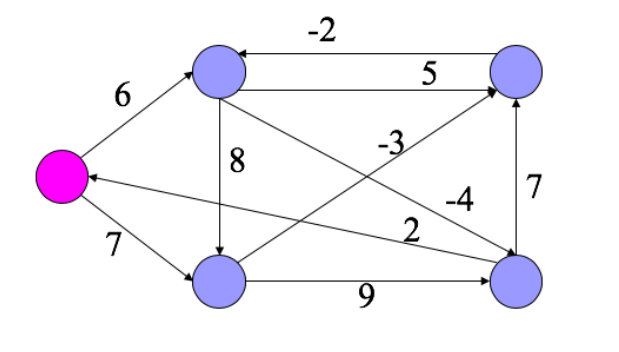
3805ICT Advanced Algorithms – Assignment 1

Chris Vuong – s5193954

**Question 2:** Write a C++ program that uses the Bellman-Ford algorithm to find the

shortest paths from the pink node to all other nodes:



B1

A0

C2

D3

E4

**Solution:** This is the solution that explains step by steps for solving the problem.

As we know, the shortest path of A to all other vertexes must be less than the number of vertex -1.

Let’s make a table of each V(vertex):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| A | B | C | D | E |
| 0 | ∞ | ∞ | ∞ | ∞ |

Go through the edges A-B we need 6pts:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| A | B | C | D | E |
| 0 | 6 | ∞ | ∞ | ∞ |

Go through the edges B-C we need 6+5 = 11 pts, we have A-B-C:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| A | B | C | D | E |
| 0 | 6 | 11 | ∞ | ∞ |

Go through the edges C-B we need 11+(-2) = 9 pts but more than the weight of 6, it is omitted:

Go through the edges B-D we need 6+(-4) = 2 pts, we have A-B-D:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| A | B | C | D | E |
| 0 | 6 | 11 | 2 | ∞ |

Go through the edges D-C we need 2+7 = 9 pts, we have A-B-D-C:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| A | B | C | D | E |
| 0 | 6 | 9 | 2 | ∞ |

Go through the edges B-E we need 6+8 = 14 pts, we have A-B-E:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| A | B | C | D | E |
| 0 | 6 | 9 | 2 | 14 |

Go through the edges A-E we need 0+7 = 7 pts, and 7 is less than 14 ,we have A-E:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| A | B | C | D | E |
| 0 | 6 | 9 | 2 | 7 |

Go through the edges A-E we need 0+7 = 7 pts, and 7 is less than 14 , we have A-E:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| A | B | C | D | E |
| 0 | 6 | 9 | 2 | 7 |

Go through the edges E-D we need 7+9 = 16 pts, and 16 is more than 2 , we omit it.

Go through the edges E-C we need 7+(-3) = 4 pts, and 4 is less than 9 , we have A-E-C.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| A | B | C | D | E |
| 0 | 6 | 4 | 2 | 7 |

We have done the first round of finding the path from A to other vertexes, but it is not the shortest path. For a faster solution, let’s omit some of the edges that does not change our result.

Go through the edges C-B we need 4+(-2) = 2 pts, and 2 is less than 6 , we have A-E-C-B.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| A | B | C | D | E |
| 0 | 2 | 4 | 2 | 7 |

Go through the edges B-D we need 2+(-4) = -2 pts, and -2 is less than 2, we have A-E-C-B-D.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| A | B | C | D | E |
| 0 | 2 | 4 | -2 | 7 |

Now, we have the shortest path. However, to check whether the path becomes a “negative cycle”, we must iterate the above solution to V times and if the path still changes the value at the V time.

The result must look like the graph below:

7

-3

-2

-4

**Applying in code:**

**Data structure:**

First, we may need to design some data structure for the edge and the graph:

Struct Edge{

Int src // source from

Int des // to destination

Int weight// the weight of the edge

}

Struct graph {

Int V // number of the vertexes

Int E // number of the edges

Edge edge[E] // a list of edges

}

Assuming each vertex name is a number as follow: A,B,C,D,E are 0,1,2,3,4

Applying Bellman-Ford, we have this pseudo-code:

Function Bell\_ford (Graph graph, int src):

Int dist[graph.V] = {infinity}

Int node[graph.V][graph.V]

Let node[src][0] = src

Let dist[src] = 0

Loop i=0 -> Graph.V-1: //O(V)

Loop through Graph.E as edge: //O(E)

If dist[edge.src] + edge.weight < dis[edge.des] :

dis[edge.des] = dist[edge.src] + edge.weight

node[edge.des] = node[edge.src] *// copy the path of the src to the des*

node[edge.des].add(edge.src)

If (dist does not changes in the edge loop): finish the i loop

In this function, we assign a variable dist to record the distance of each path. The node variable is recording the path of the vertex. We loop in the maximum V-1 times and each time we loop through all the edges. We will check if the src edge adds the weight is less than the weight of the previous distance of the des edge. If so, we replace the path and update the distance. Otherwise, we ignore this edge. For a better performance, we do not need to loop again.

**Performance:**

Let n is the number of the vertex: n = V

The worst case in the first loop to V-1 can reach to maximum time => O(n).

The worst case in the second loop to E in scenario that the number of maximum E can be V(V-1) which has O(n^2).

If both worst cases are happened in the same graph, the worst-case time complexity is O(V^3) or O(n^3)

The average time complexity happened if the second case isn’t happened is O(V\*E)

The best time complexity happened if the shortest path can be found in the first loop at the time complexity is O(E).

**The result of C++ implementation:**

A , total weight is 0

A E C B , total weight is 2

A E C , total weight is 4

A E C B D , total weight is -2

A E , total weight is 7