Design and Analysis of Graph Algorithms

David Shin

California Polytechnic University, Pomona

Computer Science 331

Professor Perez

Abstract

In this project, we analyzed four different types of algorithms: Dijkstra, Floyd, Prim, and Kruskal. Dijkstra is a greedy algorithm that finds the shortest path from one vertex to every other vertex. Floyd’s algorithm is a dynamic programming approach that finds the shortest path from every vertex to every other vertex. Both Dijkstra and Floyd algorithms operate on digraphs. Prim and Kruskals algorithm are types of greedy algorithms that find the minimum spanning tree of a given undirected graph. Dijkstra’s algorithm was experimentally found to have an overall faster runtime than Floyd’s, as expected. For our first experiment, we passed in the same directed graph into both Dijkstra’s and Floyd’s, and repeatedly ran them for 20 consecutive cycles. Then we analyzed the data that was returned. The idea would be that Floyd’s algorithm would run slow for the first time and start to become faster finding the shortest path every time it ran for the same directed graph. For our second experiment Kruskals was found to have a faster runtime when finding the minimum spanning tree of an undirected graph that contained fewer than 76 vertices. This conclusion was made by first plotting points on a graph of vertices versus time in nanoseconds and calculating the intersection of two logarithmic trendlines, one from each algorithm. The data points that were used were the average time it took for both Prim and Kruskals to complete.

*Keywords*: Dijkstra, Floyd-Warshall, Floyd, Prim, Kruskal, minimum spanning tree, shortest path, dynamic programming, greedy, graphs, principle of optimality

Algorithm Discussion

**Dijkstra’s Algorithm:**

Dijkstra’s algorithm will find the shortest path from one vertex to any other vertex.

Every Case:

Space Complexity:

**Floyd’s Algorithm:**

Every Case:

Space Complexity:

**Prim’s Algorithm:**

Every Case:

Space Complexity:

**Kruskal’s Algorithm**

Worst Case:

Space Complexity:

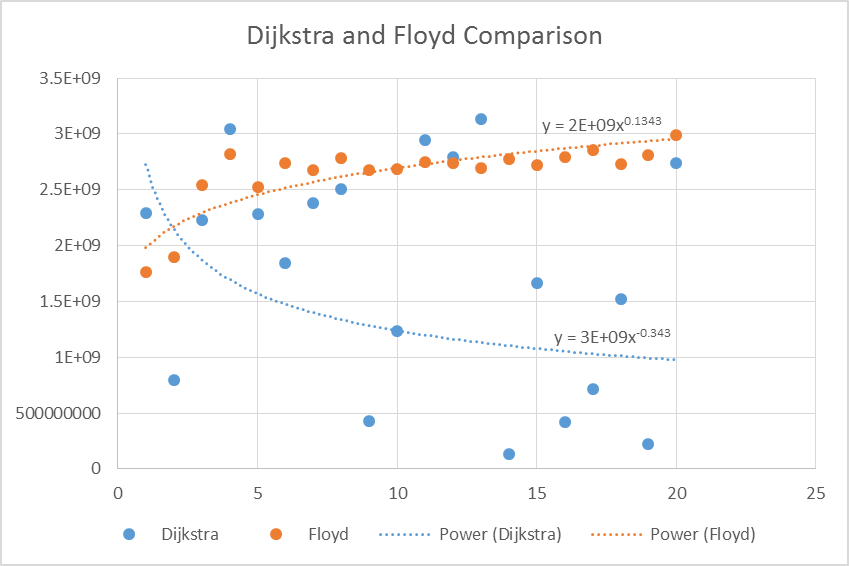
Experimental Results

Presented below are runtime data collected by running three separate tests and taking averages. Each runtime data point is the time it takes for an algorithm to perform in nanoseconds. For the shortest path algorithms (Dijkstra’s and Floyd’s), each data point was tested given a random source vertex and a random destination vertex that was calculated by java’s pseudorandom number generator. For the minimum spanning tree algorithms (Kruskals and Prims), each data point represents the time a randomly generated undirected graph took to be converted into a minimum spanning tree using those algorithms. The variable n represents the number of vertices being tested and the corresponding percentage is how “dense” a graph was.



Comparing Floyd’s and Dijkstra’s Algorithm

For this experiment, I used a 25% dense graph with 1024 vertices and repeatedly calculated the runtimes for finding the shortest path for both Dijkstra and Floyds. I passed in the same directed graph for every iteration and saved the resulting runtime data.



The intersection of the lines and calculated by Wolfram Alpha came out to be , which is approximately 2.3385. From my understanding of this intersection, Floyd’s algorithm would have to be requested at least 2.33 times before it would start to perform at least as well as Dijkstras.

A Closer Look at Prim’s and Kruskal’s Algorithm

Using the averaged data set with a 50% dense undirected graph, I have plotted Prim’s and Kruskal’s time in nanoseconds versus the number of vertices. Then, I have added a logarithmic trend line to experimentally calculate the intersection of the two lines. For Prim’s we are presented with the equation of the line: . For Kruskal’s, we are presented with the equation: . After finding the intersection of both lines, we are presented with, , which is approximately equal to 76.198. From this we can conclude that Kruskal’s algorithm will outperform Prims’s algorithm when the process of finding the minimum spanning tree is performed on a relatively low number of vertices. Kruskal’s algorithm will work better when the number of edges is sparse. With a relatively low number of vertices we can guarantee that there will be a lower number of edges. For example in a 50% dense graph, we have the formula (n(n-1))/2 to calculate the maximum number of edges there may be in that graph. For a graph containing 64 vertices we can have a maximum of 2016 number of edges with a 50% density. For a graph containing 32 vertices, we can have a maximum of 496 edges in a 50% dense graph. Given this fact, we can see under what conditions Kruskal’s will outperform Prim’s.

Conclusion

After analyzing the runtimes of each algorithm, I plotted them on an X Y scatter plot and found that the resulting logarithmic trend line equations of Prims algorithm was: y = 2E+07ln(x) - 7E+07 and Kruskal’s algorithm was: y = 5E+07ln(x) - 2E+08. Using Wolfram Alpha, I calculated the intersection of both lines and found that the resulting meeting points was , which was approximately equal to 76. This number was found to be the threshold for Kruskal’s algorithm perform better than Prims. Then I found how many times Floyd’s algorithm would have to be requested before it would perform as well as Dijkstra’s Algorithm and I experimentally found that number to be 2.33 times.

My experience with the project was even more challenging than the first project, especially because it was my first time implementing all the graph algorithms in code. I understood how each algorithm worked on paper but it was difficult to translate that into code. I have to say that this project made me way more knowledgeable about the different types of graph algorithms and will resort to Dijkstra’s algorithm when given the choice between that and Floyds.

References

Dijkstra's algorithm. (2017, March 12). Retrieved March 13, 2017, from <https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm>

Floyd–Warshall algorithm. (2017, March 11). Retrieved March 13, 2017, from <https://en.wikipedia.org/wiki/Floyd%E2%80%93Warshall_algorithm>

Kruskal's algorithm. (2017, March 05). Retrieved March 13, 2017, from <https://en.wikipedia.org/wiki/Kruskal%27s_algorithm>

Neapolitan, R. E. (2015). Foundations of algorithms (5th ed.). Burlington, MA: Jones & Bartlett Learning.

Prim's algorithm. (2017, March 12). Retrieved March 13, 2017, from <https://en.wikipedia.org/wiki/Prim%27s_algorithm>

Appendix

Dijkstra's 1 Time: 2287685480 Floyd's 1 Time: 1760839693

Dijkstra's 2 Time: 799806367 Floyd's 2 Time: 1899577610

Dijkstra's 3 Time: 2225315765 Floyd's 3 Time: 2538934744

Dijkstra's 4 Time: 3044417595 Floyd's 4 Time: 2823638752

Dijkstra's 5 Time: 2278358474 Floyd's 5 Time: 2526015034

Dijkstra's 6 Time: 1847997984 Floyd's 6 Time: 2740537292

Dijkstra's 7 Time: 2384357795 Floyd's 7 Time: 2679366667

Dijkstra's 8 Time: 2506001145 Floyd's 8 Time: 2780448185

Dijkstra's 9 Time: 425404486 Floyd's 9 Time: 2680073690

Dijkstra's 10 Time: 1238531968 Floyd's 10 Time: 2683426348

Dijkstra's 11 Time: 2943418184 Floyd's 11 Time: 2744571887

Dijkstra's 12 Time: 2789819664 Floyd's 12 Time: 2741836732

Dijkstra's 13 Time: 3137630628 Floyd's 13 Time: 2695676097

Dijkstra's 14 Time: 132212207 Floyd's 14 Time: 2777322457

Dijkstra's 15 Time: 1664137158 Floyd's 15 Time: 2719909889

Dijkstra's 16 Time: 423731008 Floyd's 16 Time: 2791040419

Dijkstra's 17 Time: 715535467 Floyd's 17 Time: 2851876919

Dijkstra's 18 Time: 1522615051 Floyd's 18 Time: 2729035620

Dijkstra's 19 Time: 218682861 Floyd's 19 Time: 2812998052

Dijkstra's 20 Time: 2739159167 Floyd's 20 Time: 2994700177







