# The Algorithm

This is just the algorithm taught in class for minimum path cost between two nodes in a graph without negative cycles. The reason we imposed that we use this algorithm on a graph without negative cycles is that the minimum cost then might be negative infinity because you could just loop over the negative cycle infinitely many times to get decreasing costs each time, so you can’t limit the edge count to . However, putting an edge limit on it would still make it work. The algorithm stays the same:

Given the graph G, let’s call the set of all the vertices of the graph V and all the edges E. Let us denote the optimal path from a node v to node t (the ending node given in the problem) as using at most i edges as OPT(i, v). We now need to express OPT(i,v) using smaller subproblems. There are two different options:

1. If the optimal path uses at most i-1 edges, then OPT(i,v) = OPT(i-1, v).
2. If the path P uses i edges, and the first edge is (v,w), then OPT(i,v) = edge\_weightcw + OPT(i-1,w).

This leads to the following recursive formula:

Using this recurrence, we get the following dynamic programming algorithm to compute the value OPT(k,s).

Shortest-Path(G,s,t,k)

Array M[0 ... k, V]

Define M[0,t] = 0 and M[0,v] = ∞ for all over v ∈ V

For i = 1, ..., k

For v ∈ V in any order

node w = the node that minimizes M[i-1,w] + edge\_weight(v,w)

M[i,v] = min(M[i-1,v], M[i-1,w] + edge\_weight(v,w))

Endfor

Endfor

Return M[k,s]

# Proof of Correctness

We prove by strong induction on i (number of edges used).

## Base Case

We want to prove that for all nodes v, M[0,v] is the optimal path cost if we were allowed to use at most 0 edges and started from v and ended at t. By definition,

M[0,t] = 0

M[0,v] = ∞ for all nodes not t

Which is correct because the shortest path from t to t using no edges should be 0, and the shortest path from other nodes v to t using no edges is impossible, thus infinity.

## Inductive Case

We need to prove M[i,v] = OPT(i,v).

Since we’re using strong induction, we can assume that for all h < i and for all v V, M[h,v] is the optimal path weight from v to t using at most h edges. Thus,

Which is a more complicated way of saying

Which was our recurrence definition given above.

# Runtime Analysis

Iterating through M is O(nk) because there’s k columns for the k edges we’re allowed to traverse and n nodes. Updating each cell in M is O(n) because there’s at most n nodes we have to consider. So altogether, the complexity is

1. Let’s consider the first part for now. Finding the minimum cost path from s to t if G has no negative cycles is just having columns in our table M and returning because, as proved in class and stated in the textbook, “If G has no negative cycles, then there is a shortest path from s to t that is simple (i.e., does not repeat nodes), and hence has at most n − 1 edges”. The time complexity for this is (also proved in class).

Now let’s consider detecting a negative cycle. To return the negative cycle, we need two things, to know that there IS a negative cycle, and the actual path (i.e. node sequence), not just the weight, from each node to t. To address the former, section 6.10 of the textbook proves that if we let the algorithm above run for one more iteration (have a column ), and see if any of the cells in that column are different from their corresponding cells in the same row but prior column (column ). If anything differs, then we have a negative cycle (according to Piazza post [@101](https://piazza.com/class/jzqwd6s59yh6bm?cid=101), we don’t have to prove what’s in the textbook already).