Bellman Ford and Shortest Path Fast Algorithm

Hello, and welcome back.

Today we will talk about the Bellman Ford algorithm and an optimization called Shortest Path Fast Algorithm.

Objectives

Your objectives are to be able to describe why we need these algorithms and to be able to implement them.

Normal Dijstra.

As a review, look at this graph from the Dijkstra lecture. I've modified it just a bit so that the example will be more interesting. You should pause the video and work out what Dijkstra's algorithm would do to this graph. Resume when you are ready.

Right. So here is the graph and the MST that Dijkstra would produce.

Now we are going to introduce a wrinkle: we are going to change one of these edges to make it negative.

Negative Edge Dijkstra.

The `d-f` edge is negative now, at -6, and for good measure I made the `f-c` edge to be 1. Try doing Dijkstra's algorithm on this graph and see what happens. Resume when you are ready.

Here's the final result. You noticed that the `c` node got updated as a result of the negative edge. Since the edge from `c` to `f` is greater than 4, we don't end up updating `d` as a result, so the algorithm terminates.

Negative Cycle Dijkstra

Now let's go crazy. The nodes `c`, `d`, and `f` are part of a cycle of negative weight. Try Dijkstra on this and see what happens. Resume when you are ready.

So, we were able to get good distances on some of the nodes, but the `c-d-f` cycle would cause Dijkstra to loop forever.

So, if there is a possibility of a negative cycle, then Dijstra is not a good idea. The classic algorithm for this is called Bellman Ford.

Bellman Ford implementation

Here is the code for Bellman Ford. I don't think you actually need an animation for it. The idea is simple: for every vertex in the graph, we relax *every single edge*. Any paths that are valid will have correct answers, and nodes that are part of negative cycles will have what amounts to garbage in them. We can take one more pass through the edges if we want; anything that would cause another relax operation to occur would indicate that the node is part of a negative weight cycle.

The problem with this algorithm, though, is that it is slow. The painful thing is that for normal graphs, most of the relax checks are going to do nothing. Fortunately, there is a nice optimization that can help.

Shortest Path Fast Algorithm.

Here is the Shortest Path Fast Algorithm. It is very similar to the Dijkstra's algorithm you know and love but we make two changes. First, we no longer enqueue a pair consisting of the node and its distance, we just enqueue the node. Second, we only enqueue a node if it is not currently in the queue. We use the `in_queue` vector to keep track of this for us.

Finally, when we dequeue something, we try relaxing all the edges, and anything that gets relaxed gets queued.

If there is a negative edge that relaxes a node, it will get put back into the queue for another round. But if there is a negative cycle, then all the nodes in the cycle will get added back to the queue. To detect if there is a negative cycle, we can keep track of how many times a node gets added to the queue. If it is V or greater we know there is such a cycle.

The nice thing about this algorithm is that on normal graphs it runs as fast as Dijkstra's algorithm, and on graphs with negative cycles it usually runs faster than Bellman Ford, since only the nodes affected by the negative weight cycle are going to be reprocessed. There's no guarantee that this won't involve the whole graph, but if it doesn't, it will benefit from that situation.

Next
Transcript — BFS for Shortest Paths →