Bellman-Ford and Dijkstra's Algorithm

José Tobias

June 2018

1 Introduction

On this report I will present a quick analysis of my implementation of the Bellman-Ford and Dijkstra's algorithm.

The code can be found at: https://goo.gl/YB4NWC.

The document is divided as follows: on the first part I will discuss the performance analysing a few input instances and the way it was implemented, followed by a short conclusion.

2 Analysis

As a complexity analysis, we see that on Bellman-Ford algorithms we have two nested loops, an external loop with time $\Theta(V)$, and a most internal loop with time $\Theta(E)$, therefore the algorithm runs in $O(VE)$.

The Dijkstra's algorithms also has two nested loops. The most external one with time $\Theta(V)$. The most internal one require a deep analysis, but his worst case is to iterate over a whole array of all edges at the start which decrease one unity in size every external loop repetition, which would lead us to a complexity of $O(V^2 + E) = O(V^2)$. Sparse matrix, implementations and case-scenarios can change this time, the full discussion can be found at the reference book.

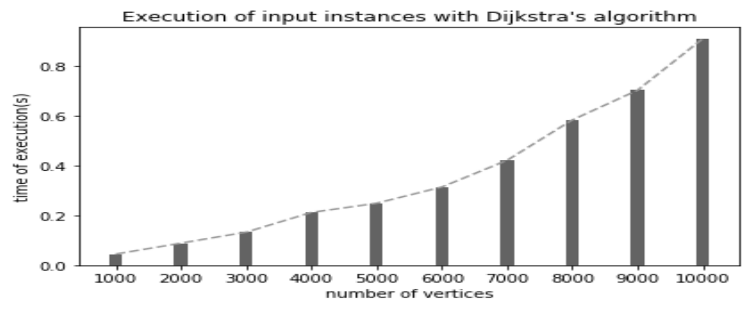

Figure 1: Time consumption to algorithm execution with multiple input instances

Running the algorithm on instances of vertices, that forms a simple and complete graph, we see on Figure 1, that there is no linearity on execution time, since the edge numbers go up exponentially. Even though I did used a fairly fast implementation of the priority-queue for the problem, we can just measure that this is the part that the algorithm consumes more time, by selecting which vertex to take out. All loops on the algorithm run on $O(1)$, while my priority queue has a complexity of $O(lgN)$.

By the nature of a simple and complete graph, the visual observation of any single-source shortest path algorithm on time against number of vertices will look like the Figure 2 below, just because of the fact that the shortest path between two points is a straight line.

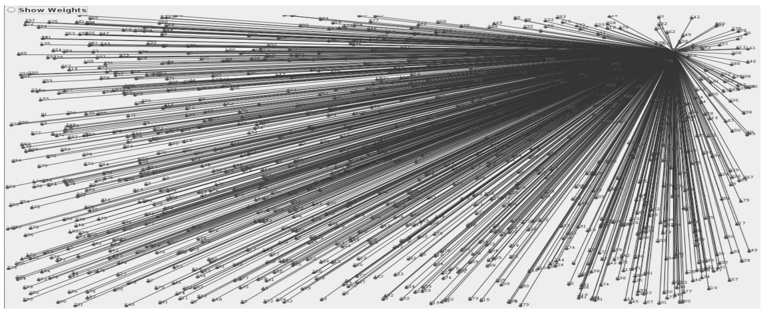

Figure 2: Example of pattern to be followed by any simple complete graph when processed by a single-source shortest path algorithm

3 Conclusion

The implementation is fairly simple and the code produced by the needed operations is small. Even though, those implementations of a shortest path algorithm are really useful as showed on classes, used by an immense amount of applications to produce solutions to many theoretical Computer Science areas and also widely used real-world applications.