

Announcements

- Homework 5 Due on Friday
 - Slight correct to Q3: substring -> subsequence

Last Time

- Dynamic Programs
 - Family of Subproblems
 - Recursion Relation
 - Tabulate and Solve
- Proof of Correctness by Induction
- Runtime: [#subproblems][time/subproblem]
- Finding right subproblems is key

Today

- Chain Matrix Multiplication
- All-Pairs Shortest Paths

Chain Matrix Multiplication

How long does it take to multiply matrices?

Recall if $C = A \cdot B$ then

$$C_{xz} = \sum A_{xy} B_{yz}.$$

Suppose A is an $n \times m$ matrix and B is $m \times k$. Then for each entry of C (of which there are nk), we need to sum m terms.

Runtime $O(nmk)$ *

*Can do slightly better with Strassen, but we'll ignore this for now.

More than two Matrices

Next suppose that you want to multiply three matrices ABC.

Can do it two different ways,

$A(BC)$ OR $(AB)C$.

How long does it take?

Example

A is 2x3,

B is 3x3,

C is 3x1.

$$2 \cdot 3 \cdot 1 = 6 \quad \underbrace{\quad \quad \quad}_{A(BC)} \quad 3 \cdot 3 \cdot 1 = 9$$

Runtime: $9 + 6 = 15$

Multiplication
order matters!

$$2 \cdot 3 \cdot 3 = 18 \quad \underbrace{\quad \quad \quad}_{(AB)C} \quad \underbrace{\quad \quad \quad}_{2 \cdot 3 \cdot 1 = 6}$$

Runtime: $18 + 6 = 24$

Problem Statement

Problem: Find the order to multiply matrices $A_1, A_2, A_3, \dots, A_m$ that requires the fewest total operations.

In particular, assume A_1 is an $n_0 \times n_1$ matrix, A_2 is $n_1 \times n_2$, generally A_k is an $n_{k-1} \times n_k$ matrix.

Recursion

- We need to find a recursive formulation.
- Often we do this by considering the last step.
- For some value of k , last step:
 $(A_1 A_2 \dots A_k) \cdot (A_{k+1} A_{k+2} \dots A_m)$
- Number of steps:
 - $\text{CMM}(A_1, A_2, \dots, A_k)$ to compute first product
 - $\text{CMM}(A_{k+1}, \dots, A_m)$ to compute second product
 - $n_0 n_k n_m$ to do final multiply
- Recursion $\text{CMM}(A_1, \dots, A_m) =$
 $\min_k [\text{CMM}(A_1, \dots, A_k) + \text{CMM}(A_{k+1}, \dots, A_m) + n_0 n_k n_m]$

Subproblems

- What subproblems do we need to solve?
 - We cannot afford to solve all possible CMM problems.
- $\text{CMM}(A_1, \dots, A_m)$ requires $\text{CMM}(A_1, \dots, A_k)$ and $\text{CMM}(A_k, \dots, A_m)$.
- These require $\text{CMM}(A_i, A_{i+1}, \dots, A_j)$, but nothing else.
- Only need subproblems $C(i, j) = \text{CMM}(A_i, A_{i+1}, \dots, A_j)$ for $1 \leq i \leq j \leq m$.
 - Fewer than m^2 total subproblems.
 - Critical: Subproblem reuse.

Full Recursion

Base Case: $C(i,i) = 0$.

(With a single matrix, we don't have to do anything)

Recursive Step:

$$C(i,j) = \min_{i \leq k < j} [C(i,k) + C(k+1,j) + n_i n_k n_j]$$

Solution order: Solve subproblems with smaller $j-i$ first. This ensures that the recursive calls will already be in your table.

Example

Compute ABCD where

A is 2x5, B is 5x4, C is 4x3, D is 3x5

Finish

	A	B	C	D
A				
B				
C				
D				

$$2 \cdot 5 \cdot 5 + 5 \cdot 5 \cdot 3 + 5 \cdot 3 \cdot 5 = 180$$

$$2 \cdot 4 \cdot 5 + 5 \cdot 4 \cdot 3 + 5 \cdot 3 \cdot 5 = 165$$

$$= 140 \quad ((AB)C)D$$

$$2 \cdot 3 \cdot 5 + 64 = 94$$

Start

Runtime

Number of Subproblems: One for each
 $1 \leq i \leq j \leq m$. Total: $O(m^2)$.

Time per Subproblem: Need to check each
 $i \leq k < j$. Each check takes constant time. $O(m)$.

Final Runtime: $O(m^3)$

DP Setup

Sometimes there are many ways to create a DP for a given problem, and how exactly you set it up will have a large effect on runtime.

All Pairs Shortest Paths

Problem: Given a graph G with (possibly negative) edge weights, compute the length of the shortest path between every pair of vertices.

Note: Bellman-Ford computes single-source shortest paths. Namely, for some fixed vertex s it computes all of the shortest paths lengths $d(s,v)$ for every v .

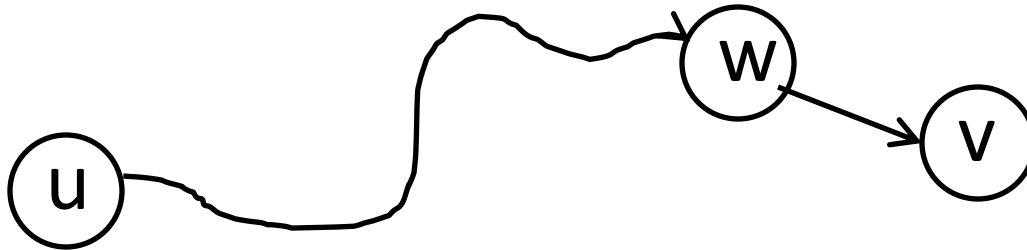
Repeated Bellman-Ford

Easy Algorithm: Run Bellman-Ford with source s for each vertex s .

Runtime: $O(|V|^2|E|)$

Dynamic Program

- Let $d_k(u,v)$ be the length of the shortest u - v path using at most k edges.

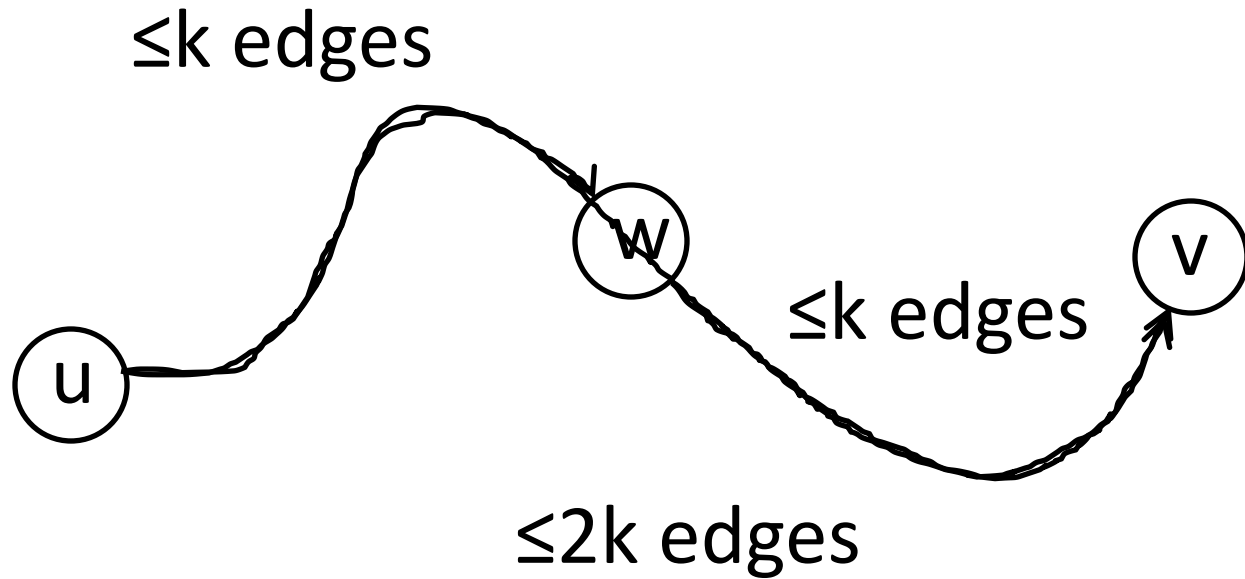


- Consider last edge.
- Length $k-1$ path from u to w , edge from w to v .
- $d_k(u,v) = \min_w [d_{k-1}(u,w) + \ell(w,v)]$

Matrix Multiplication Method

- Bellman-Ford is slow in part because we can only increase k by one step at a time.
- This happens because we cut off only the last edge of the optimal path.
- What if instead we cut it in the middle?

Recursion



$$d_{2k}(u, v) = \min_{w \in V} (d_k(u, w) + d_k(w, v)).$$

Algorithm

Runtime: $O(|V|^3 \log |V|)$

$O(|V|^2)$

Base Case:

$$d_1(u, v) = \begin{cases} 0 & \text{if } u = v \\ \ell(u, v) & \text{if } (u, v) \in E \\ \infty & \text{otherwise} \end{cases}$$

Recursion: Given $d_k(u, v)$ for all u, v compute

$$d_{2k}(u, v) \text{ using } d_{2k}(u, v) = \min_{w \in V} (d_k(u, w) + d_k(w, v)).$$

$O(|V|^3)$

End Condition: Compute $d_1, d_2, d_4, \dots, d_m$ with
 $m > |V|$.

$O(\log |V|)$ iterations

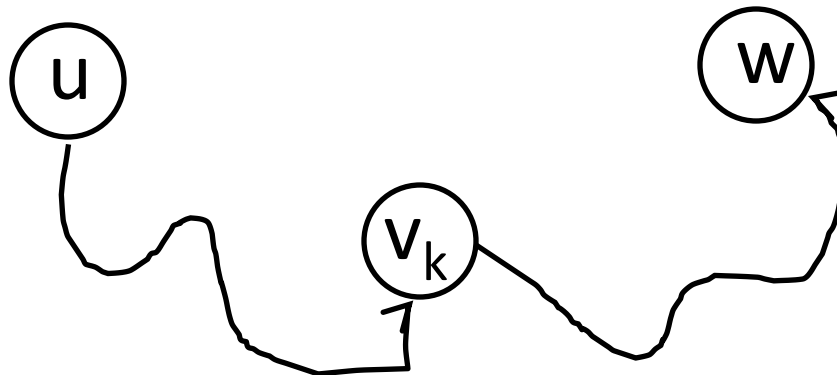
Floyd-Warshall Algorithm

- Label vertices v_1, v_2, \dots, v_n .
- Let $d_k(u, w)$ be the length of the shortest u - w path using only v_1, v_2, \dots, v_k as intermediate vertices.
- **Base Case:**
$$d_0(u, w) = \begin{cases} 0 & \text{if } u = w \\ \ell(u, w) & \text{if } (u, w) \in E \\ \infty & \text{otherwise} \end{cases}$$

Recursion

Break into cases based on whether shortest path uses v_k .

- The shortest path not using v_k has length $d_{k-1}(u,w)$.
- The shortest path using v_k has length $d_{k-1}(u,v_k) + d_{k-1}(v_k,w)$.



Algorithm

Runtime: $O(|V|^3)$

Base Case:

$$d_0(u, w) = \begin{cases} 0 & \text{if } u = w \\ \ell(u, w) & \text{if } (u, w) \in E \\ \infty & \text{otherwise} \end{cases}$$

$O(|V|^2)$

Recursion: For each u, w compute:

$O(|V|^2)$

$$d_k(u, w) = \min(d_{k-1}(u, w), d_{k-1}(u, v_k) + d_{k-1}(v_k, w)).$$

End Condition: $d(u, w) = d_n(u, w)$ where $n = |V|$.

$O(|V|)$ Iterations

Best Known Algorithm

Actually isn't DP.

1. Run `Bellman-Ford` once to compute $d(v)$.
2. Problem equivalent to using
$$\ell'(u,w) = \ell(u,w) + d(u) - d(w) \geq 0.$$
3. Run `Dijkstra` from every source.

Runtime: $O(|V||E| + |V|^2 \log |V|)$