\*/

printf("select a source node(0-%d)",V-1);

scanf("%d",&n);

dijkstra(graph,n);

return 0;

}

void dijkstra(int graph[V][V],int src)

{

int dist[V],i,j; /\* holds the distances from source to respective vertex \*/

\_Bool sptSet[V]; /\* holds the vertices whose minimum distances from source are finalized \*/

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

{

dist[i]=INT\_MAX; /\*initilizing all distances to infinity \*/

sptSet[i]=false;

/\*initialising all vertices in set to false as distance from source to respective vertex is not finalized \*/

}

dist[src]=0; /\* src distance(itself) is zero \*/

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

{

int u=minDistance(sptSet,dist); /\* finding adjacent vertex with minimum distance which is not in sptSet \*/

sptSet[u]=true; /\* mark the u vertex as processed \*/

/\* updating the distances of adjacent vertices of processed vertex \*/

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

{

/\* sptSet[j]==false --> the vertex should not be part of finalised distance of vertex from source (sptSet)

graph[u][j]!=0 --> the vertex j should be connected with vertex u

dist[u]!=INT\_MAX --> for the source vertex as all its adjacent vertices is infinity at first

dist[u]+graph[u][j]<dist[v] ---> checking least distance of vertex j from source as of now

\*/

if(sptSet[j]==false && graph[u][j]!=0 && dist[u]!=INT\_MAX && dist[u]+graph[u][j]< dist[j])

{

dist[j]=dist[u]+graph[u][j];

/\* assigning least distance from source as of now \*/

}

}

//printSolution(dist);

/\*if you want to see step by step procedure uncomment above line\*/

}

printSolution(dist);

}

int minDistance(\_Bool sptSet[],int dist[])

{

int min=INT\_MAX,min\_index,i; /\* INT\_MAX is highest value for integer according from <limits.h> \*/

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

{

if(sptSet[i]==false && dist[i]<=min)

{

min=dist[i]; min\_index=i;

}

}

return min\_index; /\* returning the index of adjacent vertex which as least minimum distance \*/

}

void printSolution(int dist[])

{

int i;

printf("distances from source vertex :\n");

printf("vertex | distance \n");

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

printf("%d\t\t%d\n",i,dist[i]);

}