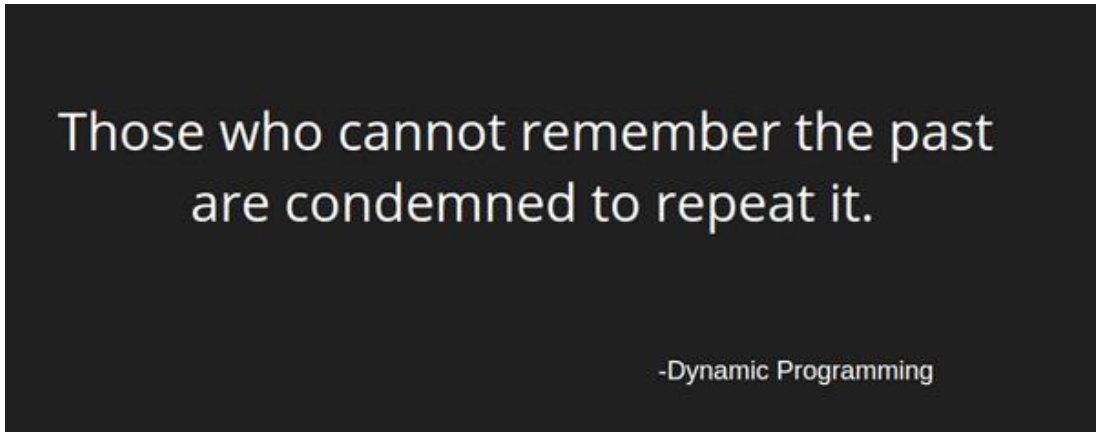
Dynamic Programming

Yulia Burkatovskaya

Outline

- The idea of dynamic programming
- The shortest path problem. Bellman algorithm



The idea of dynamic programming

- **How should I explain dynamic programming to a 4-year-old?**
- Jonathan Paulson, Software Engineer at Jump Trading

writes down "1+1+1+1+1+1+1+1 =" on a sheet of paper
"What's that equal to?"
counting "Eight!"
writes down another "1+" on the left
"What about that?"
quickly "Nine!"
"How'd you know it was nine so fast?"
"You just added one more"
"So you didn't need to recount because you remembered there were eight! *Dynamic Programming* is just a fancy way to say 'remembering stuff to save time later'"

The idea of dynamic programming

Dynamic Programming is a method for solving a complex problem by

- breaking it down into a collection of simpler subproblems,
- solving each of those subproblems just once, and
- storing their solutions using a memory-based data structure

If we don't know how to solve a problem, we divide it into simpler subproblems which we also don't know how to solve.

Richard E. Bellman

The idea of dynamic programming

Dynamic Programming can be used:

- If the given problem can be broken up in to smaller subproblems and these smaller subproblems are in turn divided in to smaller ones, and in this process, if you observe some overlapping subproblems;
- If the optimal solutions to the subproblems contribute to the optimal solution of the given problem.is a method for solving a complex problem by

The idea of dynamic programming

- Given a N×M matrix where each cell has a cost associated with it, find the maximum cost for a path from the top-left cell to the bottom-right cell.
- You can move only right and down.

7	2	8	6	8
3	8	6	5	7
2	11	13	6	5
5	7	12	9	11
8	14	3	13	2

The idea of dynamic programming

7	9	17	23	31

7	9	17	23	31
10	18	24	29	38

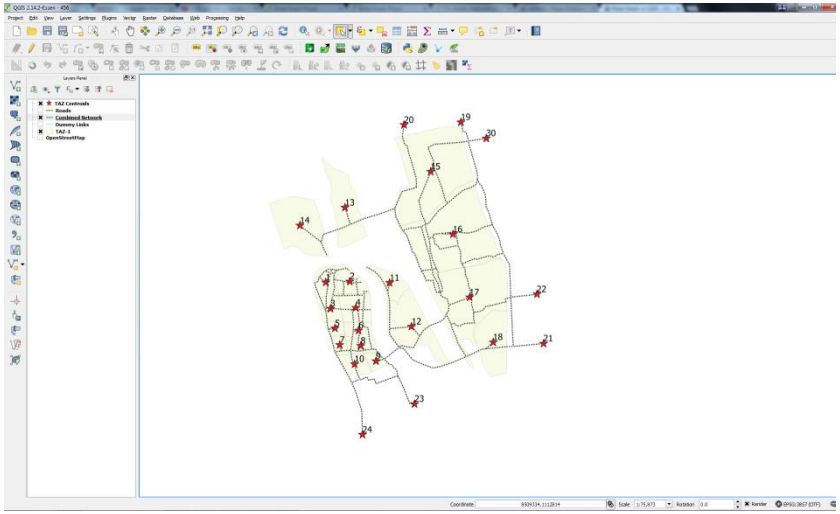
7	9	17	23	31
10	18	24	29	38
12	29	42	48	53

7	9	17	23	31
10	18	24	29	38
12	29	42	48	53
17	36	54	63	74
25	50	57	76	78

7	2	8	6	8
3	8	6	5	7
2	11	13	6	5
5	7	12	9	11
8	14	3	13	2

The shortest path problem

- The **shortest path problem** is one of the classical problems in graph theory that is due to its practical importance.
- Indeed, we constantly come across the search for optimal paths. For example, GPS-navigators search for a time-minimum path between two objects.
- **Example.** Q-GIS (road graph).



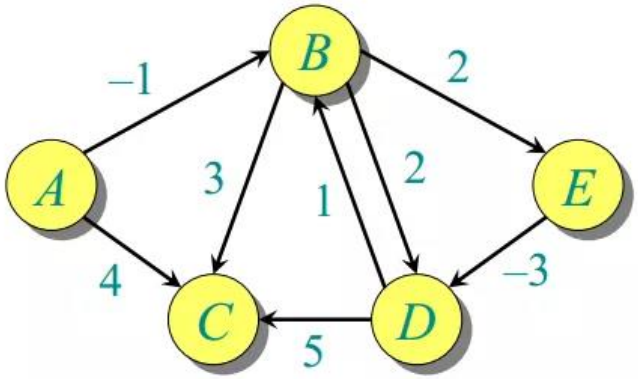
The shortest path problem

The Shortest paths problems

- The **single-pair shortest path problem**, in which we have to find shortest paths from a source vertex v to a single destination vertex u.
- The **single-source shortest path problem**, in which we have to find shortest paths from a source vertex v to all other vertices in the graph.
- The **single-destination shortest path problem**, in which we have to find shortest paths from all vertices in the directed graph to a single destination vertex v.
- The **all-pairs shortest path problem**, in which we have to find shortest paths between every pair of vertices v, u in the graph.

The shortest path problem

- A **weighted graph** associates a label (**weight**) with every edge in the graph.
- The **weight** $W(\langle u,v \rangle)$ of the path $\langle u,v \rangle$ is the sum of weights of the edges included in the path.
- The **shortest** path $\langle u,v \rangle$ in a weighted graph is a path of the minimum weight $W(\langle u,v \rangle)$.



The shortest path problem

In the **shortest path** problem:

- Any shortest path consists of shortest paths; i.e.,

if $< x, y > = x \ldots z \ldots y$

then it contains the shortest paths $< x, z >$ and $< z, y >$.

- The weight of the shortest paths it the sum of the weights of its subpaths; i.e. $W(x, y) = W(x, z) + W(z, y)$.

So, **we can use DP.**

Bellman-Ford algorithm

- **Initialization.** For the source, say s, set $\lambda(s) = 0$. For other vertices, set $\lambda(v) = \infty$.
- **Relaxation.** For every edge, say (u, v), try to decrease $\lambda(v)$:

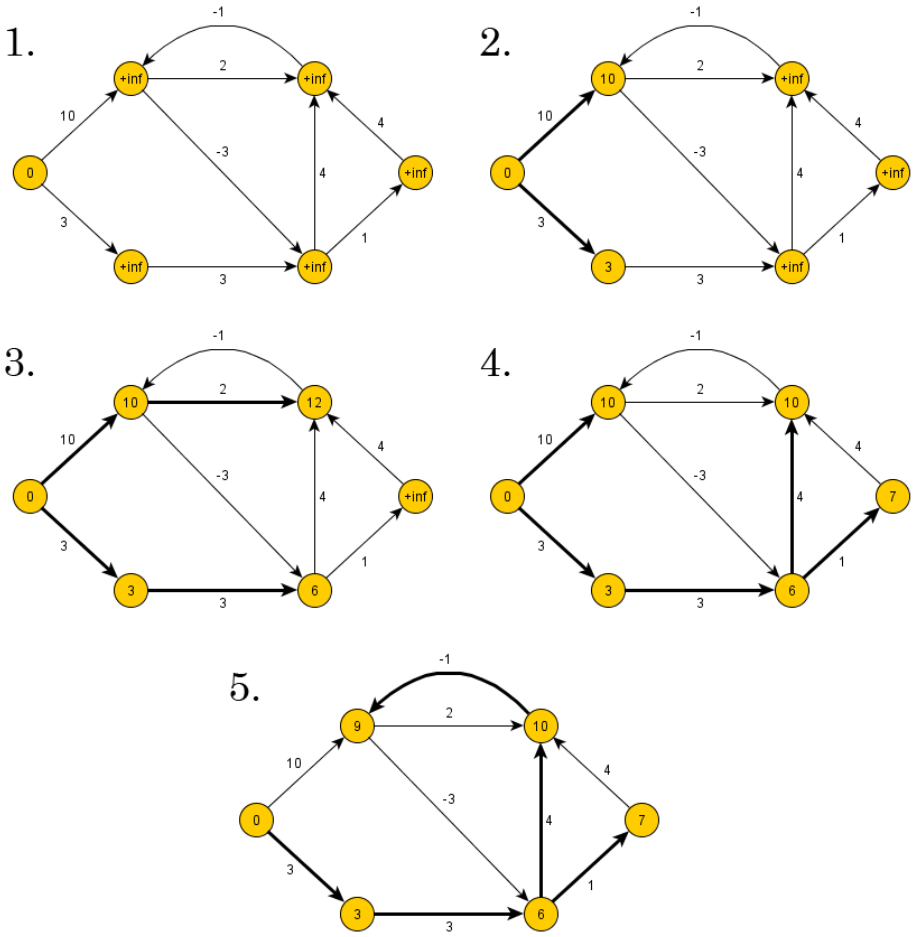
  if $\lambda(v) > \lambda(u) + W(u, v)$ then $\lambda(v) = \lambda(u) + W(u, v)$
- By using relaxation, we add **at most one edge** into the shortest path.
- If there are no negative cycles, the length of the shortest path is **at most $\boldsymbol{p - 1}$**, where p is the number of vertices.
- We repeat relaxation **$\boldsymbol{p - 1}$ times**.
- **Computational complexity** $O(qp)$.

Bellman-Ford algorithm

- **Pseudocode**

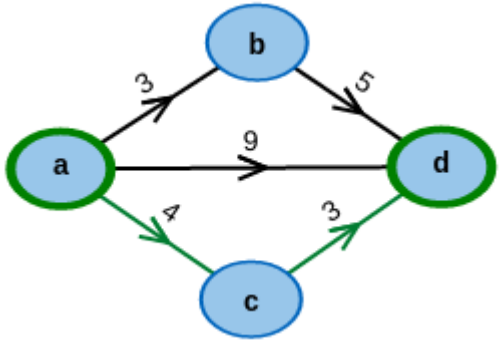
```
01.function <predecessors> Bellman-Ford(G, source)
02.for i in 1 to |U| do
03.distance[i] = +inf
04.predecessors[i] = null
05.
06.distance[source] = 0
07.
08.for i in 1 to (|U| - 1) do
09.   for each Edge e in Edges(G) do
10.           if distance[e.from] + length(e) < distance[e.to] do
11.                   distance[e.to] = distance[e.from] + length(e)
12.                   predecessors[e.to] = e.from
13.
14.for each Edge e in Edges(G) do
15.   if distance[e.from] + length(e) < distance[e.to] do
16.           error("Graph contains cycles of negative length")
17.
18.return predecessors
```

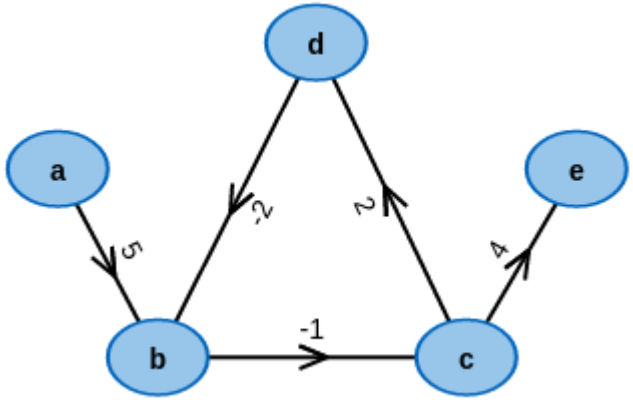
Bellman-Ford algorithm



Floyd-Warshall algorithm

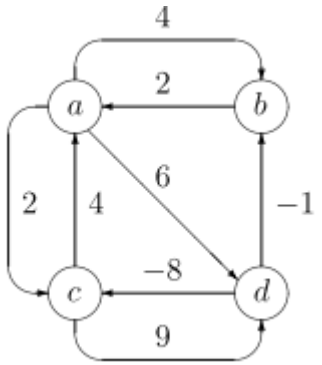
- The **Floyd-Warshall** algorithm is an example of dynamic programming. It breaks the problem down into smaller subproblems, then combines the answers to those subproblems to solve the initial problem.
- The **idea** is this: either the quickest path from A to C is the quickest path found so far from A to C, or it's the quickest path from A to B plus the quickest path from B to C.





Floyd-Warshall algorithm

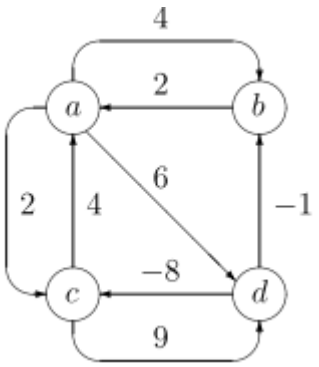
- **Pseudocode**
- Initial actions
- For i = 1 to p
  - For j = 1 to p
    - If $w_{i,j} < \infty$
    - then $\lambda_{i,j} = w_{i,j}$, $h_{i,j} = j$
      - else $\lambda_{i,j} = \infty$, $h_{i,j} = 0$
      - $\lambda_{i,i} = 0$, $h_{i,i} = i$



$$\Lambda = \begin{array}{c|cccc} & a & b & c & d \\ \hline a & 0 & 4 & 2 & 6 \\ b & 2 & 0 & \infty & \infty \\ c & 4 & \infty & 0 & 9 \\ d & \infty & -1 & -8 & 0 \end{array} \qquad H = \begin{array}{c|cccc} & a & b & c & d \\ \hline a & a & b & c & d \\ b & a & b & 0 & 0 \\ c & a & 0 & c & d \\ d & 0 & b & c & d \end{array}$$

Floyd-Warshall algorithm

- **Pseudocode**
- **Algorithm**
- For k = 1 to p
  - For i = 1 to p
    - For j = 1 to p
    - If $\lambda_{i,j} > \lambda_{i,k} + \lambda_{k,j}$ then
      - $\lambda i,j = \lambda i,k + \lambda k,j$
      - $hi,j = hi,k$



$$\Lambda = \begin{array}{c|cccc} & a & b & c & d \\ \hline a & 0 & 4 & 2 & 6 \\ b & 2 & 0 & \infty & \infty \\ c & 4 & \infty & 0 & 9 \\ d & \infty & -1 & -8 & 0 \end{array} \qquad H = \begin{array}{c|cccc} & a & b & c & d \\ \hline a & a & b & c & d \\ b & a & b & 0 & 0 \\ c & a & 0 & c & d \\ d & 0 & b & c & d \end{array}$$

Computational complexity: $O(p^3)$

$\lambda_{b,a} + \lambda_{a,b} = 2 + 4 = 6 > 0 = \lambda_{b,b};$

$\lambda_{b,a} + \lambda_{a,c} = 2 + 2 = 4 < \infty = \lambda_{b,c}, \quad \lambda_{b,c} = 4, \quad h_{b,c} = h_{b,a} = a;$

$\lambda_{b,a} + \lambda_{a,d} = 2 + 6 = 8 < \infty = \lambda_{b,d}, \quad \lambda_{b,d} = 8, \quad h_{b,d} = h_{b,a} = a;$

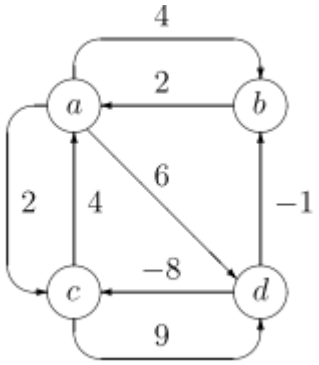
$\lambda_{c,a} + \lambda_{a,b} = 4 + 4 = 8 < \infty = \lambda_{c,b}, \quad \lambda_{c,b} = 8, \quad h_{c,b} = h_{c,a} = a;$

$\lambda_{c,a} + \lambda_{a,c} = 4 + 2 = 6 > 0 = \lambda_{c,c}$

$\lambda_{c,a} + \lambda_{a,d} = 4 + 6 = 10 > 9 = \lambda_{c,d};$

$$\Lambda = \begin{array}{c|cccc} & a & b & c & d \\ \hline a & 0 & 4 & 2 & 6 \\ b & 2 & 0 & 4 & 8 \\ c & 4 & 8 & 0 & 9 \\ d & \infty & -1 & -8 & 0 \end{array} \qquad H = \begin{array}{c|cccc} & a & b & c & d \\ \hline a & a & b & c & d \\ b & a & b & a & a \\ c & a & a & c & d \\ d & 0 & b & c & d \end{array}$$

Floyd-Warshall algorithm



$$\Lambda = \begin{array}{c|cccc} & a & b & c & d \\ \hline a & 0 & 4 & 2 & 6 \\ b & 2 & 0 & 4 & 8 \\ c & 4 & 8 & 0 & 9 \\ d & \infty & -1 & -8 & 0 \end{array} \qquad H = \begin{array}{c|cccc} & a & b & c & d \\ \hline a & a & b & c & d \\ b & a & b & a & a \\ c & a & a & c & d \\ d & 0 & b & c & d \end{array}$$

$$\Lambda = \begin{array}{c|cccc} & a & b & c & d \\ \hline a & 0 & 4 & 2 & 6 \\ b & 2 & 0 & 4 & 8 \\ c & 4 & 8 & 0 & 9 \\ d & 1 & -1 & -8 & 0 \end{array} \qquad H = \begin{array}{c|cccc} & a & b & c & d \\ \hline a & a & b & c & d \\ b & a & b & a & a \\ c & a & a & c & d \\ d & b & b & c & d \end{array}$$

$$\Lambda = \begin{array}{c|cccc} & a & b & c & d \\ \hline a & 0 & 4 & 2 & 6 \\ b & 2 & 0 & 4 & 8 \\ c & 4 & 8 & 0 & 9 \\ d & -4 & -1 & -8 & 0 \end{array} \qquad H = \begin{array}{c|cccc} & a & b & c & d \\ \hline a & a & b & c & d \\ b & a & b & a & a \\ c & a & a & c & d \\ d & c & b & c & d \end{array}$$

Links

- https://medium.com/@codingfreak/top-50-dynamic-programming-practice-problems-4208fed71aa3
- http://www.programming-algorithms.net/article/47389/Bellman-Ford-algorithm
- https://www.youtube.com/watch?v=obWXjtg0L64
- https://www.youtube.com/watch?v=4OQeCuLYj-4