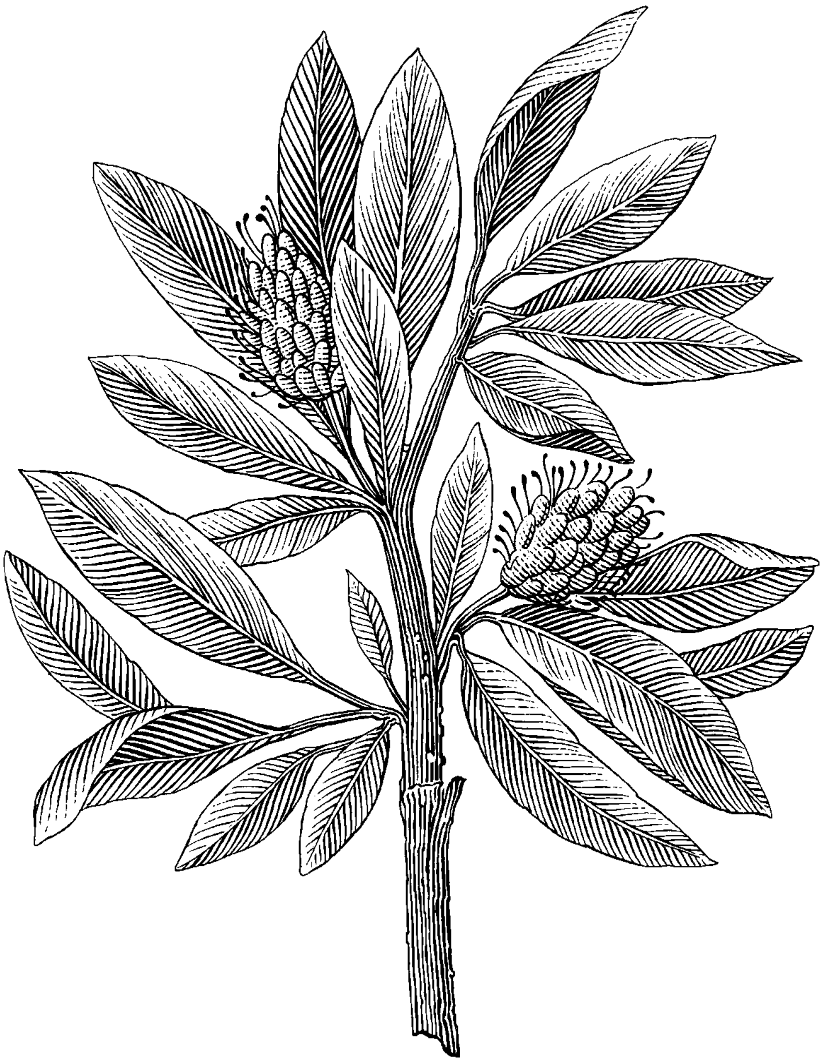
Linnéuniversitetet
Kalmar Växjö

Report

Assignment 4

1DV516



Supervisor: Morgan Ericsson
Semester: Autumn 2023

Author1: Katarina Simakina
Email es225hi@student.lnu.se

Author2: Zejian Wang
Email zw222bb@student.lnu.se

Table of Contents

Task 4

In this task, Dijkstra and Bellman-Ford algorithm are implemented to find the shortest path from a source vertex to other vertices in a graph. However Dijkstra cannot work with graphs with edges of negative weight because of its greedy nature which cannot handle negative values, hence in our implementations of Dijkstra, there are only non-negative weight edges. In Dijkstra java class main method, all the edges in a graph performed are non-negative weight to ensure the result of the algorithm performs is correct. In our Dijkstra implementation, it uses a priority queue which is based on min heap to maintain the set of vertices to be explored, it selects the vertex with the smallest distance from the source and updates the distances of its adjacent vertices.

Bellman-Ford can work with graph with negative edges but cannot handle ones with negative cycles because during the relaxation step, it iterates through all edges in the graph and updates the shortest distance to each vertex if a shorter path is found. This process takes into account negative edge weights and adjusts the distances accordingly. The iteration will take at most v-1 times(v is the number of vertices in a graph) and the distances of each vertex is not changed, however if the graph has negative cycle, the distances will continue to update after v-1 iterations and on. Because of this property of Bellman-ford, it can be used to detect negative cycle. In our implementation in Bellman-Ford class, isNegativeCycle method is to check if there is a negative cycle in a graph. In Bellman-Ford class main method, to test if there is a negative cycle in a graph and to be more straightforward and clearly, instead we manually creates two graphs, one has negative edges but no negative cycle, one has negative edges and negative cycles.

In the Main class, we measured the time taken by the two algorithms performing some randomly generated graphs. To compare the time two algorithms taking, the same graph should be performed. As Dijkstra only work with non-negative edges, all edges the randomly generated graphs have are non-negative. In our experiments, the number of vertices in a graph are randomly generated is set in the range of 50-100, the weight of an edge is randomly generated is set in the range of 100-200. Ten graphs are computed and one of the running result shows in table below:

Graph size(number of vertices)	Dijkstra Performed time(ns)	Bellman-Ford performed time(ns)
79	71899	249100
93	149500	747901
54	51101	103599
69	65000	217399
63	117801	263800
64	79700	231600
89	135900	736800
83	113399	444100
81	181600	522100
98	2231000	3935900

The above represents the execution time of Dijkstra's algorithm and Bellman-Ford algorithm on graphs with varying numbers of vertices. Bellman-Ford algorithm shows significantly higher execution times compared to Dijkstra's algorithm, Dijkstra's algorithm consistently outperforms Bellman-Ford algorithm in terms of execution time across all graphs.

The reason for the time difference between the two algorithm can be obtained by the analysis of the time complexity of each algorithm. The time complexity of the our Dijkstra's algorithm implementation is O((V + E) log V), where V is the number of vertices and E is the number of edges in the graph, because it visits all vertices, for each visits it explores the adjacency list of edges which takes O(E) time, and updating an vertex in the priority queue (heap operations) takes O(log V) time.

Comparing to the time complexity of Dijkstra, Bellman-Ford algorithm implementation is O(V*E), because it runs for V - 1 iterations, and each edge is relaxed in each iteration in the worst case, resulting in E operations. Hence, Dijkstra is faster than Bellman-ford, the same as it showed from the actual execution time.

In conclusion, the performance analysis clearly demonstrates that Dijkstra's algorithm is more efficient than Bellman-Ford algorithm, Dijkstra's algorithm is a preferred choice for finding shortest paths in graphs with non-negative edges. Bellman-Ford algorithm, while capable of handling negative edge weights and detecting negative cycles, shows slower performance, particularly in larger graphs. When choosing between these algorithms, one should take into account the particular needs, such as the demand for real-time responses or the existence of negative weights.