Dijkstra's Algorithm and Floyd Warshall's Algorithm

Nawal Ahmed

Kennesaw State University

November 15, 2018

ABSTRACT

The purpose of this paper is to explore common solutions to the Shortest Path Problem in Graph Theory. The Shortest Path Problem relates to finding the shortest path given a weighted graph from a node to another. Two popular algorithms that are made for solving this problem are Dijkstra's Algorithm and Floyd Warshall's Algorithm. The following topics presented in the paper will investigate time complexities among the two algorithms as well as their effectiveness in real-world applications.

INTRODUCTION

For smaller graphs, there is no problem in brute forcing the answer into finding the shortest path; however, for graphs representing numerous paths such as a road system in the United States, it is time consuming, ineffective, and does not necessarily give confirmation as to whether the selected path is in fact the shortest – the real shortest path might have been skipped over. Pathfinding algorithms that decrease the amount of time it takes to select the shortest path from a certain starting point to a certain destination also give confirmation to the correct result.

All pathfinding algorithms must be capable of determining two results from a given problem:

1) A solution for finding a shortest path from a given starting point to a given end point does exist.
2) No such solution for finding a shortest path exists.

Pathfinding algorithms are used in multiple applications such as GIS applications (Geographic Information System), Satellite Navigations Systems, Network Routing, and Telephony Systems. There are a wide range of pathfinding algorithms that attempt to accomplish this with varying degrees of complexities and effectiveness. The A* path finding search algorithm for example is an extension of Dijkstra's algorithm that significantly reduces the time it takes to find the correct answer. As stated earlier, this paper will focus on comparing Dijkstra's algorithm and Floyd Warshall's algorithm.

HISTORY – Dijkstra's Algorithm

Dijkstra's Algorithm was devised by Edsger W. Dijkstra. It is an extremely popular algorithm that was created long before modern computing in 1956. Despite this, Dijkstra found his algorithm still bring used in several practical situations. Today, Dijkstra's algorithm has its most common use in routing protocols such as the Open Shortest Path First protocol where the algorithm is used to update tables that determine the shortest path from a source router to a destination router.

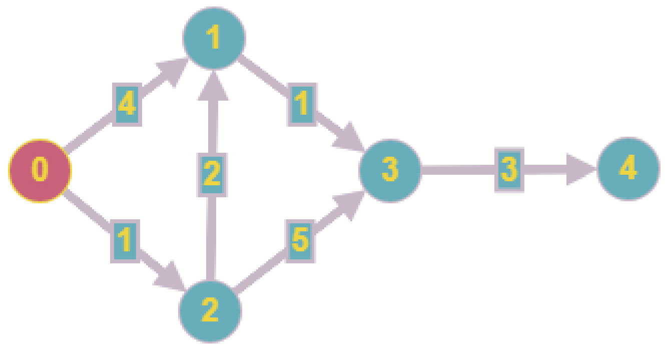
DIJKSTRA APPROACH

Dijkstra's algorithm in a sense is similar to a brute-force field approach with the exception that it is performed in an order that makes the most sense based on the weights of the graph. For Dijkstra's algorithm to work, the graph must contain non-negative edge weights. This is done to ensure that once a node has been visited, its optimal distance cannot be improved any further. This property makes Dijkstra's algorithm to become a greedy algorithm. A greedy algorithm is more of a technique in that it will always make the most optimal decision at the current time. The idea is that if at smaller ranges, the best choice is made, then for the entire picture, all the best choices would have been made. Greedy algorithms tend to be easier to implement and measure as opposed to other types such as Divide and Conquer algorithms.

Dijkstra’s algorithm is a Single Source Shortest Path algorithm. It tells the shortest path from one node to every other node. It is different from Prim’s algorithm and Kruskal’s algorithm which result in minimum spanning trees.

At a high level, here is how Dijkstra’s algorithm works:

1) All of the nodes on the graph are listed, and a starting node is selected.



2) The weights on the graph aside from the starting point are infinite since they have not been visited yet, and therefore it must be assumed every node is unreachable. The starting node has a value of 0. These values are typically stored as a distance array.

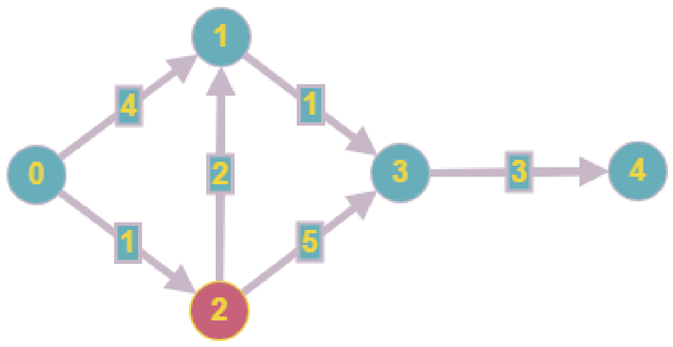
0	1	2	3	4
0	∞	∞	∞	∞

3) The edges leaving the starting node are examined and their weights are counted. A priority queue can be maintained to organize the key-value pairs of the node indexes and their distances.

0	1	2	3	4
0	**4**	**1**	∞	∞

(index, distance)
~~(0,0)~~
(1,4)

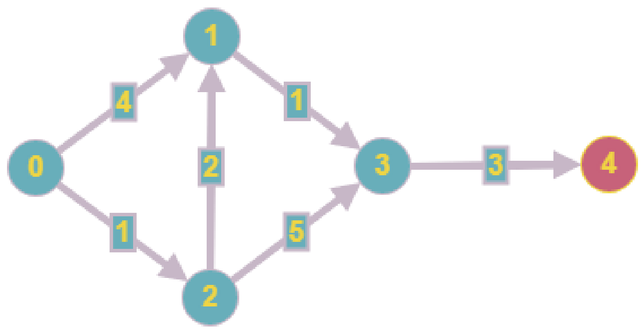
4) Next, the edge with the lowest weight of which the vertex that has not yet been chosen is selected in order to move forward.

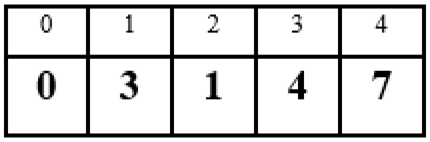


0	1	2	3	4
0	**3**	**1**	∞	∞

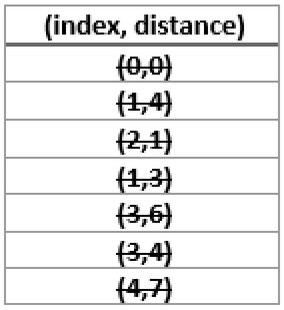
(index, distance)
~~(0,0)~~
(1,4)
(2,1)
(1,3)

5) The cycle of visiting vertices, logging the weights of the edges connected to them and them selecting the edge with the smallest weight of an unvisited node, continues until the all of the vertices have been visited at which point the shortest path is revealed.





0	1	2	3	4
0	3	1	4	7



(index, distance)
~~(0,0)~~
~~(1,4)~~
~~(2,1)~~
~~(1,3)~~
~~(3,6)~~
~~(3,4)~~
~~(4,7)~~

The Lazy Dijkstra implementation is by far the most popular. It inserts duplicate key-value pairs and lazily deletes them. This is done because it is more efficient to insert a new key-value pair in logarithmic time into the priority queue than it is to update an existing key's value in linear time. The time complexity of the Lazy approach is O(V^2). The Lazy approach, while easy to implement, is not as effect as it could be. If a dense graph is introduced, then in the end, there are stale, outdated key-value pairs in the party queue. The Eager Dijkstra implementation aims to solve the above issue by avoiding duplicate key value pairs and supporting efficient value updates in logarithmic time using an indexed priority queue, which is a priority queue variant which allows access to key value pairs within the priority queue in constant time, and updates in the logarithmic time if a min heap is used. Instead of the time complexity being O(V^2), the binary heap causes a decrease in time, reducing the complexity to a competitive O(ElogV).

HISTORY – Floyd-Warshall's Algorithm

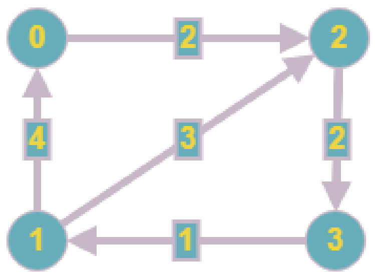
The algorithm that is commonly called as the Floyd-Warshall algorithm was created by Robert Floyd in 1962. It was developed independently from Stephen Warshall's approach to the algorithm that coincidentally was created sometime in 1962. Both methods are quite like each other, and as such, the algorithm is known as Floyd-Warshall's algorithm. On a rare occasion, it might be referred to as simply Floyd's algorithm.

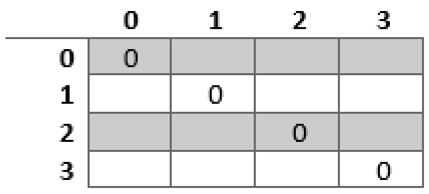
FLOYD-WARSHALL APPROACH

The Floyd-Warshall algorithm is an All-Pairs Shortest Path algorithm. As opposed to Dijkstra's algorithm, which finds the shortest path from one node to all other nodes, the Floyd-Warshall algorithm finds the shortest path between all pairs of nodes, and negative edges are allowed. The primary idea behind the Floyd-Warshall algorithm is to gradually build up all intermediate routes between two select nodes to find the optimal path.

Here is how Floyd-Warshall's algorithm works:

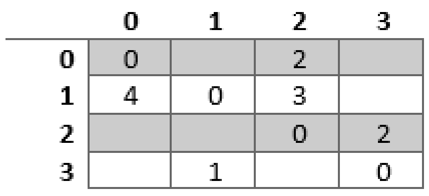
1) A graph that is intended to be used with Floyd-Warshall's algorithm is optimally represented by a 2D adjacency matrix. It is assumed that a distance from a node to itself is zero, and that is why there are diagonal zeros.





	0	1	2	3
0	0			
1		0		
2			0	
3				0

2) Loop through the edges of the graph and fill in the matrix with the corresponding weights.



	0	1	2	3
0	0		2	
1	4	0	3	
2			0	2
3		1		0

3) Next, the bulk of the algorithm is used, which involves a triple nested for-loop. There are three variables of concern: i,j, and k. i and j starting and destination nodes, and k involves intermediate nodes. The following if statement is used to update the matrix:

   a. if matrix[i][j] > matrix[i][k] + matrix[k][j]

      matrix[i][j] ← matrix[i][k] + matrix[k][j]

   Here is an example of the if statement in action. All three variables start at a value of 1. The if condition is applied until it is met, such as the following:

   b. k = 0 1 2 3 | i = 0 1 2 3 | j = 0 1 2 3

      matrix[i][j] > matrix[i][k] + matrix[k][j]

      matrix[0][1] > matrix[0][0] + matrix[0][0]

      $\infty > 0 + 0$ (Paths that have not yet been visited must presumably be infinite.)

      Now, the path from [0][1] can be updated with a smaller weight, which is 0.

	0	**1**	**2**	**3**
0	0	0	2	
1	4	0	3	
2			0	2
3		1		0

4) The end result is a table of shortest paths between all pairs of vertices.

	0	**1**	**2**	**3**
0	0	0	2	0
1	4	0	3	0
2	0	0	0	2
3	0	1	0	0

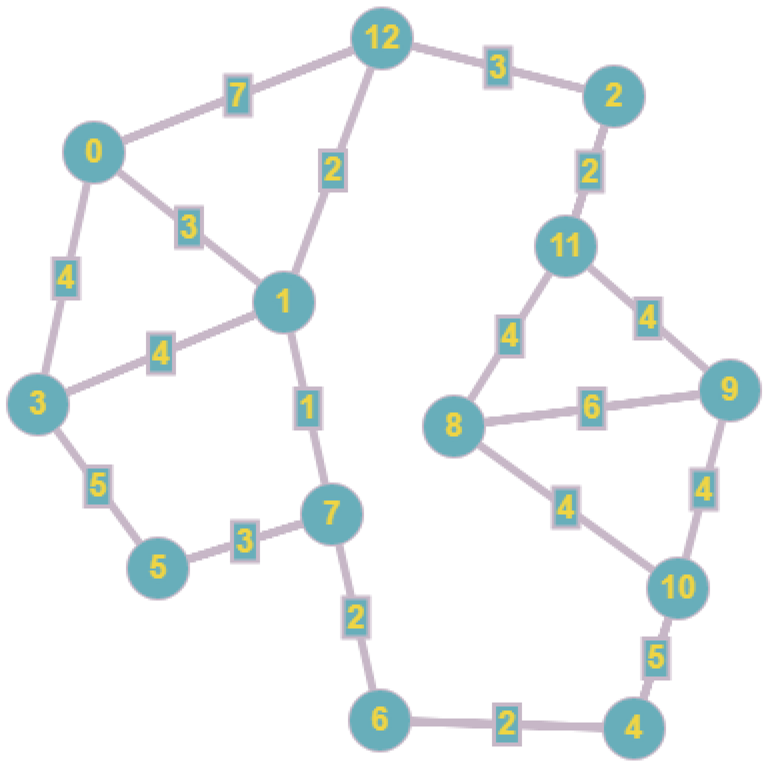
While it did not occur in the above example, it is possible to update a square to a different value than the one initially counted, however this condition occurs in graphs that contain negative edges. The time complexity of Floyd-Warshall's algorithm in $O(V^3)$, where V is the number of vertices since the bulk of the algorithm involves a triple for-loop. Due to this, Floyd-Warshall’s algorithm is rarely done by hand.

METHODOLOGY

One of the most key differences between Dijkstra’s Algorithm and Floyd Warshall’s Algorithm, are the use of negative numbers. Due to the methodology of Dijkstra’s Algorithms, negative numbers are not allowed, unlike in Floyd Warshall’s Algorithm. Luckily, when in the relevance of real-world applications, negative weighted graphs are of limited use. Therefore, to maintain an even comparison, only non-negative numbers will be used. The same graph will also be used for both implementations of the algorithms. It is important to also note, that integrally, neither algorithm return’s the specific details of the paths used for a destination vertex; however, some alterations to both algorithms can be made, so that more detailed results are given.

COMPARISON

The Dijkstra’s Algorithm program included utilizes its most basic implementation. At its core, the time complexity of the program is O(V^2). While, a random set of elements for the array could have been made, for the sake of consistency, the following graph was inserted:



For a more thorough analysis, a larger set of data could have been used, however the code shows the algorithm’s crucial steps that are the same for any type of implementation of Dijkstra’s algorithm.

Output:

```
Node--Dist
0------0
1------3
2------8
3------4
4------8
5------7
6------6
7------4
8------14
9------14
10------13
11------10
12------5
```

```
Process returned 0 (0x0)   execution time : 0.014 s
Press any key to continue.
```

The Floyd-Warshall algorithm program can represent more information, but only selective in respects from one vertex to another. The code uses the triple nested loops and has a time complexity of O(V^3). Despite not being a large graph, the Floyd-Warshall algorithm was slower in running than the Dijkstra algorithm. Again, to witness more remarkable results, are far larger graph is needed. Implementation of the Floyd-Warshall algorithm was easier as well.

Output:

```
    0    3    8    4    7    7    6    4    14   14   12   10   5
    3    0    5    4    4    4    3    1    11   11   9    7    2
    8    5    0    9    9    9    8    6    6    6    10   2    3
    4    4    9    0    5    5    7    5    14   14   10   11   6
    4    4    9    0    0    5    2    4    9    9    5    11   6
    4    4    9    0    0    0    2    3    9    9    5    11   6
    6    3    8    2    2    5    0    2    11   11   7    10   5
    4    1    6    3    3    3    2    0    12   12   8    8    3
    13   11   6    9    9    14   11   12   0    6    4    4    9
    13   11   6    9    9    14   11   12   6    0    4    4    9
    9    9    10   5    5    10   7    9    4    4    0    8    11
    10   7    2    11   11   11   10   8    4    4    8    0    5
    5    2    3    6    6    6    5    3    9    9    11   5    0

Process returned 0 (0x0)   execution time : 0.037 s
Press any key to continue.
```

CONCLUSION

Both algorithms can be best used in different cases. Using the most common Lazy approach to Dijkstra’s algorithm, the worst-case scenario is O(V^2). The time complexity can be improved depending on the implementation method. Floyd-Warshall’s algorithm has a time complexity of O(V^3), making it quite slower than the base implementation of Dijkstra. Naturally, Dijkstra could also be used to find the shortest path of all pairs of vertices, however, this would alter the time complexity, and it is not as effective as an All Pairs Shortest Path algorithm as Floyd-Warshall's algorithm is. If the goal is compute the shortest path on a graph with weighted edges, then the Floyd-Warshall algorithm is not suitable. Floyd-Warshall's algorithm is better suited towards smaller graphs of only a few hundred nodes since it uses more memory. Dijkstra's algorithm is good for larger to medium-sized graphs, where the goal is to calculate the shortest path between two nodes.

REFERENCES

[1] Singal, Pooja, and R.S. Chhillar. "Dijkstra Shortest Path Algorithm Using Global Positioning System." Semantics Scholar, International Journal of Computer Applications, 6 Nov. 2014, pdfs.semanticscholar.org/1540/945b4b16d9b485e410ff4a6730a7c103f399.pdf.

[2] Singh, Akanksha, and Pramod Kumar Mishra. "Performance Analysis of Floyd Warshall Algorithm." Semantics Scholar, International Journal of Computer Applications, 16 Nov. 2014, pdfs.semanticscholar.org/b1bf/92813e986099b1298bf45cb5d17454725161.pdf.

[3] Cormen, Thomas H., et al. Introduction to Algorithms. 3rd ed., Mit Press, 1990.

[4] Dai, Liang. "Fast Shortest Path Algorithm for Road Network and Implementation ." Carleton University School of Computer Science Publications, Carleton University, 2005, people.scs.carleton.ca/~maheshwa/Honor-Project/Fall05-ShortestPaths.pdf.

[5] Sanders, Peter, and Dominik Schultes. "Engineering Fast Route Planning Algorithms."

Karlsruhe Institute of Technology Publications, Karlsruhe Institute of Technology, June 2007, algo2.iti.kit.edu/documents/routeplanning/weaOverview.pdf.

[6] Erickson, Jeff. “Algorithms, Etc.” University of Illinois College of Computer Science, January 2015, http://jeffe.cs.illinois.edu/teaching/algorithms/