Department of
Computer Science and Engineering

Title: Finding Shortest Path using Dijkstra's Algorithm

Algorithms Lab

CSE 206

Green University of Bangladesh

1 Objective(s)

- To understand the shortest path of a graph.
- To implement Dijkstra's algorithm for finding the shortest path of a graph and analyse it.

2 Problem analysis

Given a graph and a source vertex in the graph, we have to 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.

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*** does not include all vertices
   - Pick a vertex u which is not there in ***sptSet*** and has the minimum distance value among all nodes in the set.
   - Include u to ***sptSet***.
   - Update distance value of all adjacent vertices of u. To update the distance values, iterate through all adjacent vertices each is denoted as v. 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. This step is known as ***Relaxation***.

2.1 Time Complexity

Let N and E be the numbers of vertices and edges respectively. The *while* loop in line 6 of the algorithm will run N times, giving O(N). Extracting the minimum distance node in line 7 gives O(log N). The *for* loop running in line 9 and for the update of distance value gives O(E log N). So total running time is O(N log N + E log N) = O(E log N).

3 Algorithm

Algorithm 1: Dijkstra (Graph G, all weights w, source vertex s)

Input: Directed or undirected weighted graph

```
/* Dijkstra's algorithm for single source shortest path                    */
```

```
Initialize sptSet[] as empty
for each vertex of G do
    distance, d[v] = infinite
end
d[s] = 0
while sptSet[] does not include all vertex from G do
    select u not in sptSet[] and has minimum distance from source s
    insert u into sptSet[]
    for each vertex v as adjacent of u do
        if current distance, d[v] > distance, d[u] + weight, w(u,v) then
            d[v] = d[u] + weight, w(u,v)
        end
    end
end
```

4 Implementation in Java

```java
// A Java program for Dijkstra's single source shortest path algorithm.
// The program is for adjacency matrix representation of the graph

import java.util.*;
import java.lang.*;
import java.io.*;

class ShortestPath {
  // A utility function to find the vertex with minimum distance value,
  // from the set of vertices not yet included in shortest path tree
  static final int V = 5;
  int minDistance(int dist[], Boolean sptSet[])
  {
    // Initialize min value
    int min = Integer.MAX_VALUE, min_index = -1;

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

    return min_index;
  }

  // A utility function to print the constructed distance array
  void printSolution(int dist[])
  {
    System.out.println("Vertex \t\t Distance from Source");
    for (int i = 0; i < V; i++)
      System.out.println(i + " \t\t " + dist[i]);
  }

  // Function that implements Dijkstra's single source shortest path
```

```
35  // algorithm for a graph represented using adjacency matrix
36  // representation
37  void dijkstra(int graph[][], int src)
38  {
39    int dist[] = new int[V]; // The output array. dist[i] will hold
40    // the shortest distance from src to i
41
42    // sptSet[i] will true if vertex i is included in shortest
43    // path tree or shortest distance from src to i is finalized
44    Boolean sptSet[] = new Boolean[V];
45
46    // Initialize all distances as INFINITE and stpSet[] as false
47    for (int i = 0; i < V; i++) {
48      dist[i] = Integer.MAX_VALUE;
49      sptSet[i] = false;
50    }
51
52    // Distance of source vertex from itself is always 0
53    dist[src] = 0;
54
55    // Find shortest path for all vertices
56    for (int count = 0; count < V - 1; count++) {
57      // Pick the minimum distance vertex from the set of vertices
58      // not yet processed. u is always equal to src in first
59      // iteration.
60      int u = minDistance(dist, sptSet);
61
62      // Mark the picked vertex as processed
63      sptSet[u] = true;
64
65      // Update dist value of the adjacent vertices of the
66      // picked vertex.
67      for (int v = 0; v < V; v++)
68
69        // Update dist[v] only if is not in sptSet, there is an
70        // edge from u to v, and total weight of path from src to
71        // v through u is smaller than current value of dist[v]
72        if (!sptSet[v] && graph[u][v] != 0 && dist[u] != Integer.MAX_VALUE &&
              dist[u] + graph[u][v] < dist[v])
73          dist[v] = dist[u] + graph[u][v];
74    }
75
76    // print the constructed distance array
77    printSolution(dist);
78  }
79
80  // Driver method
81  public static void main(String[] args)
82  {
83    /* Let us create the example graph discussed above */
84    int graph[][] = new int[][] { { 0, 10, 5, 0, 0 },
85                    { 0, 0, 2, 1, 0 },
86                    { 0, 3, 0, 9, 2 },
87                    { 0, 0, 0, 0, 4 },
88                    { 7, 0, 0, 6, 0 }};
89    ShortestPath t = new ShortestPath();
90    t.dijkstra(graph, 0);
91  }
```

```
92 }
```

5 Sample Input/Output (Compilation, Debugging & Testing)

6 Discussion & Conclusion

Based on the focused objective(s) to understand finding the shortest path to every vertices from a single source using Dijkstra's algorithm, the additional lab exercise will increase confidence towards the fulfilment of the objectives(s).

7 Lab Task (Please implement yourself and show the output to the instructor)

1. Prove that, if there exists negative edge, Dijkstra's shortest path algorithm may fail to find the shortest path. Please add Bellman Ford Algorithm
2. Print the shortest path for Dijkstra's algorithm
3. Find out how many shortest paths are there from source to destination

8 Lab Exercise (Submit as a report)

- Suppose you are given a graph where each edge represents the path cost and each vertex has also a cost which represents that, if you select a path using this node, the cost will be added with the path cost. Implement this graph using Dijkstra's algorithm.

9 Policy

Copying from internet, classmate, seniors, or from any other source is strongly prohibited. 100% marks will be *deducted* if any such copying is detected.