All pairs shortest paths:
calculate the shortest paths between all vertices

Easy solution?
All pairs shortest paths

All pairs shortest paths: calculate the shortest paths between all vertices

Run Bellman-Ford from each vertex!

\( O(VE) \)
- Bellman-Ford: \( O(VE) \)
- \( V \) calls, one for each vertex

Floyd-Warshall: key idea

Label all vertices with a number from 1 to \( V \)

\( d_{ij}^k = \text{shortest path from vertex } i \text{ to vertex } j \)
using only vertices \( \{1, 2, \ldots, k\} \)

What is \( d_{15}^{10} \)?

\( d_{15}^4 = 1 \). Can’t use vertex 4.
Floyd-Warshall: key idea

Label all vertices with a number from 1 to V

\[ d_{ij} = \text{shortest path from vertex } i \text{ to vertex } j \text{ using only vertices } \{1, 2, \ldots, k\} \]

If we want all possibilities, how many values are there (i.e., what is the size of \( d_{ij} \))?

\[ d_{ij}^{k} = \text{shortest path from vertex } i \text{ to vertex } j \text{ using only vertices } \{1, 2, \ldots, k\} \]

Floyd-Warshall: key idea

Recursive relationship

Assume we know \( d_{ij}^{k} \)

How can we calculate \( d_{ij}^{k+1} \), i.e., shortest path now including vertex \( k+1 \) (Hint: in terms of \( d_{ij}^{k} \))

Two options:
1) Vertex \( k+1 \) doesn't give us a shorter path
2) Vertex \( k+1 \) does give us a shorter path
Recursive relationship

\[ d_{i,j}^k = \text{shortest path from vertex } i \text{ to vertex } j \]

using only vertices \{1, 2, ..., k\}

Two options:
1) Vertex \( k+1 \) doesn't give us a shorter path
2) Vertex \( k+1 \) does give us a shorter path

\[ d_{i,j}^{k+1} = \] ?

13

14

Recursive relationship

\[ d_{i,j}^k = \text{shortest path from vertex } i \text{ to vertex } j \]

using only vertices \{1, 2, ..., k\}

Two options:
1) Vertex \( k+1 \) doesn't give us a shorter path
2) Vertex \( k+1 \) does give us a shorter path

\[ d_{i,j}^{k+1} = d_{i,j}^k \]

15

16

Recursive relationship

\[ d_{i,j}^k = \text{shortest path from vertex } i \text{ to vertex } j \]

using only vertices \{1, 2, ..., k\}

Two options:
1) Vertex \( k+1 \) doesn't give us a shorter path
2) Vertex \( k+1 \) does give us a shorter path

\[ d_{i,j}^{k+1} = \] ?

some vertices \{1,2,...,k\}

some vertices \{1,2,...,k\}

What is the cost of this path?
Recursive relationship

Two options:
1) Vertex \( k+1 \) doesn’t give us a shorter path
2) Vertex \( k+1 \) does give us a shorter path

\[
d_{ij}^{k+1} = d_{ij}^{k} + d_{ik}^{k+1}
\]

How do we combine these two options?

Floyd-Warshall

Calculate \( d_{ij}^{k} \) for increasing \( k \), i.e. \( k = 1 \) to \( V \)

Floyd-Warshall\( (G = (V,E,W)) \):

\[
d_{i,j}^{0} = W \quad // \text{initialize with edge weights}
\]

for \( k = 1 \) to \( V \)

for \( i = 1 \) to \( V \)

for \( j = 1 \) to \( V \)

\[
d_{ij}^{k} = \min(d_{ij}^{k-1}, d_{ik}^{k-1} + d_{kj}^{k-1})
\]

return \( d_{ij}^{V} \)
Fluor-Warshall() = (G = (V, E, W)):

\[ d^0 = W \]

// initialize with edge weights
for \( k = 1 \) to \( V \)
for \( j = 1 \) to \( V \)
for \( i = 1 \) to \( V \)
\[ d_{ij} = \min(d_{ij}, d_{ik} + d_{kj}) \]

return \( d^V \)

Fluor-Warshall() = (G = (V, E, W)):

\[ d^0 = W \]

// initialize with edge weights
for \( k = 1 \) to \( V \)
for \( j = 1 \) to \( V \)
for \( i = 1 \) to \( V \)
\[ d_{ij} = \min(d_{ij}, d_{ik} + d_{kj}) \]

return \( d^V \)
Floyd-Warshall Algorithm:

\( d^0 = W \) // Initialize with edge weights
\[
\text{for } k = 1 \text{ to } V \\
\text{for } i = 1 \text{ to } V \\
\text{for } j = 1 \text{ to } V \\
\quad d_{ij} = \min(d_{ij}, d_{ik} + d_{kj})
\]
return \( d^V \)

minimum

Found a shorter path!
Floyd-Warshall($G(V,W)$):

\( d^W = W \) // initialize with edge weights

for \( k = 1 \) to \( V \)

for \( i = 1 \) to \( V \)

for \( j = 1 \) to \( V \)

\[ d_{ij}^k = \min(d_{ij}^{k-1}, d_{ik}^{k-1} + d_{kj}^{k-1}) \]

return \( d^V \)

---

\( k = 2 \)

\[
\begin{array}{cccccc}
1 & 2 & 3 & 4 & 5 \\
0 & 4 & -1 & 9 & 0 \\
0 & 0 & 0 & 5 & 0 \\
3 & 0 & 2 & 2 & 0 \\
0 & 0 & 0 & 0 & -3 \\
1 & 0 & 0 & 0 & 5 \\
\end{array}
\]

\( k = 3 \)

\[
\begin{array}{cccccc}
1 & 2 & 3 & 4 & 5 \\
0 & 4 & -1 & 9 & 0 \\
0 & 0 & 0 & 5 & 0 \\
3 & 0 & 2 & 2 & 0 \\
0 & 0 & 0 & 0 & -3 \\
1 & 0 & 0 & 0 & 5 \\
\end{array}
\]

---

Found a shorter path!
Floyd-Warshall: \( G = (V, E, W) \):

\[
\begin{align*}
\text{for } k = 1 \text{ to } V & \quad \text{do}
\end{align*}
\]

return \( d^V \)

minimum

Found a shorter path!
\[ d^0 = W \] // initialize with edge weights
for \( k = 1 \) to \( V \)
  for \( i = 1 \) to \( V \)
    for \( j = 1 \) to \( V \)
      \( d_{ij} = \min(d_{ij}, d_{ik} + d_{kj}) \)

return \( d^V \)
Floyd-Warshall analysis

Is it correct?

Floyd-Warshall analysis

Is it correct?

Floyd-Warshall analysis

Is it correct?
Floyd-Warshall analysis

Is it correct?

Any assumptions?

Floyd-Warshall(G = (V,E,W))

$\delta^0 = W$         // initiate with edge weights
for $k = 1$ to $V$
    for $i = 1$ to $V$
        for $j = 1$ to $V$
            $d_{ij}^{k} = \min(\delta_{ij}, \delta_{ik}^{k-1} + \delta_{kj}^{k-1})$

return $d_{VV}$

45

Floyd-Warshall analysis

Is it correct?

Assuming the graph has no negative cycles!

What happens if there is a negative cycle?

Floyd-Warshall(G = (V,E,W))

$\delta^0 = W$         // initiate with edge weights
for $k = 1$ to $V$
    for $i = 1$ to $V$
        for $j = 1$ to $V$
            $d_{ij}^{k} = \min(\delta_{ij}, \delta_{ik}^{k-1} + \delta_{kj}^{k-1})$

return $d_{VV}$

46

Floyd-Warshall analysis

If the graph has a negative weight cycle, at the end, at least one of the diagonal entries will be a negative number, i.e., there’s a way to get back to a vertex using all of the vertices that results in a negative weight

1 2 3 4 5
1 0 2 -1 1 -2
2 0 7 9 5
3 3 0 2 -1
4 1 -2 0 -3
5 0 1 = 1 = 0

47

Floyd-Warshall analysis

Run-time?

Floyd-Warshall(G = (V,E,W))

$\delta^0 = W$         // initiate with edge weights
for $k = 1$ to $V$
    for $i = 1$ to $V$
        for $j = 1$ to $V$
            $d_{ij}^{k} = \min(\delta_{ij}, \delta_{ik}^{k-1} + \delta_{kj}^{k-1})$

return $d_{VV}$

48
Floyd-Warshall analysis

Run-time: $O(V^3)$

Floyd-Warshall(G = (V,E,W))

$d^0 = W$ // initialize with edge weights

for $k = 1$ to $V$
    for $i = 1$ to $V$
        for $j = 1$ to $V$
            $d_{ij}^{k} = \min(d_{ij}^{k-1}, d_{ik}^{k-1} + d_{kj}^{k-1})$

return $d^V$

Floyd-Warshall analysis

What type of algorithm is Floyd-Warshall?

Floyd-Warshall(G = (V,E,W))

d^0 = W // initialize with edge weights

for $k = 1$ to $V$
    for $i = 1$ to $V$
        for $j = 1$ to $V$
            $d_{ij}^{k} = \min(d_{ij}^{k-1}, d_{ik}^{k-1} + d_{kj}^{k-1})$

return $d^V$

Floyd-Warshall analysis

Dynamic programming!!

Build up solutions to larger problems using solutions to smaller problems. Use a table to store the values.

Floyd-Warshall(G = (V,E,W))

d^0 = W // initialize with edge weights

for $k = 1$ to $V$
    for $i = 1$ to $V$
        for $j = 1$ to $V$
            $d_{ij}^{k} = \min(d_{ij}^{k-1}, d_{ik}^{k-1} + d_{kj}^{k-1})$

return $d^V$

Floyd-Warshall analysis

Space usage?

Floyd-Warshall(G = (V,E,W))

d^0 = W // initialize with edge weights

for $k = 1$ to $V$
    for $i = 1$ to $V$
        for $j = 1$ to $V$
            $d_{ij}^{k} = \min(d_{ij}^{k-1}, d_{ik}^{k-1} + d_{kj}^{k-1})$

return $d^V$
Floyd-Warshall: key idea

Label all vertices with a number from 1 to \( V \)

\( d_{i,j}^k \) = shortest path from vertex \( i \) to vertex \( j \)
using only vertices \( \{1, 2, \ldots, k\} \)

If we want all possibilities, how many values are there
(i.e. what is the size of \( d_{i,j}^k \)?)

\( d_{i,j}^k = \) shortest path from vertex \( i \) to vertex \( j \)
using only vertices \( \{1, 2, \ldots, k\} \)

V^3

- \( i \): all vertices
- \( j \): all vertices
- \( k \): all vertices

Can we do better?

Floyd-Warshall analysis

Space usage: \( \Theta(V^3) \)

Only need the current value and the previous

Floyd-Warshall analysis

\( d_{i,j}^0 = W \)         // initialize with edge weights
for \( k = 1 \) to \( V \)
for \( i = 1 \) to \( V \)
for \( j = 1 \) to \( V \)
    
    \[ d_{i,j}^k = \min(d_{i,j}^{k-1}, d_{i,k} + d_{k,j}) \]

return \( d_{ij}^V \)

All pairs shortest paths

V * Bellman-Ford: \( O(V^2E) \)

Floyd-Warshall: \( \Theta(V^3) \)
All pairs shortest paths

All pairs shortest paths for positive weight graphs:
calculate the shortest paths between all points

Easy solution?

Run Dijkstra's from each vertex!

Running time (in terms of $E$ and $V$)?

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

$\cdot$ $V$ calls to Dijkstra's
$\cdot$ Dijkstra's: $O(V \log V + E)$

Is this any better?
4/16/24

All pairs shortest paths

- Bellman-Ford: $O(V^2 E)$
- Floyd-Warshall: $O(V^3)$
- Dijkstra's: $O(V^2 \log V + V E)$

If the graph is sparse!

All pairs shortest paths for positive weight graphs:
calculate the shortest paths between all points
- Run Dijkstra’s from each vertex
- Challenge: Dijkstra’s assumes positive weights

Johnson’s: key idea

Reweight the graph to make all edges positive such that shortest paths are preserved

What’s the shortest path from A to D?

Lemma

Let $h$ be any function mapping a vertex to a real value
- If we change the graph weights as: $\hat{w}(u, v) = w(u, v) + h(u) - h(v)$
- The shortest paths are preserved
Let \( u, v, x_1, ..., x_k \) be a path from \( s \) to \( t \).

The weight in the reweighted graph is:

\[
\tilde{w}(u,v) = w(u,v) + h(u) - h(v)
\]

Lemma: proof

\( \tilde{w}(u,v) = w(u,v) + h(u) - h(v) \)

Let \( u, v, x_1, ..., x_k \) be a path from \( s \) to \( t \).

The weight in the reweighted graph is:

\[
\tilde{w}(u,v) = w(u,v) + h(u) - h(v)
\]

Lemma: proof

\( \tilde{w}(u,v) = w(u,v) + h(u) - h(v) \)
Claim: the weight change preserves shortest paths, i.e. if a path was the shortest from s to t in the original graph it will still be the shortest path from s to t in the new graph. Let $s, v_1, v_2, ..., v_k, t$ be a path from s to t. The weight in the reweighted graph is:

$$\hat{u}(s, v_1, ..., v_k, t) = w(s, v_1, ..., v_k, t) + h(v_1) - h(v_k)$$

The shortest paths are preserved. Justification? $h(s) - h(t)$ is a constant and will be the same for all paths from s to t, so the absolute ordering of all paths from s to t will not change.

If we change the graph weights as:

$$\hat{u}(u, v) = w(u, v) + h(u) - h(v)$$

The shortest paths are preserved. Big question: how do we pick $h$?
Selecting $h$

Need to pick $h$ such that the resulting graph has all weights as positive:

$$h(u,v) = w(u,v) + h(u) - h(v)$$

Johnson's algorithm

Create $G'$ with one extra node $s$ with 0 weight edges to all nodes

run Bellman-Ford($G', s$)

if no negative-weight cycle

reweight edges in $G$ with $h(v)$ = shortest path from $s$ to $v$

run Dijkstra's from every vertex

reweight shortest paths based on $G$
Create $G'$
run Bellman-Ford($G',s$)
if no negative-weight cycle
   reweight edges in $G$ with $h(v)$=shortest path from $s$ to $v$
run Dijkstra's from every vertex
   reweight shortest paths based on $G$

Create $G'$
run Bellman-Ford($G',s$)
if no negative-weight cycle
   reweight edges in $G$ with $h(v)$=shortest path from $s$ to $v$
run Dijkstra's from every vertex
   reweight shortest paths based on $G$
Create $G'$
run Bellman-Ford($G',s$)
if no negative-weight cycle
reweight edges in $G$ with $h(v)$=shortest path from $s$ to $v$
run Dijkstra's from every vertex
reweight shortest paths based on $G$

$h(v)$ in blue

\[ \hat{w}(u,v) = w(u,v) + h(u) - h(v) \]
Create $G'$
run Bellman-Ford($G'$, s)
if no negative-weight cycle
  reweight edges in $G$ with $h(v)$=shortest path from $s$ to $v$
run Dijkstra's from every vertex
reweight shortest paths based on $G$
$h(v)$ in blue

$\hat{w}(u,v) = w(u,v) + h(u) - h(v)$
Create $G'$
run Bellman-Ford$(G',s)$
if no negative-weight cycle
  reweight edges in $G$ with $h(v)$=shortest path from $s$ to $v$
run Dijkstra’s from every vertex
reweight shortest paths based on $G$
$h(v)$ in blue

Create $G'$
run Bellman-Ford$(G',s)$
if no negative-weight cycle
  reweight edges in $G$ with $h(v)$=shortest path from $s$ to $v$
run Dijkstra’s from every vertex
reweight shortest paths based on $G$
$h(v)$ in blue
Create $G'$

run Bellman-Ford($G'$)

if no negative-weight cycle

reweight edges in $G'$ with $h(v) =$ shortest path from $s$ to $v$

run Dijkstra's from every vertex

reweight shortest paths based on $G$

**Selecting $h$**

Need to pick $h$ such that the resulting graph has all weights as positive

Create $G'$ with one extra node $s$ with 0 weight edges to all nodes

run Bellman-Ford($G'$)

if no negative-weight cycle

reweight edges in $G'$ with $h(v) =$ shortest path from $s$ to $v$

run Dijkstra's from every vertex

reweight shortest paths based on $G$

**Reweighted graph is positive**

Take two nodes $u$ and $v$

$h(u)$ shortest distance from $s$ to $u$

$h(v)$ shortest distance from $s$ to $v$

Claim: $h(v) = h(u) + w(u,v)$

Why?
Reweighted graph is positive

Take two nodes u and v
h(u) shortest distance from s to u
h(v) shortest distance from s to v
Claim: h(v) ≤ h(u) + w(u, v)

If this weren't true, we could have made a shorter path s to v using u
... but this is in contradiction with how we defined h(v)

All edge weights in reweighted graph are non-negative

Johnson's algorithm

Create G'
run Bellman-Ford(G')
if no negative-weight cycle
reweight edges in G
run Dijkstra's from every vertex
reweight shortest paths based on G

Run-time?
Johnson's algorithm

Create $G'$
run Bellman-Ford($G', s$)
if no negative-weight cycle
  reweight edges in $G$
run Dijkstra's from every vertex
reweight shortest paths based on $G$

Run-time?

All pairs shortest paths

$V \times$ Bellman-Ford: $O(V^2E)$
Floyd-Warshall: $O(V^3)$
Johnson's: $O(V^2 \log V + VE)$

DAGs

Can represent dependency graphs

Topological sort

A linear ordering of all the vertices such that for all edges $(u, v)$
$u$ appears before $v$ in the ordering
An ordering of the nodes that "obeys" the dependencies, i.e.
on activity can't happen until it's dependent activities have happened
Topological sort

1. Find a node \( v \) with no incoming edges
2. Delete \( v \) from \( G \)
3. Add \( v \) to linked list
4. Topological-Sort(\( G \))

Topological sort

1. Find a node \( v \) with no incoming edges
2. Delete \( v \) from \( G \)
3. Add \( v \) to linked list
4. Topological-Sort(\( G \))

Topological sort

1. Find a node \( v \) with no incoming edges
2. Delete \( v \) from \( G \)
3. Add \( v \) to linked list
4. Topological-Sort(\( G \))

Topological sort

1. Find a node \( v \) with no incoming edges
2. Delete \( v \) from \( G \)
3. Add \( v \) to linked list
4. Topological-Sort(\( G \))
Topological sort

1. Find a node \( v \) with no incoming edges
2. Delete \( v \) from \( G \)
3. Add \( v \) to linked list
4. Topological-sort(G)

Running time?

1. Find a node \( v \) with no incoming edges
2. Delete \( v \) from \( G \)
3. Add \( v \) to linked list
4. Topological-sort(G)

Running time?

\( O(|V| + |E|) \)

Running time?

\( O(E) \) overall
Running time?

1. Find a node $v$ with no incoming edges
2. Delete $v$ from $G$
3. Add $v$ to linked list

How many calls? $|V|$

Overall running time?

$O(|V|^2 + |V| |E|)$

Can we do better?

1. Find a node $v$ with no incoming edges
2. Insert $v$ into list
3. Add $v$ to linked list
4. Topological-sort$(G)$

Topological sort 2

1. for all edges $(u, v) \in E$
2. active$[v] = active[v] + 1$
3. for all $v \in V$
4. if active$[v] = 0$
5. Enqueue$(S, v)$
6. while !Empty$(S)$
7. $s = \text{Dequeue}(S)$
8. add $s$ to linked list
9. for each edge $(u, v) \in E$
10. active$[u] = active[u] - 1$
11. if active$[u] = 0$
12. Enqueue$(S, u)$
Topological sort 2

```
TopologicalSort2(G)
1 for all edges (u, v) ∈ E
2 active[v] ← active[v] + 1
3 for all u ∈ V
4 if active[v] = 0
5 Enqueue(S, v)
6 while !EMPTY(S)
7 u ← Dequeue(S)
8 add u to linked list
9 for each edge (u, v) ∈ E
10 active[v] ← active[v] - 1
11 if active[v] = 0
12 Enqueue(S, v)
```

Running time?

How many times do we process each node?
How many times do we process each edge?

\( O(|V| + |E|) \)
Detecting cycles

Undirected graph
- BFS or DFS. If we reach a node we’ve seen already, then we’ve found a cycle (have to be a bit careful about the node we just came from)

Directed graph
- Call TopologicalSort
- If the length of the list returned ≠ |V| then a cycle exists

What are the shortest paths from S to each of the vertices?

REWIGHT the graph on the right based on the h values

\[ \hat{w}(u,v) = w(u,v) + h(u) - h(v) \]