Announcements

- Homework 4 Solutions online
- Homework 5 online due Friday

Last Time

- Dynamic Programming

Dynamic Programming

Our final general algorithmic technique:

1. Break problem into smaller subproblems.
2. Find recursive formula solving one subproblem in terms of simpler ones.
3. Tabulate answers and solve all subproblems.

Notes about DP

- General Correct Proof Outline:
  - Prove by induction that each table entry is filled out correctly
  - Use base-case and recursion
- Runtime of DP:
  - Usually
    [Number of subproblems]x[Time per subproblem]

More Notes about DP

- Finding Recursion
  - Often look at first or last choice and see what things look like without that choice
- Key point: Picking right subproblem
  - Enough information stored to allow recursion
  - Not too many

Today

- Chain Matrix Multiplication
- All Pairs Shortest Path

Chain Matrix Multiplication

How long does it take to multiply matrices?

Chain Matrix Multiplication

How long does it take to multiply matrices?

Recall if $C = A \cdot B$ then

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

Chain Matrix Multiplication

How long does it take to multiply matrices?

Recall if $C = A \cdot B$ then

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

Suppose A is an nxm matrix and B is mxk. Then for each entry of C (of which there are nk), we need to sum m terms.

Chain Matrix Multiplication

How long does it take to multiply matrices?

Recall if $C = A \cdot B$ then

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

Suppose A is an nxm matrix and B is mxk. 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.

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.

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.

Example

A is 2x3,

B is 3x3,

C is 3x1.

A(BC)

Example

A is 2x3,

B is 3x3,

C is 3x1.

A(BC)

$3 \cdot 3 \cdot 1 = 9$

Example

A is 2x3,

B is 3x3,

C is 3x1.

A(BC)

$3 \cdot 3 \cdot 1 = 9$

$2 \cdot 3 \cdot 1 = 6$

Example

A is 2x3,

B is 3x3,

C is 3x1.

A(BC)

$3 \cdot 3 \cdot 1 = 9$

$2 \cdot 3 \cdot 1 = 6$

Runtime: 9 + 6 = 15

Example

A is 2x3,

B is 3x3,

C is 3x1.

A(BC)

$3 \cdot 3 \cdot 1 = 9$

$2 \cdot 3 \cdot 1 = 6$

Runtime: $9 + 6 = 15$

(AB)C

Example

A is 2x3,

B is 3x3,

C is 3x1.

A(BC)

$3 \cdot 3 \cdot 1 = 9$

$2 \cdot 3 \cdot 1 = 6$

Runtime: 9 + 6 = 15

(AB)C

$2 \cdot 3 \cdot 3 = 18$

Example

A is 2x3,

B is 3x3,

C is 3x1.

A(BC)

$3 \cdot 3 \cdot 1 = 9$

$2 \cdot 3 \cdot 1 = 6$

Runtime: $9 + 6 = 15$

(AB)C

$2 \cdot 3 \cdot 3 = 18$

$2 \cdot 3 \cdot 1 = 6$

Example

A is 2x3,

B is 3x3,

C is 3x1.

A(BC)

$3 \cdot 3 \cdot 1 = 9$

$2 \cdot 3 \cdot 1 = 6$

Runtime: 9 + 6 = 15

(AB)C

$2 \cdot 3 \cdot 3 = 18$

$2 \cdot 3 \cdot 1 = 6$

Runtime: 18 + 6 = 24

Example

A is 2x3,

B is 3x3,

C is 3x1.

$A(BC)$

$3 \cdot 3 \cdot 1 = 9$

$2 \cdot 3 \cdot 1 = 6$

Runtime: $9 + 6 = 15$

Multiplication order matters!

$(AB)C$

$2 \cdot 3 \cdot 3 = 18$

$2 \cdot 3 \cdot 1 = 6$

Runtime: $18 + 6 = 24$

Problem Statement

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

Problem Statement

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

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

Recursion

- We need to find a recursive formulation.

Recursion

- We need to find a recursive formulation.
- Often we do this by considering the last step.

Recursion

- We need to find a recursive formulation.
- Often we do this by considering the last step.
- For some value of k, last step:
  $(A_1 A_2 \ldots A_k) \cdot (A_{k+1} A_{k+2} \ldots A_m)$

Recursion

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

Recursion

- We need to find a recursive formulation.
- Often we do this by considering the last step.
- For some value of k, last step: $(A_1A_2\ldots A_k)\cdot(A_{k+1}A_{k+2}\ldots A_m)$
- Number of steps:
  - $CMM(A_1,A_2,\ldots,A_k)$ to compute first product
  - $CMM(A_{k+1},\ldots,A_m)$ to compute second product
  - $n_0n_kn_m$ to do final multiply
- Recursion $CMM(A_1,\ldots,A_m) = \min_k[CMM(A_1,\ldots,A_k)+CMM(A_{k+1},\ldots,A_m)+n_0n_kn_m]$

Subproblems

- What subproblems do we need to solve?
  - We cannot afford to solve all possible CMM problems.

Subproblems

- What subproblems do we need to solve?
  - We cannot afford to solve <u>all</u> possible CMM problems.
- $CMM(A_1,\ldots,A_m)$ requires $CMM(A_1,\ldots,A_k)$ and $CMM(A_k,\ldots,A_m)$.

Subproblems

- What subproblems do we need to solve?
  - We cannot afford to solve all possible CMM problems.
- CMM(A_1,…,A_m) requires CMM(A_1,…,A_k) and CMM(A_k,…,A_m).
- These require CMM(A_i,A_{i+1},…,A_j), but nothing else.

Subproblems

- What subproblems do we need to solve?
  - We cannot afford to solve <u>all</u> possible CMM problems.
- CMM(A_1,…,A_m) requires CMM(A_1,…,A_k) and CMM(A_k,…,A_m).
- These require CMM(A_i,A_{i+1},…,A_j), but nothing else.
- Only need subproblems C(i,j) = CMM(A_i,A_{i+1},…,A_j) for $1 \leq i \leq j \leq m$.

Subproblems

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

Subproblems

- What subproblems do we need to solve?
  - We cannot afford to solve all possible CMM problems.
- $CMM(A_1,\ldots,A_m)$ requires $CMM(A_1,\ldots,A_k)$ and $CMM(A_k,\ldots,A_m)$.
- These require $CMM(A_i,A_{i+1},\ldots,A_j)$, but nothing else.
- Only need subproblems $C(i,j) = CMM(A_i,A_{i+1},\ldots,A_j)$ 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)

Full Recursion

Base Case: C(i,i) = 0.
(With a single matrix, we don't have to do anything)

Recursive Step:

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

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

Example

Compute ABCD where

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

Finish

Start

	A	B	C	D
A				
B				
C				
D				

Example

Compute ABCD where

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

Finish

Start

	A	B	C	D
A				
B	X			
C	X	X		
D	X	X	X	

Illegal calls

Example

Compute ABCD where

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

Finish

Start

	A	B	C	D
A	0			
B	X	0		
C	X	X	0	
D	X	X	X	0

Base

Case

Example

Compute ABCD where

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

Finish

Start

	A	B	C	D
A	0	40		
B	X	0		
C	X	X	0	
D	X	X	X	0

$2 \cdot 5 \cdot 4 = 40$

Example

Compute ABCD where

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

Finish

Start

	A	B	C	D
A	0	40		
B	X	0	60	
C	X	X	0	
D	X	X	X	0

$5 \cdot 4 \cdot 3 = 60$

Example

Compute ABCD where

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

Finish

Start

	A	B	C	D
A	0	40		
B	X	0	60	
C	X	X	0	60
D	X	X	X	0

$4 \cdot 3 \cdot 6 = 60$

Example

Compute ABCD where

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

Finish

Start

	A	B	C	D
A	0	40	64	
B	X	0	60	
C	X	X	0	60
D	X	X	X	0

$2 \cdot 5 \cdot 3 + 60 = 90$

$2 \cdot 4 \cdot 3 + 40 = 64$

Example

Compute ABCD where

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

Finish

Start

	A	B	C	D
A	0	40	64	
B	X	0	60	135
C	X	X	0	60
D	X	X	X	0

$5 \cdot 4 \cdot 5 + 60 = 160$

$5 \cdot 3 \cdot 5 + 60 = 135$

Example

Compute ABCD where

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

Finish

Start

	A	B	C	D
A	0	40	64	94
B	X	0	60	135
C	X	X	0	60
D	X	X	X	0

$2 \cdot 5 \cdot 5 + 135 = 185$

$2 \cdot 4 \cdot 5 + 40 + 60$

$= 140$

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

Example

Compute ABCD where

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

Finish

Start

	A	B	C	D
A	0	40	64	94
B	X	0	60	135
C	X	X	0	60
D	X	X	X	0

(((AB)C)D)

Runtime

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

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. $O(m)$.

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.

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.

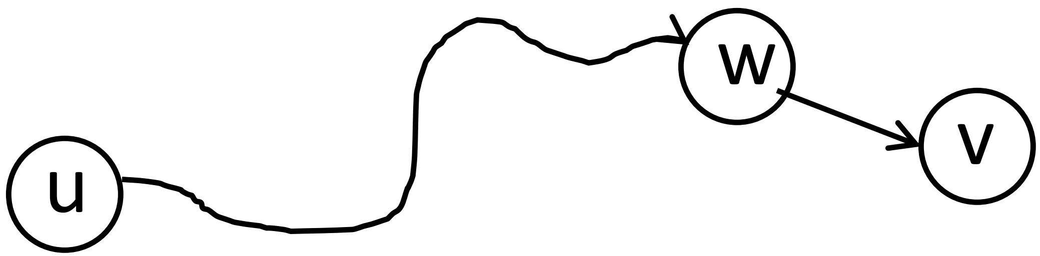
Runtime: $O(|V|^2|E|)$

Dynamic Program

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

Dynamic Program

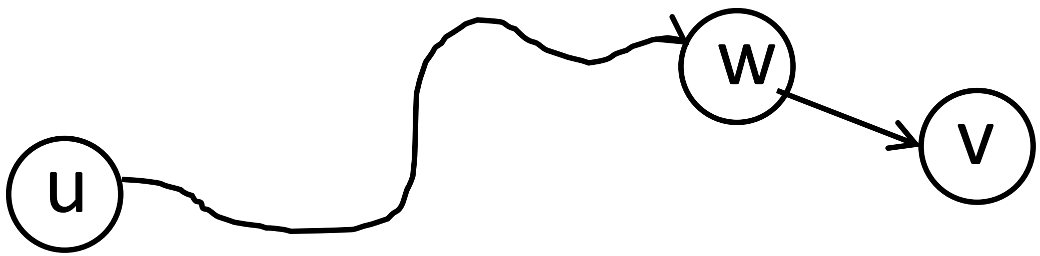
- Let $d_k(u,v)$ be the length of the shortest u-v path using at most k edges.



- Consider last edge.

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.

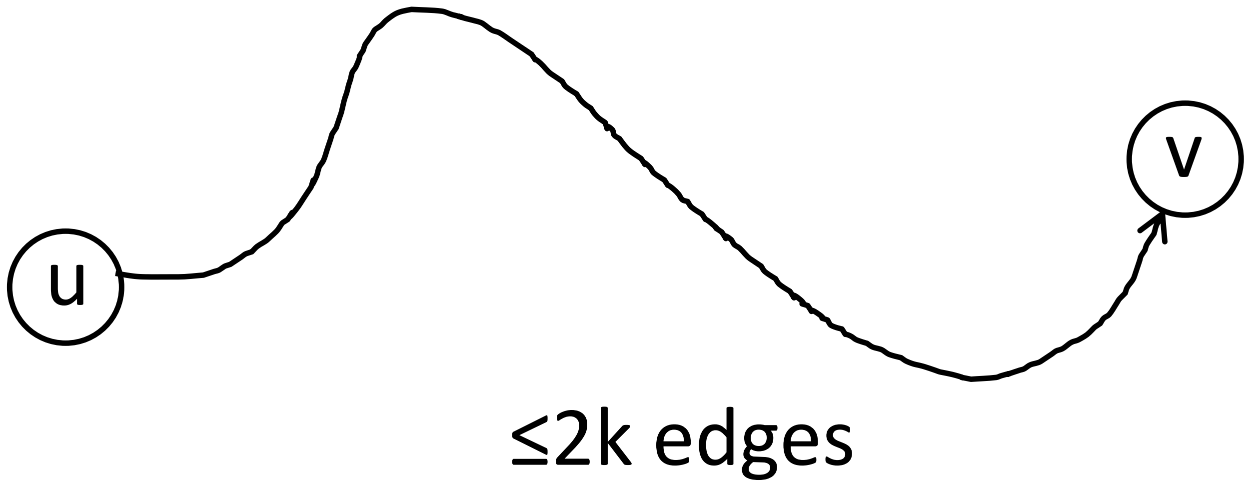
Matrix Multiplication Method

- `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.

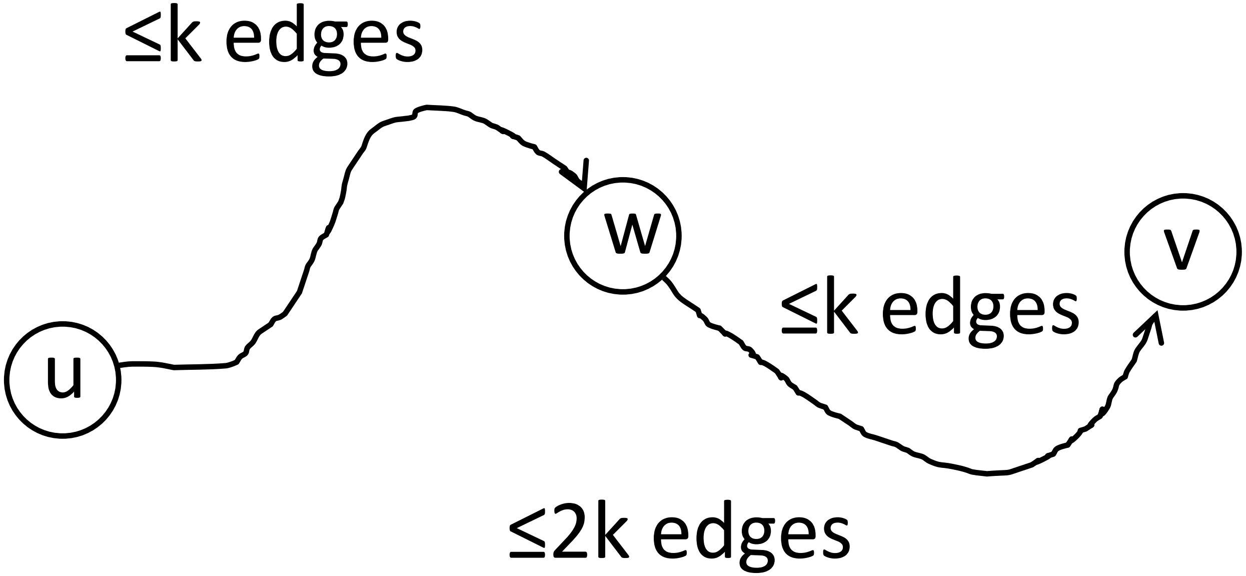
Matrix Multiplication Method

- `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?

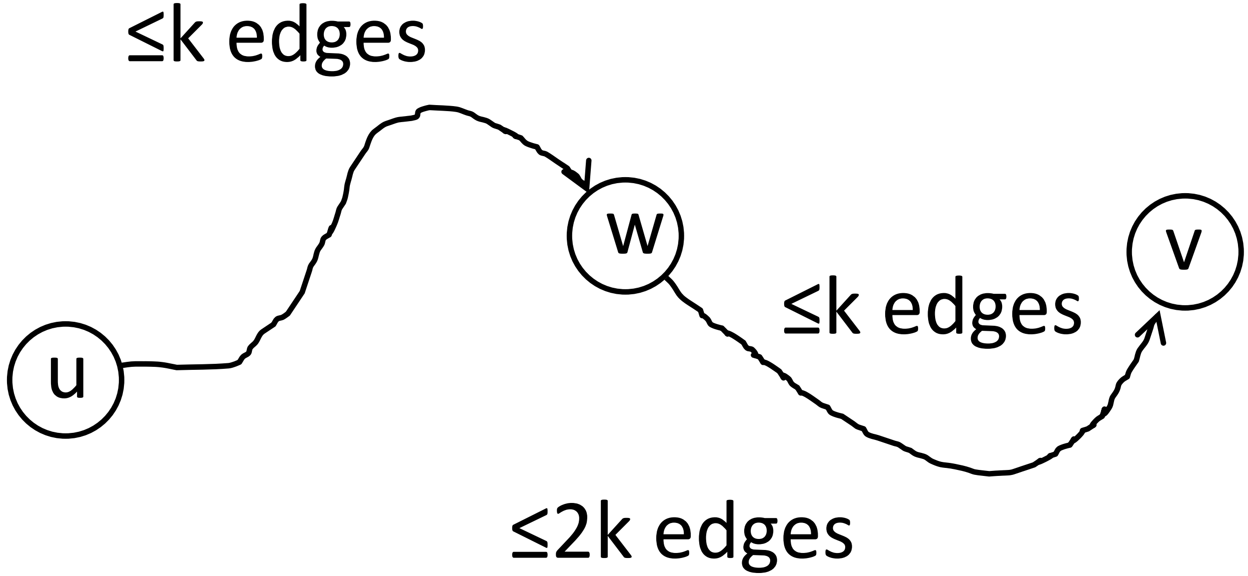
Recursion



Recursion



Recursion



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

Algorithm

Base Case:

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

Algorithm

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

Algorithm

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

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

Algorithm

$O(|V|^2)$

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

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

Algorithm

$O(|V|^2)$

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

$O(|V|^3)$

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

Algorithm

$O(|V|^2)$

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

$O(|V|^3)$

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

$O(\log|V|)$ iterations

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

$O(|V|^3)$

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

$O(\log|V|)$ iterations

Floyd-Warshall Algorithm

- Label vertices $v_1, v_2, \ldots, v_n$.

Floyd-Warshall Algorithm

- Label vertices v_1, v_2, …, v_n.
- Let $d_k(u,w)$ be the length of the shortest u-w path using only v_1, v_2,…,v_k as intermediate vertices.

Floyd-Warshall Algorithm

- Label vertices $v_1, v_2, \ldots, v_n$.
- Let $d_k(u,w)$ be the length of the shortest u-w path using only $v_1, v_2, \ldots, v_k$ 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.

Recursion

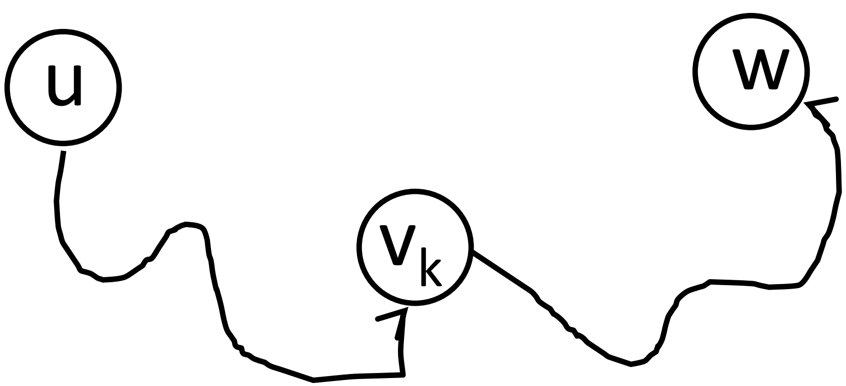
Break into cases based on whether shortest path uses v_k.

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

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

Base Case:

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

Algorithm

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: For each u, w compute:

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

Algorithm

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: For each u, w compute:

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

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

Algorithm

$O(|V|^2)$

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: For each u, w compute:

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

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

Algorithm

$O(|V|^2)$

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: 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 $n = |V|$.

Algorithm

O(|V|²)

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:

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

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

O(|V|) Iterations

Algorithm

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

$O(|V|^2)$

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:

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

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

$O(|V|)$ Iterations

Best Known Algorithm

Actually isn't DP.

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.

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