SHORTEST PATHS

David Kauchak
CS 140 – Spring 2024

1

Admin

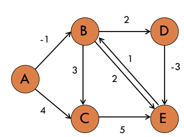
Assignment 8

Assignment 9

2

All pairs shortest paths

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



3

All pairs shortest paths

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

Easy solution?

4

All pairs shortest paths

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

Run Bellman-Ford from each vertex!

$O(V^2E)$
- Bellman-Ford: O(VE)
- V calls, one for each vertex

5

Floyd-Warshall: key idea

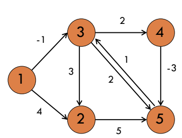
Label all vertices with a number from 1 to V

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

6

Floyd-Warshall: key idea

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

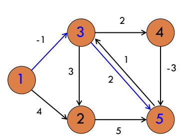


What is $d_{15}{}^3$?

7

Floyd-Warshall: key idea

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



$d_{15}{}^3$ = 1. Can't use vertex 4.

8

Floyd-Warshall: key idea

Label all vertices with a number from 1 to V

$d_{ij}{}^{k}$ = 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 $d_{ij}{}^{k}$)?

9

Floyd-Warshall: key idea

Label all vertices with a number from 1 to V

$d_{ij}{}^{k}$ = 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

10

Floyd-Warshall: key idea

Label all vertices with a number from 1 to V

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

What is $d_{ij}{}^{V}$?

- Distance of the shortest path from i to j
- If we can calculate this, for all (i, j), we're done!

11

Recursive relationship

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

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

12

Recursive relationship

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

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

$d_{ij}{}^{k+1} = ?$

13

Recursive relationship

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

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

$d_{ij}{}^{k+1} = d_{ij}{}^{k}$

14

Recursive relationship

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

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

$d_{ij}{}^{k+1} = ?$

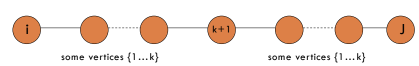
15

Recursive relationship

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

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

$d_{ij}{}^{k+1} = ?$



What is the cost of this path?

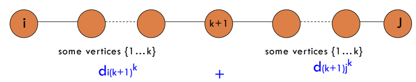
16

Recursive relationship

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

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

$$d_{ij}^{k+1} = d_{i(k+1)}{}^k + d_{(k+1)j}{}^k$$



17

Recursive relationship

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

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

d_{ij}^{k+1} = ?

How do we combine these two options?

18

Recursive relationship

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

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

$$d_{ij}^{k+1} = \min(d_{ij}k, d_{i(k+1)}{}^k + d_{(k+1)j}{}^k)$$

Pick whichever is shorter

19

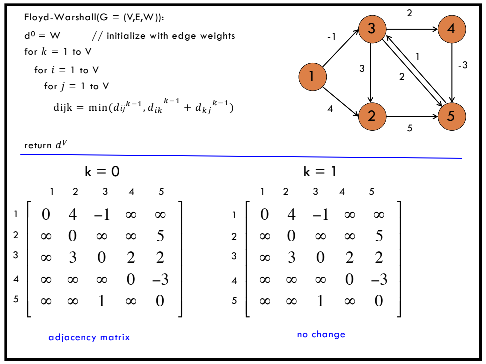
Floyd-Warshall

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

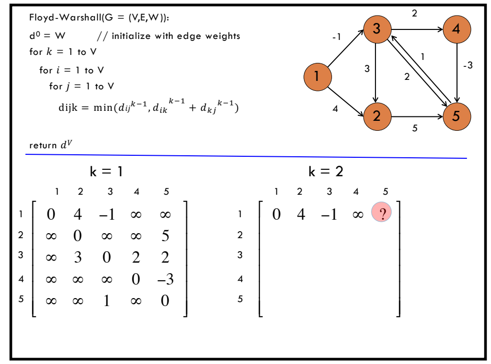
Floyd-Warshall(G = (V,E,W)):
d^0 = W // initialize with edge weights
for k = 1 to V
  for i = 1 to V
    for j = 1 to V
      $d_{ij}k = \min(d_{ij}^{k-1}, d_{ik}{}^{k-1} + d_{kj}{}^{k-1})$

return d^V

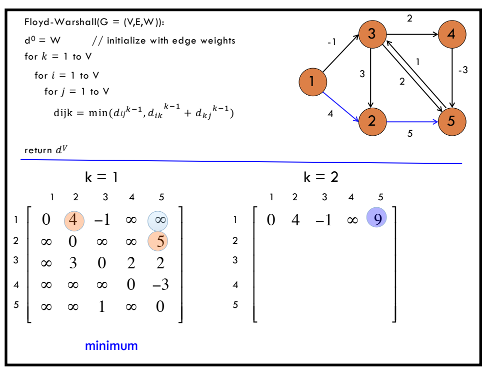
20


Floyd-Warshall(G = (V,E,W)):
d⁰ = W // initialize with edge weights
for k = 1 to V
for i = 1 to V
for j = 1 to V
dijk = min(dij^{k−1}, dik^{k−1} + dkj^{k−1})
return d^V
1
2
3
4
5
2
-1
3
1
2
-3
4
5
k = 0
1 2 3 4 5
1 0 4 −1 ∞ ∞
2 ∞ 0 ∞ ∞ 5
3 ∞ 3 0 2 2
4 ∞ ∞ ∞ 0 −3
5 ∞ ∞ 1 ∞ 0
adjacency matrix
k = 1
1 2 3 4 5
1 0 4 −1 ∞ ∞
2 ∞ 0 ∞ ∞ 5
3 ∞ 3 0 2 2
4 ∞ ∞ ∞ 0 −3
5 ∞ ∞ 1 ∞ 0
no change

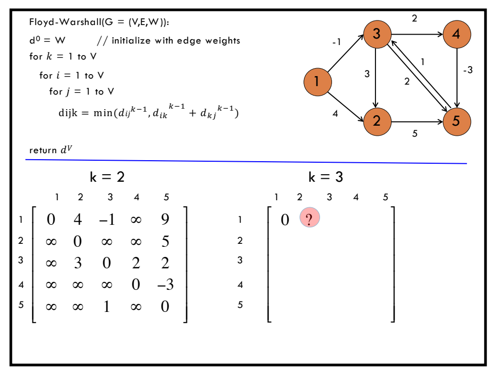
21


Floyd-Warshall(G = (V,E,W)):
d⁰ = W // initialize with edge weights
for k = 1 to V
for i = 1 to V
for j = 1 to V
dijk = min(dij^{k−1}, dik^{k−1} + dkj^{k−1})
return d^V
1
2
3
4
5
2
-1
3
1
2
-3
4
5
k = 1
1 2 3 4 5
1 0 4 −1 ∞ ∞
2 ∞ 0 ∞ ∞ 5
3 ∞ 3 0 2 2
4 ∞ ∞ ∞ 0 −3
5 ∞ ∞ 1 ∞ 0
k = 2
1 2 3 4 5
1 0 4 −1 ∞ ?
2
3
4
5

22


Floyd-Warshall(G = (V,E,W)):
d⁰ = W // initialize with edge weights
for k = 1 to V
for i = 1 to V
for j = 1 to V
dijk = min(dij^{k−1}, dik^{k−1} + dkj^{k−1})
return d^V
1
2
3
4
5
2
-1
3
1
2
-3
4
5
k = 1
1 2 3 4 5
1 0 4 −1 ∞ ∞
2 ∞ 0 ∞ ∞ 5
3 ∞ 3 0 2 2
4 ∞ ∞ ∞ 0 −3
5 ∞ ∞ 1 ∞ 0
minimum
k = 2
1 2 3 4 5
1 0 4 −1 ∞ 9
2
3
4
5

23


Floyd-Warshall(G = (V,E,W)):
d⁰ = W // initialize with edge weights
for k = 1 to V
for i = 1 to V
for j = 1 to V
dijk = min(dij^{k−1}, dik^{k−1} + dkj^{k−1})
return d^V
1
2
3
4
5
2
-1
3
1
2
-3
4
5
k = 2
1 2 3 4 5
1 0 4 −1 ∞ 9
2 ∞ 0 ∞ ∞ 5
3 ∞ 3 0 2 2
4 ∞ ∞ ∞ 0 −3
5 ∞ ∞ 1 ∞ 0
k = 3
1 2 3 4 5
1 0 ?
2
3
4
5

24

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

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

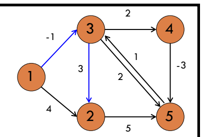
for k = 1 to V

for i = 1 to V

for j = 1 to V

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

return d^V



k = 2

	1	2	3	4	5
1	0	4	−1	∞	9
2	∞	0	∞	∞	5
3	∞	3	0	2	2
4	∞	∞	∞	0	−3
5	∞	∞	1	∞	0

minimum

k = 3

	1	2	3	4	5
1	0	2			
2					
3					
4					
5					

Found a shorter path!

25

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

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

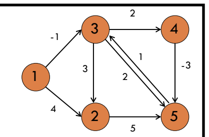
for k = 1 to V

for i = 1 to V

for j = 1 to V

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

return d^V



k = 2

	1	2	3	4	5
1	0	4	−1	∞	9
2	∞	0	∞	∞	5
3	∞	3	0	2	2
4	∞	∞	∞	0	−3
5	∞	∞	1	∞	0

k = 3

	1	2	3	4	5
1	0	2			
2					
3					
4					
5					

26

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

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

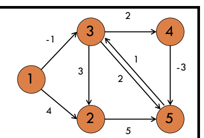
for k = 1 to V

for i = 1 to V

for j = 1 to V

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

return d^V



k = 2

	1	2	3	4	5
1	0	4	−1	∞	9
2	∞	0	∞	∞	5
3	∞	3	0	2	2
4	∞	∞	∞	0	−3
5	∞	∞	1	∞	0

k = 3

	1	2	3	4	5
1	0	2	−1	?	
2					
3					
4					
5					

27

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

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

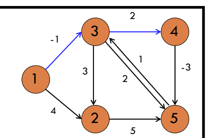
for k = 1 to V

for i = 1 to V

for j = 1 to V

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

return d^V



k = 2

	1	2	3	4	5
1	0	4	−1	∞	9
2	∞	0	∞	∞	5
3	∞	3	0	2	2
4	∞	∞	∞	0	−3
5	∞	∞	1	∞	0

minimum

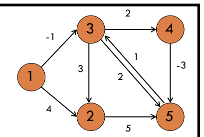
k = 3

	1	2	3	4	5
1	0	2	−1	1	
2					
3					
4					
5					

28

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

return d^V



k = 2

	1	2	3	4	5
1	0	4	−1	∞	9
2	∞	0	∞	∞	5
3	∞	3	0	2	2
4	∞	∞	∞	0	−3
5	∞	∞	1	∞	0

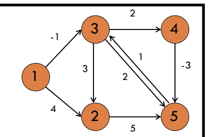
k = 3

	1	2	3	4	5
1	0	2	−1	1	
2					
3					
4					
5					

29

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

return d^V



k = 2

	1	2	3	4	5
1	0	4	−1	∞	9
2	∞	0	∞	∞	5
3	∞	3	0	2	2
4	∞	∞	∞	0	−3
5	∞	∞	1	∞	0

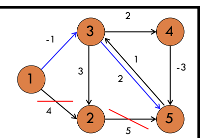
k = 3

	1	2	3	4	5
1	0	2	−1	1	?
2					
3					
4					
5					

30

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

return d^V



k = 2

	1	2	3	4	5
1	0	4	−1	∞	9
2	∞	0	∞	∞	5
3	∞	3	0	2	2
4	∞	∞	∞	0	−3
5	∞	∞	1	∞	0

minimum

k = 3

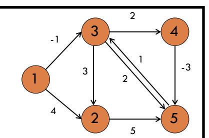
	1	2	3	4	5
1	0	2	−1	1	1
2					
3					
4					
5					

Found a shorter path!

31

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

return d^V



k = 2

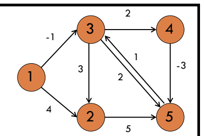
	1	2	3	4	5
1	0	4	−1	∞	9
2	∞	0	∞	∞	5
3	∞	3	0	2	2
4	∞	∞	∞	0	−3
5	∞	∞	1	∞	0

k = 3

	1	2	3	4	5
1	0	2	−1	1	1
2	∞	0	∞	∞	5
3	∞	3	0	2	2
4	∞	∞	∞	0	−3
5	∞	?			

32

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
      dijk = $\min(d_{ij}^{k-1}, d_{ik}^{k-1} + d_{kj}^{k-1})$
return d^V



k = 2

	1	2	3	4	5
1	0	4	−1	∞	9
2	∞	0	∞	∞	5
3	∞	3	0	2	2
4	∞	∞	∞	0	−3
5	∞	∞	1	∞	0

minimum

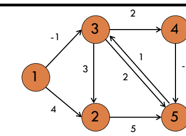
k = 3

	1	2	3	4	5
1	0	2	−1	1	1
2	∞	0	∞	∞	5
3	∞	3	0	2	2
4	∞	∞	∞	0	−3
5	∞	4			

Found a shorter path!

33

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
      dijk = $\min(d_{ij}^{k-1}, d_{ik}^{k-1} + d_{kj}^{k-1})$
return d^V



k = 2

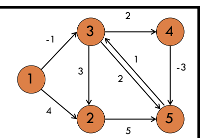
	1	2	3	4	5
1	0	4	−1	∞	9
2	∞	0	∞	∞	5
3	∞	3	0	2	2
4	∞	∞	∞	0	−3
5	∞	∞	1	∞	0

k = 3

	1	2	3	4	5
1	0	2	−1	1	1
2	∞	0	∞	∞	5
3	∞	3	0	2	2
4	∞	∞	∞	0	−3
5	∞	4	1	?	

34

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
      dijk = $\min(d_{ij}^{k-1}, d_{ik}^{k-1} + d_{kj}^{k-1})$
return d^V



k = 2

	1	2	3	4	5
1	0	4	−1	∞	9
2	∞	0	∞	∞	5
3	∞	3	0	2	2
4	∞	∞	∞	0	−3
5	∞	∞	1	∞	0

minimum

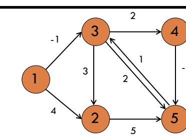
k = 3

	1	2	3	4	5
1	0	2	−1	1	1
2	∞	0	∞	∞	5
3	∞	3	0	2	2
4	∞	∞	∞	0	−3
5	∞	4	1	3	

Found a shorter path!

35

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
      dijk = $\min(d_{ij}^{k-1}, d_{ik}^{k-1} + d_{kj}^{k-1})$
return d^V



k = 2

	1	2	3	4	5
1	0	4	−1	∞	9
2	∞	0	∞	∞	5
3	∞	3	0	2	2
4	∞	∞	∞	0	−3
5	∞	∞	1	∞	0

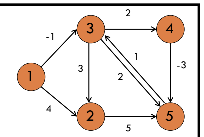
k = 3

	1	2	3	4	5
1	0	2	−1	1	1
2	∞	0	∞	∞	5
3	∞	3	0	2	2
4	∞	∞	∞	0	−3
5	∞	4	1	3	0

36

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

return d^V



k = 3

	1	2	3	4	5
1	0	2	−1	1	1
2	∞	0	∞	∞	5
3	∞	3	0	2	2
4	∞	∞	∞	0	−3
5	∞	∞	1	∞	0

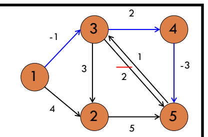
k = 4

	1	2	3	4	5
1	0	2	−1	1	?
2					
3					
4					
5					

37

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

return d^V



k = 3

	1	2	3	4	5
1	0	2	−1	1	1
2	∞	0	∞	∞	5
3	∞	3	0	2	2
4	∞	∞	∞	0	−3
5	∞	∞	1	∞	0

k = 4

	1	2	3	4	5
1	0	2	−1	1	−2
2					
3					
4					
5					

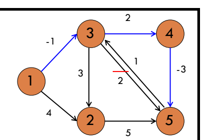
minimum

Found a shorter path!

38

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

return d^V



k = 3

	1	2	3	4	5
1	0	2	−1	1	1
2	∞	0	∞	∞	5
3	∞	3	0	2	2
4	∞	∞	∞	0	−3
5	∞	∞	1	∞	0

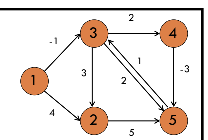
k = 4

	1	2	3	4	5
1	0	2	−1	1	−2
2					
3					
4					
5					

39

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

return d^V



k = 3

	1	2	3	4	5
1	0	2	−1	1	1
2	∞	0	∞	∞	5
3	∞	3	0	2	2
4	∞	∞	∞	0	−3
5	∞	∞	1	∞	0

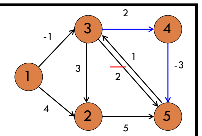
k = 4

	1	2	3	4	5
1	0	2	−1	1	−2
2	∞	0	∞	∞	5
3	∞	3	0	2	?
4					
5					

40

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

return d^V



k = 3

	1	2	3	4	5
1	0	2	−1	1	1
2	∞	0	∞	∞	5
3	∞	3	0	2	2
4	∞	∞	∞	0	−3
5	∞	∞	1	∞	0

minimum

k = 4

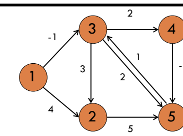
	1	2	3	4	5
1	0	2	−1	1	−2
2	∞	0	∞	∞	5
3	∞	3	0	2	−1
4					
5					

Found a shorter path!

41

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

return d^V



k = 3

	1	2	3	4	5
1	0	2	−1	1	1
2	∞	0	∞	∞	5
3	∞	3	0	2	2
4	∞	∞	∞	0	−3
5	∞	∞	1	∞	0

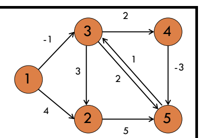
k = 4

	1	2	3	4	5
1	0	2	−1	1	−2
2	∞	0	∞	∞	5
3	∞	3	0	2	−1
4	∞	∞	∞	0	−3
5	∞	4	1	3	0

42

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

return d^V



k = 4

	1	2	3	4	5
1	0	2	−1	1	−2
2	∞	0	∞	∞	5
3	∞	3	0	2	−1
4	∞	∞	∞	0	−3
5	∞	∞	1	∞	0

k = 5

	1	2	3	4	5
1	0	2	−1	1	−2
2	∞	0	6	8	5
3	∞	3	0	2	−1
4	∞	1	−2	0	−3
5	∞	4	1	3	0

Done!

43

Floyd-Warshall analysis

Is it correct?

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

return d^V

44

Floyd-Warshall analysis

Is it correct?

Any assumptions?

```
Floyd-Warshall(G = (V,E,W)):
d⁰ = W        // initialize with edge weights
for k = 1 to V
  for i = 1 to V
    for j = 1 to V
      dijk = min(dij^(k-1), dik^(k-1) + dkj^(k-1))

return d^V
```

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)):
d⁰ = W        // initialize with edge weights
for k = 1 to V
  for i = 1 to V
    for j = 1 to V
      dijk = min(dij^(k-1), dik^(k-1) + dkj^(k-1))

return d^V
```

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., we 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	∞	∞	1	∞	0

47

Floyd-Warshall analysis

Run-time?

```
Floyd-Warshall(G = (V,E,W)):
d⁰ = W        // initialize with edge weights
for k = 1 to V
  for i = 1 to V
    for j = 1 to V
      d_ij k = min(dij^(k-1), dik^(k-1) + dkj^(k-1))

return d^V
```

48

Floyd-Warshall analysis

Run-time: $\theta(V^3)$

```
Floyd-Warshall(G = (V,E,W)):
d0 = 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
```

49

Floyd-Warshall analysis

What type of algorithm is Floyd-Warshall?

```
Floyd-Warshall(G = (V,E,W)):
d0 = 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
```

50

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)):
d0 = 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
```

51

Floyd-Warshall analysis

Space usage?

```
Floyd-Warshall(G = (V,E,W)):
d0 = 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
```

52

Floyd-Warshall: key idea

Label all vertices with a number from 1 to V

$d_{ij}{}^{k}$ = 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 $d_{ij}{}^{k}$)?

53

Floyd-Warshall: key idea

Label all vertices with a number from 1 to V

$d_{ij}{}^{k}$ = 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?

54

Floyd-Warshall analysis

Space usage: $\theta(V^2)$

Only need the current value and the previous

```
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
```

55

All pairs shortest paths

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

Floyd-Warshall: $\theta(V^3)$

56

All pairs shortest paths

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

Easy solution?

57

All pairs shortest paths

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

Run Dijsktras from each vertex!

Running time (in terms of E and V)?

58

All pairs shortest paths

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

Run Dijsktras from each vertex!

$O(V^2 \log V + V E)$
- V calls to Dijkstras
- Dijkstras: $O(V \log V + E)$

59

All pairs shortest paths

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

Floyd-Warshall: $\theta(V^3)$

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

Is this any better?

60

All pairs shortest paths

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

Floyd-Warshall: $\theta(V^3)$

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

If the graph is sparse!

61

All pairs shortest paths

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

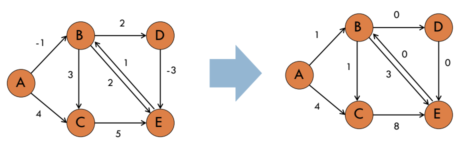
Run Dijsktras from each vertex!

Challenge: Dijkstras assumes positive weights

62

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?

63

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

64

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

Let $s, v_1, v_2, \ldots, v_k, t$ be a path from s to t

The weight in the reweighted graph is:

$$\hat{w}(s,v_1,\ldots,v_k,t) = \underbrace{w(s,v_1) + h(s) - h(v_1)}_{\text{weight for first edge}} + \underbrace{\hat{w}(v_1,\ldots,v_k,t)}_{\text{weight for remaining edges}}$$

65

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

Let $s, v_1, v_2, \ldots, v_k, t$ be a path from s to t

The weight in the reweighted graph is:

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

$$= \underbrace{w(s,v_1) + h(s) - h(v_1)}_{\text{weight for first edge}} + \underbrace{w(v_1,v_2) + h(v_1) - h(v_2)}_{\text{weight for second edge}} + \hat{w}(v_2,\ldots,v_k,t)$$

weight for remaining edges

66

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

Let $s, v_1, v_2, \ldots, v_k, t$ be a path from s to t

The weight in the reweighted graph is:

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

$$= w(s,v_1) + h(s) - h(v_1) + w(v_1,v_2) + h(v_1) - h(v_2) + \hat{w}(v_2,\ldots,v_k,t)$$

$$= w(s,v_1) + h(s) + w(v_1,v_2) - h(v_2) + \hat{w}(v_2,\ldots,v_k,t)$$

67

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

Let $s, v_1, v_2, \ldots, v_k, t$ be a path from s to t

The weight in the reweighted graph is:

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

$$= w(s,v_1) + h(s) - h(v_1) + w(v_1,v_2) + h(v_1) - h(v_2) + \hat{w}(v_2,\ldots,v_k,t)$$

$$= w(s,v_1) + h(s) + w(v_1,v_2) - h(v_2) + \hat{w}(v_2,\ldots,v_k,t)$$

$$= w(s,v_1) + h(s) + w(v_1,v_2) - h(v_2) + \underbrace{w(v_2,v_3) + h(v_2) - h(v_3)}_{\text{weight for third edge}} + \hat{w}(v_3,\ldots,v_k,t)$$

weight for remaining edges

68

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

Let $s, v_1, v_2, \ldots, v_k, t$ be a path from s to t

The weight in the reweighted graph is:

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

$$= w(s,v_1) + h(s) - h(v_1) + w(v_1,v_2) + h(v_1) - h(v_2) + \hat{w}(v_2,\ldots,v_k,t)$$

$$= w(s,v_1) + h(s) + w(v_1,v_2) - h(v_2) + \hat{w}(v_2,\ldots,v_k,t)$$

$$= w(s,v_1) + h(s) + w(v_1,v_2) - h(v_2) + w(v_2,v_3) + h(v_2) - h(v_3) + \hat{w}(v_3,\ldots,v_k,t)$$

$$= w(s,v_1) + h(s) + w(v_1,v_2) + w(v_2,v_3) - h(v_3) + \hat{w}(v_3,\ldots,v_k,t)$$

$$\ldots$$

$$= w(s,v_1,\ldots,v_k,t) + h(s) - h(t)$$

69

Lemma: proof

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

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.

Justification?

70

Lemma: proof

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

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.

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.

71

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

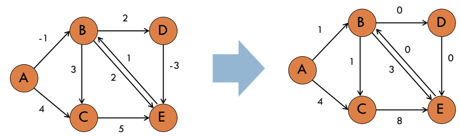
Big question: how do we pick h?

72

Selecting h

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

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



73

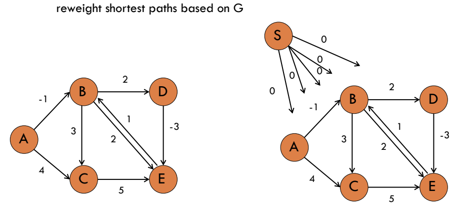
Johnson's algorithm

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

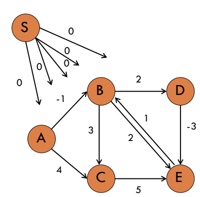
74

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



75

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



S→A: ?
S→B:
S→C:
S→D:
S→E:

76

Create G'

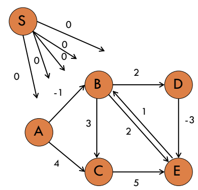
run Bellman-Ford(G',s)

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



S→A: 0
S→B:
S→C:
S→D:
S→E:

77

Create G'

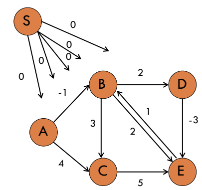
run Bellman-Ford(G',s)

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



S→A: 0
S→B: ?
S→C:
S→D:
S→E:

78

Create G'

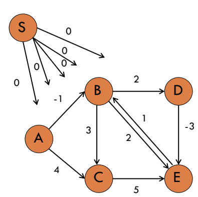
run Bellman-Ford(G',s)

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



S→A: 0
S→B: -2
S→C:
S→D:
S→E:

79

Create G'

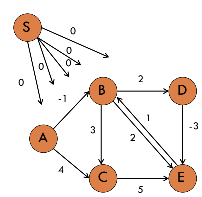
run Bellman-Ford(G',s)

if no negative-weight cycle

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

run Dijkstra's from every vertex

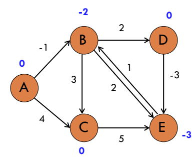
reweight shortest paths based on G



S→A: 0
S→B: -2
S→C: 0
S→D: 0
S→E: -3

80

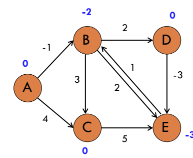
S➔A: 0
S➔B: -2
S➔C: 0
S➔D: 0
S➔E: -3

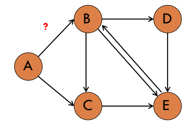


81

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

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





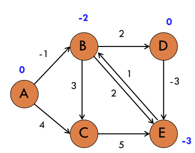
h(v) in blue

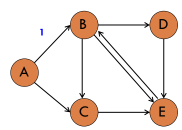
82

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

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

-1 + 0 - -2



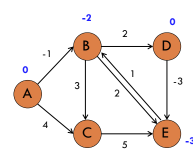


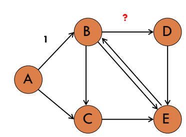
h(v) in blue

83

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

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





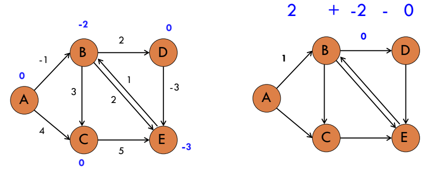
h(v) in blue

84

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

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

2 + -2 - 0

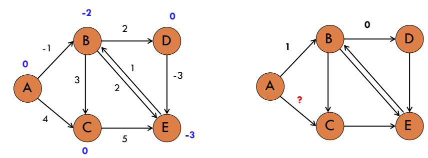


h(v) in blue

85

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

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



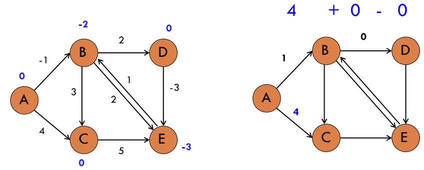
h(v) in blue

86

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

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

4 + 0 - 0

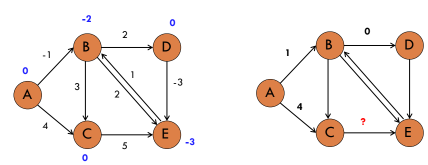


h(v) in blue

87

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

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



h(v) in blue

88

Create G'

run Bellman-Ford(G',s)

if no negative-weight cycle

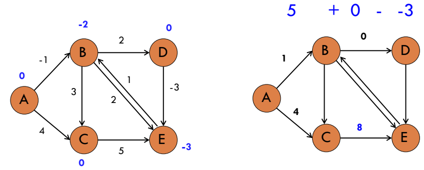
reweight edges in G with h(v)=shortest path from s to v

run Dijkstra's from every vertex

reweight shortest paths based on G

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

5 + 0 - -3



h(v) in blue

89

Create G'

run Bellman-Ford(G',s)

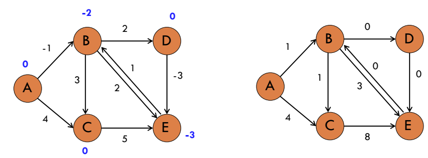
if no negative-weight cycle

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

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



h(v) in blue

90

Create G'

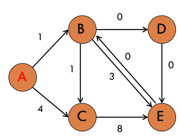
run Bellman-Ford(G',s)

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



91

Create G'

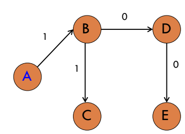
run Bellman-Ford(G',s)

if no negative-weight cycle

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

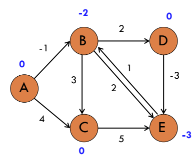
run Dijkstra's from every vertex

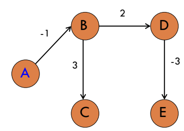
reweight shortest paths based on G



92

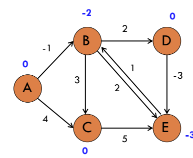
Create G'
run Bellman-Ford(G',s)
if no negative-weight cycle
reweight edges in G with h(v)=shortest path from s to v
run Dijkstra's from every vertex
reweight shortest paths based on G

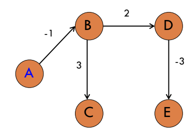




93

A→B: -1
A→C: 2
A→D: 1
A→E: -2





94

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',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

Why does this work (i.e. how do we guarantee that reweighted graph has only positive edges)?

95

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: $h(v) \le h(u) + w(u,v)$

Why?

96

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: $h(v) \le h(u) + w(u,v)$

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)

97

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

$h(v) \le h(u) + w(u,v)$

$\underbrace{w(u,v) + h(u) - h(v)}_{\text{What is this?}} \ge 0$

98

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

$h(v) \le h(u) + w(u,v)$

$\underbrace{w(u,v) + h(u) - h(v)}_{} \ge 0$

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

$\hat{w}(u,v) = w(u,v) + h(u) - h(v) \ge 0$

All edge weights in reweighted graph are non-negative

99

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?

100

Johnson's algorithm

Create G'	θ(V)
run Bellman-Ford(G',s)	θ(VE)
if no negative-weight cycle	
reweight edges in G	θ(E)
run Dijkstra's from every vertex	$O(V^2 \log V + VE)$
reweight shortest paths based on G	θ(E)

Run-time?

101

All pairs shortest paths

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

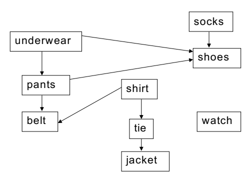
Floyd-Warshall: $\theta(V^3)$

Johnson's: $O(V^2 \log V + V E)$

102

DAGs

Can represent dependency graphs

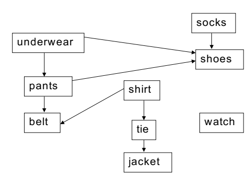


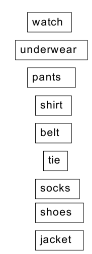
103

Topological sort

A linear ordering of all the vertices such that for all edges (u,v) ∈ E, u appears before v in the ordering

An ordering of the nodes that "obeys" the dependencies, i.e. an activity can't happen until it's dependent activities have happened





104

Topological sort

TOPOLOGICAL-SORT1(G)

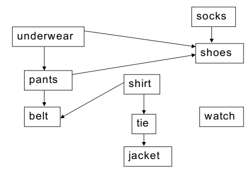
Find a node v with no incoming edges
Delete v from G
Add v to linked list
TOPOLOGICAL-SORT1(G)

105

Topological sort

TOPOLOGICAL-SORT1(G)

Find a node v with no incoming edges
Delete v from G
Add v to linked list
TOPOLOGICAL-SORT1(G)

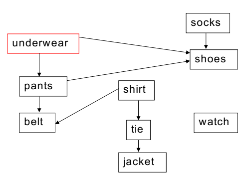


106

Topological sort

TOPOLOGICAL-SORT1(G)

Find a node v with no incoming edges
Delete v from G
Add v to linked list
TOPOLOGICAL-SORT1(G)



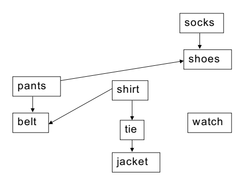
107

Topological sort

TOPOLOGICAL-SORT1(G)

Find a node v with no incoming edges
Delete v from G
Add v to linked list
TOPOLOGICAL-SORT1(G)

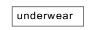
underwear

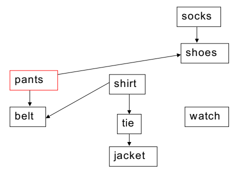


108

Topological sort

TOPOLOGICAL-SORT1(G)
Find a node v with no incoming edges
Delete v from G
Add v to linked list
TOPOLOGICAL-SORT1(G)

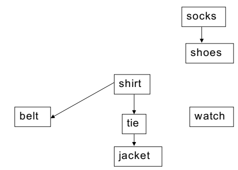




109

Topological sort

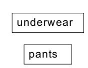
TOPOLOGICAL-SORT1(G)
Find a node v with no incoming edges
Delete v from G
Add v to linked list
TOPOLOGICAL-SORT1(G)

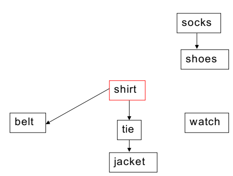


110

Topological sort

TOPOLOGICAL-SORT1(G)
Find a node v with no incoming edges
Delete v from G
Add v to linked list
TOPOLOGICAL-SORT1(G)

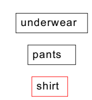


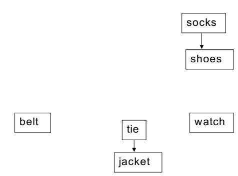


111

Topological sort

TOPOLOGICAL-SORT1(G)
Find a node v with no incoming edges
Delete v from G
Add v to linked list
TOPOLOGICAL-SORT1(G)



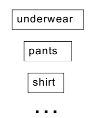


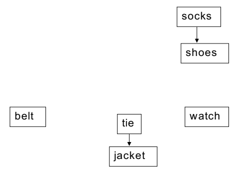
112

Topological sort

TOPOLOGICAL-SORT1(G)

Find a node v with no incoming edges
Delete v from G
Add v to linked list
TOPOLOGICAL-SORT1(G)





113

Running time?

TOPOLOGICAL-SORT1(G)

Find a node v with no incoming edges
Delete v from G
Add v to linked list
TOPOLOGICAL-SORT1(G)

114

Running time?

TOPOLOGICAL-SORT1(G)

Find a node v with no incoming edges — O(|V|+|E|)
Delete v from G
Add v to linked list
TOPOLOGICAL-SORT1(G)

115

Running time?

TOPOLOGICAL-SORT1(G)

Find a node v with no incoming edges
Delete v from G — O(E) overall
Add v to linked list
TOPOLOGICAL-SORT1(G)

116

Running time?

```
TOPOLOGICAL-SORT1(G)
  Find a node v with no incoming edges
  Delete v from G
  Add v to linked list
  TOPOLOGICAL-SORT1(G)
```

How many calls? |V|

117

Running time?

```
TOPOLOGICAL-SORT1(G)
  Find a node v with no incoming edges
  Delete v from G
  Add v to linked list
  TOPOLOGICAL-SORT1(G)
```

Overall running time?

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

118

Can we do better?

```
TOPOLOGICAL-SORT1(G)
  Find a node v with no incoming edges
  Delete v from G
  Add v to linked list
  TOPOLOGICAL-SORT1(G)
```

119

Topological sort 2

```
TOPOLOGICAL-SORT2(G)
  for all edges (u, v) ∈ E
      active[v] ← active[v] + 1
  for all v ∈ V
      if active[v] = 0
          ENQUEUE(S, v)
  while !EMPTY(S)
      u ← DEQUEUE(S)
      add u to linked list
      for each edge (u, v) ∈ E
          active[v] ← active[v] − 1
          if active[v] = 0
              ENQUEUE(S, v)
```

120

Topological sort 2

```
Topological-Sort2(G)
for all edges (u, v) ∈ E
    active[v] ← active[v] + 1
for all v ∈ V
    if active[v] = 0
        Enqueue(S, v)
while !Empty(S)
    u ← Dequeue(S)
    add u to linked list
    for each edge (u, v) ∈ E
        active[v] ← active[v] − 1
        if active[v] = 0
            Enqueue(S, v)
```

121

Topological sort 2

```
Topological-Sort2(G)
for all edges (u, v) ∈ E
    active[v] ← active[v] + 1
for all v ∈ V
    if active[v] = 0
        Enqueue(S, v)
while !Empty(S)
    u ← Dequeue(S)
    add u to linked list
    for each edge (u, v) ∈ E
        active[v] ← active[v] − 1
        if active[v] = 0
            Enqueue(S, v)
```

122

Topological sort 2

```
Topological-Sort2(G)
for all edges (u, v) ∈ E
    active[v] ← active[v] + 1
for all v ∈ V
    if active[v] = 0
        Enqueue(S, v)
while !Empty(S)
    u ← Dequeue(S)
    add u to linked list
    for each edge (u, v) ∈ E
        active[v] ← active[v] − 1
        if active[v] = 0
            Enqueue(S, v)
```

123

Running time?

How many times do we process each node?

How many times do we process each edge?

O(|V| + |E|)

```
Topological-Sort2(G)
for all edges (u, v) ∈ E
    active[v] ← active[v] + 1
for all v ∈ V
    if active[v] = 0
        Enqueue(S, v)
while !Empty(S)
    u ← Dequeue(S)
    add u to linked list
    for each edge (u, v) ∈ E
        active[v] ← active[v] − 1
        if active[v] = 0
            Enqueue(S, v)
```

124

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

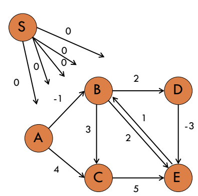
- Call TopologicalSort
- If the length of the list returned ≠ |V| then a cycle exists

125

Handout

126

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



S→A: ?

S→B:

S→C:

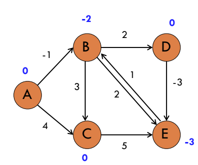
S→D:

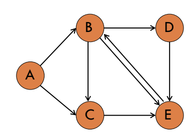
S→E:

127

Reweight the graph on the right based on the h values

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





h(v) in blue

128