**Bellman-Ford algorithm – shortest path – a practical approach**

Student: Damir Rahmani

Mentor: prof. dr. Andrej Stefanov

**Extended Abstract**

The Bellman-Ford algorithm is a popular algorithm used to find a shortest path in a graph that may have negative cycles. This algorithm works by iteratively relaxing edges in the graph, meaning that it tries to improve the estimate of the shortest path between two nodes by considering longer paths that have more edges.

In this extended abstract, we present a practical approach to implementing the Bellman-Ford algorithm in three parts:

The first part of our implementation is a C++ implementation of the basic structure of the Bellman-Ford algorithm. We start by defining a struct to represent a single edge in the graph, which contains the source and destination nodes as well as the weight of the edge. We then define a function that takes a graph represented as a vector of edges and the number of nodes in the graph, and returns the shortest distances from a starting node to all other nodes in the graph. This function uses a nested loop to iterate through all the nodes in the graph, and within each iteration, it iterates through all the edges in the graph and relaxes each edge.

The second part of our implementation is an Octave implementation of the Bellman-Ford algorithm. Octave is a high-level programming language that is similar to Matlab, and is particularly well suited to numerical computations. Our implementation in octave closely follows the structure of the C++ implementation. In particular, we represent the graph as a matrix where each row represents an edge, and we use matrix multiplication to perform the edge relaxation step.

The third and final part of our implementation is an improved version of the C++ Bellman-Ford algorithm that uses priority queues to speed up the algorithm. The basic idea behind this improvement is to prioritize edges that are likely to lead to shorter paths, so that we can terminate the algorithm as soon as we have found the shortest path to a node. Specifically, we maintain a priority queue of nodes that need to be relaxed, where each node is associated with a distance estimate. We initially insert the starting node into the queue with a distance estimate of zero, and then iteratively extract the node with the smallest distance estimate and relax all the edges incident to that node. This process continues until the queue is empty or we have found the shortest path to the target node.

The Bellman-Ford algorithm is a popular algorithm used to find the shortest path in a graph that may have negative weights. This algorithm works by iteratively relaxing edges in the graph, meaning that it tries to improve the estimate of the shortest path between two nodes by considering longer paths that have more edges.

In this extended abstract, we present a practical approach to implementing the Bellman-Ford algorithm in three parts.

The first part of our implementation is a C++ implementation of the basic structure of the Bellman-Ford algorithm. We start by defining a struct to represent a single edge in the graph, which contains the source and destination nodes as well as the weight of the edge. We then define a function that takes a graph represented as a vector of edges and the number of nodes in the graph, and returns the shortest distances from a starting node to all other nodes in the graph. This function uses a nested loop to iterate through all the nodes in the graph, and within each iteration, it iterates through all the edges in the graph and relaxes each edge.

The second part of our implementation is an Octave implementation of the Bellman-Ford algorithm. Octave is a high-level programming language that is similar to Matlab, and is particularly well-suited to numerical computations. Our implementation in Octave closely follows the structure of the C++ implementation, but takes advantage of Octave's built-in matrix operations to simplify some of the computations. In particular, we represent the graph as a matrix where each row represents an edge, and we use matrix multiplication to perform the edge relaxation step.

The third and final part of our implementation is an improved version of the C++ Bellman-Ford algorithm that uses priority queues to speed up the algorithm. The basic idea behind this improvement is to prioritize edges that are likely to lead to shorter paths, so that we can terminate the algorithm as soon as we have found the shortest path to a node. Specifically, we maintain a priority queue of nodes that need to be relaxed, where each node is associated with a distance estimate. We initially insert the starting node into the queue with a distance estimate of zero, and then iteratively extract the node with the smallest distance estimate and relax all the edges incident to that node. This process continues until the queue is empty or we have found the shortest path to the target node.

In conclusion, we have presented a practical approach to implementing the Bellman-Ford algorithm in three parts: a basic C++ implementation, an Octave implementation, and an improved C++ implementation using priority queues. These implementations can be used to solve a wide range of problems that involve finding the shortest path in a graph, particularly in scenarios where the graph may have negative weights.

**References**

“Introduction to Algorithms” - Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein – Chapter 24

GeeksforGeeks website - <https://www.geeksforgeeks.org/bellman-ford-algorithm-dp-23/>