Ama Freeman

Shortest Path Algorithm Comparisons

Goal

Which shortest path algorithm works better and faster? The answer to this question depends on many factors. These include time spent running on a set of data and how many iterations are needed to process that data, to name a few. The task is to compare the efficiency of two shortest path algorithms: Dijkstra's Shortest Path Algorithm and the Floyd-Warshall Shortest Path Algorithm.

Dijkstra's algorithm finds the shortest path between two chosen vertices and returns the distance between the two.

The Floyd-Warshall algorithm is used to find a vertex's shortest paths between all other vertices and returns a matrix with the shortest paths.

Plan and Why Floyd-Warshall Algorithm

I will make use of two classes: Dijkstra and Floyd-Warshall. The Dijkstra class will contain all attributes of an adjacency matrix and also the method for finding the shortest path. The Floyd-Warshall class will also be quite similar to the Dijkstra's class, but will carry its own shortest path algorithm.

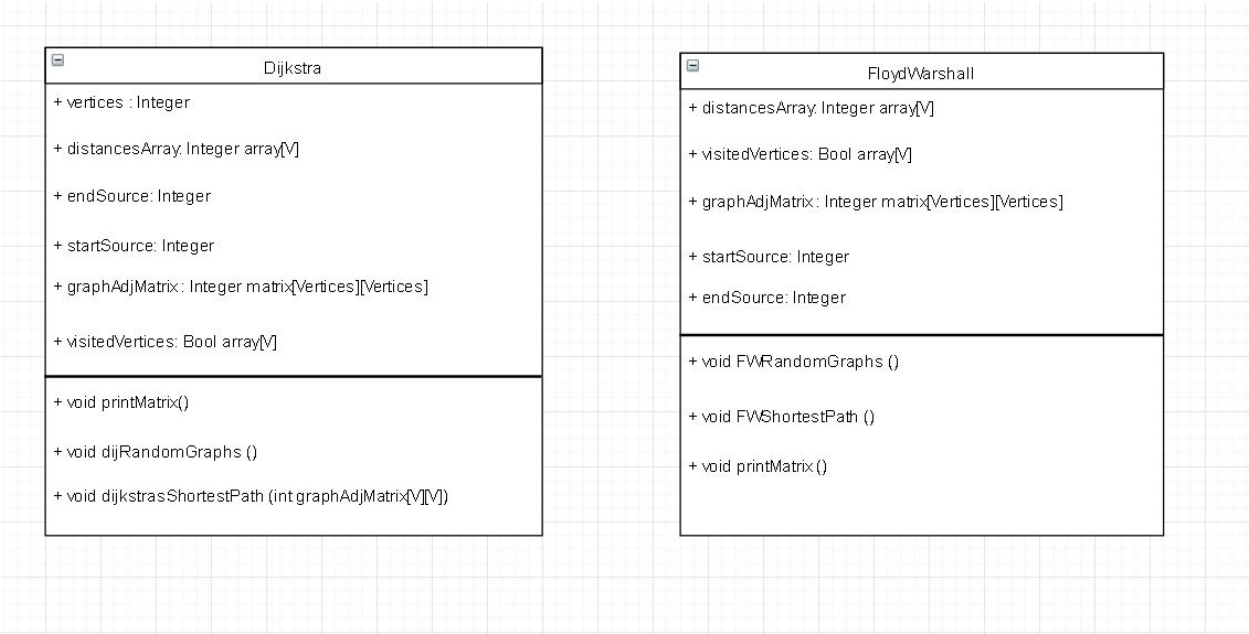
My choice for shortest path algorithm was between Bellman-Ford's and Floyd-Warshall's. Bellman-Ford is used to find the shortest path a source vertex has between one target vertex. On the other hand, Floyd-Warshall's algorithm is used to find all paths between a source and its sibling vertices. I specifically chose Floyd-Warshall over Bellman-Ford because I wanted to observe how to implement Floyd-Warshall's algorithm by producing data for the shortest path between a source and a target. I am hoping that using this algorithm will give me insight into changing a piece of code to find what I need, while also showing me how to make sure the code is reusable post changes that I have made.

Part 1 - OOA and OOD

A graph has an adjacency matrix. In order for it to be represented in a fashion both compact and tangible when the graph is not sparse, a matrix is the most plausible structure. The adjacency matrix is a a V by V matrix, defined by the number of vertices in the graph.

The Dijkstra's and Floyd-Warshall algorithms both have their own members and are classes isolated from each other.

UML Class Diagram



Part 2 - Testing

Test 1 - Generate a Single Graph, Print It, and Run it Through Dijkstra's Algorithm

This test is for checking the accuracy of my implemented Dijkstra's algorithm. Below are the results of the test where the code successfully runs.

My Dijkstra's implementation outputs the shortest path between each other vertex in the graph itself. In order to make sure results are true, it is imperative to check the results with all vertices. Notice that a distance of "0" stands for no connection between two nodes. My implementation allows me to use "0" as a standin for no connection.

```
1. Create and Print Matrix
2. Create Matrix and Run Dijkstra's Shortest Path Algorithm
3. Create Matrix and Run Floyd-Warshall's Shortest Path Algorithm
4. Quit
2
Creating a Matrix and Finding the Shortest Path using Dijkstra's Algorithm...
Printing Matrix:
0 3 1 0 0 2
0 0 4 4 1 3
4 2 0 4 0 2
4 4 0 0 4 0
2 1 0 2 0 4
4 3 4 0 0 0
Vertex 0 distance from source vertex 0: 0
Vertex 1 distance from source vertex 0: 3
Vertex 2 distance from source vertex 0: 1
Vertex 3 distance from source vertex 0: 5
Vertex 4 distance from source vertex 0: 4
Vertex 5 distance from source vertex 0: 2
What would you like to do? (Enter a number.)
1. Create and Print Matrix
2. Create Matrix and Run Dijkstra's Shortest Path Algorithm
3. Create Matrix and Run Floyd-Warshall's Shortest Path Algorithm
4. Quit
```

Test 2 - Generate a Single Graph, Print It, and Run it Through Floyd-Warshall's Algorithm

This test is for the accuracy of my implementation of Floyd-Warshall's Algorithm. Below are the results of the test.

```
2. Create Matrix and Run Dijkstra's Shortest Path Algorithm
3. Create Matrix and Run Floyd-Warshall's Shortest Path Algorithm
4. Quit
3
Creating a Matrix and Finding the Shortest Path using Floyd-Warshall's Algorithm
...
Printing Matrix:
0 4 0 3 3 0
4 0 4 2 0 4
3 3 0 1 0 4
2 0 0 0 2 2
1 4 2 3 0 2
0 4 0 3 1 0
Vertex 0 distance from source vertex 0: 4
Vertex 1 distance from source vertex 0: 4
Vertex 2 distance from source vertex 0: 5
Vertex 3 distance from source vertex 0: 3
Vertex 4 distance from source vertex 0: 3
Vertex 5 distance from source vertex 0: 5
What would you like to do? (Enter a number.)
1. Create and Print Matrix
2. Create Matrix and Run Dijkstra's Shortest Path Algorithm
3. Create Matrix and Run Floyd-Warshall's Shortest Path Algorithm
4. Quit
```

Floyd-Warshall's algorithm does not exclude cycles. Therefore, in the case from source "0" to "0," the algorithm found the shortest cycle between "0" and itself. In the case of my project, I will collect the data only for the distances from source "0" to the last vertex "V-1" simply for consistency when comparing both algorithms.

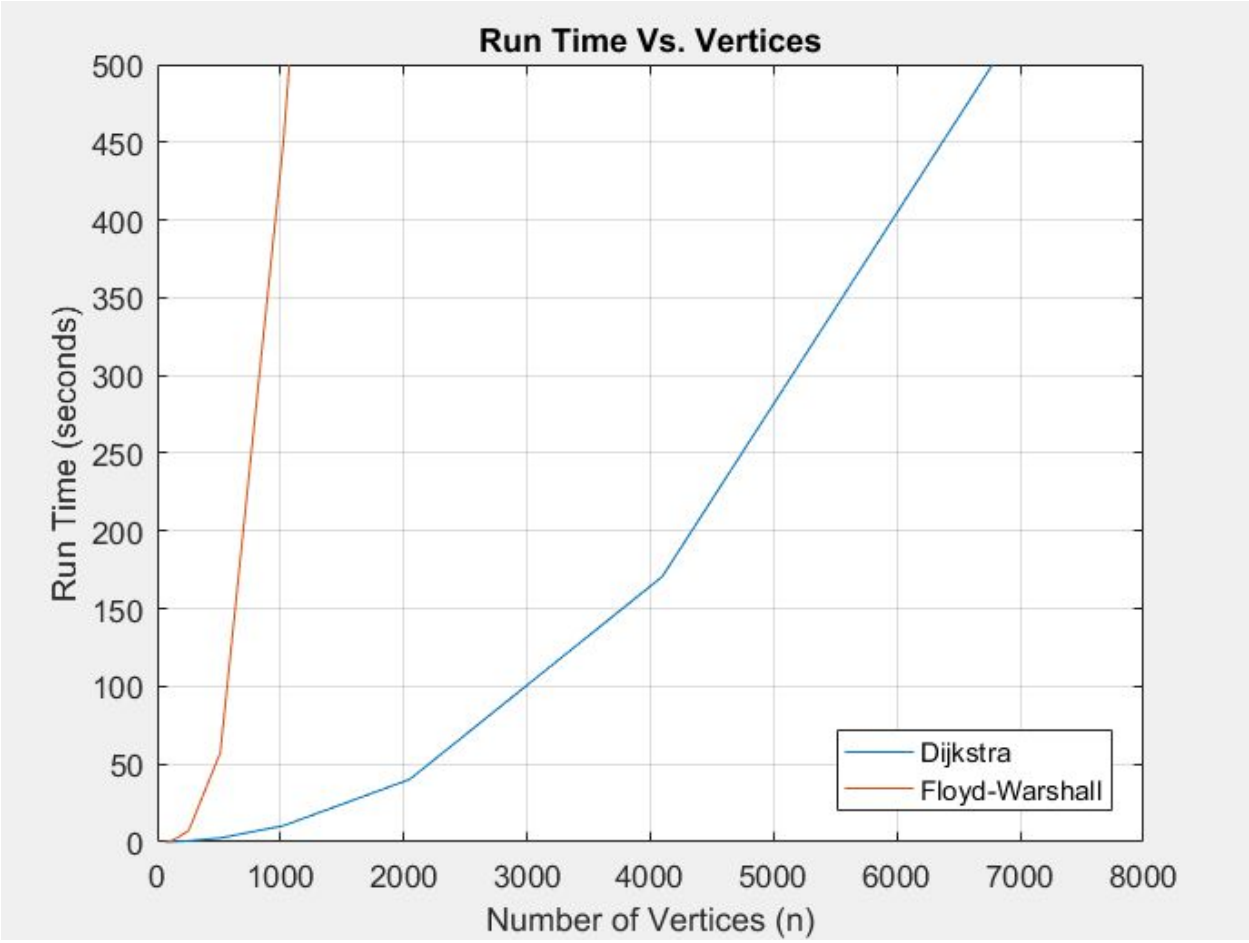
Generate 100 Graphs, Test Each Using Dijkstra's and Floyd-Warshall's Algorithms, Process Averages

In order for me to plot the comparisons of each algorithm, I will be using MATLAB for plotting and analysing the data.

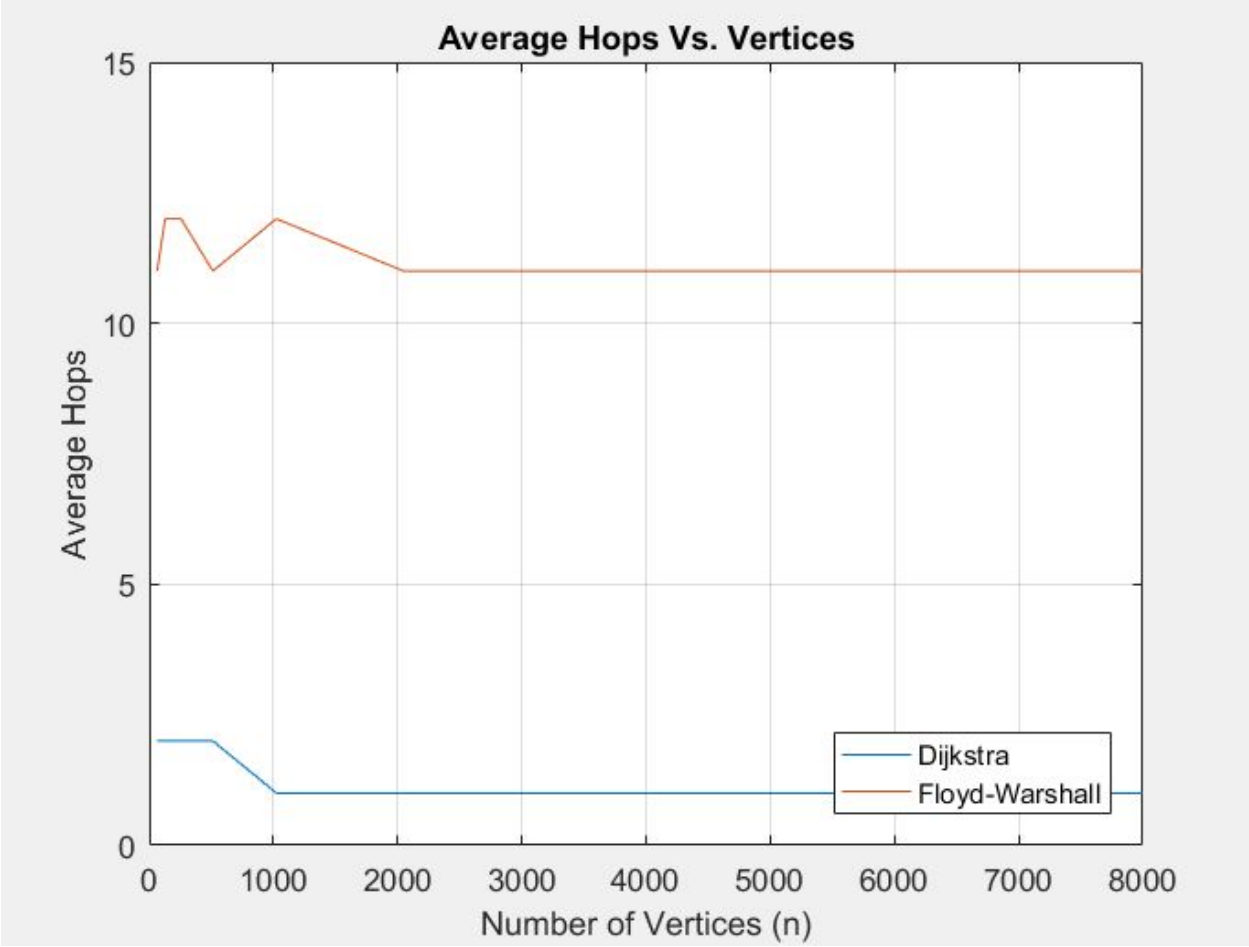
Collected Data

	A	B	C	D	E	F
1	Dijkstra's	Hops	Avg Hops	Distances	Avg Distance	Average Run Time (seconds)
2	64	200	2	200	2	0.044
3	128	200	2	200	2	0.175
4	256	200	2	200	2	0.634
5	512	200	2	200	2	2.507
6	1024	165	1	180	1	10.425
7	2048	160	1	175	1	40.234
8	4096	169	1	181	1	170.625
9	8192	163	1	181	1	674.231
10						
11	Floyd-Warshall	Hops	Avg Hops	Distances	Avg Distance	Average Run Time (seconds)
12	64	100	11	200	2	0.157
13	128	100	12	200	2	1.037
14	256	100	12	179	1	7.286
15	512	100	11	185	1	57.487
16	1024	100	12	179	1	450.373
17	2048	100	11	179	1	1534.445
18	4096	100	11	180	1	12402.581
19	8192	100	11	183	1	243001.239

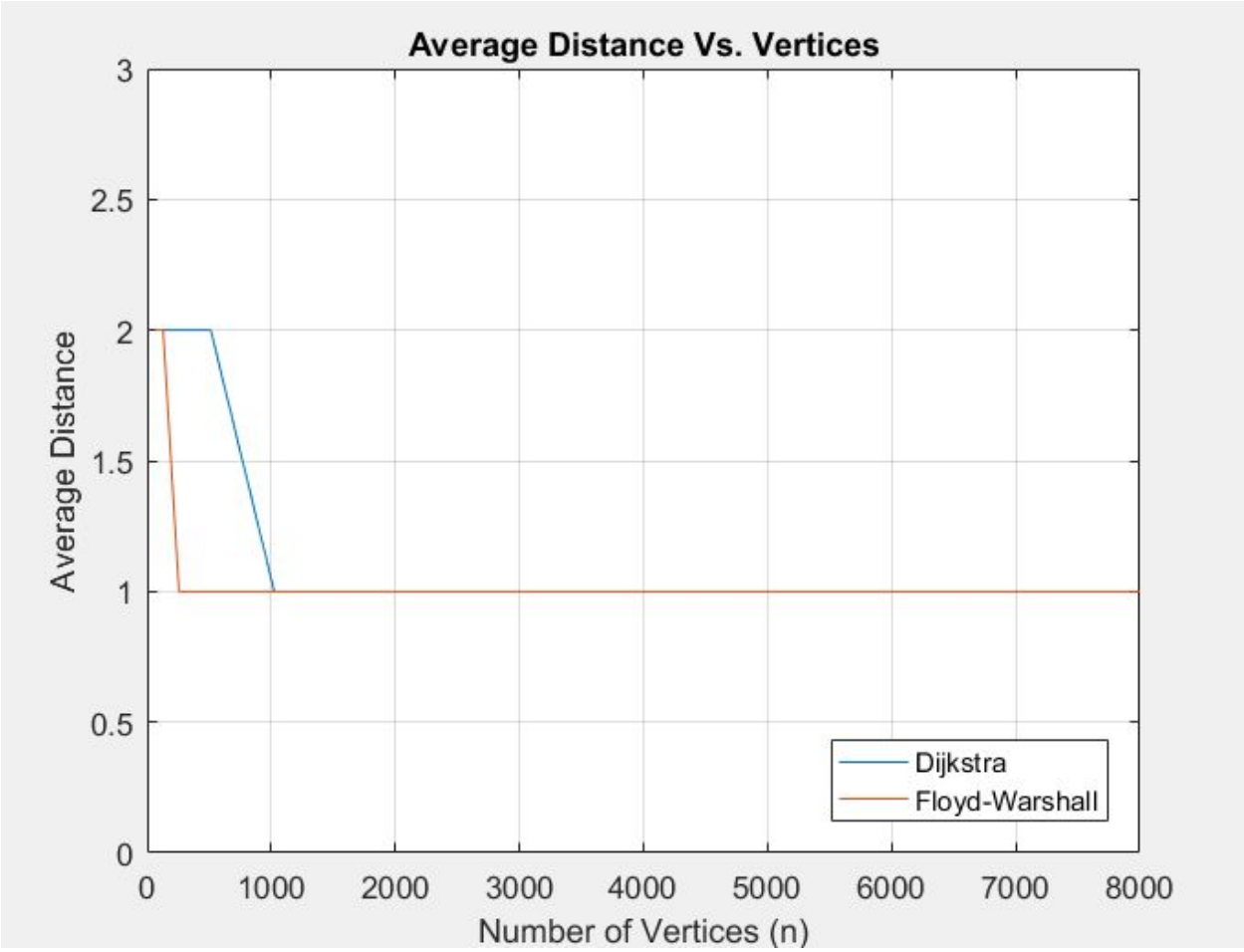
Runtime vs. Vertices



Average Hops vs. Vertices



Average Distances vs. Vertices



Memory Requirements and Big O

Floyd-Warshall's algorithm has a worst, best, and average case performance of $O(V^3)$. Due to the three nested for loops in the core of the algorithm, a run time this undesirable is quite unavoidable. This is also due to that fact that the algorithm finds *all* shortest paths in a provided graph. In terms of memory, Floyd-Warshall's can be implemented using one auxiliary matrix, or two. The former is required only when it is desired to see the shortest path pairs. The second case has another matrix in order to see the paths taken to reach another vertex. So big O for memory can be either $O(V^2)$ or $O(V^2 + V^2)$.

Dijkstra's algorithm has multiple time complexities depending on the implementation. However, for my algorithm, the average case is $O(|E| + |V|\log|V|)$, as with most other implementations. The algorithm requires an adjacency matrix and two arrays that contains elements equal to the number of vertices. Summed up, the memory complexity becomes $O(V^2 + 2V)$ which simplifies to $O(V^2)$. Compared to Floyd-Warshall's algorithm, Dijkstra's is much prefer in terms of both memory and performance. In real world scenarios, Dijkstra's proves to be the superior algorithm as well.

Real World Discussion

If memory were no object, graphs that implement an adjacency matrix, or two, will find great use where the total amount of vertices is extremely large. In other words, the matrix must not be sparse, to the point where almost every vertex is connected to almost all other vertices.

However, there are many cases where there are both memory and performance constraints. Further elaborating on one case, a school's curriculum and course path can be represented as a graph. In order for a student to pursue a certain degree, they are constrained to specific courses (vertices in the graph) and a few other paths that can lead to their degree (edges in the graph). The average community college offers nearly 500 or more courses per year, each course a part of a curriculum to be followed for the desired degree.

In the case of biology and chemistry, when medicines are being developed, it is required to know the many different interactions molecules and chemicals have with each other. The interactions can be unique and almost infinite, making it very memory hungry. This shows how a graph data structure would be very much needed, but also possibly extremely inefficient.

There is the case where a graph would be well implemented in aerospace engineering. Airplanes can use a graph data structure to carefully consider possible collision points between other airplanes in real time. The number of planes in the atmosphere above the United States at any given moment ranges between 5,000 to 6,000 (according to the NOAA). In such a case, memory is second to performance, as the most efficient shortest path algorithm will be the most desired for use.