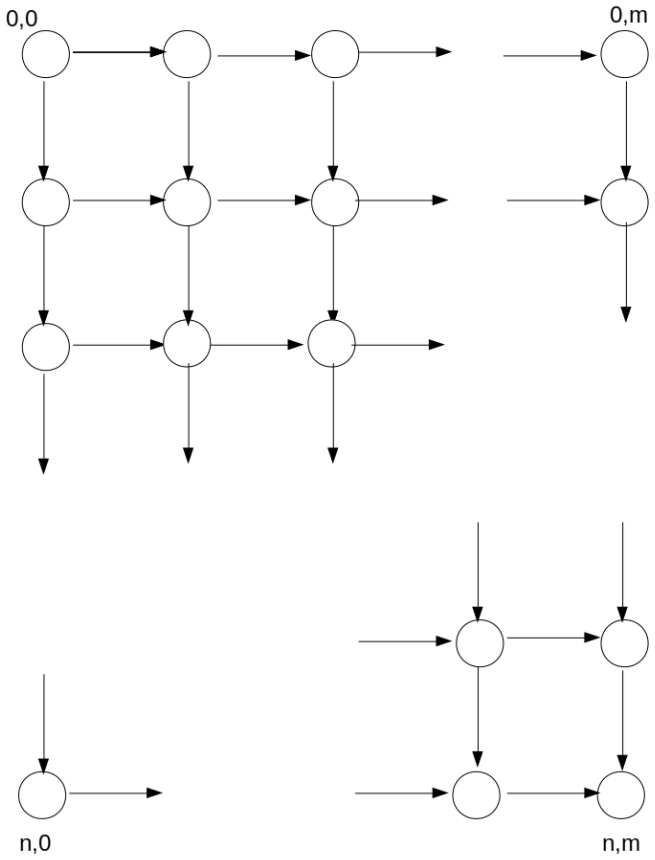
21090226

A shortest path problem:

Suppose we have a rectangular grid with rows numbered 0,...,n and columns numbered 0,...,m

- with (0,0) in the top left corner and (n,m) in the bottom right corner
- with costs associated with all edges - notation: w(a,b : c,d) is the edge-cost (or weight) of the edge from (a,b) to (c,d)
- with all horizontal edges pointing to the right
- and with all vertical edges pointing downward



Our goal is to find the least-cost path from (0,0) to (n,m) (NB: this is one of the few instances where 0-based indexing is actually useful - it simplifies the calculation of the number of

possible solutions.)

From the structure of the grid we can see that every path from (0,0) to (n,m) contains exactly n "down" edges and exactly m "right" edges. The total number of different paths from (0,0) to (n,m) is equal to the number of ways of interleaving the "down" and "right" edges. To get a sense of how many paths exist we can apply the following somewhat casual analysis:

For simplicity assume that n = m . Then the number of different paths is clearly $\binom{2n}{n}$ which is $\frac{(2n)!}{(n!n!)}$ This equals $\frac{((2n)(2n-1)...(n+1))}{(n(n-1)(n-2)...1)}$. Pairing each term in the numerator with the corresponding term in the denominator, we get

$$(\frac{2n}{n}) * (\frac{2n-1}{n-1}) * (\frac{2n-2}{n-2}) * ... * (\frac{n+2}{2}) * (\frac{n+1}{1})$$

Each term in this expression is clearly ≥ 2, so the product is $\geq 2^n$. Thus there are exponentially many potential paths - enumerating them all and choosing the least expensive (the BFI approach) is infeasible.

Instead, we make a clever observation: no matter what the optimum path looks like, the last step to (n,m) either comes downward from (n-1,m) or rightward from (n.m-1). So the Min_Cost path to (n,m) can be computed as follows

Min_Cost(n,m) = min { Min_Cost(n-1,m) + w(n-1,m : n,m),
Min_Cost(n,m-1) + w(n,m-1 : n,m) }

And each of Min_Cost(n-1,m) and Min_Cost(n,m-1) is based in the same simple fashion on the vertices immediately above and immediately to the left. In fact, the same holds true for almost all of the vertices in the grid. The only exceptions are the vertices in row 0, which can only be reached from their immediate left, and the vertices in column 0, which can only be reached from immediately above.

So if we compute the Min_Cost values to all vertices in row 0 (since there is only one path to each of these vertices, this is easy) and to all vertices in column 0 (ditto), we can then compute the Min_Cost value for all other vertices, one row at a time, using the values we have previously calculated.

Thus the Min_Cost value for each vertex is computed exactly once, and requires a constant amount of work to compute. So the total time to compute all the Min_Cost values, including

Min_Cost(n,m) is in O(n*m)

The key to formalizing the solution to this problem lies in correctly and precisely expressing it as a recurrence relation. We did part of this above, but we will now do it more completely.

Let Min_Cost(x,y) be the minimum total cost of a path from (0,0) to (x,y). It is important to observe (and obvious) that Min_Cost(x,y) is well defined - there is a minimum cost path from (0,0) to (x,y)

Min_Cost(0,0) = 0
Min_Cost(i,0) = Min_Cost(i-1,0) + w(i-1,0 : i,0) for i > 0
Min_Cost(0,j) = Min_Cost(0,j-1) + w(0,j-1 : 0,j) for j > 0
Min_Cost(i,j) = min { Min_Cost(i-1,j) + w(i-1,j : i,j), for i,j > 0
Min_Cost(i,j-1) + w(i,j-1 : i,j)
}

We can store the Min_Cost values in a 2-dimensional array MC, with MC[i][j] = Min_Cost(i,j)

Now we can fill in the table one row at a time, working left to right in each row. Each element depends on one or two previous elements, the values of which are already available. Thus each element is computed in constant time and the whole algorithm runs in O(n*m) time.

It is reasonable to ask why we need this algorithm when we already know that Dijkstra's Algorithm will find the shortest path from any vertex to any other vertex. People often forget that Dijkstra's Algorithm has a limitation – all edge-weights must be ≥ 0. If any edge-weights are < 0, Dijkstra's algorithm is not guaranteed to find the optimal solution.

However, our new algorithm ***is*** guaranteed to find the optimal solution for this problem, regardless of the edge-weights.