# SHORTEST PATHS

David Kauchak
CS 140 – Spring 2023

1

## Admin

Assignment 9

No office hours Friday

2

## Minimum spanning trees

What is the lowest weight set of edges that connects all vertices of an undirected graph with positive weights

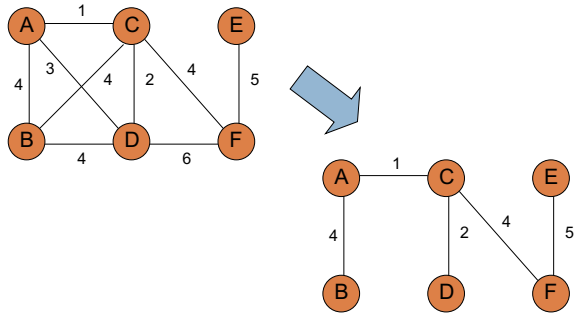
Input: An undirected, positive weight graph, G=(V,E)

Output: A tree T=(V,E') where E' $\subseteq$ E that minimizes

$$weight(T) = \sum_{e \in E'} w_e$$

3

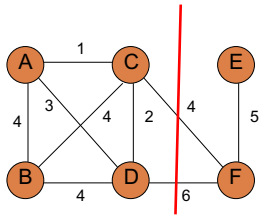
## MST example




4

## Minimum cut property

Given a partition S, let edge e be the minimum cost edge that **crosses** the partition. *Every* minimum spanning tree contains edge e.




5

## Prim's algorithm

Start at some root node and build out the MST by adding the lowest weighted edge out of the MST constructed so far

```
PRIM(G, r)
1   for all v ∈ V
2         key[v] ← ∞
3         prev[v] ← null
4   key[r] ← 0
5   H ← MAKEHEAP(key)
6   while !Empty(H)
7         u ← EXTRACT-MIN(H)
8         visited[u] ← true
9         for each edge (u, v) ∈ E
10              if !visited[v] and w(u, v) < key(v)
11                    DECREASE-KEY(v, w(u, v))
12                    prev[v] ← u
```

6

## Correctness of Prim's?

Can we use the min-cut property?

- Given a partion S, let edge e be the minimum cost edge that **crosses** the partition. *Every* minimum spanning tree contains edge e.

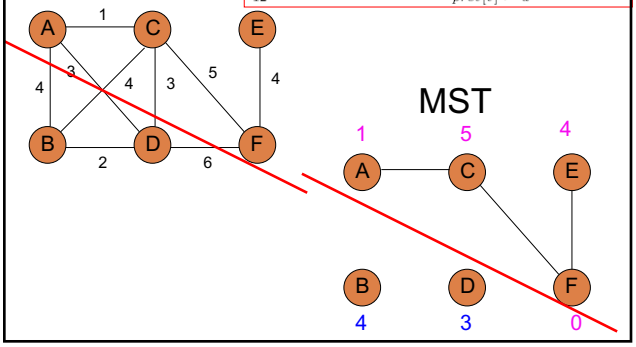
Let S be the set of vertices visited so far

The only time we add a new edge is if it's the lowest weight edge from S to V-S

7

## Prim's

```
6   while !Empty(H)
7         u ← EXTRACT-MIN(H)
8         visited[u] ← true
9         for each edge (u, v) ∈ E
10              if !visited[v] and w(u, v) < key(v)
11                    DECREASE-KEY(v, w(u, v))
12                    prev[v] ← u
```




8

## Running time of Prim's

```
PRIM(G, r)
1  for all v ∈ V
2        key[v] ← ∞
3        prev[v] ← null
4  key[r] ← 0
5  H ← MAKEHEAP(key)
6  while !Empty(H)
7        u ← EXTRACT-MIN(H)
8        visited[u] ← true
9        for each edge (u, v) ∈ E
10             if !visited[v] and w(u, v) < key(v)
11                   DECREASE-KEY(v, w(u, v))
12                   prev[v] ← u
```

9

## Running time of Prim's

```
PRIM(G, r)
1  for all v ∈ V
2        key[v] ← ∞
3        prev[v] ← null
4  key[r] ← 0
5  H ← MAKEHEAP(key)
6  while !Empty(H)
7        u ← EXTRACT-MIN(H)
8        visited[u] ← true
9        for each edge (u, v) ∈ E
10             if !visited[v] and w(u, v) < key(v)
11                   DECREASE-KEY(v, w(u, v))
12                   prev[v] ← u
```

Θ(|V|)

1 call to MakeHeap

|V| calls to Extract-Min

|E| calls to Decrease-Key

10

## Running time of Prim's

| | 1 MakeHeap | \|V\| ExtractMin | \|E\| DecreaseKey | Total |
|---|---|---|---|---|
| Array | θ(\|V\|) | O(\|V\|²) | O(\|E\|) | O(\|V\|²) |
| Bin heap | θ(\|V\|) | O(\|V\| log \|V\|) | O(\|E\| log \|V\|) | O((\|V\|+\|E\|) log \|V\|)<br>O(\|E\| log \|V\|) |
| Fib heap | θ(\|V\|) | O(\|V\| log \|V\|) | O(\|E\|) | **O(\|V\| log \|V\| + \|E\|)** |

Kruskal's: O(|E| log |E| )

11

## Running time of Prim's

When should we use Kruskal's or Prim's?

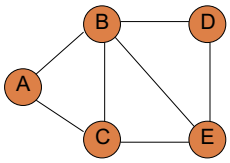
| | 1 MakeHeap | \|V\| ExtractMin | \|E\| DecreaseKey | Total |
|---|---|---|---|---|
| Array | θ(\|V\|) | O(\|V\|²) | O(\|E\|) | O(\|V\|²) |
| Bin heap | θ(\|V\|) | O(\|V\| log \|V\|) | O(\|E\| log \|V\|) | O((\|V\|+\|E\|) log \|V\|)<br>O(\|E\| log \|V\|) |
| Fib heap | θ(\|V\|) | O(\|V\| log \|V\|) | O(\|E\|) | **O(\|V\| log \|V\| + \|E\|)** |

Kruskal's: O(|E| log |E| )

12

## Shortest paths

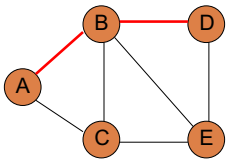
What is the shortest path from a to d?




13

## Shortest paths

How can we find this?

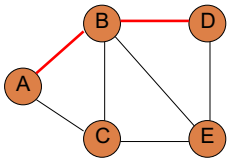



14

## Shortest paths

BFS

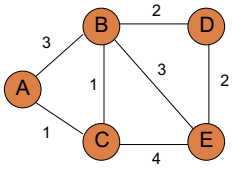
```
BFS(G, s)
1  for each v ∈ V
2        dist[v] = ∞
3  dist[s] = 0
4  ENQUEUE(Q, s)
5  while !EMPTY(Q)
6        u ← DEQUEUE(Q)
7        VISIT(u)
8        for each edge (u, v) ∈ E
9              if dist[v] = ∞
10                   ENQUEUE(Q, v)
11                   dist[v] ← dist[u] + 1
```




15

## Shortest paths

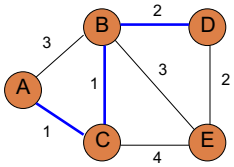
What is the shortest path from a to d?




16

## Shortest paths

What is the shortest path from a to d?




17

## Shortest path algorithms?

18

## Dijkstra's algorithm

What is dist?

What is prev?

How does it work?

What is the run-time?

How do we get the shortest path?

```
DIJKSTRA(G, s)
1  for all v ∈ V
2          dist[v] ← ∞
3          prev[v] ← null
4  dist[s] ← 0
5  Q ← MAKEHEAP(V)
6  while !EMPTY(Q)
7          u ← EXTRACTMIN(Q)
8          for all edges (u, v) ∈ E
9                  if dist[v] > dist[u] + w(u, v)
10                         dist[v] ← dist[u] + w(u, v)
11                         DECREASEKEY(Q, v, dist[v])
12                         prev[v] ← u
```

19

## Dijkstra's algorithm

```
DIJKSTRA(G, s)
1  for all v ∈ V
2          dist[v] ← ∞
3          prev[v] ← null
4  dist[s] ← 0
5  Q ← MAKEHEAP(V)
6  while !EMPTY(Q)
7          u ← EXTRACTMIN(Q)
8          for all edges (u, v) ∈ E
9                  if dist[v] > dist[u] + w(u, v)
10                         dist[v] ← dist[u] + w(u, v)
11                         DECREASEKEY(Q, v, dist[v])
12                         prev[v] ← u
```

```
BFS(G, s)
1  for each v ∈ V
2          dist[v] = ∞
3  dist[s] = 0
4  ENQUEUE(Q, s)
5  while !EMPTY(Q)
6          u ← DEQUEUE(Q)
7          VISIT(U)
8          for each edge (u, v) ∈ E
9                  if dist[v] = ∞
10                         ENQUEUE(Q, v)
11                         dist[v] ← dist[u] + 1
```

20

## Dijkstra's algorithm

prev keeps track of the shortest path

```
DIJKSTRA(G, s)
1  for all v ∈ V
2          dist[v] ← ∞
3          prev[v] ← null
4  dist[s] ← 0
5  Q ← MAKEHEAP(V)
6  while !EMPTY(Q)
7          u ← EXTRACTMIN(Q)
8          for all edges (u, v) ∈ E
9                  if dist[v] > dist[u] + w(u, v)
10                         dist[v] ← dist[u] + w(u, v)
11                         DECREASEKEY(Q, v, dist[v])
12                         prev[v] ← u
```

```
BFS(G, s)
1  for each v ∈ V
2          dist[v] = ∞
3  dist[s] = 0
4  ENQUEUE(Q, s)
5  while !EMPTY(Q)
6          u ← DEQUEUE(Q)
7          VISIT(u)
8          for each edge (u, v) ∈ E
9                  if dist[v] = ∞
10                         ENQUEUE(Q, v)
11                         dist[v] ← dist[u] + 1
```

21

## Dijkstra's algorithm

```
DIJKSTRA(G, s)
1  for all v ∈ V
2          dist[v] ← ∞
3          prev[v] ← null
4  dist[s] ← 0
5  Q ← MAKEHEAP(V)
6  while !EMPTY(Q)
7          u ← EXTRACTMIN(Q)
8          for all edges (u, v) ∈ E
9                  if dist[v] > dist[u] + w(u, v)
10                         dist[v] ← dist[u] + w(u, v)
11                         DECREASEKEY(Q, v, dist[v])
12                         prev[v] ← u
```

```
BFS(G, s)
1  for each v ∈ V
2          dist[v] = ∞
3  dist[s] = 0
4  ENQUEUE(Q, s)
5  while !EMPTY(Q)
6          u ← DEQUEUE(Q)
7          VISIT(u)
8          for each edge (u, v) ∈ E
9                  if dist[v] = ∞
10                         ENQUEUE(Q, v)
11                         dist[v] ← dist[u] + 1
```

22

## Dijkstra's algorithm

```
DIJKSTRA(G, s)
1  for all v ∈ V
2          dist[v] ← ∞
3          prev[v] ← null
4  dist[s] ← 0
5  Q ← MAKEHEAP(V)
6  while !EMPTY(Q)
7          u ← EXTRACTMIN(Q)
8          for all edges (u, v) ∈ E
9                  if dist[v] > dist[u] + w(u, v)
10                         dist[v] ← dist[u] + w(u, v)
11                         DECREASEKEY(Q, v, dist[v])
12                         prev[v] ← u
```

```
BFS(G, s)
1  for each v ∈ V
2          dist[v] = ∞
3  dist[s] = 0
4  ENQUEUE(Q, s)
5  while !EMPTY(Q)
6          u ← DEQUEUE(Q)
7          VISIT(u)
8          for each edge (u, v) ∈ E
9                  if dist[v] = ∞
10                         ENQUEUE(Q, v)
11                         dist[v] ← dist[u] + 1
```

23

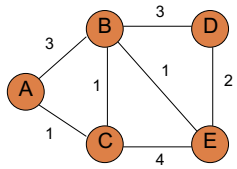
## Dijkstra's algorithm

```
DIJKSTRA(G, s)
1  for all v ∈ V
2          dist[v] ← ∞
3          prev[v] ← null
4  dist[s] ← 0
5  Q ← MAKEHEAP(V)
6  while !EMPTY(Q)
7          u ← EXTRACTMIN(Q)
8          for all edges (u, v) ∈ E
9                  if dist[v] > dist[u] + w(u, v)
10                         dist[v] ← dist[u] + w(u, v)
11                         DECREASEKEY(Q, v, dist[v])
12                         prev[v] ← u
```

```
BFS(G, s)
1  for each v ∈ V
2          dist[v] = ∞
3  dist[s] = 0
4  ENQUEUE(Q, s)
5  while !EMPTY(Q)
6          u ← DEQUEUE(Q)
7          VISIT(u)
8          for each edge (u, v) ∈ E
9                  if dist[v] = ∞
10                         ENQUEUE(Q, v)
11                         dist[v] ← dist[u] + 1
```

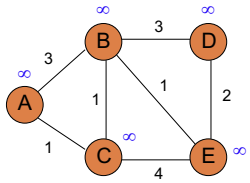
24

```
DIJKSTRA(G, s)
1  for all v ∈ V
2          dist[v] ← ∞
3          prev[v] ← null
4  dist[s] ← 0
5  Q ← MAKEHEAP(V)
6  while !EMPTY(Q)
7          u ← EXTRACTMIN(Q)
8          for all edges (u, v) ∈ E
9                  if dist[v] > dist[u] + w(u, v)
10                         dist[v] ← dist[u] + w(u, v)
11                         DECREASEKEY(Q, v, dist[v])
12                         prev[v] ← u
```



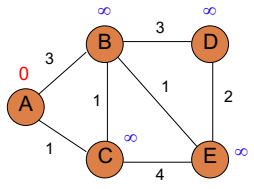

25

```
DIJKSTRA(G, s)
1  for all v ∈ V
2          dist[v] ← ∞
3          prev[v] ← null
4  dist[s] ← 0
5  Q ← MAKEHEAP(V)
6  while !EMPTY(Q)
7          u ← EXTRACTMIN(Q)
8          for all edges (u, v) ∈ E
9                  if dist[v] > dist[u] + w(u, v)
10                         dist[v] ← dist[u] + w(u, v)
11                         DECREASEKEY(Q, v, dist[v])
12                         prev[v] ← u
```




26

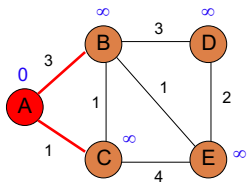
```
DIJKSTRA(G, s)
1  for all v ∈ V
2          dist[v] ← ∞
3          prev[v] ← null
4  dist[s] ← 0
5  Q ← MAKEHEAP(V)
6  while !EMPTY(Q)
7          u ← EXTRACTMIN(Q)
8          for all edges (u, v) ∈ E
9                  if dist[v] > dist[u] + w(u, v)
10                         dist[v] ← dist[u] + w(u, v)
11                         DECREASEKEY(Q, v, dist[v])
12                         prev[v] ← u
```




| Heap |
|---|
| A 0 |
| B ∞ |
| C ∞ |
| D ∞ |
| E ∞ |

27

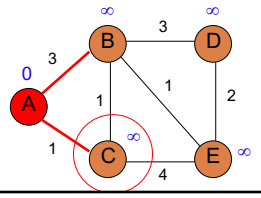
```
DIJKSTRA(G, s)
1  for all v ∈ V
2          dist[v] ← ∞
3          prev[v] ← null
4  dist[s] ← 0
5  Q ← MAKEHEAP(V)
6  while !EMPTY(Q)
7          u ← EXTRACTMIN(Q)
8          for all edges (u, v) ∈ E
9                  if dist[v] > dist[u] + w(u, v)
10                         dist[v] ← dist[u] + w(u, v)
11                         DECREASEKEY(Q, v, dist[v])
12                         prev[v] ← u
```




| Heap |
|---|
| B ∞ |
| C ∞ |
| D ∞ |
| E ∞ |

28

```
DIJKSTRA(G, s)
1  for all v ∈ V
2          dist[v] ← ∞
3          prev[v] ← null
4  dist[s] ← 0
5  Q ← MAKEHEAP(V)
6  while !EMPTY(Q)
7          u ← EXTRACTMIN(Q)
8          for all edges (u, v) ∈ E
9                  if dist[v] > dist[u] + w(u, v)
10                         dist[v] ← dist[u] + w(u, v)
11                         DECREASEKEY(Q, v, dist[v])
12                         prev[v] ← u
```

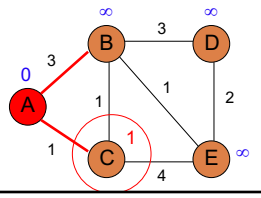



Heap

B ∞
C ∞
D ∞
E ∞

29

```
DIJKSTRA(G, s)
1  for all v ∈ V
2          dist[v] ← ∞
3          prev[v] ← null
4  dist[s] ← 0
5  Q ← MAKEHEAP(V)
6  while !EMPTY(Q)
7          u ← EXTRACTMIN(Q)
8          for all edges (u, v) ∈ E
9                  if dist[v] > dist[u] + w(u, v)
10                         dist[v] ← dist[u] + w(u, v)
11                         DECREASEKEY(Q, v, dist[v])
12                         prev[v] ← u
```

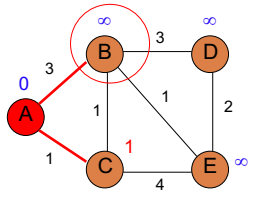



Heap

C 1
B ∞
D ∞
E ∞

30

```
DIJKSTRA(G, s)
1  for all v ∈ V
2          dist[v] ← ∞
3          prev[v] ← null
4  dist[s] ← 0
5  Q ← MAKEHEAP(V)
6  while !EMPTY(Q)
7          u ← EXTRACTMIN(Q)
8          for all edges (u, v) ∈ E
9                  if dist[v] > dist[u] + w(u, v)
10                         dist[v] ← dist[u] + w(u, v)
11                         DECREASEKEY(Q, v, dist[v])
12                         prev[v] ← u
```

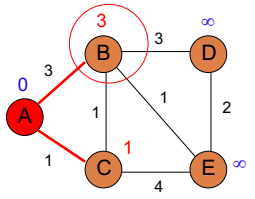



Heap

C 1
B ∞
D ∞
E ∞

31

```
DIJKSTRA(G, s)
1  for all v ∈ V
2          dist[v] ← ∞
3          prev[v] ← null
4  dist[s] ← 0
5  Q ← MAKEHEAP(V)
6  while !EMPTY(Q)
7          u ← EXTRACTMIN(Q)
8          for all edges (u, v) ∈ E
9                  if dist[v] > dist[u] + w(u, v)
10                         dist[v] ← dist[u] + w(u, v)
11                         DECREASEKEY(Q, v, dist[v])
12                         prev[v] ← u
```

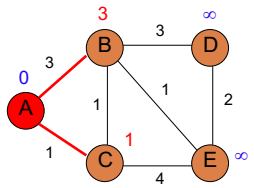



Heap

C 1
B 3
D ∞
E ∞

32

```
DIJKSTRA(G, s)
1  for all v ∈ V
2          dist[v] ← ∞
3          prev[v] ← null
4  dist[s] ← 0
5  Q ← MAKEHEAP(V)
6  while !EMPTY(Q)
7          u ← EXTRACTMIN(Q)
8          for all edges (u, v) ∈ E
9                  if dist[v] > dist[u] + w(u, v)
10                         dist[v] ← dist[u] + w(u, v)
11                         DECREASEKEY(Q, v, dist[v])
12                         prev[v] ← u
```

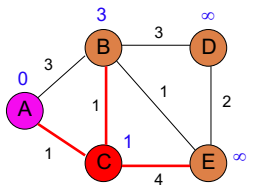



Heap

C 1
B 3
D ∞
E ∞

33

```
DIJKSTRA(G, s)
1  for all v ∈ V
2          dist[v] ← ∞
3          prev[v] ← null
4  dist[s] ← 0
5  Q ← MAKEHEAP(V)
6  while !EMPTY(Q)
7          u ← EXTRACTMIN(Q)
8          for all edges (u, v) ∈ E
9                  if dist[v] > dist[u] + w(u, v)
10                         dist[v] ← dist[u] + w(u, v)
11                         DECREASEKEY(Q, v, dist[v])
12                         prev[v] ← u
```

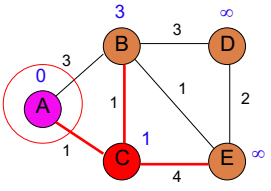



Heap

B 3
D ∞
E ∞

34

```
DIJKSTRA(G, s)
1  for all v ∈ V
2          dist[v] ← ∞
3          prev[v] ← null
4  dist[s] ← 0
5  Q ← MAKEHEAP(V)
6  while !EMPTY(Q)
7          u ← EXTRACTMIN(Q)
8          for all edges (u, v) ∈ E
9                  if dist[v] > dist[u] + w(u, v)
10                         dist[v] ← dist[u] + w(u, v)
11                         DECREASEKEY(Q, v, dist[v])
12                         prev[v] ← u
```

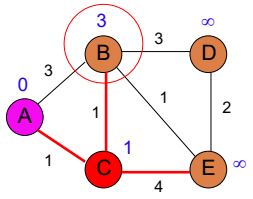



Heap

B 3
D ∞
E ∞

35

```
DIJKSTRA(G, s)
1  for all v ∈ V
2          dist[v] ← ∞
3          prev[v] ← null
4  dist[s] ← 0
5  Q ← MAKEHEAP(V)
6  while !EMPTY(Q)
7          u ← EXTRACTMIN(Q)
8          for all edges (u, v) ∈ E
9                  if dist[v] > dist[u] + w(u, v)
10                         dist[v] ← dist[u] + w(u, v)
11                         DECREASEKEY(Q, v, dist[v])
12                         prev[v] ← u
```




Heap

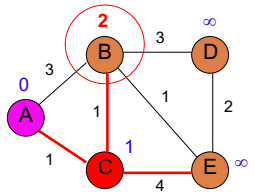
B 3
D ∞
E ∞

36

```
DIJKSTRA(G, s)
1  for all v ∈ V
2          dist[v] ← ∞
3          prev[v] ← null
4  dist[s] ← 0
5  Q ← MAKEHEAP(V)
6  while !EMPTY(Q)
7          u ← EXTRACTMIN(Q)
8          for all edges (u, v) ∈ E
9                  if dist[v] > dist[u] + w(u, v)
10                         dist[v] ← dist[u] + w(u, v)
11                         DECREASEKEY(Q, v, dist[v])
12                         prev[v] ← u
```

Heap

B 2
D ∞
E ∞

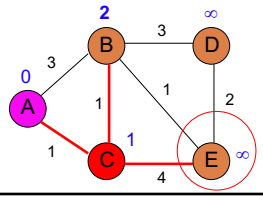



37

```
DIJKSTRA(G, s)
1  for all v ∈ V
2          dist[v] ← ∞
3          prev[v] ← null
4  dist[s] ← 0
5  Q ← MAKEHEAP(V)
6  while !EMPTY(Q)
7          u ← EXTRACTMIN(Q)
8          for all edges (u, v) ∈ E
9                  if dist[v] > dist[u] + w(u, v)
10                         dist[v] ← dist[u] + w(u, v)
11                         DECREASEKEY(Q, v, dist[v])
12                         prev[v] ← u
```

Heap

B 2
D ∞
E ∞

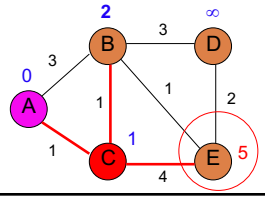



38

```
DIJKSTRA(G, s)
1  for all v ∈ V
2          dist[v] ← ∞
3          prev[v] ← null
4  dist[s] ← 0
5  Q ← MAKEHEAP(V)
6  while !EMPTY(Q)
7          u ← EXTRACTMIN(Q)
8          for all edges (u, v) ∈ E
9                  if dist[v] > dist[u] + w(u, v)
10                         dist[v] ← dist[u] + w(u, v)
11                         DECREASEKEY(Q, v, dist[v])
12                         prev[v] ← u
```

Heap

B 2
E 5
D ∞

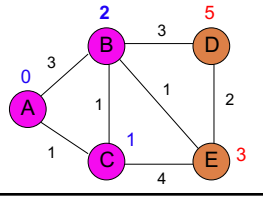



39

```
DIJKSTRA(G, s)
1  for all v ∈ V
2          dist[v] ← ∞
3          prev[v] ← null
4  dist[s] ← 0
5  Q ← MAKEHEAP(V)
6  while !EMPTY(Q)
7          u ← EXTRACTMIN(Q)
8          for all edges (u, v) ∈ E
9                  if dist[v] > dist[u] + w(u, v)
10                         dist[v] ← dist[u] + w(u, v)
11                         DECREASEKEY(Q, v, dist[v])
12                         prev[v] ← u
```

Heap

E 3
D 5

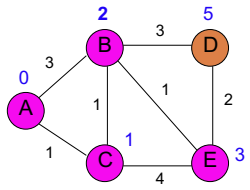



40

```
DIJKSTRA(G, s)
1  for all v ∈ V
2          dist[v] ← ∞
3          prev[v] ← null
4  dist[s] ← 0
5  Q ← MAKEHEAP(V)
6  while !EMPTY(Q)
7          u ← EXTRACTMIN(Q)
8          for all edges (u, v) ∈ E
9                  if dist[v] > dist[u] + w(u, v)
10                         dist[v] ← dist[u] + w(u, v)
11                         DECREASEKEY(Q, v, dist[v])
12                         prev[v] ← u
```

Heap

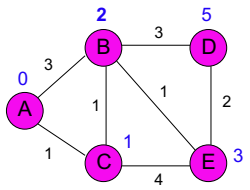
D 5




41

```
DIJKSTRA(G, s)
1  for all v ∈ V
2          dist[v] ← ∞
3          prev[v] ← null
4  dist[s] ← 0
5  Q ← MAKEHEAP(V)
6  while !EMPTY(Q)
7          u ← EXTRACTMIN(Q)
8          for all edges (u, v) ∈ E
9                  if dist[v] > dist[u] + w(u, v)
10                         dist[v] ← dist[u] + w(u, v)
11                         DECREASEKEY(Q, v, dist[v])
12                         prev[v] ← u
```

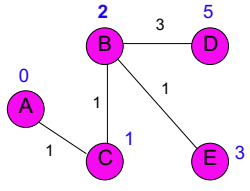
Heap




42

```
DIJKSTRA(G, s)
1  for all v ∈ V
2          dist[v] ← ∞
3          prev[v] ← null
4  dist[s] ← 0
5  Q ← MAKEHEAP(V)
6  while !EMPTY(Q)
7          u ← EXTRACTMIN(Q)
8          for all edges (u, v) ∈ E
9                  if dist[v] > dist[u] + w(u, v)
10                         dist[v] ← dist[u] + w(u, v)
11                         DECREASEKEY(Q, v, dist[v])
12                         prev[v] ← u
```

Heap

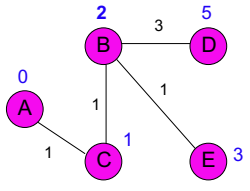



Prev

43

```
DIJKSTRA(G, s)
1  for all v ∈ V
2          dist[v] ← ∞
3          prev[v] ← null
4  dist[s] ← 0
5  Q ← MAKEHEAP(V)
6  while !EMPTY(Q)
7          u ← EXTRACTMIN(Q)
8          for all edges (u, v) ∈ E
9                  if dist[v] > dist[u] + w(u, v)
10                         dist[v] ← dist[u] + w(u, v)
11                         DECREASEKEY(Q, v, dist[v])
12                         prev[v] ← u
```

Heap




Prev

How do we get the actual paths?

44

## Is Dijkstra's algorithm correct?

Invariant: For every vertex removed from the heap, dist[v] is the actual shortest distance from s to v

```
DIJKSTRA(G, s)
 1  for all v ∈ V
 2          dist[v] ← ∞
 3          prev[v] ← null
 4  dist[s] ← 0
 5  Q ← MAKEHEAP(V)
 6  while !EMPTY(Q)
 7          u ← EXTRACTMIN(Q)
 8          for all edges (u, v) ∈ E
 9                  if dist[v] > dist[u] + w(u, v)
10                          dist[v] ← dist[u] + w(u, v)
11                          DECREASEKEY(Q, v, dist[v])
12                          prev[v] ← u
```

proof?

45

## Is Dijkstra's algorithm correct?

Invariant: For every vertex removed from the heap, dist[v] is the actual shortest distance from s to v

- The only time a vertex gets visited is when the distance from s to that vertex is smaller than the distance to any remaining vertex
- Therefore, there cannot be any other path that hasn't been visited already that would result in a shorter path

46

## Running time?

```
DIJKSTRA(G, s)
 1  for all v ∈ V
 2          dist[v] ← ∞
 3          prev[v] ← null
 4  dist[s] ← 0
 5  Q ← MAKEHEAP(V)
 6  while !EMPTY(Q)
 7          u ← EXTRACTMIN(Q)
 8          for all edges (u, v) ∈ E
 9                  if dist[v] > dist[u] + w(u, v)
10                          dist[v] ← dist[u] + w(u, v)
11                          DECREASEKEY(Q, v, dist[v])
12                          prev[v] ← u
```

47

## Running time?

```
DIJKSTRA(G, s)
 1  for all v ∈ V
 2          dist[v] ← ∞
 3          prev[v] ← null
 4  dist[s] ← 0
 5  Q ← MAKEHEAP(V)
 6  while !EMPTY(Q)
 7          u ← EXTRACTMIN(Q)
 8          for all edges (u, v) ∈ E
 9                  if dist[v] > dist[u] + w(u, v)
10                          dist[v] ← dist[u] + w(u, v)
11                          DECREASEKEY(Q, v, dist[v])
12                          prev[v] ← u
```

1 call to MakeHeap

48

## Running time?

```
DIJKSTRA(G, s)
1  for all v ∈ V
2          dist[v] ← ∞
3          prev[v] ← null
4  dist[s] ← 0
5  Q ← MAKEHEAP(V)
6  while !EMPTY(Q)
7          u ← EXTRACTMIN(Q)
8          for all edges (u, v) ∈ E
9                  if dist[v] > dist[u] + w(u, v)
10                         dist[v] ← dist[u] + w(u, v)
11                         DECREASEKEY(Q, v, dist[v])
12                         prev[v] ← u
```

|V| iterations

49

## Running time?

```
DIJKSTRA(G, s)
1  for all v ∈ V
2          dist[v] ← ∞
3          prev[v] ← null
4  dist[s] ← 0
5  Q ← MAKEHEAP(V)
6  while !EMPTY(Q)
7          u ← EXTRACTMIN(Q)
8          for all edges (u, v) ∈ E
9                  if dist[v] > dist[u] + w(u, v)
10                         dist[v] ← dist[u] + w(u, v)
11                         DECREASEKEY(Q, v, dist[v])
12                         prev[v] ← u
```

|V| calls

50

## Running time?

```
DIJKSTRA(G, s)
1  for all v ∈ V
2          dist[v] ← ∞
3          prev[v] ← null
4  dist[s] ← 0
5  Q ← MAKEHEAP(V)
6  while !EMPTY(Q)
7          u ← EXTRACTMIN(Q)
8          for all edges (u, v) ∈ E
9                  if dist[v] > dist[u] + w(u, v)
10                         dist[v] ← dist[u] + w(u, v)
11                         DECREASEKEY(Q, v, dist[v])
12                         prev[v] ← u
```

O(|E|) calls

51

## Running time?

Depends on the heap implementation

| | 1 MakeHeap | \|V\| ExtractMin | \|E\| DecreaseKey | Total |
|---|---|---|---|---|
| Array | O(\|V\|) | O(\|V\|$^2$) | O(\|E\|) | O(\|V\|$^2$) |
| Bin heap | O(\|V\|) | O(\|V\| log \|V\|) | O(\|E\| log \|V\|) | O((\|V\|+\|E\|) log \|V\|)<br>O(\|E\| log \|V\|) |

52

## Running time?

Depends on the heap implementation

| | 1 MakeHeap | \|V\| ExtractMin | \|E\| DecreaseKey | Total |
|---|---|---|---|---|
| Array | O(\|V\|) | O(\|V\|$^2$) | O(\|E\|) | O(\|V\|$^2$) |
| Bin heap | O(\|V\|) | O(\|V\| log \|V\|) | O(\|E\| log \|V\|) | O((\|V\|+\|E\|) log \|V\|)<br>O(\|E\| log \|V\|) |

Is this an improvement? If |E| < |V|$^2$ / log |V|

53

## Running time?

Depends on the heap implementation

| | 1 MakeHeap | \|V\| ExtractMin | \|E\| DecreaseKey | Total |
|---|---|---|---|---|
| Array | O(\|V\|) | O(\|V\|$^2$) | O(\|E\|) | O(\|V\|$^2$) |
| Bin heap | O(\|V\|) | O(\|V\| log \|V\|) | O(\|E\| log \|V\|) | O((\|V\|+\|E\|) log \|V\|)<br>O(\|E\| log \|V\|) |
| Fib heap | O(\|V\|) | O(\|V\| log \|V\|) | O(\|E\|) | O(\|V\| log \|V\| + \|E\|) |

54

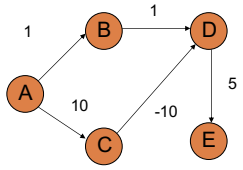
## Dijkstra's vs Prim's

```
DIJKSTRA(G, s)
1  for all v ∈ V
2          dist[v] ← ∞
3          prev[v] ← null
4  dist[s] ← 0
5  Q ← MAKEHEAP(V)
6  while !EMPTY(Q)
7          u ← EXTRACTMIN(Q)
8          for all edges (u, v) ∈ E
9                  if dist[v] > dist[u] + w(u, v)
10                         dist[v] ← dist[u] + w(u, v)
11                         DECREASEKEY(Q, v, dist[v])
12                         prev[v] ← u
```

```
PRIM(G, r)
1  for all v ∈ V
2          key[v] ← ∞
3          prev[v] ← null
4  key[r] ← 0
5  H ← MAKEHEAP(key)
6  while !Empty(H)
7          u ← EXTRACT-MIN(H)
8          visited[u] ← true
9          for each edge (u, v) ∈ E
10                 if !visited[v] and w(u, v) < key(v)
11                         DECREASE-KEY(v, w(u, v))
12                         prev[v] ← u
```

55

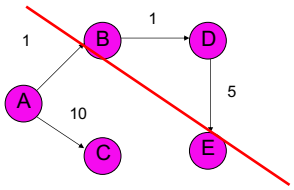
## What about Dijkstra's on…?




56

## What about Dijkstra's on…?

Dijkstra's algorithm only works for positive edge weights




57

## Is Dijkstra's algorithm correct?

Invariant: For every vertex removed from the heap, dist[v] is the actual shortest distance from s to v

- The only time a vertex gets visited is when the distance from s to that vertex is smaller than the distance to any remaining vertex
- Therefore, there cannot be any other path that hasn't been visited already that would result in a shorter path

We relied on having positive edge weights for correctness!

58

## Bounding the distance

Another invariant: For each vertex v, dist[v] is an upper bound on the actual shortest distance

```
DIJKSTRA(G, s)
1  for all v ∈ V
2          dist[v] ← ∞
3          prev[v] ← null
4  dist[s] ← 0
5  Q ← MAKEHEAP(V)
6  while !EMPTY(Q)
7          u ← EXTRACTMIN(Q)
8          for all edges (u, v) ∈ E
9                  if dist[v] > dist[u] + w(u, v)
10                         dist[v] ← dist[u] + w(u, v)
11                         DECREASEKEY(Q, v, dist[v])
12                         prev[v] ← u
```

Is this a valid invariant?

59

## Bounding the distance

Another invariant: For each vertex v, dist[v] is an upper bound on the actual shortest distance

- start off at ∞
- only update the value if we find a shorter distance

An update procedure

$$dist[v] = \min\{dist[v], dist[u] + w(u,v)\}$$

60

$$dist[v] = \min\{dist[v], dist[u] + w(u,v)\}$$

Can we ever go wrong applying this update rule?

- We can apply this rule as many times as we want and will never underestimate dist[v]

When will dist[v] be right?

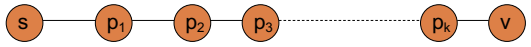
- If u is along the shortest path to v and dist[u] is correct

61

$$dist[v] = \min\{dist[v], dist[u] + w(u,v)\}$$

dist[v] will be right if u is along the shortest path to v and dist[u] is correct

Consider the shortest path from s to v

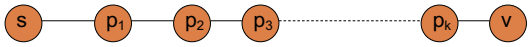



62

$$dist[v] = \min\{dist[v], dist[u] + w(u,v)\}$$

dist[v] will be right if u is along the shortest path to v and dist[u] is correct

What happens if we update all of the vertices with the above update?

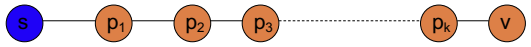



63

$$dist[v] = \min\{dist[v], dist[u] + w(u,v)\}$$

dist[v] will be right if u is along the shortest path to v and dist[u] is correct

What happens if we update all of the vertices with the above update?



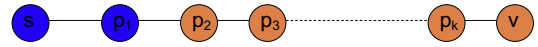

correct

64

$$dist[v] = \min\{dist[v], dist[u] + w(u,v)\}$$

dist[v] will be right if u is along the shortest path to v and dist[u] is correct

What happens if we update all of the vertices with the above update?



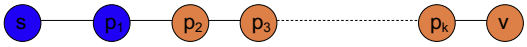

correct correct

65

$$dist[v] = \min\{dist[v], dist[u] + w(u,v)\}$$

dist[v] will be right if u is along the shortest path to v and dist[u] is correct

Does the order that we update the vertices matter?




correct correct

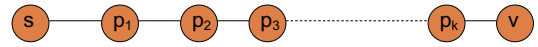
66

$$dist[v] = \min\{dist[v], dist[u] + w(u,v)\}$$

dist[v] will be right if u is along the shortest path to v and dist[u] is correct

How many times do we have to do this for vertex $p_i$ to have the correct shortest path from s?

- i times



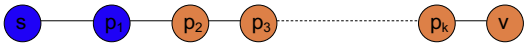

67

$$dist[v] = \min\{dist[v], dist[u] + w(u,v)\}$$

dist[v] will be right if u is along the shortest path to v and dist[u] is correct

How many times do we have to do this for vertex $p_i$ to have the correct shortest path from s?

- i times




correct correct

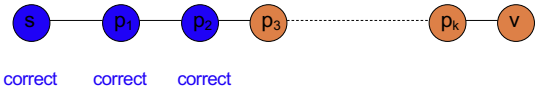
68

$$dist[v] = \min\{dist[v], dist[u] + w(u,v)\}$$

dist[v] will be right if u is along the shortest path to v and dist[u] is correct

How many times do we have to do this for vertex $p_i$ to have the correct shortest path from s?

- i times



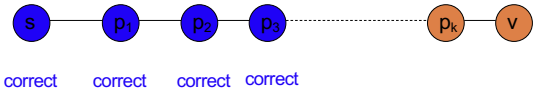

69

$$dist[v] = \min\{dist[v], dist[u] + w(u,v)\}$$

dist[v] will be right if u is along the shortest path to v and dist[u] is correct

How many times do we have to do this for vertex $p_i$ to have the correct shortest path from s?

- i times



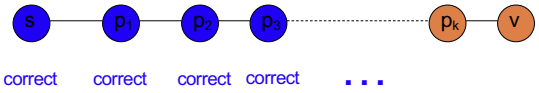

70

$$dist[v] = \min\{dist[v], dist[u] + w(u,v)\}$$

dist[v] will be right if u is along the shortest path to v and dist[u] is correct

How many times do we have to do this for vertex $p_i$ to have the correct shortest path from s?

- i times



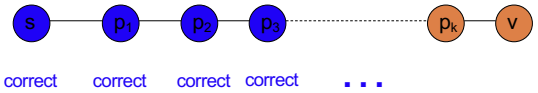

71

$$dist[v] = \min\{dist[v], dist[u] + w(u,v)\}$$

dist[v] will be right if u is along the shortest path to v and dist[u] is correct

What is the longest (vertex-wise) the path from s to any node v can be?

- |V| - 1 edges/vertices




72

## Bellman-Ford algorithm

```
BELLMAN-FORD(G, s)
1  for all v ∈ V
2          dist[v] ← ∞
3          prev[v] ← null
4  dist[s] ← 0
5  for i ← 1 to |V| − 1
6          for all edges (u, v) ∈ E
7                  if dist[v] > dist[u] + w(u, v)
8                          dist[v] ← dist[u] + w(u, v)
9                          prev[v] ← u
10 for all edges (u, v) ∈ E
11         if dist[v] > dist[u] + w(u, v)
12                 return false
```

73

## Bellman-Ford algorithm

```
BELLMAN-FORD(G, s)
1  for all v ∈ V
2          dist[v] ← ∞
3          prev[v] ← null
4  dist[s] ← 0
5  for i ← 1 to |V| − 1
6          for all edges (u, v) ∈ E
7                  if dist[v] > dist[u] + w(u, v)
8                          dist[v] ← dist[u] + w(u, v)
9                          prev[v] ← u
10 for all edges (u, v) ∈ E
11         if dist[v] > dist[u] + w(u, v)
12                 return false
```

Initialize all the distances

do it |V| -1 times

iterate over all edges/vertices and apply update rule

74

## Bellman-Ford algorithm

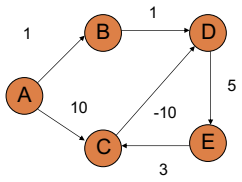
```
BELLMAN-FORD(G, s)
1  for all v ∈ V
2          dist[v] ← ∞
3          prev[v] ← null
4  dist[s] ← 0
5  for i ← 1 to |V| − 1
6          for all edges (u, v) ∈ E
7                  if dist[v] > dist[u] + w(u, v)
8                          dist[v] ← dist[u] + w(u, v)
9                          prev[v] ← u
10 for all edges (u, v) ∈ E
11         if dist[v] > dist[u] + w(u, v)
12                 return false
```

check for negative cycles

75

## Negative cycles

What is the shortest path from a to e?



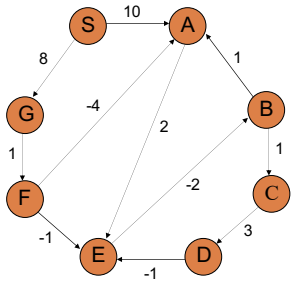

76

## Bellman-Ford algorithm

```
BELLMAN-FORD(G, s)
1  for all v ∈ V
2          dist[v] ← ∞
3          prev[v] ← null
4  dist[s] ← 0
5  for i ← 1 to |V| − 1
6          for all edges (u, v) ∈ E
7                  if dist[v] > dist[u] + w(u, v)
8                          dist[v] ← dist[u] + w(u, v)
9                          prev[v] ← u
10 for all edges (u, v) ∈ E
11         if dist[v] > dist[u] + w(u, v)
12                 return false
```

77

## Bellman-Ford algorithm




How many edges is the shortest path from s to:

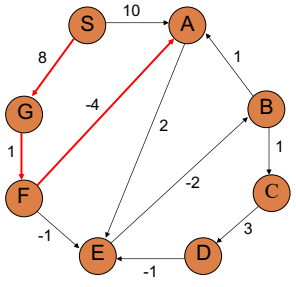
A:

B:

D:

78

## Bellman-Ford algorithm

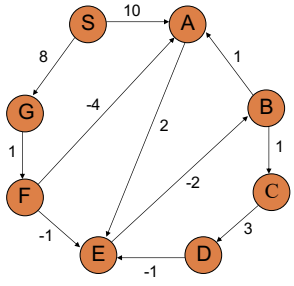



How many edges is the shortest path from s to:

A: 3

79

## Bellman-Ford algorithm



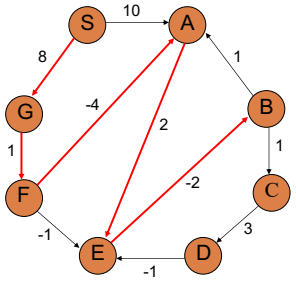

How many edges is the shortest path from s to:

A: 3

B:

80

## Bellman-Ford algorithm



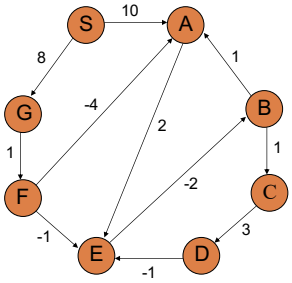

How many edges is the shortest path from s to:

A: 3

B: 5

81

## Bellman-Ford algorithm




How many edges is the shortest path from s to:

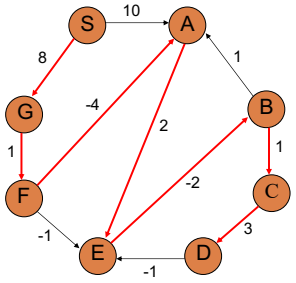
A: 3

B: 5

D:

82

## Bellman-Ford algorithm




How many edges is the shortest path from s to:

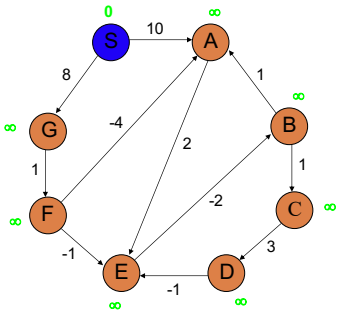
A: 3

B: 5

D: 7

83

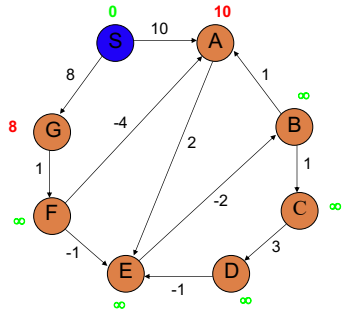
## Bellman-Ford algorithm




Iteration: 0

84

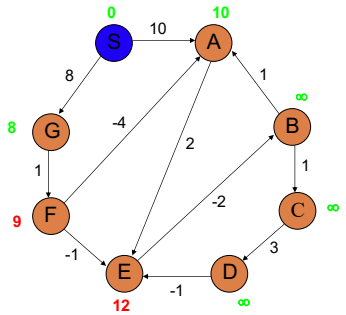
## Bellman-Ford algorithm




Iteration: 1

85

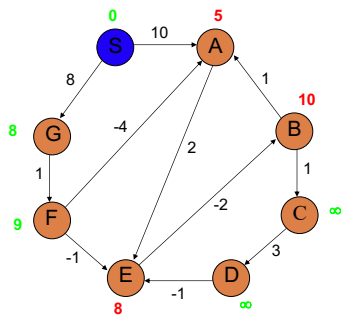
## Bellman-Ford algorithm




Iteration: 2

86

## Bellman-Ford algorithm

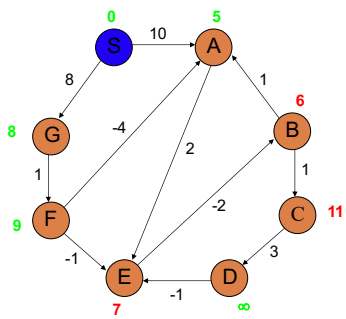



Iteration: 3

A has the correct distance and path

87

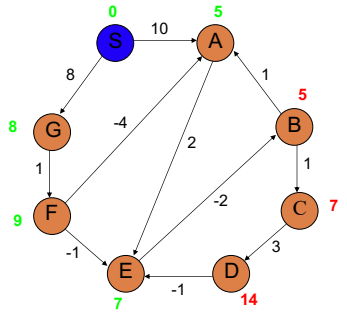
## Bellman-Ford algorithm




Iteration: 4

88

## Bellman-Ford algorithm

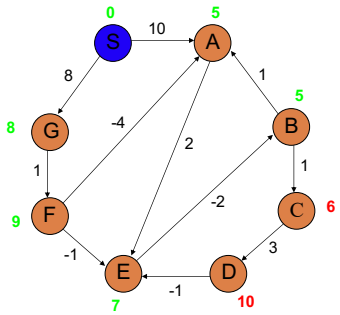



Iteration: 5

B has the correct distance and path

89

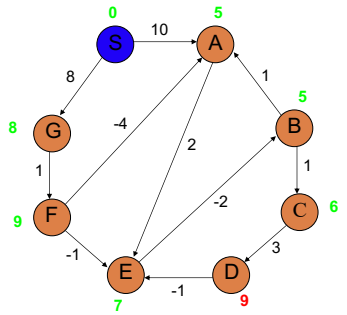
## Bellman-Ford algorithm




Iteration: 6

90

## Bellman-Ford algorithm




Iteration: 7

D (and all other nodes) have the correct distance and path

91

## Correctness of Bellman-Ford

Loop invariant: After iteration i, all vertices with shortest paths from s of length i edges or less have correct distances

```
BELLMAN-FORD(G, s)
1   for all v ∈ V
2           dist[v] ← ∞
3           prev[v] ← null
4   dist[s] ← 0
5   for i ← 1 to |V| − 1
6           for all edges (u, v) ∈ E
7                   if dist[v] > dist[u] + w(u, v)
8                           dist[v] ← dist[u] + w(u, v)
9                           prev[v] ← u
10  for all edges (u, v) ∈ E
11          if dist[v] > dist[u] + w(u, v)
12                  return false
```

92

## Runtime of Bellman-Ford

```
BELLMAN-FORD(G, s)
1   for all v ∈ V
2           dist[v] ← ∞
3           prev[v] ← null
4   dist[s] ← 0
5   for i ← 1 to |V| − 1
6           for all edges (u, v) ∈ E
7                   if dist[v] > dist[u] + w(u, v)
8                           dist[v] ← dist[u] + w(u, v)
9                           prev[v] ← u
10  for all edges (u, v) ∈ E
11          if dist[v] > dist[u] + w(u, v)
12                  return false
```

O(|V| |E|)

93

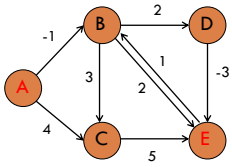
## Runtime of Bellman-Ford

```
BELLMAN-FORD(G, s)
1   for all v ∈ V
2           dist[v] ← ∞
3           prev[v] ← null
4   dist[s] ← 0
5   for i ← 1 to |V| − 1
6           for all edges (u, v) ∈ E
7                   if dist[v] > dist[u] + w(u, v)
8                           dist[v] ← dist[u] + w(u, v)
9                           prev[v] ← u
10  for all edges (u, v) ∈ E
11          if dist[v] > dist[u] + w(u, v)
12                  return false
```

Can you modify the algorithm to run faster (in some circumstances)?

94

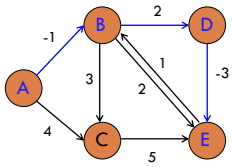
## Shortest Paths




What is the shortest path from A to E?

95

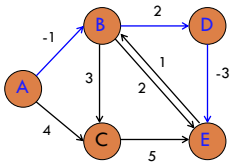
## Shortest Paths




-2

96

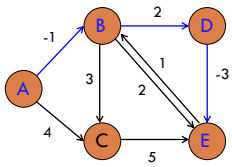
## Shortest Paths




What algorithm would we use to calculate this?

97

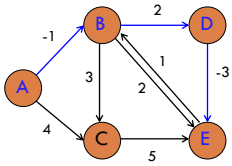
## Shortest Paths




- Bellman-Ford (since the graph has negative edges)
- O(VE)

98

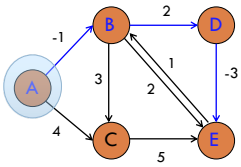
## Shortest Paths




- Bellman-Ford (since the graph has negative edges)
- O(VE)
- Called a single-source shortest path algorithm. Why?

99

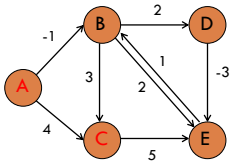
## Shortest Paths




- Bellman-Ford (since the graph has negative edges)
- O(VE)
- Calculate all paths from a **single vertex.**

100

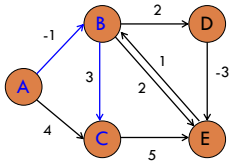
## Shortest Paths




What is the shortest path from A to C?
If we already calculated A to E using Bellman-Ford do we need to do any work?

101

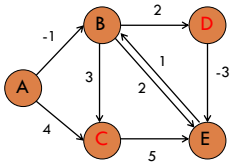
## Shortest Paths




No new calculations!
Bellman-Ford calculates all shortest paths starting at A.

102

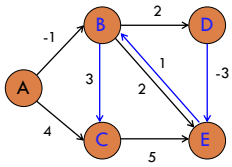
## Shortest Paths




What is the shortest path from D to C?
If we already calculated A to E using Bellman-Ford do we need to do any work?

103

## Shortest Paths

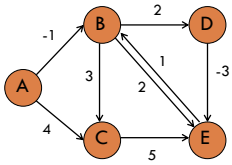



Different source.
Have to run Bellman-Ford again!

104

## All pairs shortest paths

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




105

## All pairs shortest paths

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

Easy solution?

106

## All pairs shortest paths

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

Run Bellman-Ford from each vertex!

Running time (in terms of E and V)?

107

## 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

108

## Floyd-Warshall: key idea

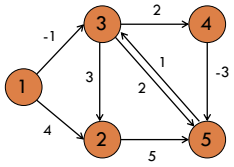
Label all vertices with a number from 1 to V

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

109

## Floyd-Warshall: key idea

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




What is $d_{15}^{2}$?

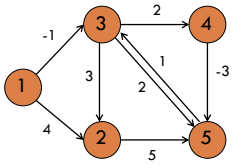
What is $d_{15}^{3}$?

What is $d_{41}^{4}$?

110

## Floyd-Warshall: key idea

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

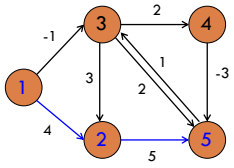



What is $d_{15}^{2}$?

111

## Floyd-Warshall: key idea

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

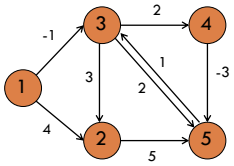



$d_{15}^{2} = 9$. Can only use 2.

112

## Floyd-Warshall: key idea

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

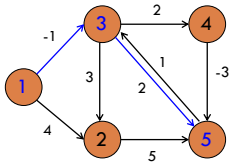



What is $d_{15}^{3}$?

113

## Floyd-Warshall: key idea

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

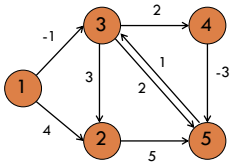



$d_{15}^{3} = 1$. Can't use vertex 4.

114

## Floyd-Warshall: key idea

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

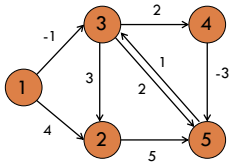



What is $d_{41}^{4}$?

115

## Floyd-Warshall: key idea

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

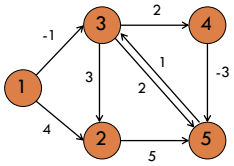



$d_{41}^{4} = \infty$. No possible path.

116

## Floyd-Warshall: key idea

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

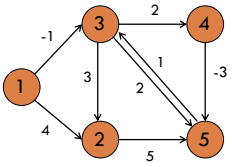



What is $d_{33}{}^5$?

117

## Floyd-Warshall: key idea

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




$d_{33}{}^5 = 0$. $d_{ii}{}^k = 0$ for all $i$.

118

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

119

## 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

120

## 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!

121

## 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

122

## 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

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

123

## 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

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

124

## 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

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

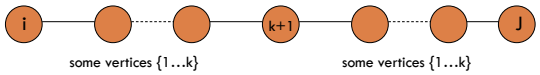
125

## 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

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




some vertices {1…k} some vertices {1…k}

What is the cost of this path?

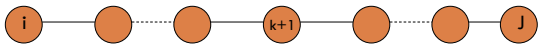
126

## 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

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




some vertices {1…k} some vertices {1…k}

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

127

## 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?

128

## 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

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

Pick whichever is shorter

129

## Floyd-Warshall

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

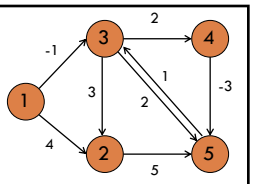
Floyd-Warshall(G = (V,E,W)):
$d^0$ = W // initialize with edge weights
for $k$ = 1 to V
  for $i$ = 1 to V
    for $j$ = 1 to V
      $\text{dijk} = \min(d_{ij}{}^{k-1}, d_{ik}{}^{k-1} + d_{kj}{}^{k-1})$

return $d^V$

130

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

return $d^V$




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

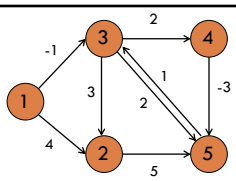
| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 4 | −1 | ∞ | ∞ |
| 2 | ∞ | 0 | ∞ | ∞ | 5 |
| 3 | ∞ | 3 | 0 | 2 | 2 |
| 4 | ∞ | ∞ | ∞ | 0 | −3 |
| 5 | ∞ | ∞ | 1 | ∞ | 0 |

no change

131

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

return $d^V$




k = 1

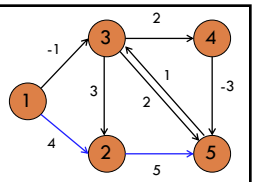
| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 4 | −1 | ∞ | ∞ |
| 2 | ∞ | 0 | ∞ | ∞ | 5 |
| 3 | ∞ | 3 | 0 | 2 | 2 |
| 4 | ∞ | ∞ | ∞ | 0 | −3 |
| 5 | ∞ | ∞ | 1 | ∞ | 0 |

k = 2

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

132

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




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 |

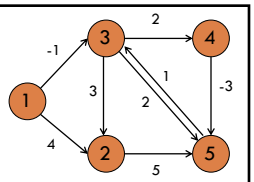
k = 2

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

minimum

133

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




k = 2

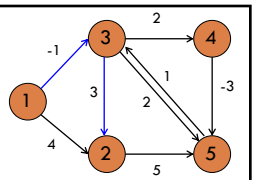
| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 4 | −1 | ∞ | 9 |
| 2 | ∞ | 0 | ∞ | ∞ | 5 |
| 3 | ∞ | 3 | 0 | 2 | 2 |
| 4 | ∞ | ∞ | ∞ | 0 | −3 |
| 5 | ∞ | ∞ | 1 | ∞ | 0 |

k = 3

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | ? | | | |
| 2 | | | | | |
| 3 | | | | | |
| 4 | | | | | |
| 5 | | | | | |

134

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

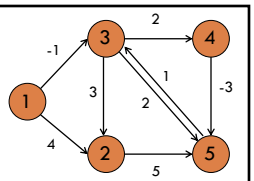
| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 2 | | | |
| 2 | | | | | |
| 3 | | | | | |
| 4 | | | | | |
| 5 | | | | | |

minimum

Found a shorter path!

135

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




k = 2

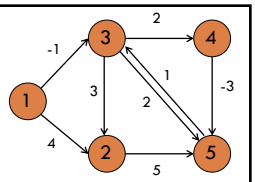
| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 4 | −1 | ∞ | 9 |
| 2 | ∞ | 0 | ∞ | ∞ | 5 |
| 3 | ∞ | 3 | 0 | 2 | 2 |
| 4 | ∞ | ∞ | ∞ | 0 | −3 |
| 5 | ∞ | ∞ | 1 | ∞ | 0 |

k = 3

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 2 | | | |
| 2 | | | | | |
| 3 | | | | | |
| 4 | | | | | |
| 5 | | | | | |

136

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




k = 2

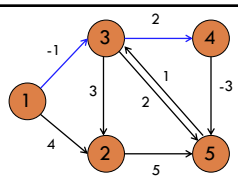
| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 4 | −1 | ∞ | 9 |
| 2 | ∞ | 0 | ∞ | ∞ | 5 |
| 3 | ∞ | 3 | 0 | 2 | 2 |
| 4 | ∞ | ∞ | ∞ | 0 | −3 |
| 5 | ∞ | ∞ | 1 | ∞ | 0 |

k = 3

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

137

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

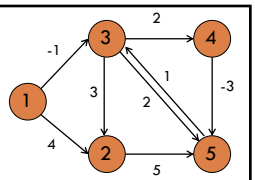
k = 3

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 2 | −1 | 1 | |
| 2 | | | | | |
| 3 | | | | | |
| 4 | | | | | |
| 5 | | | | | |

minimum

138

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




k = 2

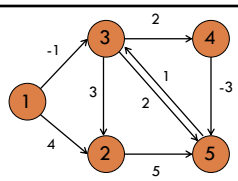
| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 4 | −1 | ∞ | 9 |
| 2 | ∞ | 0 | ∞ | ∞ | 5 |
| 3 | ∞ | 3 | 0 | 2 | 2 |
| 4 | ∞ | ∞ | ∞ | 0 | −3 |
| 5 | ∞ | ∞ | 1 | ∞ | 0 |

k = 3

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 2 | −1 | 1 | |
| 2 | | | | | |
| 3 | | | | | |
| 4 | | | | | |
| 5 | | | | | |

139

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




k = 2

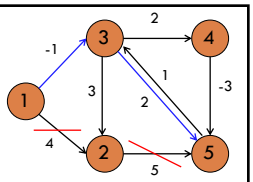
| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 4 | −1 | ∞ | 9 |
| 2 | ∞ | 0 | ∞ | ∞ | 5 |
| 3 | ∞ | 3 | 0 | 2 | 2 |
| 4 | ∞ | ∞ | ∞ | 0 | −3 |
| 5 | ∞ | ∞ | 1 | ∞ | 0 |

k = 3

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

140

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

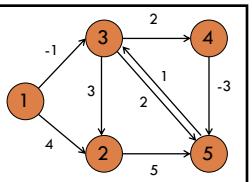
k = 3

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 2 | –1 | 1 | 1 |
| 2 | | | | | |
| 3 | | | | | |
| 4 | | | | | |
| 5 | | | | | |

Found a shorter path!

141

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

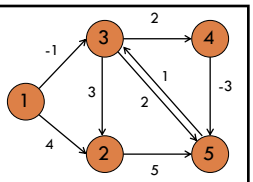
| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 4 | –1 | ∞ | 9 |
| 2 | ∞ | 0 | ∞ | ∞ | 5 |
| 3 | ∞ | 3 | 0 | 2 | 2 |
| 4 | ∞ | ∞ | ∞ | 0 | –3 |
| 5 | ∞ | ∞ | 1 | ∞ | 0 |

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 | ∞ | ∞ | 1 | ∞ | 0 |

142

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

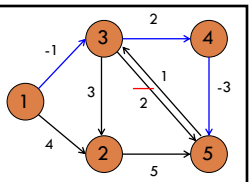
| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 2 | –1 | 1 | 1 |
| 2 | ∞ | 0 | ∞ | ∞ | 5 |
| 3 | ∞ | 3 | 0 | 2 | 2 |
| 4 | ∞ | ∞ | ∞ | 0 | –3 |
| 5 | ∞ | ∞ | 1 | ∞ | 0 |

k = 4

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 2 | –1 | 1 | ? |
| 2 | | | | | |
| 3 | | | | | |
| 4 | | | | | |
| 5 | | | | | |

143

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

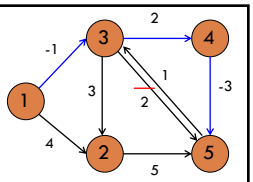
k = 4

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 2 | –1 | 1 | –2 |
| 2 | | | | | |
| 3 | | | | | |
| 4 | | | | | |
| 5 | | | | | |

Found a shorter path!

144

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




k = 3

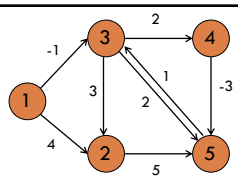
| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 2 | −1 | 1 | 1 |
| 2 | ∞ | 0 | ∞ | ∞ | 5 |
| 3 | ∞ | 3 | 0 | 2 | 2 |
| 4 | ∞ | ∞ | ∞ | 0 | −3 |
| 5 | ∞ | ∞ | 1 | ∞ | 0 |

k = 4

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 2 | −1 | 1 | −2 |
| 2 | | | | | |
| 3 | | | | | |
| 4 | | | | | |
| 5 | | | | | |

145

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




k = 3

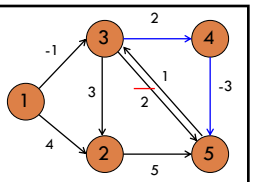
| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 2 | −1 | 1 | 1 |
| 2 | ∞ | 0 | ∞ | ∞ | 5 |
| 3 | ∞ | 3 | 0 | 2 | 2 |
| 4 | ∞ | ∞ | ∞ | 0 | −3 |
| 5 | ∞ | ∞ | 1 | ∞ | 0 |

k = 4

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

146

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

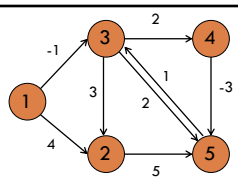
| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 2 | −1 | 1 | −2 |
| 2 | ∞ | 0 | ∞ | ∞ | 5 |
| 3 | ∞ | 3 | 0 | 2 | −1 |
| 4 | | | | | |
| 5 | | | | | |

minimum

Found a shorter path!

147

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




k = 3

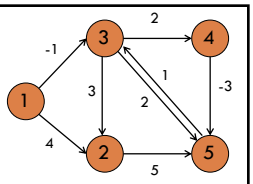
| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 2 | −1 | 1 | 1 |
| 2 | ∞ | 0 | ∞ | ∞ | 5 |
| 3 | ∞ | 3 | 0 | 2 | 2 |
| 4 | ∞ | ∞ | ∞ | 0 | −3 |
| 5 | ∞ | ∞ | 1 | ∞ | 0 |

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 | ∞ | ∞ | 1 | ∞ | 0 |

148

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
      $\text{dijk} = \min(d_{ij}^{k-1}, d_{ik}^{k-1} + d_{kj}^{k-1})$
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 | 7 | 9 | 5 |
| 3 | ∞ | 3 | 0 | 2 | −1 |
| 4 | ∞ | 1 | −2 | 0 | −3 |
| 5 | ∞ | ∞ | 1 | ∞ | 0 |

Done!

149

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

150

## Floyd-Warshall analysis

Is it correct?

Any assumptions?

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

151

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

152

## 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 |

153

## Floyd-Warshall analysis

Run-time?

```
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
```

154

## Floyd-Warshall analysis

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

```
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
      dijk = min(d_ij^(k-1), d_ik^(k-1) + d_kj^(k-1))
return d^V
```

155

## 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
      dijk = min(d_ij^(k-1), d_ik^(k-1) + d_kj^(k-1))
return d^V
```

156

## 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
      dijk = min(d_ij^{k-1}, d_ik^{k-1} + d_kj^{k-1})

return d^V
```

157

## 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
      dijk = min(d_ij^{k-1}, d_ik^{k-1} + d_kj^{k-1})

return d^V
```

158

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

159

## 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?

160

## 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
      dijk = min(d_ij^(k-1), d_ik^(k-1) + d_kj^(k-1))
return d^V
```

161

## All pairs shortest paths

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

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

162

## All pairs shortest paths

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

Easy solution?

163

## 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)?

164

## 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 do Dijkstras
- Dijkstras: $O(V \log V + E)$

165

## 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?

166

## 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!

167

## All pairs shortest paths

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

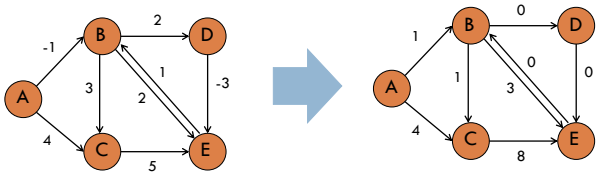
Run Dijsktras from each vertex!

Challenge: Dijkstras assumes positive weights

168

## Johnson's: key idea

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




169

## 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

170

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

Let *s*, $v_1$, $v_2$, …, $v_k$, *t* be a path from *s* to *t*

The weight in the reweighted graph is:

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

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

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

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

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

$$\ldots$$

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

171

## 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?

172

## Lemma: proof

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

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.

173

## 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

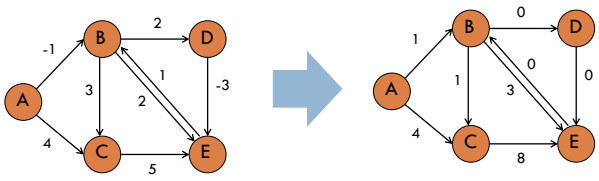
Big question: how do we pick h?

174

## Selecting h

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

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




175

## 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

176

Create G'

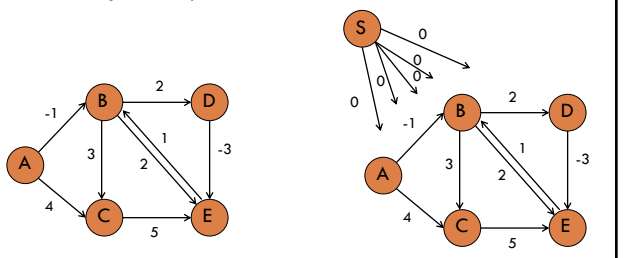
run Bellman-Ford(G',s)

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




177

Create G'

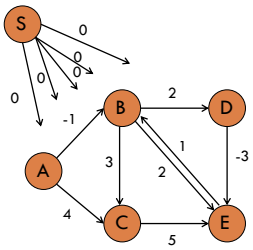
run Bellman-Ford(G',s)

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:

178

Create G'

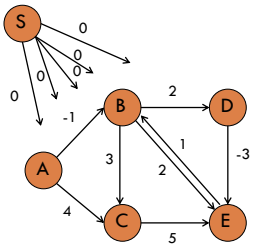
run Bellman-Ford(G',s)

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:

179

Create G'

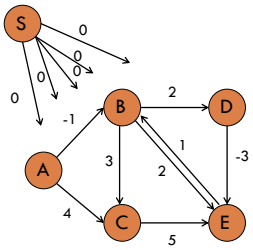
run Bellman-Ford(G',s)

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:

180

Create G'

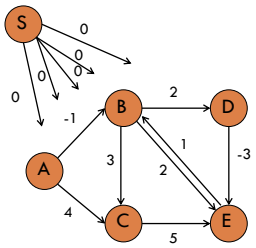
run Bellman-Ford(G',s)

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:

181

Create G'

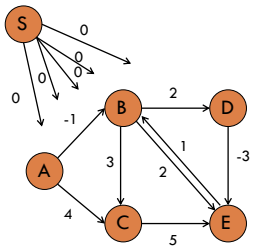
run Bellman-Ford(G',s)

if no negative-weight cycle

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

run Dijkstra's from every vertex

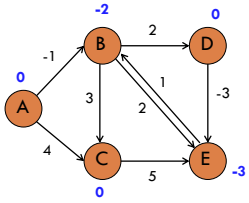
reweight shortest paths based on G




S➔A: 0
S➔B: -2
S➔C: 0
S➔D: 0
S➔E: -3

182

S➔A: 0
S➔B: -2
S➔C: 0
S➔D: 0
S➔E: -3




183

Create G'

run Bellman-Ford(G',s)

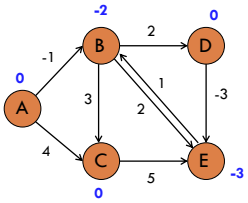
if no negative-weight cycle

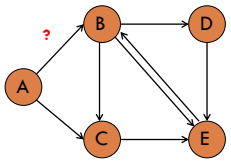
reweight edges in G with h(v)=shortest path from s to v

run Dijkstra's from every vertex

reweight shortest paths based on G

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



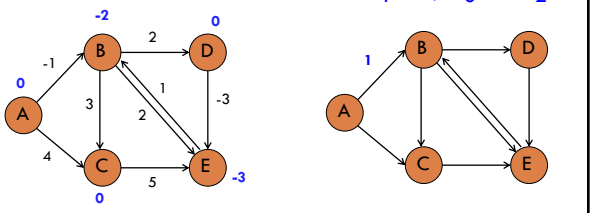




184

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

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

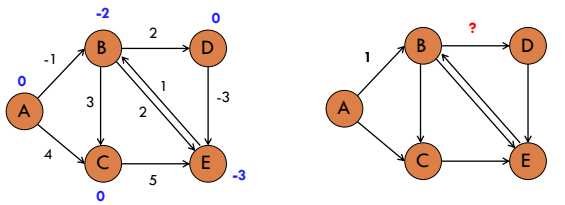
-1 + 0 - -2




185

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

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

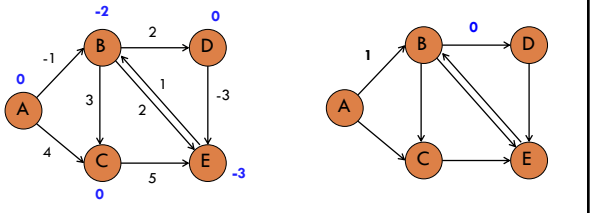



186

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

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

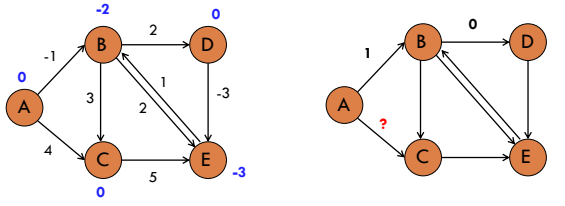
2 + -2 - 0




187

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

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

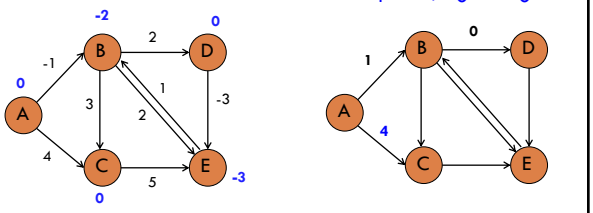



188

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

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

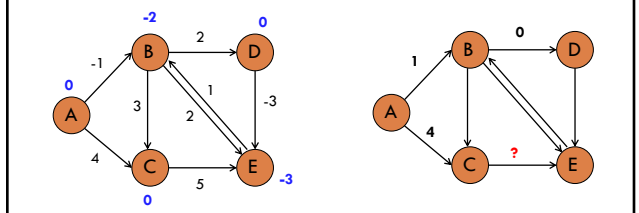
4 + 0 - 0




189

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

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

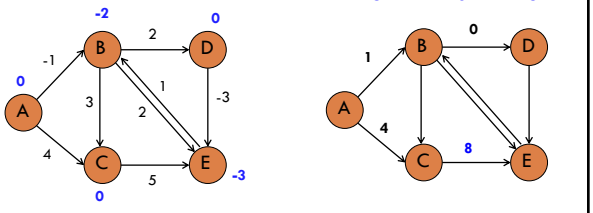



190

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

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

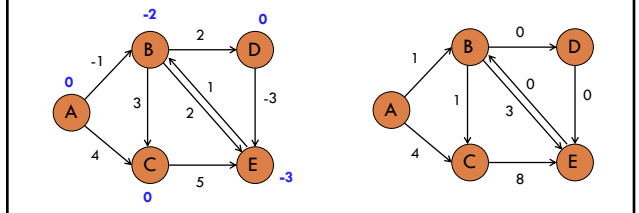
5 + 0 - -3




191

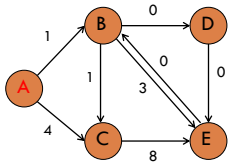
Create G'
run Bellman-Ford(G',s)
if no negative-weight cycle
reweight edges in G with h(v)=shortest path from s to v
run Dijkstra's from every vertex
reweight shortest paths based on G

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



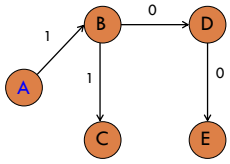

192

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



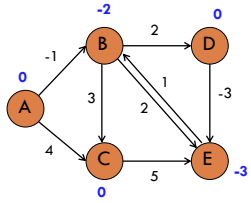

193

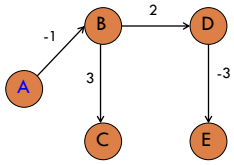
Create G'
run Bellman-Ford(G',s)
if no negative-weight cycle
reweight edges in G with h(v)=shortest path from s to v
run Dijkstra's from every vertex
reweight shortest paths based on G




194

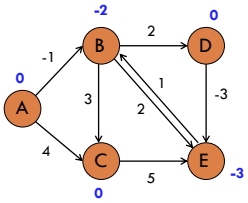
Create G'
run Bellman-Ford(G',s)
if no negative-weight cycle
reweight edges in G with h(v)=shortest path from s to v
run Dijkstra's from every vertex
reweight shortest paths based on G

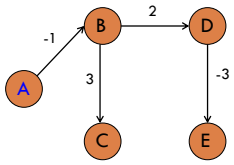



195

A➔B: -1
A➔C: 2
A➔D: 1
A➔E: -2




196

## 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)?

197

## 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?

198

## 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)

199

## 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

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

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

What is this?

200

## 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

201

## 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?

202

## Johnson's algorithm

Create G' — θ(V)
run Bellman-Ford(G',s) — $O(V^2)$
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?

203

## All pairs shortest paths

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

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

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

204