ALL PAIRS SHORTEST PATHS (APSP) PROBLEM

Instance: A directed graph \( G = (V, E) \) and a weight matrix \( W \), where \( W[i, j] \) denotes the weight of edge \( e_{ij} \), for all \( i, j \in V, i \neq j \).

Find: For all pairs of vertices \( u, v \in V \), a directed path \( P \) from \( u \) to \( v \) such that

\[
\omega(P) = \sum_{e_{ij} \in P} W[i, j]
\]

is minimized.

We allow edges to have negative weights, but we assume there are no negative-weight directed cycles in \( G \).

EASY SOLUTION

Run Bellman-Ford \( n \) times, once for each possible source.

Output: Matrix \( D \) of shortest path lengths

\[
D[i, j] = \begin{cases} 0 & \text{if } i = j \\ \infty & \text{otherwise} \end{cases}
\]

Time complexity: \( O(n^3) \)

Space complexity is more tricky...

Home exercise: do we need to keep both \( D_{n-1} \) and \( D_n \)? Or can we reuse \( D_{n-1} \) directly as \( D_n \) entry, and modify it in place?

Note: this is asymptotically the same as input size for dense graphs where \( |E| \approx |V|^2 \).

A Dynamic Programming Approach

Suppose we successively consider paths of length \( 1, 2, \ldots, n-1 \). Let \( L_m[i, j] \) denote the minimum-weight \((i, j)\)-path having at most \( m \) edges.

We want to compute \( L_{n-1} \).

- Base case: \( L_0[i, j] = W[i, j] \)
- General case: How to express solution in terms of optimal solution to subproblems?

Express shortest path with \( m \) edges in terms of shortest paths with \( m-1 \) edges.

For \( m \geq 2 \),

\[
L_m[i, j] = \min\{L_{m-1}[i, k] + E_k[k, j] : 1 \leq k \leq n \}.
\]

Problem: we don’t know the predecessor of \( j \) on the optimal path \( P \).

Try all possible predecessors.

Arguing optimal substructure

Let \( P = \text{minimum weight } (i, j)\)-path with \( m \) edges

\[
w(P) = \min_{k \in V} \{w[P_k(k, j)] + E_k[k, j] : 1 \leq k \leq n \}.
\]

Then \( P = \text{minimum weight } (i, j)\)-path with \( m - 1 \) edges (or could think \( w(P) \) constant)

Algorithm: FastestSolutionPaths(W)

\[
L_n[i, j] = \begin{cases} 0 & \text{if } i = j \\ \infty & \text{otherwise} \end{cases}
\]

for \( m = 1 \) to \( n-1 \) do

\[
L_{m-1}[i, j] = \min_{k \in V} \{L_{m-1}[i, k] + E_k[k, j] : 1 \leq k \leq n \}.
\]

return \( L_{n-1} \)

Time complexity: \( O(n^3) \)

Home exercise: do we need to keep \( L_{n-1} \) and \( L_n \)? Or can we reuse \( L_{n-1} \) directly as \( L_n \) entry, and modify it in place?

Note: this is asymptotically the same as input size for dense graphs where \( |E| \approx |V|^2 \)...
**SUMMARY & WHAT'S NEXT**

**FLOYD-WARSHALL ALGORITHM**

Let $D_k(i, j)$ denote the minimum-weight $(i, j)$-path in which all interior nodes are in the set $\{1, \ldots, k\}$.

- **Base case:** $D_0 = W$

- **Recurrence:** $D_k(i, j) = \min(D_{k-1}(i, j) + D_{k-1}(k, j))$

**EXAMPLE**

$D_0 = (0 \ 3 \ \infty \ \infty) \ 
(\infty \ 0 \ 12 \ 5) \ 
(4 \ \infty \ 0 \ -1) \ 
(2 \ \infty \ \infty \ \infty)$

$D_1 = (0 \ 3 \ \infty \ \infty) \ 
(\infty \ 0 \ 12 \ 5) \ 
(4 \ \infty \ 0 \ -1) \ 
(2 \ \infty \ \infty \ \infty)$

$D_2 = (0 \ 3 \ 15 \ \infty) \ 
(\infty \ 0 \ 12 \ 5) \ 
(4 \ 7 \ 0 \ -1) \ 
(2 \ 4 \ 8 \ 0)$

$D_3 = (0 \ 3 \ 15 \ 8) \ 
(7 \ 0 \ 12 \ 5) \ 
(1 \ 5 \ 0 \ -1) \ 
(2 \ 4 \ 8 \ 0)$
FORMULATING GRAPH PROBLEMS

The RootBear Problem:
Suppose we have a canyon with perpendicular walls on either side of a forest.
- We assume a north wall and a south wall.
Viewed from above we see the A&I RootBear attempting to get through the canyon.
- We assume trees are represented by points.
- We assume the bear is a circle of given diameter d.
- We are given a list of coordinates for the trees.
Find an algorithm that determines whether the bear can get through the forest.

Bear cannot get through the canyon if North and South walls are connected.
Test connectivity using BFS from any point on the North wall, and checking if any point on the South wall is visited.

Exercise: what if each tree had radius 𝑟?

Reliable network routing:
- Suppose we have a computer network with many links.
- Every link has an assigned reliability.
  - The reliability is a probability between 0 and 1 that the link will operate correctly.
- Given nodes u and v, we want to choose a route between nodes u and v with the highest reliability.
  - The reliability of a route is a product of the reliabilities of all its links.

Reliability of path 𝑎−𝑏−𝑐−𝑑 = 0.5 * 0.9 * 0.75 = 0.3375
Higher reliability via path 𝑎−𝑏−𝑐− 𝑑 = 0.5 * 0.8 = 0.4

Graphs are a very important formalism in computer science. Efficient algorithms are available for many important problems:
- exploration,
- shortest paths,
- minimum spanning trees, etc.
If we formulate a problem as a graph problem, chances are that an efficient non-trivial algorithm for solving the problem is known.
Some problems have a natural graph formulation.
- For others we need to choose a less intuitive graph formulation.
- Some problems that do not seem to be graph problems at all can be formulated as such.

Can we turn this into a shortest path problem?
Problem 1: need product of weight, not sum
Use logs to turn product of weights into a sum.
Recall: log 𝑥 + log 𝑦 = log 𝑥𝑦. So log 𝑤 = log 𝑤.
log 𝑤 = log (2) = log 1 − log (−log 𝑤).
We want to minimize this!
Solution: create a new graph where each weight 𝑤 is replaced with weight⁻log 𝑤.

For each input point (x, y), add vertices (x, 0), (x, h), (x, y) to V.
For all pairs of vertices u, v in V:
If dist(u, v) < d, add edge uv.
Also add edges between all vertices on each canyon wall.

Exercise: what if each tree had radius 𝑟?

Reliability of path 𝑎−𝑏−𝑐− 𝑑 = 0.5 * 0.9 * 0.75 = 0.3375
Higher reliability via path 𝑎−𝑏− 𝑑 = 0.5 * 0.8 = 0.4

śniad

Problem 2: want to maximize the product
A path 𝑉 has maximum |𝑤|
iff it has maximum log |𝑤|
iff it has minimum log |−𝑤|.
Solution: create a new graph where each weight 𝑤 is replaced with weight⁻log 𝑤.

If weight 𝑤 ≤ 1 then log 𝑤 ≤ 0
⁻log 𝑤 ≥ 0
So we can use Dijkstra!
A MORE FORMAL OPTIMALITY ARGUMENT FOR YOUR NOTES

**Case 1:** $m$ is not used in $P$
- Interior nodes are all in $\{1, \ldots, m-1\}$
- Then $w(P) = D_{m-1}[i, j]$ by I.H. and $D_m[i, j] = D_{m-1}[i, j]$.

**Case 2:** $m$ is used in $P$
- Consider $P'$
- By I.H., $w(P') = D_{m-1}[i, j]$ and $w(P) = D_m[i, j]$
- And $w(P') + w(P) = D_{m-1}[i, j] + D_{m-1}[m, j] = D_m[i, j]$.

Reduce $P_1, P_2$ to subproblems but what if $m \in P_1, P_2$?

**(Details in slide notes)**

If $m$ appears twice in $P$, it creates a cycle which can be removed to get $P'$ with same or better weight.

Base case $D_0[i, j]$ is left as an exercise.