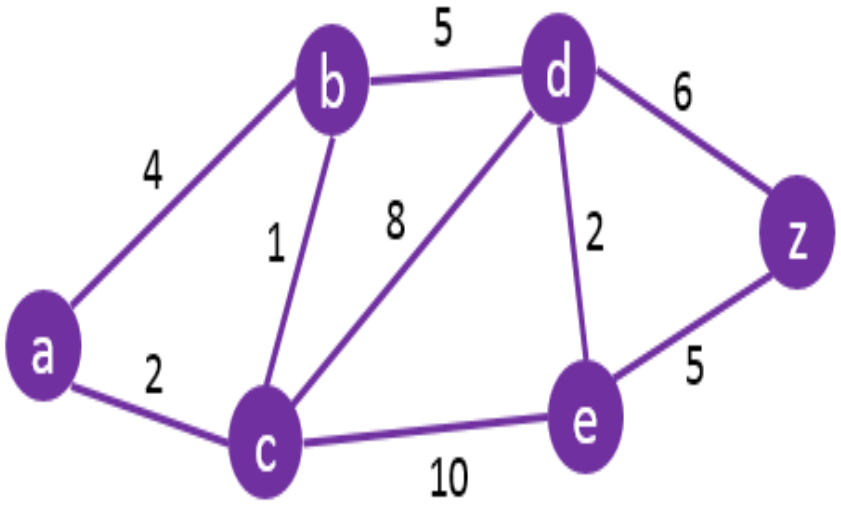
Map Navigation Using The Dijkstra's Algorithm



Dijkstra’s Algorithm

What is the shortest path to travel from A to Z?

Introduction

Dijkstra's algorithm is a widely used algorithm in computer science and graph theory for finding the shortest path between nodes in a graph, which may represent, for example, road networks, computer networks, or any other system of interconnected nodes. The goal of Dijkstra's algorithm is to determine the shortest path from a specified starting node to all other nodes in the graph. It works by iteratively selecting the node with the smallest tentative distance (initially set to infinity) from the source node and updating the distances of its neighboring nodes.

Step-by-step overview of Dijkstra's algorithm:

1.Initialization: Set the distance to the source node as 0 and the distances to all other nodes as infinity. Create an empty set to store the vertices whose shortest distance from the source is known.
2.Selection: Choose the node with the smallest tentative distance from the set of nodes not yet processed. Initially, this will be the source node.
3.Update: For the selected node, update the tentative distances of its neighboring nodes. If the sum of the tentative distance from the source to the selected node and the edge weight to a neighboring node is smaller than the current tentative distance of that neighboring node, update it.
4.Mark as Processed: Mark the selected node as processed by adding it to the set of nodes whose shortest distance from the source is known.
5.Repeat: Repeat steps 2-4 until all nodes are processed.
6.Result: The final distances represent the shortest paths from the source node to all other nodes.

Code:

```c
#include <stdio.h>
#include <limits.h>

#define V 9 // Number of vertices in the graph

// Function to find the vertex with the minimum distance value
// from the set of vertices not yet included in the shortest
// path tree
int minDistance(int dist[], int sptSet[]) {
    int min = INT_MAX, min_index;

    for (int v = 0; v < V; v++) {
        if (sptSet[v] == 0 && dist[v] <= min) {
            min = dist[v];
            min_index = v;
        }
    }

    return min_index;
}

// Function to print the final solution
void printSolution(int dist[], int n) {
    printf("Vertex   Distance from Source\n");
    for (int i = 0; i < V; i++)
        printf("%d \t\t %d\n", i, dist[i]);
}

// Function to implement Dijkstra's algorithm for a given graph
void dijkstra(int graph[V][V], int src) {
    int dist[V];     // The output array dist[i] holds the shortest distance
                     from src to i
    int sptSet[V];   // sptSet[i] will be true if vertex i is included in the
                     shortest
                     // path tree or the shortest distance from src to i is
                        finalized

    // Initialize all distances as INFINITE and sptSet[] as false
    for (int i = 0; i < V; i++) {
        dist[i] = INT_MAX;
        sptSet[i] = 0;
    }

    // Distance of source vertex from itself is always 0
    dist[src] = 0;

    // Find shortest path for all vertices
    for (int count = 0; count < V - 1; count++) {
        // Pick the minimum distance vertex from the set of vertices not
        // yet processed. u is always equal to src in the first iteration.
        int u = minDistance(dist, sptSet);

        // Mark the picked vertex as processed
        sptSet[u] = 1;

        // Update dist value of the adjacent vertices of the picked vertex
        for (int v = 0; v < V; v++) {
            // Update dist[v] only if it is not in the sptSet, there is an
            // edge from u to v, and the total weight of path from src to
            // v through u is less than the current value of dist[v]
            if (!sptSet[v] && graph[u][v] && dist[u] != INT_MAX &&
                dist[u] + graph[u][v] < dist[v]) {
                dist[v] = dist[u] + graph[u][v];
            }
        }
    }

    // Print the constructed distance array
    printSolution(dist, V);
}

// Driver program to test the above functions
int main() {
    // Example graph represented as an adjacency matrix
    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}
    };

    dijkstra(graph, 0);

    return 0;
}
```

Description of the code:

- min distance () function: This function identifies the vertex with the minimum distance value among those not yet included in the shortest path tree.

- printSolution() function: This function displays the final shortest distances from the source node to all other nodes.

- dijkstra() function: The Dijkstra function takes an adjacency matrix (graph) representing the weighted edges between vertices and the source vertex (src) as input.
  - Initializes arrays dist (to store the shortest distances) and sptSet (to keep track of included vertices).
  - Initializes distances to all vertices as INT_MAX except the source vertex, which is set to 0.
  - The algorithm iteratively selects the vertex with the minimum distance (u) from the set of vertices not processed.
  - Marks u as processed (sptSet[u] = 1) and updates the distances to its adjacent vertices if a shorter path is found.
  - The process is repeated until all vertices are included in the shortest path tree.

THANK YOU!