Finding the shortest path ending in every node in a directed graph in O(VE)

Liviu Silion

Let G = (V, E) be a directed graph and w : E ⟶ R a weight function. We want to find for every node n, the shortest path starting from every other node and ending in n. An algorithm with O(VE) time complexity will be presented.

# The graph has no cycles

Just like in Johnson’s algorithm, we will add a fictitious node s which will have an edge to every node from the original graph with weight 0. We will then run an algorithm for computing shortest paths with negative weights and obtain the array d[i] = shortest path from s to i. This can be done in O(VE) using Bellman Ford, which is the best-known algorithm for unbounded weights. Because the graph is acyclic, the answer for every node n will be min(d[x]+w(x,n)), where x is a node such that there exists an edge from x to n. The last part takes O(E) time, making the whole algorithm O(VE+E)=O(VE).

# The graph only has negative cycles

If we encounter a negative cycle, the answer for every node accessible from that cycle will be negative infinity. Therefore, we can apply the rest of the algorithm as usual after ignoring the affected nodes.

# The graph has non-negative cycles

The issue we can run into by obliviously applying the previous steps is that when calculating d[x]+w(x,n) the shortest path to a node x already includes the node n. In fact, the real shortest path we calculated would start from n and end in n. This follows immediately from the fact that there are no negative cycles left. This shortcoming can be fixed by recalculating d for every node.

We will consider the shortest path (weighted) tree created by running the shortest path algorithm starting from the fictitious node s. Some key facts about this tree are:

* Every node n for which we run into the aforementioned issue will be a direct child of s (s is considered to have depth 0 and n depth 1).
* Every edge between a node with depth 1 and 2 will have negative weight.
* What is more, if x is a node with depth > 1 then the shortest path from s to x will have negative weight
* We only need to recalculate d for descendants of n in the tree, which we’ll consider the current subtree; we only need to look from edges outside the subtree into the subtree and edges entirely in the subtree. By removing the edges coming out of n in the subtree (which we know are negative), the length of the shortest path to other nodes in the subtree increases, and more importantly an edge from inside the subtree to outside the subtree contradicts the construction of the tree itself.

By cutting every node coming from n and reapplying Bellman Ford each time on just the current subtree, we can recalculate the needed parts of d in O(VE) for all nodes. This part doesn’t increase the time complexity.

In practice, using an optimised version of Bellman Ford (SPFA or similar) yields an average time complexity of O(E).

Diagram, schematic

Description automatically generated