Efficient **solution** of All Pair Shortest Path

Shakil Mahmud, Amit Khan, Abeda Sultana Khanam

2016-1-60-007, 2016-1-60-012, 2016-1-60-011

Department of Computer Science and Engineering, East West University, Dhaka-1212, Bangladesh.

Email: shakilmahmud13@gmail.com, uchihaamit1@gmail.com, abeda2152@gmail.com

*Abstract*— **Routing is important to forward information from one node to another in communication network. It is accomplished by shortest path algorithm. There are different types of algorithm for finding paths in network such as Dijkstra’s algorithm, Bellman-ford, Floyd algorithm etc. In search algorithm when nodes get increased, number of search and complexity is also get increased. In this paper, analysis of shortest path is done in terms of minimizing the time complexity to solve any shortest path problem.**

Keywords—Bellman Ford’s; Dijkstra’s algorithm; Floyd Warshall; Johnson’s.

# Introduction

The shortest path computations are one of the problems in graph theory. If G(V,E) is directed weighted graph, where V represents the set of vertices of graph & E represents the set of edges of graph. |V | represents the total number of vertices in graph & |E| represents the total number of edges in the graph. In shortest path problems, a directed weighted graph is given & the goal is to determine the shortest path among vertices. The shortest path problem can be categorized in to two different problems; single source shortest path problem and all pair shortest algorithm. In this paper, we will work on all pair shortest path problem. In all pair shortest path problems, the goal is to finding the shortest paths between all pairs of vertices of a graph.

# All pair shortest path

## Floyd Warshall Algorithm

The Floyd-Warshall algorithm is a graph analysis algorithm for finding shortest paths in a weighted & graph. A single execution of the algorithm will find the shortest paths between all pairs of vertices. The Floyd–Warshall algorithm is named after Robert Floyd and Stephen Warshall; it is an example of dynamic programming &optimal substructure[1]. A single execution of the algorithm will find the lengths (summed weights) of the shortest paths between all pairs of vertices.

The algorithm considers the "intermediate" vertices of a shortest path, where an intermediate vertex of a simple path p = v1, v2,..., vl is any vertex of p other than v1 or vl, that is any vertex in the set {v2, v3,...,vl-1}.The Floyd-Warshall algorithm is based on the following observation. The vertices of G are V = {1, 2,..., n}, let us consider a subset {1, 2,..., k} of vertices for some k. For any pair of vertices i, j Є V, consider all paths from i to j whose intermediate vertices are all drawn from {1, 2,..., k}, and let p be a minimum weight path from among them.

* If k is not an intermediate vertex of path p, then all intermediate vertices of path p are in the set {1, 2,..., k - 1}. Thus, a shortest path from vertex i to vertex j with all intermediate vertices in the set {1, 2,..., k - 1} is also a shortest path from i to j with all intermediate vertices in the set {1, 2,..., k}.
* If k is an intermediate vertex of path p, then we break p down into i→(p1) k→(p2)j. p1 is a shortest path from i to k with all intermediate vertices in the set {1, 2,..., k}. Because vertex k is not an intermediate vertex of path p1, we see that p1 is a shortest path from I to k with all intermediate vertices in the set {1, 2,..., k-1}. Similarly, p2 is a shortest path from vertex k to vertex j with all intermediate vertices in the set {1, 2,..., k - 1}.

Let dij (k) be the weight of a shortest path from vertex i to vertex j for which all intermediate vertices are in the set {1, 2,..., k}. When k = 0, a path from vertex i to vertex j with no intermediate vertex numbered higher than 0 has no intermediate vertices at all. Such a path has at most one edge, and hence dij(0) =wij . A recursive definition is:

Dij(k) = wij if k = 0,

min (dij(k-1).dik(k-1) + dkj(k-1)) if k ≥ 0

Pseudocode for this basic version follows:

procedure [array] FloydWarshall(D, P)

for k in 1 to n do

for i in 1 to n do

for j in 1 to n do

if D[i][j] > D[i][k] + D[k][j] then

D[i][j] = D[i][k] + D[k][j]

P[i][j] = P[k][j]

return P

This algorithm runs on O(V3) time complexity.

## Johnson’s Algorithm

Johnson‘s algorithm can be used to solve all pair shortest path problems within the time complexity of O(V2logV+VE) time. It allows some of the edge weights to be negative numbers, but no negative-weight cycles may exist. It works by using the Bellman–Ford algorithm to compute a transformation of the input graph that removes all negative weights, allowing Dijkstra's algorithm to be used on the transformed graph. It is named after Donald B. Johnson, who first published the technique in 1977. If the graph contain negative cycle then Johnson‘s algorithm reports that the graph contains negative cycle. If the graph does not contain negative cycle then Johnson‘s algorithm returns a particular matrix which shows the shortest distance among vertices.

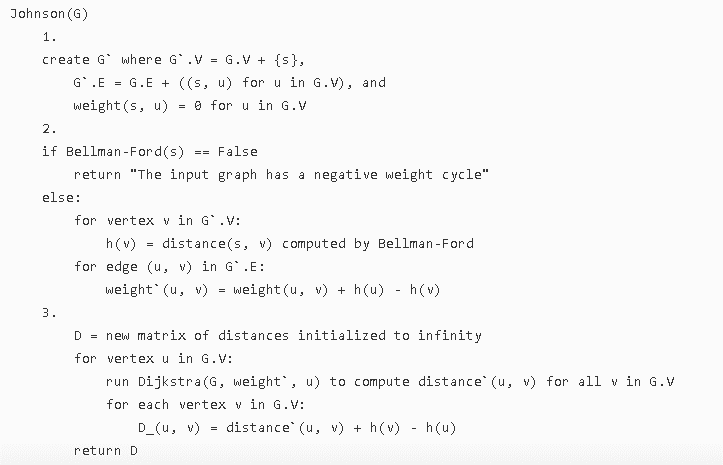
Johnson's algorithm is a hybrid of Dijkstra's algorithm and the Bellman-Ford algorithm. Johnson's algorithm uses the technique of re-weighting. Johnson's algorithm consists of the following steps:

1. First, a new node q is added to the graph, connected by zero-weight edge to each other node.

2. Second, the Bellman-Ford algorithm is used, starting from the new vertex q, to find for each vertex v the least weight h (v) of a path from q to v. If this step detects a negative cycle, the algorithm is terminated.

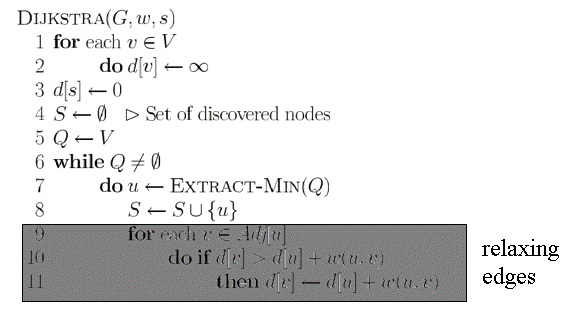
3. Next the edges of the original graph are reweighted using the values computed by the Bellman-Ford algorithm: an edge from u to v, having length w(u,v), is given the new length w(u,v) + h(u) −h(v). (h: V -> R be any function mapping vertices to real Numbers.)

4. Finally, for each node s, Dijkstra's algorithm is used to find the shortest paths from s to each other vertex in the reweighted graph



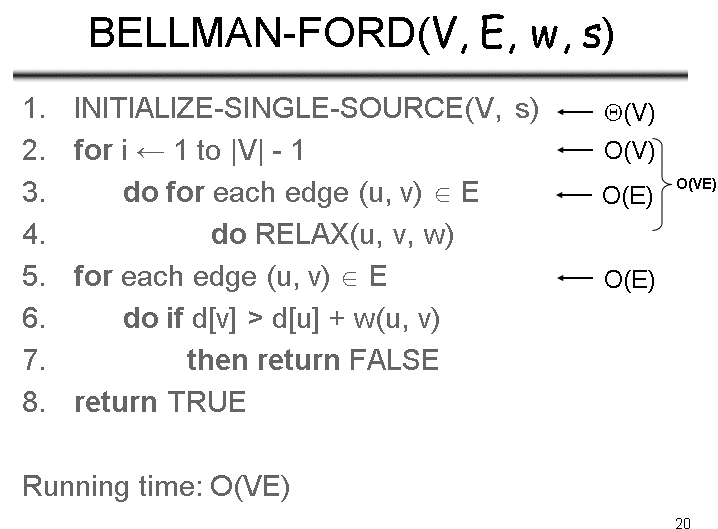
## Dijkstra’s Algorithm

Dijkstra algorithm can also be used to solve the single source shortest path problems on a given weighted, directed graph G(V,E) if and only if all the weights of edges are positive The time complexity of Dijkstra algorithm depends upon the implementation of min priority queue. If the Min priority queue is being implemented by using binary heap, then the time complexity of Dijkstra algorithm is O((V+E)logV). But if the min priority queue is being implemented by using Fibonacci Heap, then the time complexity for Dijkstra algorithm is O(VlogV+E).



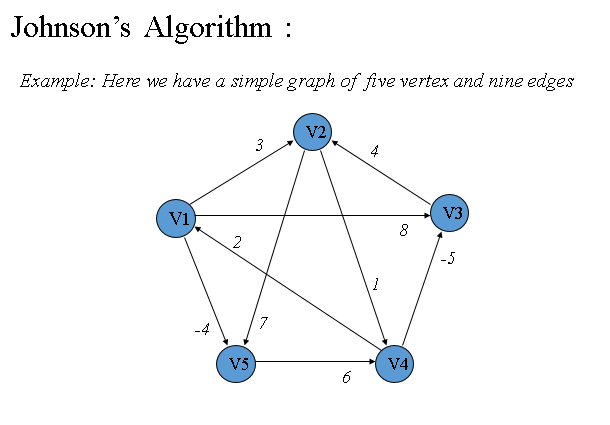
## Bellman Ford’s Algorithm

Bellman ford algorithm can be used to solve the single source shortest path problems in which edges of a given graph can have negative weight as long as the graph contains no negative cycle. The computational time of this algorithm increases exponentially with the increase in the number of nodes. This algorithm returns a Boolean value TRUE if the given graph contains no negative cycles that are reachable from source vertex otherwise it returns Boolean value FALSE.

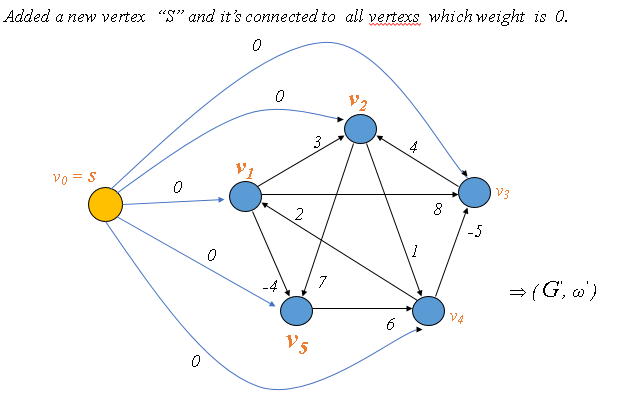


# Example of Johnson’s Algorithm

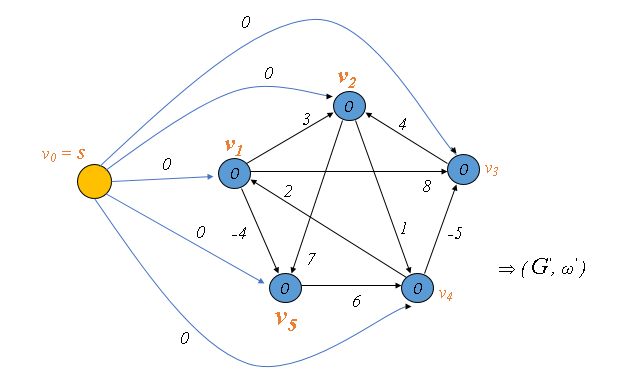
Here we have a simple graph of five vertex and nine edge:

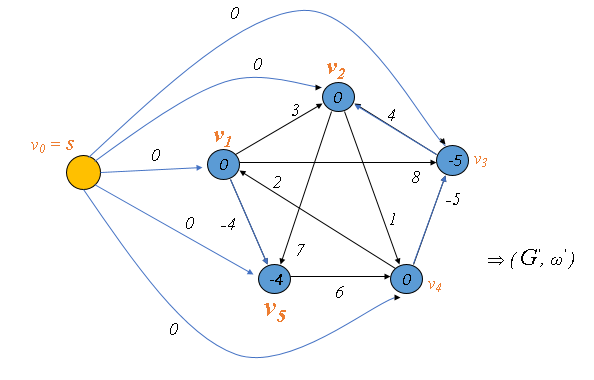


Added a new vertex “S” and it’s connected to all vertexs which is 0.



Applying Bellman Ford algorithm:

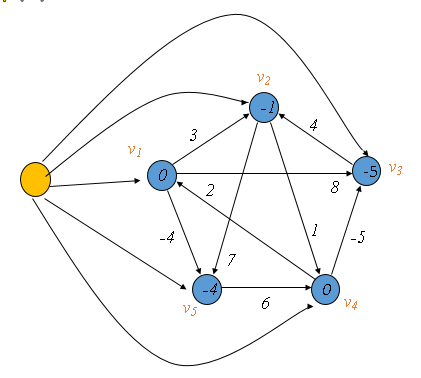


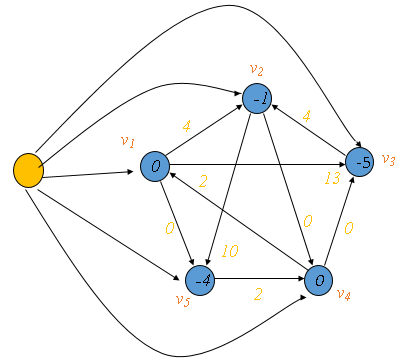


Edge Reweighting:

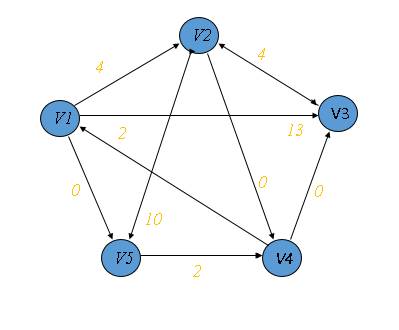
w\*(v,u) =w(v,u)+p(v)–p(u)

* (V1,V2)=3+0-(-1)=4
* (V1,V3)=8+0-(-5)=13
* (V1,V5)=-4+0-(-4)=0
* (V5,V4)=6-4-0=2
* (V4,V3)=-5+0-(-5)=0
* (V3,V2)=4-5-(-1)=0
* (V4,V1)=2+0-0=2
* (V2,V5)=7-1-(-4)=10
* (V2,V4)=1-1-0=0

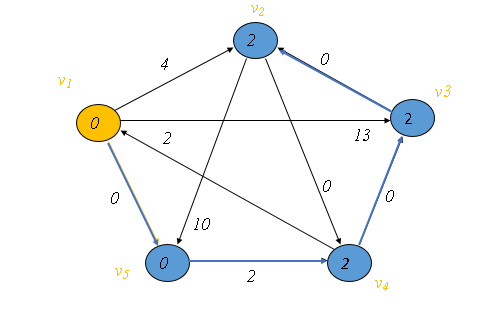




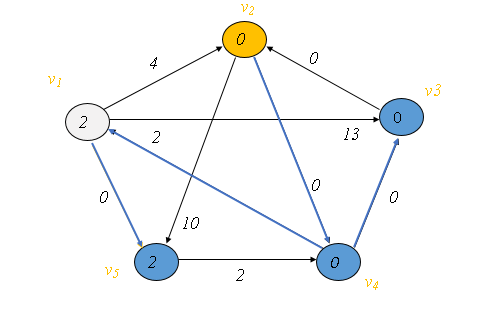
Simplifying the graph with every positive edge:



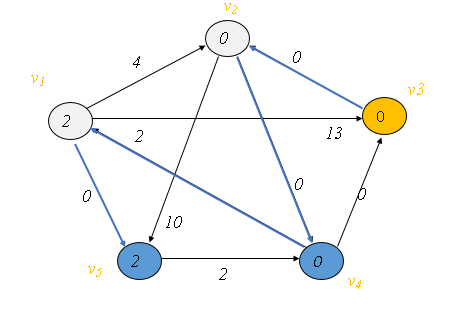
Run Dijkstra reweighted value of edge considering *V1* as source.



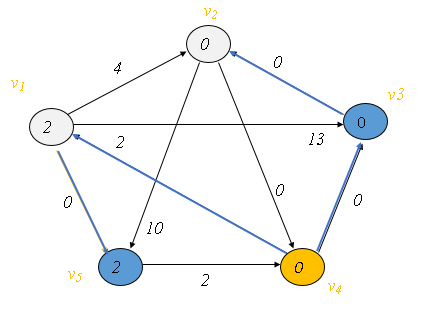
Run Dijkstra reweighted value of edge considering *V2* as source.



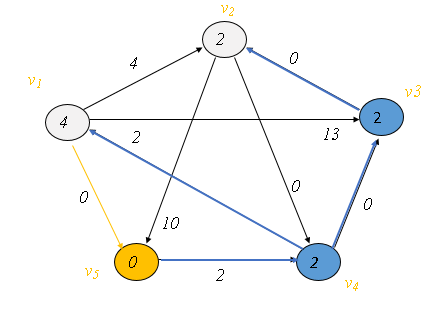
Run Dijkstra reweighted value of edge considering *V3* as source.



Run Dijkstra reweighted value of edge considering *V4* as source.



Run Dijkstra reweighted value of edge considering *V5* as source.



# Comparison of Algorithm

The time complexity[2] of Bellman Ford algorithm is O(|V|+|E|), the time complexity of Dijkstra algorithm using Fibonacci heap is O(VlogV+E). So the complexity of Johnson using Fibonacci heaps in the implementation of Dijkstra algorithm is O(V2logV+VE). But the time complexity of Floyd-Warshall algorithm is O(V3)

## Figures and Tables

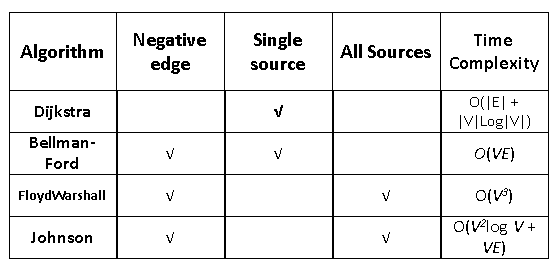


Figure: Time complexitiy of different algorithms

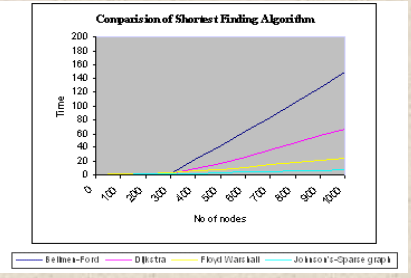


Figure: Comparison of the algorithms

##### Acknowledgment

We thank Md. Shamsujjoha sir for helping us in both theoretical and computational aspects of this project.

##### References

1. Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L. (1990). Introduction to Algorithms (1st ed.). MIT Press and McGraw-Hill. ISBN 0-262-03141-8. See in particular Section 26.2, "The Floyd–Warshall algorithm", pp. 558–565 and Section 26.4, "A general framework for solving path problems in directed graphs", pp. 570–576.J.
2. Kenneth H. Rosen (2003). Discrete Mathematics and Its Applications, 5th Edition. Addison Wesley. ISBN 0-07-119881-4.I.
3. https://en.wikipedia.org/wiki/Johnson%27s\_algorithm