MIT Art Design and Technology University

MIT School of Computing, Pune

Department of Computer Science and Engineering

Second Year B. Tech

Academic Year 2022-2023. (SEM-II)

Subject: Advance Data Structures Laboratory

Assignment 6

Assignment Title: Represent a given graph using an adjacency list or array and find the shortest path using Dijkstra's algorithm.

Aim: Implement Graph and Dijkstra's algorithm to find the shortest path.

Prerequisite:

1. Basic concepts of Graph.
2. Dijkstra's algorithm of Graph to find the shortest path

Objectives:

Implement a Program in C++ for the following operations on Graph:

1. Create a Graph of N nodes using Adjacency Matrix/adjacency list.
2. Recognize and define the basic attributes of a Graph.
3. Print shortest path from given single source to all vertices in graph using Dijkstra's algorithm.

Outcomes:

Upon Completion of the assignment the students will be able to

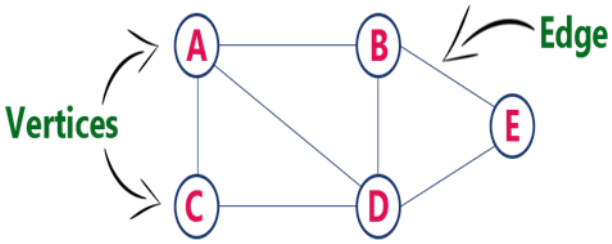
1. Create and implement Graph using adjacency list/adjacency matrix.
2. Understand and analyse Dijkstra's algorithm.

Theory:

Graph: Graph is a non-linear data structure. It contains a set of points known as nodes (or vertices) and a set of links known as edges (or Arcs). Here edges are used to connect the vertices.

Generally, a graph G is represented as G = (V , E), where V is set of vertices and E is set of edges.

Example:



The figure shows is a graph with 5 vertices and 6 edges.

This graph G can be defined as G = (V , E)

Where V = {A,B,C,D,E} and

E = {(A,B),(A,C)(A,D),(B,D),(C,D),(B,E),(E,D)}.

Storage representation:

1. **Adjacency Matrix**
2. **Adjacency list**

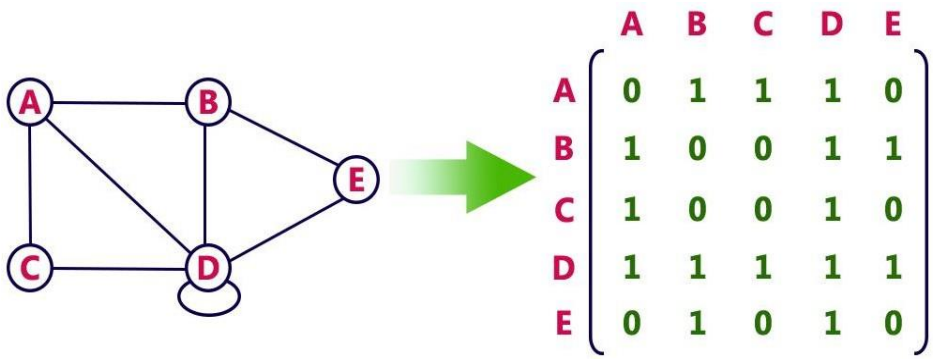
1.Adjacency Matrix

Let G = (V,E) be a graph with 'v' vertices v >= 1.

The adjacency matrix of graph G is a 2-dimensional n x n array say A[n:n]., with the property that,

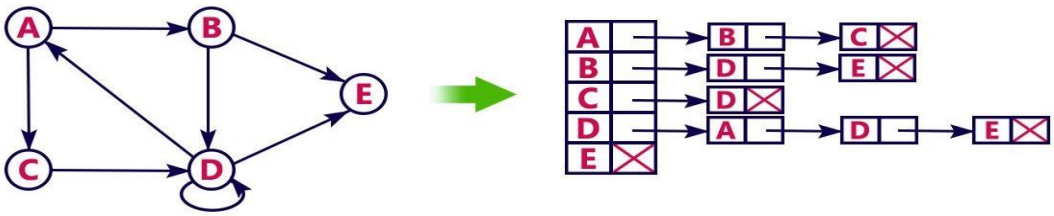
A(i, j) = 1 iff the edge (Vi, Vj) is in E(G) for undirected graph or the edge <Vi, Vj> is in E(G) for directed graph and

A(i, j) = 0 if there is no such edge in graph G from vertex Vi to Vj.



2. Adjacency list

In this representation the ‘n’ rows of adjacency matrix are represented as ‘n’ linked lists. There is one list for each vertex in a graph. The nodes in list i represent the vertices that are adjacent to vertex i.



Dijkstra’s algorithm:

Dijkstra's Algorithm basically starts at the node that you choose (the source node) and it analyzes the graph to find the shortest path between that node and all the other nodes in the graph.

The algorithm keeps track of the currently known shortest distance from each node to the source node and it updates these values if it finds a shorter path.

Once the algorithm has found the shortest path between the source node and another node, that node is marked as "visited" and added to the path.

The process continues until all the nodes in the graph have been added to the path. This way, we have a path that connects the source node to all other nodes following the shortest path possible to reach each node.

Algorithm

1. Mark the source node with a current distance of 0 and the rest with infinity.
2. Set the non-visited node with the smallest current distance as the current node, lets say C.
3. For each neighbor N of the current node C: add the current distance of C with the weight of the edge connecting C-N. If it is smaller than the current distance of N, set it as the new current distance of N.
4. Mark the current node C as visited.
5. Go to step 2 if there are any nodes are unvisited.

Dijkstra Algorithm Time and Space Complexity

Complexity analysis for Dijkstra's algorithm with adjacency matrix representation of graph.

Time complexity of Dijkstra's algorithm is **O(V2)** where **V** is the number of vertices in the graph.

It can be explained as below:

First thing we need to do is find the unvisited vertex with the smallest path. For that we require O(V) time as we need check all the vertices.

Now for each vertex selected as above, we need to relax its neighbors which means to update each neighbor's path to the smaller value between its current path or to the newly found. The time required to relax one neighbor comes out to be of order of **O (1)** (constant time).

For each vertex we need to relax all of its neighbors, and a vertex can have at most V-1 neighbors, so the time required to update all neighbors of a vertex comes out to be [O(V) * O(1)] = O(V)

So now following the above conditions, we get:

Time for visiting all vertices =O(V)

Time required for processing one vertex=O(V)

Time required for visiting and processing all the vertices = O(V)*O(V) =O(V^2)

So the time complexity of dijkstra's algorithm using adjacency matrix representation comes out to be **O(V2)O(V2)**

Space complexity of adjacency matrix representation of graph in the algorithm is also **O(V2)** as a V*V matrix is required to store the representation of the graph. An additional array of V length will also be used by the algorithm to maintain the states of each vertex but the total space complexity will remain **O(V2)**.

The time complexity of dijkstra's algorithm can be reduced to **O((V+E) log V)** using adjacency list representation of the graph and a min-heap to store the unvisited vertices, where E is the number of edges in the graph and V is the number of vertices in the graph.

With this implementation, the time to visit each vertex becomes **O(V+E)** and the time required to process all the neighbors of a vertex becomes **O(log V)**.

Time for visiting all vertices =O(V+E)

Time required for processing one vertex=O(logV)

Time required for visiting and processing all the vertices = O(V+E)*O(logV) = O((V+E)logV)

The space complexity in this case will also improve to **O(V)** as both the adjacency list and min-heap require **O(V)** space. So the total space complexity becomes

$O(V)+O(V)=O(2V) = O(V)$

Conclusion:

Dijkstra is an uninformed algorithm - it should be used when you have no knowledge on the graph, and cannot estimate the distance from each node to the target.

Frequently asked questions:

1. Write Dijkstra’s algorithm.
2. How do you solve Dijkstra's problem? Give one example
3. What is the time and space complexity of Dijkstra algorithm?

MIT SCHOOL OF ENGINEERING
Rajbaug, Loni-Kalbhor, Pune

MIT-ADT UNIVERSITY
PUNE, INDIA

Assignment - 6

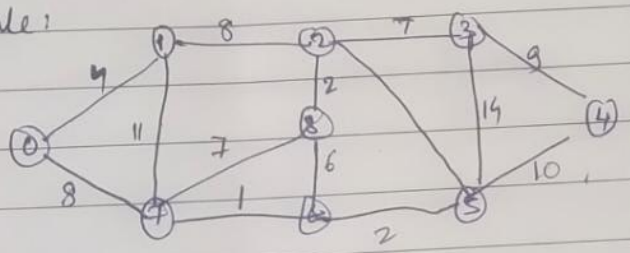
1) Write Dijkstra's algorithm ?

Ans - Create a set of unvisited node and set the distance of each node to infinity.
- Set the distance of the starting node to 0 and add it to the set of unvisited node.
- While the set of unvisited node is not empty:
  - Select the node with smallest tentative distance from the set of unvisited node
  - For each neighbour of visited nodes:
    - Calculate the distance from the starting node to the neighbour node via selected node.
    - If the tentative distance is less than the current distance of the neighbour node, update its distance.
  - Mark the selected node as visited and remove it from the set of unvisited nodes.
- Return the shortest distance to the target node.

2) How do you solve Dijsktra's problem ? Give an example.

Ans To solve a problem using Dijsktra's algorithm, you need to represent a problem as a graph, where the nodes represented states or locations and the edges represent the transition or path between them along with their weights and costs.

Example:



Source = 0.

The minimum distance from 0 to 1 = 4, 0 → 1
The minimum distance from 0 to 2 = 12, 0 → 7 → 2
The minimum distance from 0 to 3 = 19, 0 → 1 → 2 → 3
The minimum distance from 0 to 4 = 21, 0 → 7 → 6 → 5 → 4
The minimum distance from 0 to 5 = 11, 0 → 7 → 6 → 5
The minimum distance from 0 to 6 = 9, 0 → 7, → 6
The minimum distance from 0 to 7 = 8, 0 → 7
The minimum distance from 0 to 8 = 14, 0 → 1, → 2 → 8

3] What is the time and space complexity of Dijkstra's algorithm?

Ans Time complexity : O ((E + V) log V)
space complexity : O (V).

Code:

```cpp
#include <iostream>
#include <climits>

using namespace std;

int minDistance(int dist[], bool visited[], int V) {
    int minDist = INT_MAX, minIndex;
    for (int i = 0; i < V; i++) {
        if (!visited[i] && dist[i] <= minDist) {
            minDist = dist[i];
            minIndex = i;
        }
    }
    return minIndex;
}

void dijkstra(int graph[][100], int start, int V) {
    int dist[V];
    bool visited[V];

    for (int i = 0; i < V; i++) {
        dist[i] = INT_MAX;
        visited[i] = false;
    }

    dist[start] = 0;

    for (int i = 0; i < V - 1; i++) {
        int u = minDistance(dist, visited, V);
        visited[u] = true;

        for (int v = 0; v < V; v++) {
            if (!visited[v] && graph[u][v] && dist[u] != INT_MAX && dist[u]
+ graph[u][v] < dist[v]) {
                dist[v] = dist[u] + graph[u][v];
            }
        }
    }
```

```
    cout << "Vertex\tDistance from start\n";
    for (int i = 0; i < V; i++) {
        cout << i << "\t\t" << dist[i] << endl;
    }
}


int main() {
    int V, start;
    cout << "Enter the number of vertices: ";
    cin >> V;
    int graph[100][100];


    cout << "Enter the adjacency matrix:\n";
    for (int i = 0; i < V; i++) {
        for (int j = 0; j < V; j++) {
            cin >> graph[i][j];
        }
    }


    cout << "Enter the starting vertex: ";
    cin >> start;


    dijkstra(graph, start, V);


    return 0;
}
```

Output:

```
PS C:\SOURAV\CODE\C++ language codes\ADS assignment> cd "c:\SOURAV\CODE\C++ language codes\ADS assi
Enter the number of vertices: 4
Enter the adjacency matrix:
3
2
1
3
1
2
3
4
5
3
2
1
5
3
7
8
Enter the starting vertex: 3
Vertex  Distance from start
0               4
1               3
2               5
3               0
PS C:\SOURAV\CODE\C++ language codes\ADS assignment>
```