Minimum Cost Path Between Two Nodes

Livădaru Alexandru-Valentin

Politehnica University of Bucharest
Faculty of Automatic Control and Computer Science
livadarualex@gmail.com
Group: 323CA

Abstract. https://medium.com/basecs/a-gentle-introduction-to-graph-theory-77969829ead8

Keywords: Floyd-Warshall · Bellman-Ford · Dijkstra · weighted graph · shortest path

1 Introduction

1.1 Description of the resolved problem

In the everyday life there are situations where we want to get from one point to the other as fast as we can, taking the shortest road. In this sense, there are things like GPS, but how does it work?

At the core of the GPS, there are graph-based algorithms like Floyd-Warshall or Djikstra which can find the shortest distances between every pair of points on the map.

A graph is a nodes and edges formed data structure. The edges are sometimes referred to as arcs (or lines) and the nodes are also called vertices. Simply put, a graph is an ordered pair G = (V, E) consisting of: V which is a set of points and E a set of lines.

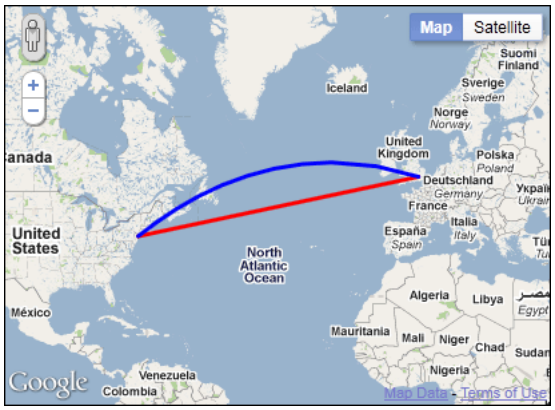

Fig.1. Minimum distance between two points

1.2 Examples of practical usage for the chosen problems

Besides the GPS, there is an even more known example of graph-based algorithms utility: connecting with friends on social media, where each user is a vertex and when they connect they create an edge. Moreover, these algorithms are used to search for webpages. We can compare a web page with a vertex and the link between two web pages with an edge.

1.3 Chosen solutions

In computer science, the Floyd–Warshall algorithm is an algorithm for finding shortest paths in a weighted graph with positive or negative edge weights (but with no negative cycles). A single execution of the algorithm will find the lengths (summed weights) of shortest paths between all pairs of vertices.

Dijkstra's algorithm is an algorithm for finding the shortest paths between nodes in a graph. For a given source node in the graph, the algorithm finds the shortest path between that node and every other. This has a better performance than the previous algorithm but it can not work on graphs with negative edges.

The Bellman–Ford algorithm is an algorithm that computes shortest paths from a single source vertex to all of the other vertices in a weighted digraph. It is slower than Dijkstra's algorithm for the same problem, but more versatile, as it is capable of handling graphs in which some of the edge weights are negative numbers.

1.4 Chosen evaluation criteria for the validation of the solutions

In order to test the correctness, the efficiency and the performance of the chosen algorithms, I created a generator which builds random inputs to see if the path generated by the algorithm is the shortest.

There more types of input tests, and these are split into two categories: performance tests and acceptance tests.

When talking about performance tests,we can compare our implementations with others to see which is the fastest one, preferably the input should be a bigger one.

In the case of acceptance tests we want to see if the algorithm has a good implementation. We do that by testing bigger or edge-case tests. At the end we check is the result is the expected one.

2 Solutions presentation

2.1 Presentation of the way the algorithms work

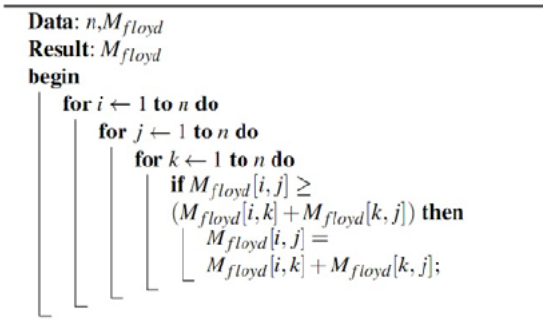

Fig.2. Pseudocode of the Floyd-Warshall algorithm

We initialize the result matrix with the cost of the edges between the nodes given by the input. After this, we update the result matrix by calculating the shortest paths and using all the nodes as intermediate (one by one) . When reaching the k vertex, we already know that vertices from 1 to k - 1 have been verified. For every pair (x, y) (source and destination) there are two possible cases: if k is not an intermediate node, we do not update the result, else, we update the result if distance[x][k] + distance[k][y] is less than distance[x][y].

A negative cycle is a cycle whose edges summed result in a negative result. There is no shortest path between any two nodes (or vertices) i, j which form part of a negative cycle, because path-lengths from i to j can be arbitrarily small (negative). For numerically meaningful output, the Floyd–Warshall algorithm assumes that there are no negative cycles. Nevertheless, if there are negative cycles, the Floyd–Warshall algorithm can be used to detect them.

Figure 4.8 **Dijkstra's shortest-path algorithm.**

```
procedure dijkstra(G, l, s)
Input:  Graph G = (V, E), directed or undirected;
        positive edge lengths {l_e : e ∈ E}; vertex s ∈ V
Output: For all vertices u reachable from s, dist(u) is set
        to the distance from s to u.

for all u ∈ V:
   dist(u) = ∞
   prev(u) = nil
dist(s) = 0


H = makequeue(V)  (using dist-values as keys)
while H is not empty:
   u = deletemin(H)
   for all edges (u, v) ∈ E:
       if dist(v) > dist(u) + l(u, v):
          dist(v) = dist(u) + l(u, v)
          prev(v) = u
          decreasekey(H, v)
```

Fig.2. Pseudocode of Djikstra algorithm

In order to get the shortest path from a single source vertex to all the others, we must take every node as a source node and then apply the algorithm. The algorithm has several steps.

Firstly, we create a shortest path tree set that keeps the record of nodes included in shortest path tree. This is an array that has all the elements set as false by default.

Secondly, Assign a distance value to all the vertices in the graph (preferably initialize all distances to infinity). The important thing is to assign distance value 0 to the source so that it is picked first.

In the last step, we need to update the distance value of all adjacent vertices of u. To update the distance values, iterate through all adjacent vertices. For every adjacent vertex v, if sum of distance value of u (from source) and weight of edge u-v, is less than the distance value of v, then update the distance value of v.

```
BellmanFord(graph, start, goal)
    foreach v in graph.vertices
        v.distance = infinity
        v.previous = null

    start.distance = 0

    for i = 1 to (graph.nvertices - 1)
        foreach v in graph.vertices
            foreach u in graph.vertices
            if there is an edge from v to u

                if (v.distance + distance(v, u)) < u.distance
                    u.distance = v.distance + distance(v, u)
                    u.previous = v

    return (goal.distance != infinity)
```

Figure 2. Pseudocode for the Bellman-Ford algorithm

Fig.3. Pseudocode of Bellman-Ford algorithm

In order to get the shortest path from a single source vertex to all the others, we must take every node as a source node and then apply the algorithm.

In the first step, we initialize distances from source to all vertices as infinite and distance to source itself as 0. Create an array dist[] of size $|V|$ with all values as infinite except dist[src] where src is source vertex.

In the second step, we calculate the shortest distances. We execute the following operation $|V|$-1 times (number of vertices minus one). For u to v edge do: if dist[v] bigger than dist[u] plus weight of edge uv, then update dist[v], dist[v] equals dist[u] plus weight of edge uv.

In the third and final step the algorithm reports if there is a negative weight cycle in graph. Do following for each edge from u to v: if dist[v] greater than dist[u] plus weight of edge uv, then "Graph contains negative weight cycle". In some fields, artificial intelligence in particular, Dijkstra's algorithm or a variant of it is known as uniform cost search and formulated as an instance of the more general idea of best-first search.

The Bellman–Ford algorithm may be improved in practice (although not in the worst case) by the observation that, if an iteration of the main loop of the algorithm terminates without making any changes, the algorithm can be immediately terminated, as subsequent iterations will not make any more changes. With this early termination condition, the main loop may in some cases use many fewer than $|V| - 1$ iterations, even though the worst case of the algorithm remains unchanged.

2.2 Analyzes of the complexity of the solutions

The algorithm named Floyd-Warshall makes a comparison between all possible edges of the graph between each two nodes. The performance of the algorithm is O(V ^{3}) (comparisons in a graph), although there might be up to O (V ^{2}) paths in a graph and every combination of edges is tested.

Time Complexity of Dijkstra's Algorithm is $O(V^2)$ but with min-priority queue it drops down to O(V + ElogV). However, if we have to find the shortest path between all pairs of vertices, both of the above methods would be expensive in terms of time.

Time Complexity of Bellman Ford algorithm is relatively high O(VE), in case E equals V^2, $O(E^3)$.

2.3 Presenting the main advantages and disadvantages of the solutions taken into consideration

The advantages of the Floyd-Warshall algorithm are: it is easy to write code for this algorithm in a program and it highlights all the shortest path pairs in a graph.

One disadvantage of this algorithm is that it is slower than other algorithms designed to perform the same task.

Djikstra algorithm is working similar with Bellman-Ford. The advantage of Djikstra over Bellman-Ford is that is faster.

A disadvantage of Djikstra is that it will not work on graphs with negative cost edges.

The main advantage of Bellman-Ford algorithm is that it works fine on a graph having negative edge weights.

The main disadvantage of Bellman-Ford is that it does not work if the graph contains negative weight cycle. Besides, it is slower than Djikstra.

Although every algorithm has its advantages and disadvantages, it really is the situation that decides which type of graph and algorithm to use. For example we will use a positive weighted graph for a map and a negative one for business sales and there are a lot of other examples.

3 Evaluation

3.1 Description of the way the validation tests were built

I created a generator which takes the number of vertices and the number of edges and generates a random input stored in the 'in' file.

The first validation tests were relatively small, and they were intended to check if the algorithms can resolve a basic graph situation.

Gradually, the dimensions of the input test increased and the tests went to considerable sizes. These sizes represented graphs that couldn't possibly be drawn on the paper.

Moreover, I generated some edge cases like negative weighted costs for Djikstra and negative weighted cycles for Floyd-Warshall and Bellman-Ford. If the input brings the algorithm to an edge case,it displays a suggestive message.

Regardless of test, we analysed the output and compared it to the expected one and we came to the conclusion that the algorithms were correctly designed and built.

3.2 Mention the system specifications on which the test were run

Operating System: Linux Ubuntu 18.04.3 LTS

Processor: Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz

MemTotal: 8004952 kB

MemFree: 324672 kB

MemAvailable: 978208 kB

3.3 Illustrate, using graphics/tables, the results of the evaluation of the solutions on the data set

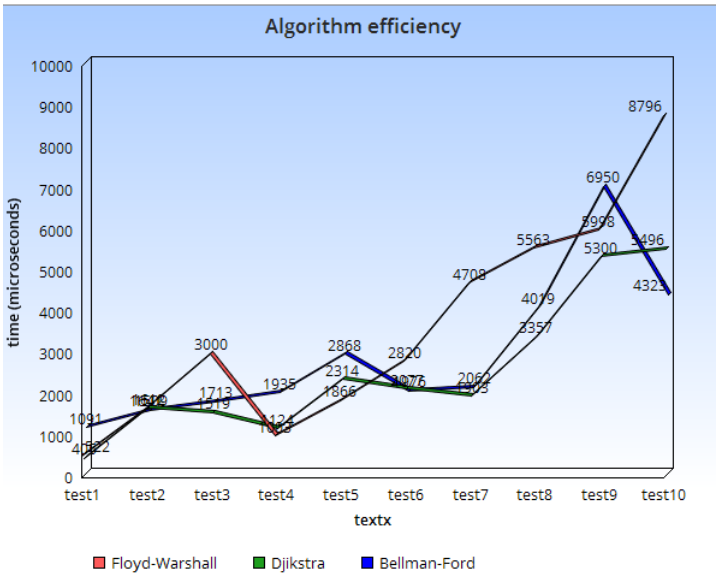

Fig.4. Comparison between time efficiency of the algorithms

3.4 Presentation of the values given by the test

Even though Djikstra and Bellman-Ford have a better performance, the nature of the test is a very important factor too. As we can see in the graphic above, the graphics tend to look a lot like each other at the beginning, although in the end, when the input is bigger, they start to differ and Dijkstra appears to be the most efficient on the bigger tests.

Another important factor is the system specification on which the program is launched. Even if the input is the same, there will never be the same output because of the differences made by the hardware.

4 Conclusions

In a real life situation, one of the most common usage of graphs and minimal paths in a graph is, as I said, the GPS. If I would develop a GPS, I would make the most of the Dijkstra algorithm because of its relatively good performance and because there are not negative weight edges on a map.

Another practical usage of a graph-based algorithm would be in the business domain. The weight if an edge could represent data used to evaluate the

effectiveness of sales campaigns. In this situations I would go for Bellman-Ford because if more efficient than Floyd-Warshall and it works on negative weight edges.

In conclusion, the choice you make for the algorithm really depends on the situation, each one has its own advantages and disadvantages and it can be used in different situations.

References

1. https://www.quora.com/What-are-the-uses-of-a-graph-in-real-life
2. https://www.geeksforgeeks.org/bellman-ford-algorithm-dp-23/
3. https://www.geeksforgeeks.org/dijkstras-algorithm-for-adjacency-list-representation-greedy-algo-8/
4. https://www.hackerearth.com/practice/algorithms/graphs/shortest-path-algorithms/tutorial/
5. https://www.geeksforgeeks.org/floyd-warshall-algorithm-dp-16/
6. https://www.geeksforgeeks.org/graph-data-structure-and-algorithms/
7. Thomas H. Cormen, Charles E. Leiserson, Roland L. Rivest, Cliord Stein: Introduction to Algorithms, 3rd edition, The MIT Press, Massachusetts Institute of Techology (2009).
8. https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/