**Shortest Paths between US Cities**

**XXX**

**Introduction**

 Graphs are widely used in our lives every day. We use them in activities such as navigation in GPS, routing of telecommunication operations, truck drivers finding shortest routes to their destinations, and many other applications in our everyday lives. The most popular algorithm used is to find is the shortest path. In our busy lives we are always looking to find a way to take the shortest path to where we are going. The shortest path problem is finding a path between two vertices such that the sum of weight of its constituent edge is minimized.  The most popular algorithms that are widely used to find the shortest path are Dijkstra’s algorithm, Bellman-Ford algorithm, Floyd-Warshall algorithm, and Johnson’s algorithm and so on.  Among them for our research we chose the Floyd-Warshall algorithm.

 Floyd-Warshall algorithm is a graph analysis algorithm for finding shortest path in a weighted graph with positive or negative edge weights and also for finding transitive closure of a relation R. A single execution of the algorithm will find the lengths of the shortest paths between all pairs of vertices. It does not return details of the paths themselves. The shortest path between two nodes of a graph is a sequence of connected nodes so that the sum of the edges that connect them is minimal. The Floyd- Warshall algorithm was published by Robert Floyd in 1962. It is very similar to the algorithms previously published by Bernard Roy in 1959 and Stephen Warshall in 1962 for finding the transitive closure. Peter Ingerman first formulated the three nested for-loop style that the Warshall algorithm uses.

**Implementation:**



**Pseudocode for Floyd-Warshall-**

**let** dist be a |V| × |V| array of minimum distances initialized to ∞ (infinity)

**for each** vertex *v*

 dist[*v*][*v*] ← 0

 **for each** edge (*u*,*v*)

 dist[*u*][*v*] ← w(*u*,*v*) *// the weight of the edge (*u*,*v*)*

 **for** *k* **from** 1 **to** |V|

 **for** *i* **from** 1 **to** |V|

 **for** *j* **from** 1 **to** |V|

 **if** dist[*i*][*j*] > dist[*i*][*k*] + dist[*k*][*j*]

 dist[*i*][*j*] ← dist[*i*][*k*] + dist[*k*][*j*]

 **end if**

 For our project we chose to use this algorithm to find the shortest path between certain US cities from West coast to East coast. We chose this theme for our project because we have a friend that took a road trip across the United States from West coast to East coast. She told us it took her two months, three oil changes and new tires. We thought about her trip and how long it took and we thought about how we could make it more efficient. By using this algorithm we increased the efficiency of her trip. While we discussed what algorithm we should choose for our project we contemplated using Dijkstra’s algorithm. At that point we were still studying the algorithms and we chose that because of popularity. Once we did more research the process led us to realize that the Floyd-Warshall algorithm was more suitable to our application. Dijkstra’s algorithm is used for sparse graphs. Dijkstra’s algorithm finds the optimal route from one node to other nodes. Our application consists of a large graph and a lot of data, 23 states. So we chose the Floyd-Warshall algorithm because it was more efficient. Floyd-Warshall finds the optimal route between all pairings. In our application we chose the state’s capital city to travel to. This made it easier to pinpoint accuracy and increased convenience. We used Washington State as our starting point. We chose the algorithm Floyd-Warshall and that is why we have to use an adjacency matrix.

In our application we used data West coast to East coast of the USA Capital Cities from a text file then insert the data into an Adjacency Matrix.

 **Distance Matrix – From West to East**

**Table to show shortest path between Washington and each state:**

|  |  |  |  |
| --- | --- | --- | --- |
| WA | To | OR | 367.4 miles |
| WA | To | CA | 1039.4 miles |
| WA | To | ID | 593.6 miles |
| WA | To | UT | 1107.7 miles |
| WA | To | NV | 898 miles |
| WA | To | AZ | 1599.2 miles |
| WA | To | MT | 1054.1 miles |
| WA | To | WY | 1111.1 miles |
| WA | To | CO | 153.6 miles |
| WA | To | NM | 1704 miles |
| WA | To | ND | 1584.4 miles |
| WA | To | SD | 1623.5 miles |
| WA | To | NE | 1868.4 miles |
| WA | To | KS | 1900 miles |
| WA | To | OK | 2277.9 miles |
| WA | To | TX | 2154 miles |
| WA | To | MN | 1939.7 miles |
| WA | To | IA | 2089.6 miles |
| WA | To | MO | 2428.8 miles |
| WA | To | AR | 2639.7 miles |
| WA | To | TN | 3765.8 miles |
| WA | To | NC | 3251.8 miles |

* We chose Washington to each state for convenience of the user. We wanted to show the results. This does not mean Washington must be the starting point. We did it this way just to show a particular result.

**Example Output: Distance From Washington to North Carolina**

****

This is our final output from WA to NC. The distance I got from my algorithm is 3,251.80 miles. I used the website <http://wdistances.com/olympia-wa/raleigh-nc> and there I got 2,914.38 miles. That website has a different amount of miles than our application because we used a different application. The difference is that we went through the capital cities and the website did not. The algorithm we used only returns the distance but not that path itself. Our algorithm gives the correct distance for the text file given.

The proof that our algorithm works can be shown using this algorithm: Let k = 0 (no iterations yet performed).

**Induction step:**

 Assume that after k iterations, D[i,j] is the cost of the lowest cost path from i to j excluding all vertices from k+1 to N.

On the next (k+1) iteration, we are allowed to include vertex k+1 in any path.

For all pairs (i,j), the lowest cost path from i to j excluding vertices k+2 through N goes through k+1 if there is a low cost path from i to k+1 and from k+1 to j, excluding vertices k+2 through N.

But the cheapest path from i to k+1 without using nodes k+2 through N is simply D[i,k+1] (by the induction hypothesis).

Similarly, the lowest cost path from k+1 to j without using nodes k+2 through N is D[k+1,j].

The Floyd-Warshall algorithm compares all paths possible in the graph between each pair of nodes. The comparisons in the graph would be O(|V|3) when every combination of edges possible is tested. The edges that are possible in the graph are Ω(|*V* |2). This helps us estimate the shortest paths between nodes, until the estimation is ideal.

**Sample Output:**



**Distance Map:**

**Conclusion:**

Floyd -Warshall is the algorithm widely used to find the all-pairs shortest path problem. In the all-pairs problems we have to find the shortest path between every pair of the vertices v, v’ in the graph. When comparing Dijkstra to Floyd-Warshall on the same problem we notice the difference in efficiency. The efficiency for Floyd Warshall is O(N^3). We can solve same problem with Dijkstra algorithm. However, it is less efficient than Floyd. To find all shortest path problem for using Dijkstra algorithm, it will be N^2 (N is the number of vertex), each time with a new starting and destination vertex. The overall efficiency of this case will be O(N^4). Finding all-pair shortest path problem, Floyd is better than Dijkstra algorithm. That’s the reason why we chose the Floyd- Warshall algorithm for our application. We wanted to find all shortest paths and we believe we made the right choice.