CS141 – Intermediate Algorithms and Data Structures
Assignment 2 – All Pairs Shortest Path

Stephanie Tong

Due on June 2 at 11:59 PM

Abstract

We will be comparing the run time of two separate algorithms for determing the shortest distance between all pairs of vertices in a graph. The first algorithm is Bellman-Ford's, which was originally developed for finding the shortest path between a single source vertex and all other vertices. We will extend this algorithm to find all pairs shortest paths. The next algorithm is Floyd-Warshall's, which was created to solve this problem faster than the extended Bellman-Ford algorithm.

1 Assignment

- Implement Bellman-Ford's algorithm to find the length of the path between a source (s) and all other vertices.
  - Extend Bellman-Ford's to find the length of the paths between all pairs of vertices.
- Implement Floyd-Warshall's algorithm to find the length of the paths between all pairs of vertices.
- Calculate the run-time of all of the implemented algorithms.
- Run all of the benchmarks on both "all pairs shortest path" algorithms.
- Fill in the report analyzing the algorithms and their run-time.
- You may remove the Assignment section (section 1) before turning in the final report

2 Introduction

- What is the problem that you are solving? We are trying to find the cheapest path for all vertices.
- What methods are you going to use to solve the problem? We will be using the Floyd-Warshall and Bellman-Ford algorithms.
- Why are these good methods to use? These methods are good because they illustrate different efficient algorithms that will find us the shortest path. These methods are also used in real-world applications.
- Why are you going to be using both of them? We will be comparing the performance of both.

3 Bellman-Ford

- What is the Bellman-Ford algorithm? It finds the shortest path from a single source vertex to every other vertex. The distance of the source vertex is initialized to 0 and the distances to all other vertices are initialized to infinity. After that step, it goes into a for loop that lasts for $|V-1|$ iterations and looks at all of the edges in the graph. Inside the loop, it performs edge relxation. If there's a cheaper edge from vertex u to vertex v, then we choose that edge to get to vertex v instead of the older one.

- Why are you using it? We are using one of many algorithms to compare performances between them.

- How did you adapt it to work for all-pairs as opposed to single source? I made all vertices as a source. In a matrix representation, the diagonals of the square matrix m[0][0], m[1][1], m[2][2], etc are initialized to 0. I added an additional for loop encapsulating the single source algorithm. This for loop runs $|V|$ times.

- What is the run-time of the algorithm before and after your adaptation? Before it's about O($|V^3|$) because we are iterating V-1 times in a V by V. After, it's abot O($|V^4|$) because we have to do this for all vertices.

4 Floyd-Warshall

- What is the Floyd-Warshall algorithm? It computes the shortest path to all vertices from all vertices in a graph. It It would initialize to 0 the diagonals of the 2D array which contains the shortest distance to all vectors. After that, it will find the shortest path from vertex i to all other vertices. Then it will do the same for vertex i+1.

- Why are you using it? To find the all pairs shortest path and compare its performance with the all pairs shortest path version of Bellman-Ford.

Table 1: Run-Time Comparison

Benchmarks	# Edges	Bellman-Ford Actual	Floyd-Warshall Actual
input4.txt	5	< 0 sec	< 0 sec
input5.txt	8	< 0 sec	< 0 sec
input10.txt	16	0.1 sec	< 0 sec
input25.txt	43	0.17 sec	0.01 sec
input50.txt	93	2.53 sec	0.06 sec
input100.txt	192	39.62 sec	0.43 sec
input250.txt	730	about 24 mins	6.25 sec
input500.txt	1532	about 9 hrs, 21 mins	51.65 sec
input1000.txt	2985	probably over 15 hrs	about 9 minutes

- How is it better than the Bellman-Ford algorithm? It requires one less for loop and doesn't need to recompute the same subproblems for every vertex.

- What is the run-time of the algorithm? The runtime is $|V^3|$.

5 Results

- Compare and contrast the two algorithms? What makes one more suited for this problem? The Floyd-Warshall approach is significantly faster. It is more suited for this problem because you don't need to redo the same subproblems for every vertex. For instance, in all pairs shortest path with Bellman-Ford you would have to recompute cell[2][0] even though you might have already computed it in cell[0][2].

- What are their theoretical run-times (from the previous sections) and how do they compare? Bellman-Ford: O($|V| * |E|$). Floyd-Warshall: O($|V^3|$)

- What are the actual run-times that you computed? Which method is better? Why? Bellman-Ford: O($|V^4|$). Floyd-Warshall: O($|V^3|$).

- Fill in table 1 with your results

6 Conclusions

- What did you find difficult about the assignment? It took some time to figure out how the professor formatted the graph G and how to access elements in G.
- What did you learn? Modifying Bellman-Ford to compute all pairs shortest path is worse than running Floyd-Warshall.
- What is one real-world problem that you think each of these problems would be good at solving? Bellman Ford is used in networking (distance vector algorithms). Floyd-Warshall could be used for determining the distance between all airports in a state.