Shortest Paths in Edge Weighted Graphs

Recall

\[ s \quad 1 \quad a \quad 3 \quad t \]

shortest path \( s \rightarrow t \) is \( s \rightarrow a \rightarrow t \) of length 3

Versions of the problem:

1. Given \( u, v \), find shortest \( u \rightarrow v \) path

2. Given \( u \), find shortest \( u \rightarrow v \) path for all \( v \)
   "single source shortest path problem"

3. Find shortest \( u \rightarrow v \) path for all \( u, v \)
   "all pairs shortest path problem" dynamic programming today

Recall—negative weight cycles are trouble!

2. Single Source Shortest Paths in Directed Graphs

- general weights (but no neg. cycle) \( O(n \cdot m) \)
  Bellman Ford — today 

- no cycles \( O(n+m) \) — today

- no negative weights \( O(m \log n) \)
  Dijkstra’s algorithm — last lecture
Single source shortest paths in a directed acyclic graph (DAG) - no directed cycle

Use topological sort \( v_1, \ldots, v_n \)

So every edge \((v_i, v_j)\) has \( i < j \)

If \( v \) comes before \( s \), there is no path \( s \rightarrow v \).

So throw those vertices away. \( s = v_i \)

Initialize \( d_i = \infty \) \( d_1 = 0 \)

for \( i = 1 \ldots n \)

for every edge \((v_i, v_j)\)

if \( d_i + w(v_i, v_j) < d_j \) then

\( d_j \leftarrow d_i + w(v_i, v_j) \)

end.

\( O(n + m) \)

**Claim** This finds shortest paths

**Proof** by induction on \( i \)
Dynamic Programming for Shortest Paths in Graphs.

Today we'll use dynamic programming for two problems:

- all pairs shortest paths, (2nd part of lecture).
  - Floyd-Warshall

- single source shortest paths where edge weights may be negative but the graph has no negative weight cycle. Bellman-Ford. The original application of dynamic programming.

Note: if there is a neg weight cycle then shortest paths are not well-defined. Go around the cycle move and move to decrease length of path arbitrarily.

Idea of dynamic programming for shortest paths:

we can try all $x$

if shortest $uw$ path goes through $xc$ then it consists of

shortest $ux$ path + shortest $xc$ path

these are subproblems
In what way are these subproblems "smaller"?

Two possibilities

1. They use fewer edges.
   This leads to dynamic programming, where we try paths of ≤1 edge, ≤2 edges, ...

2. They don't use vertex x.

We will pursue (1) for single source, (2) for all pairs.

**Single Source**

Let \( d_i(v) \) = length of shortest path from \( s \) to \( v \) using \( ≤ i \) edges.

Then \( d_1(v) = \begin{cases} 0 & \text{if } v = s \\ \infty & \text{else} \end{cases} \)

\( d_{i-1}(v) \) if \( (s,v) \in E \)

And we want \( d_{i-1}(v) \)

Why? Because a path with \( ≥ n \) edges would repeat a vertex \( s \) giving a cycle.

Since every cycle has weight \( ≥ 0 \), removing the cycle is at least as good.

We compute \( d_i \) from \( d_{i-1} \)

\[
d_i(v) = \min \left\{ di-1(v), \min_u \left( d_i-1(u) + w(u,v) \right) \right\}
\]
correctness: we consider all possibilities for $d_i$. Then correct by induction on $i$.

**Bellman Ford**

Initialize as above

For $i=2 \ldots n-1$

For each vertex $v$

$$d_i(v) \leftarrow d_{i-1}(v)$$

For each edge $(u, v)$

$$d_i(v) \leftarrow \min \{d_i(v), d_i-1(u) + w(u, v)\}$$

end.

Run Time $O(n \cdot (n + m))$

outer loop in inner loops we look at each edge and each vertex once.

We can save space — re-use same $d_i(v)$

Can also simplify code (and avoid issue * )

Initialize

$$d(v) = \infty \quad \forall v$$

$$d(s) = 0$$

For $i = 1 \ldots n$

For each edge $(u, v)$

$$d(v) \leftarrow \min \{d(v), d(u) + w(u, v)\}$$

end.

Note the curious fact that $i$ does not appear inside the loop.
Ex. See why this code does the same.
   (In this form, it is more mysterious why the code works)

In fact, we can exit from the top-level loop after an iteration in which no d value changes.
Ex. Justify this.

Can enhance the code to find actual shortest paths—just store parent pointers and update when d is updated. Allows us to recover path $s \rightarrow v$ backwards from $v$.

Can also use this code to detect negative weight cycle reachable from $s$.
How?
   Run 1 more iteration and see if any d value changes.
Ex. See why this works.
Ex. Show how to detect negative weight cycle anywhere in the graph.
   [So in add new $s'$ and add edges $(s',v) \forall v$, with weight 0.]
All pairs shortest paths.

Given digraph G with edge weights \( w : E \rightarrow \mathbb{R} \) (but no negative cycle),
find shortest path from \( u \) to \( v \) \( \forall u, v \).
Can output distances as \( n \times n \) matrix \( D[u, v] \).
Repeated Bellman-Ford gives \( O(n^2 m) \).
We'll use dynamic programming where intermediate paths use only a subset of the vertices.

Let \( V = \{ 1, \ldots, n \} \).
Let \( D_i [u, v] = \text{length of shortest } uv \text{ path using intermediate vertices in } \{1, \ldots, i\} \).

Solve subproblems \( D_i [u, v] \) for all \( u, v \)
as \( i \) goes from 0 to \( n \).
\( D_n [u, v] \) is what we really want.

Initial info:
\[
D_0 [u, v] = \begin{cases} 
0 & \text{if } u = v \\
\infty & \text{otherwise}
\end{cases}
\]

Main recursive property \( i > 0 \):
\[
D_i [u, v] = \min \left\{ D_{i-1} [u, i] + D_{i-1} [i, v] \right\} \quad \text{use vertex } i \\
D_{i-1} [u, v] \quad \text{don't}
\]

Correctness: this considers all possibilities for \( D_i \).
Then induction on \( i \).
Floyd-Warshall Algorithm
Initialize $D_0[u,v]$ as above
for $i = 1 \ldots n$
    for $u = 1 \ldots n$
        for $v = 1 \ldots n$
            
            $D_i[u,v] \leftarrow \min \{ D_{i-1}[u,v], D_{i-1}[u,i] + D_{i-1}[i,v] \}

        end
    end
end

Time $O(n^3)$
Space $O(n^3)$ - each $D_i$ for $i = 0 \ldots n$ is an $n \times n$ matrix

Exercise: Show that you can get $O(n^2)$ space by showing that the following works:
for $i = 1 \ldots n$
    for $u = 1 \ldots n$
        for $v = 1 \ldots n$
            
            $D[u,v] \leftarrow \min \{ D[u,v], D[u,i] + D[i,v] \}

        end
    end
what if we want the actual path?
Along with $D[u,v]$, compute $\text{Next}[u,v] = \text{the first vertex after } u \text{ on a shortest } u \text{ to } v \text{ path}$. 

If we update $D[u,v] \leftarrow D[u,i] + D[i,v]$ then also update $\text{Next}[u,v] \leftarrow \text{Next}[u,i]$

Example: Check how this works.
Bellman explains the reasoning behind the term *dynamic programming* in his autobiography, *Eye of the Hurricane: An Autobiography* (1984, page 159). He explains: "I spent the Fall quarter (of 1950) at RAND. My first task was to find a name for multistage decision processes. An interesting question is, Where did the name, dynamic programming, come from? The 1950s were not good years for mathematical research. We had a very interesting gentleman in Washington named Wilson. He was Secretary of Defense, and he actually had a pathological fear and hatred of the word research. I'm not using the term lightly; I'm using it precisely. His face would suffuse, he would turn red, and he would get violent if people used the term research in his presence. You can imagine how he felt, then, about the term mathematical. The RAND Corporation was employed by the Air Force, and the Air Force had Wilson as its boss, essentially. Hence, I felt I had to do something to shield Wilson and the Air Force from the fact that I was really doing mathematics inside the RAND Corporation. What title, what name, could I choose? In the first place I was interested in planning, in decision making, in thinking. But planning, is not a good word for various reasons. I decided therefore to use the word “programming”. I wanted to get across the idea that this was dynamic, this was multistage, this was time-varying. I thought, let's kill two birds with one stone. Let's take a word that has an absolutely precise meaning, namely dynamic, in the classical physical sense. It also has a very interesting property as an adjective, and that it's impossible to use the word dynamic in a pejorative sense. Try thinking of some combination that will possibly give it a pejorative meaning. It's impossible. Thus, I thought dynamic programming was a good name. It was something not even a Congressman could object to. So I used it as an umbrella for my activities."