SINGLE SOURCE SHORTEST PATHS

David Kauchak
CS 140 – Fall 2022

1

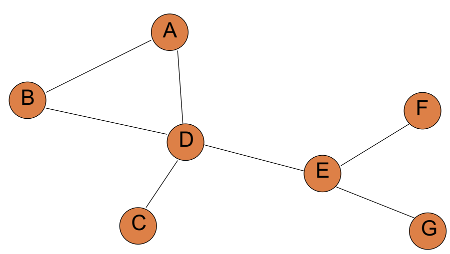
Admin

Assignment 8

2

Graphs

A graph is a set of vertices V and a set of edges (u,v) $\in$ E where u,v $\in$ V



3

Connectedness

Given an undirected graph, for every node u $\in$ V, can we reach all other nodes in the graph? Algorithm + running time

Run BFS or DFS-Visit (one pass) and mark nodes as we visit them. If we visit all nodes, return true, otherwise false.

Running time: O(|V| + |E|)

4

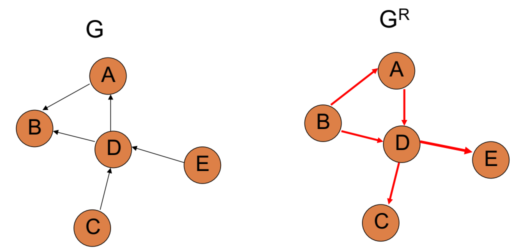
Strongly connected

Given a directed graph, can we reach any node v from any other node u?

Can we do the same thing?

Transpose of a graph

Given a graph G, we can calculate the transpose of a graph G^R by reversing the direction of all the edges



Running time to calculate G^R? $\theta(|V| + |E|)$

Strongly connected

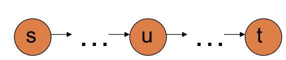
Strongly-Connected(G)
- Run DFS-Visit or BFS from some node u
- If not all nodes are visited: return false
- Create graph G^R
- Run DFS-Visit or BFS on G^R from node u
- If not all nodes are visited: return false
- return true

Is it correct?

What do we know after the first pass?
- Starting at u, we can reach every node

What do we know after the second pass?
- All nodes can reach u. Why?
- We can get from u to every node in G^R, therefore, if we reverse the edges (i.e. G), then we have a path from every node to u

Which means that any node can reach any other node. Given any two nodes s and t we can create a path through u



Runtime?

Strongly-Connected(G)

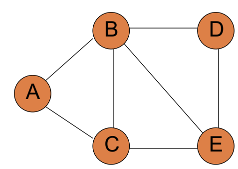
- Run DFS-Visit or BFS from some node u — O(|V| + |E|)
- If not all nodes are visited: return false — O(|V|)
- Create graph G^R — θ(|V| + |E|)
- Run DFS-Visit or BFS on G^R from node u — O(|V| + |E|)
- If not all nodes are visited: return false — O(|V|)
- return true

O(|V| + |E|)

9

Shortest paths

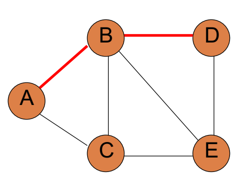
What is the shortest path from a to d?



10

Shortest paths

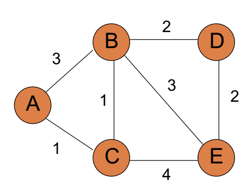
BFS



11

Shortest paths

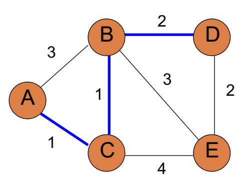
What is the shortest path from a to d?



12

Shortest paths

What is the shortest path from a to d?



13

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)
for all v ∈ V
    dist[v] ← ∞
    prev[v] ← null
dist[s] ← 0
Q ← MAKEHEAP(V)
while !EMPTY(Q)
    u ← EXTRACTMIN(Q)
    for all edges (u, v) ∈ E
        if dist[v] > dist[u] + w(u, v)
            dist[v] ← dist[u] + w(u, v)
            DECREASEKEY(Q, v, dist[v])
            prev[v] ← u
```

14

Dijkstra's algorithm

```
DIJKSTRA(G, s)
for all v ∈ V
    dist[v] ← ∞
    prev[v] ← null
dist[s] ← 0
Q ← MAKEHEAP(V)
while !EMPTY(Q)
    u ← EXTRACTMIN(Q)
    for all edges (u, v) ∈ E
        if dist[v] > dist[u] + w(u, v)
            dist[v] ← dist[u] + w(u, v)
            DECREASEKEY(Q, v, dist[v])
            prev[v] ← u
```

```
BFS(G, s)
for each v ∈ V
    dist[v] = ∞
dist[s] = 0
ENQUEUE(Q, s)
while !EMPTY(Q)
    u ← DEQUEUE(Q)
    VISIT(u)
    for each edge (u, v) ∈ E
        if dist[v] = ∞
            ENQUEUE(Q, v)
            dist[v] ← dist[u] + 1
```

15

Dijkstra's algorithm

prev keeps track of
the shortest path

```
DIJKSTRA(G, s)
for all v ∈ V
    dist[v] ← ∞
    prev[v] ← null
dist[s] ← 0
Q ← MAKEHEAP(V)
while !EMPTY(Q)
    u ← EXTRACTMIN(Q)
    for all edges (u, v) ∈ E
        if dist[v] > dist[u] + w(u, v)
            dist[v] ← dist[u] + w(u, v)
            DECREASEKEY(Q, v, dist[v])
            prev[v] ← u
```

```
BFS(G, s)
for each v ∈ V
    dist[v] = ∞
dist[s] = 0
ENQUEUE(Q, s)
while !EMPTY(Q)
    u ← DEQUEUE(Q)
    VISIT(u)
    for each edge (u, v) ∈ E
        if dist[v] = ∞
            ENQUEUE(Q, v)
            dist[v] ← dist[u] + 1
```

16

Dijkstra's algorithm

```
Dijkstra(G, s)
for all v ∈ V
        dist[v] ← ∞
        prev[v] ← null
dist[s] ← 0
Q ← MakeHeap(V)
while !Empty(Q)
        u ← ExtractMin(Q)
        for all edges (u, v) ∈ E
                if dist[v] > dist[u] + w(u, v)
                        dist[v] ← dist[u] + w(u, v)
                        DecreaseKey(Q, v, dist[v])
                        prev[v] ← u
```

```
BFS(G, s)
for each v ∈ V
        dist[v] = ∞
dist[s] = 0
Enqueue(Q, s)
while !Empty(Q)
        u ← Dequeue(Q)
        Visit(u)
        for each edge (u, v) ∈ E
                if dist[v] = ∞
                        Enqueue(Q, v)
                        dist[v] ← dist[u] + 1
```

17

Dijkstra's algorithm

```
Dijkstra(G, s)
for all v ∈ V
        dist[v] ← ∞
        prev[v] ← null
dist[s] ← 0
Q ← MakeHeap(V)
while !Empty(Q)
        u ← ExtractMin(Q)
        for all edges (u, v) ∈ E
                if dist[v] > dist[u] + w(u, v)
                        dist[v] ← dist[u] + w(u, v)
                        DecreaseKey(Q, v, dist[v])
                        prev[v] ← u
```

```
BFS(G, s)
for each v ∈ V
        dist[v] = ∞
dist[s] = 0
Enqueue(Q, s)
while !Empty(Q)
        u ← Dequeue(Q)
        Visit(u)
        for each edge (u, v) ∈ E
                if dist[v] = ∞
                        Enqueue(Q, v)
                        dist[v] ← dist[u] + 1
```


18

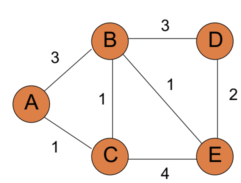
Dijkstra's algorithm

```
Dijkstra(G, s)
for all v ∈ V
        dist[v] ← ∞
        prev[v] ← null
dist[s] ← 0
Q ← MakeHeap(V)
while !Empty(Q)
        u ← ExtractMin(Q)
        for all edges (u, v) ∈ E
                if dist[v] > dist[u] + w(u, v)
                        dist[v] ← dist[u] + w(u, v)
                        DecreaseKey(Q, v, dist[v])
                        prev[v] ← u
```

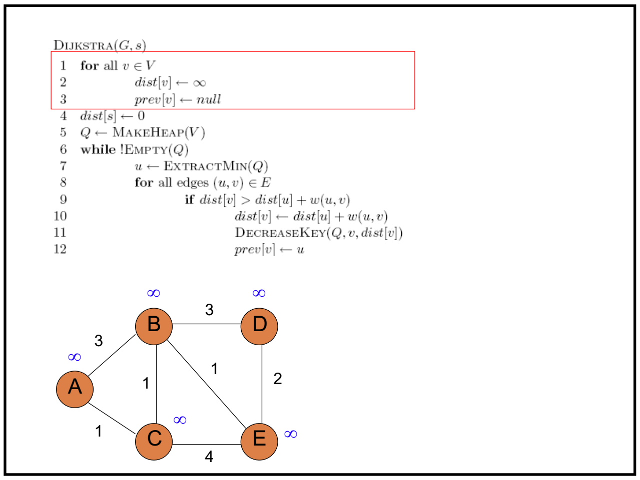
```
BFS(G, s)
for each v ∈ V
        dist[v] = ∞
dist[s] = 0
Enqueue(Q, s)
while !Empty(Q)
        u ← Dequeue(Q)
        Visit(u)
        for each edge (u, v) ∈ E
                if dist[v] = ∞
                        Enqueue(Q, v)
                        dist[v] ← dist[u] + 1
```

19

```
Dijkstra(G, s)
for all v ∈ V
        dist[v] ← ∞
        prev[v] ← null
dist[s] ← 0
Q ← MakeHeap(V)
while !Empty(Q)
        u ← ExtractMin(Q)
        for all edges (u, v) ∈ E
                if dist[v] > dist[u] + w(u, v)
                        dist[v] ← dist[u] + w(u, v)
                        DecreaseKey(Q, v, dist[v])
                        prev[v] ← u
```

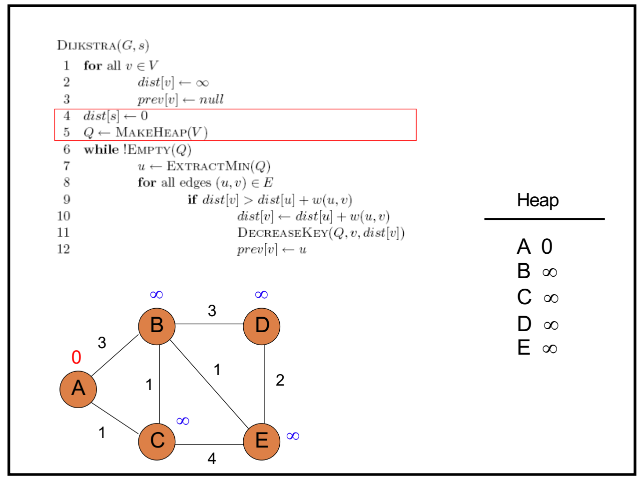


20



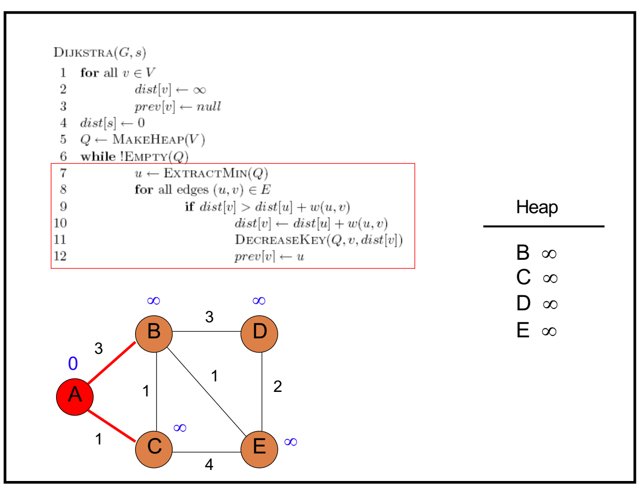
DIJKSTRA(G, s)
for all v ∈ V
dist[v] ← ∞
prev[v] ← null
dist[s] ← 0
Q ← MAKEHEAP(V)
while !EMPTY(Q)
u ← EXTRACTMIN(Q)
for all edges (u, v) ∈ E
if dist[v] > dist[u] + w(u, v)
dist[v] ← dist[u] + w(u, v)
DECREASEKEY(Q, v, dist[v])
prev[v] ← u
A
B
C
D
E

21



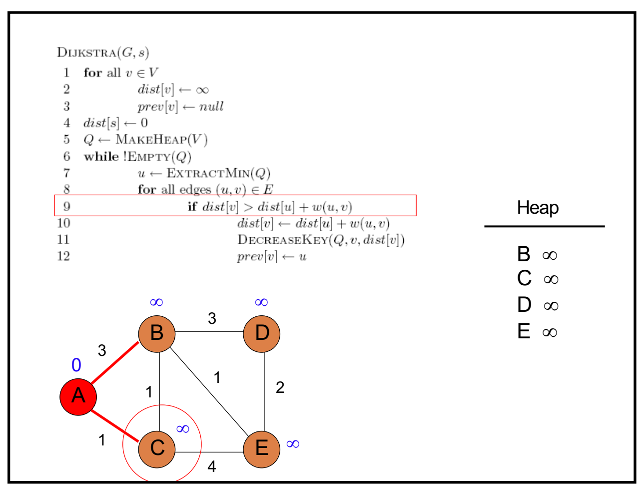
DIJKSTRA(G, s)
for all v ∈ V
dist[v] ← ∞
prev[v] ← null
dist[s] ← 0
Q ← MAKEHEAP(V)
while !EMPTY(Q)
u ← EXTRACTMIN(Q)
for all edges (u, v) ∈ E
if dist[v] > dist[u] + w(u, v)
dist[v] ← dist[u] + w(u, v)
DECREASEKEY(Q, v, dist[v])
prev[v] ← u
Heap
A 0
B ∞
C ∞
D ∞
E ∞

22



DIJKSTRA(G, s)
for all v ∈ V
dist[v] ← ∞
prev[v] ← null
dist[s] ← 0
Q ← MAKEHEAP(V)
while !EMPTY(Q)
u ← EXTRACTMIN(Q)
for all edges (u, v) ∈ E
if dist[v] > dist[u] + w(u, v)
dist[v] ← dist[u] + w(u, v)
DECREASEKEY(Q, v, dist[v])
prev[v] ← u
Heap
B ∞
C ∞
D ∞
E ∞

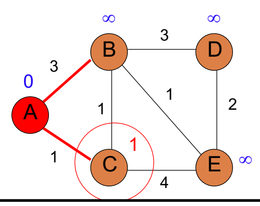
23



DIJKSTRA(G, s)
for all v ∈ V
dist[v] ← ∞
prev[v] ← null
dist[s] ← 0
Q ← MAKEHEAP(V)
while !EMPTY(Q)
u ← EXTRACTMIN(Q)
for all edges (u, v) ∈ E
if dist[v] > dist[u] + w(u, v)
dist[v] ← dist[u] + w(u, v)
DECREASEKEY(Q, v, dist[v])
prev[v] ← u
Heap
B ∞
C ∞
D ∞
E ∞

24

```
DIJKSTRA(G, s)
for all v ∈ V
    dist[v] ← ∞
    prev[v] ← null
dist[s] ← 0
Q ← MAKEHEAP(V)
while !EMPTY(Q)
    u ← EXTRACTMIN(Q)
    for all edges (u, v) ∈ E
        if dist[v] > dist[u] + w(u, v)
            dist[v] ← dist[u] + w(u, v)
            DECREASEKEY(Q, v, dist[v])
            prev[v] ← u
```

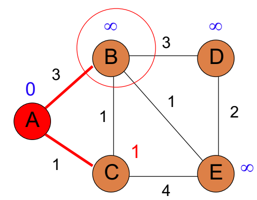


Heap

C 1
B ∞
D ∞
E ∞

25

```
DIJKSTRA(G, s)
for all v ∈ V
    dist[v] ← ∞
    prev[v] ← null
dist[s] ← 0
Q ← MAKEHEAP(V)
while !EMPTY(Q)
    u ← EXTRACTMIN(Q)
    for all edges (u, v) ∈ E
        if dist[v] > dist[u] + w(u, v)
            dist[v] ← dist[u] + w(u, v)
            DECREASEKEY(Q, v, dist[v])
            prev[v] ← u
```

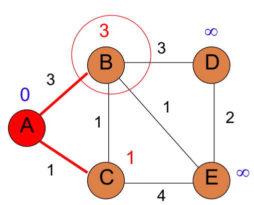


Heap

C 1
B ∞
D ∞
E ∞

26

```
DIJKSTRA(G, s)
for all v ∈ V
    dist[v] ← ∞
    prev[v] ← null
dist[s] ← 0
Q ← MAKEHEAP(V)
while !EMPTY(Q)
    u ← EXTRACTMIN(Q)
    for all edges (u, v) ∈ E
        if dist[v] > dist[u] + w(u, v)
            dist[v] ← dist[u] + w(u, v)
            DECREASEKEY(Q, v, dist[v])
            prev[v] ← u
```

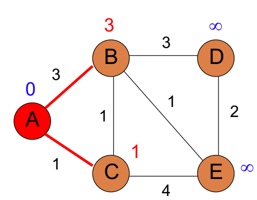


Heap

C 1
B 3
D ∞
E ∞

27

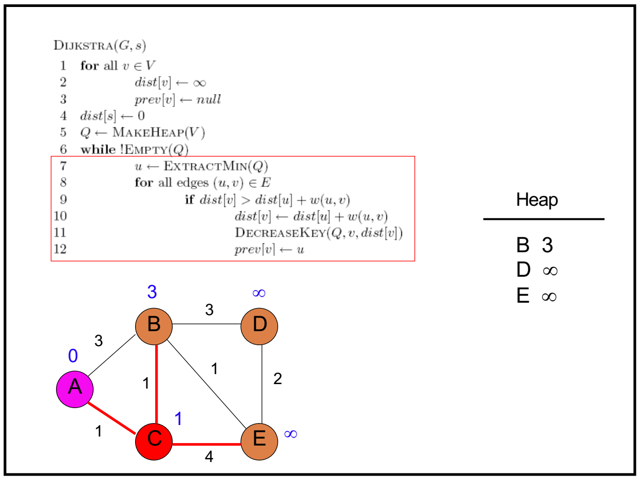
```
DIJKSTRA(G, s)
for all v ∈ V
    dist[v] ← ∞
    prev[v] ← null
dist[s] ← 0
Q ← MAKEHEAP(V)
while !EMPTY(Q)
    u ← EXTRACTMIN(Q)
    for all edges (u, v) ∈ E
        if dist[v] > dist[u] + w(u, v)
            dist[v] ← dist[u] + w(u, v)
            DECREASEKEY(Q, v, dist[v])
            prev[v] ← u
```



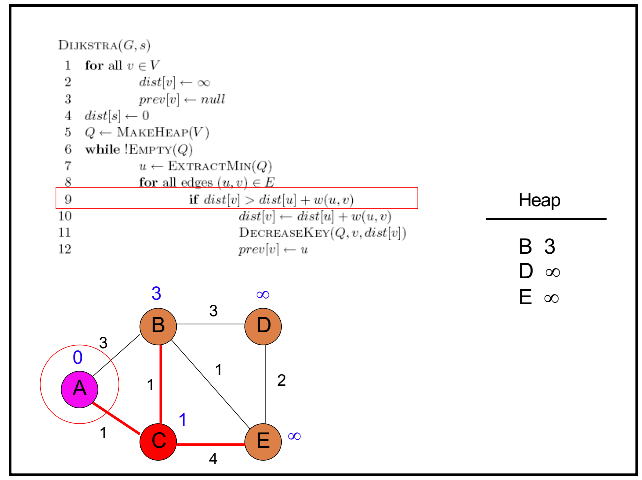
Heap

C 1
B 3
D ∞
E ∞

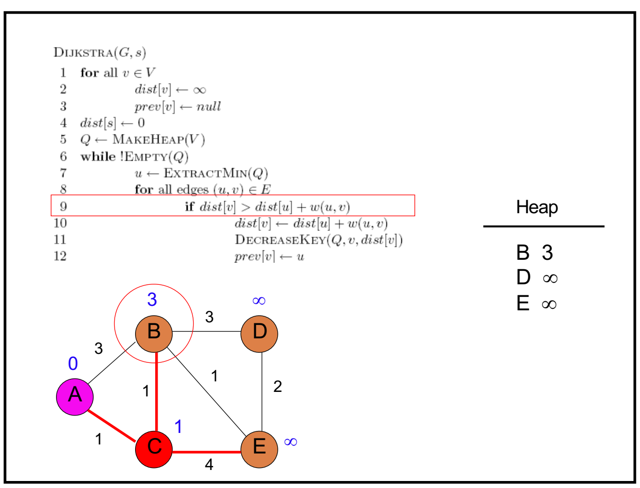
28


DIJKSTRA(G, s)
for all v ∈ V
dist[v] ← ∞
prev[v] ← null
dist[s] ← 0
Q ← MAKEHEAP(V)
while !EMPTY(Q)
u ← EXTRACTMIN(Q)
for all edges (u, v) ∈ E
if dist[v] > dist[u] + w(u, v)
dist[v] ← dist[u] + w(u, v)
DECREASEKEY(Q, v, dist[v])
prev[v] ← u
Heap
B 3
D ∞
E ∞
A B C D E
0 3 1 ∞ ∞
3 3 1 1 2 1 4

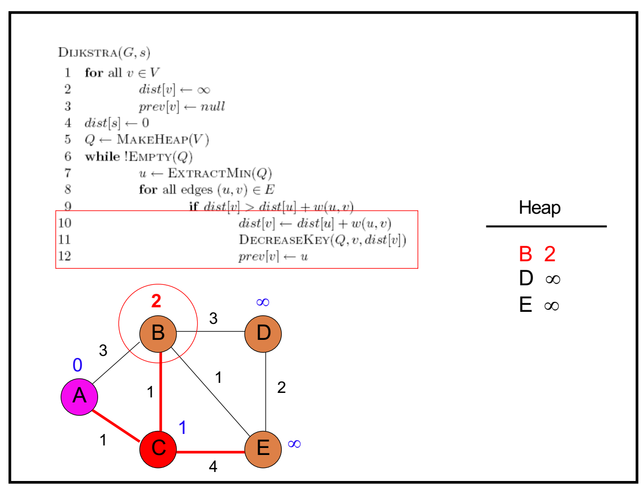
29


DIJKSTRA(G, s)
for all v ∈ V
dist[v] ← ∞
prev[v] ← null
dist[s] ← 0
Q ← MAKEHEAP(V)
while !EMPTY(Q)
u ← EXTRACTMIN(Q)
for all edges (u, v) ∈ E
if dist[v] > dist[u] + w(u, v)
dist[v] ← dist[u] + w(u, v)
DECREASEKEY(Q, v, dist[v])
prev[v] ← u
Heap
B 3
D ∞
E ∞
A B C D E
0 3 1 ∞ ∞
3 3 1 1 2 1 4

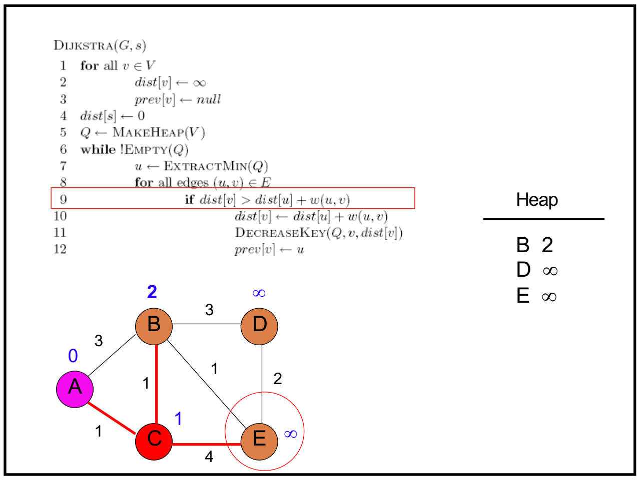
30


DIJKSTRA(G, s)
for all v ∈ V
dist[v] ← ∞
prev[v] ← null
dist[s] ← 0
Q ← MAKEHEAP(V)
while !EMPTY(Q)
u ← EXTRACTMIN(Q)
for all edges (u, v) ∈ E
if dist[v] > dist[u] + w(u, v)
dist[v] ← dist[u] + w(u, v)
DECREASEKEY(Q, v, dist[v])
prev[v] ← u
Heap
B 3
D ∞
E ∞
A B C D E
0 3 1 ∞ ∞
3 3 1 1 2 1 4

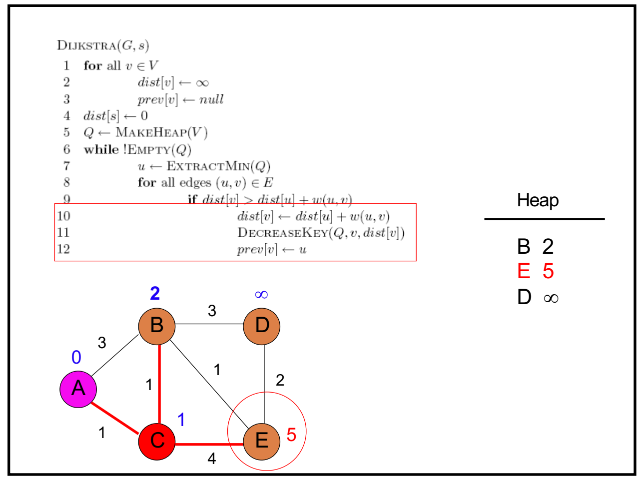
31


DIJKSTRA(G, s)
for all v ∈ V
dist[v] ← ∞
prev[v] ← null
dist[s] ← 0
Q ← MAKEHEAP(V)
while !EMPTY(Q)
u ← EXTRACTMIN(Q)
for all edges (u, v) ∈ E
if dist[v] > dist[u] + w(u, v)
dist[v] ← dist[u] + w(u, v)
DECREASEKEY(Q, v, dist[v])
prev[v] ← u
Heap
B 2
D ∞
E ∞
A B C D E
0 2 1 ∞ ∞
3 3 1 1 2 1 4

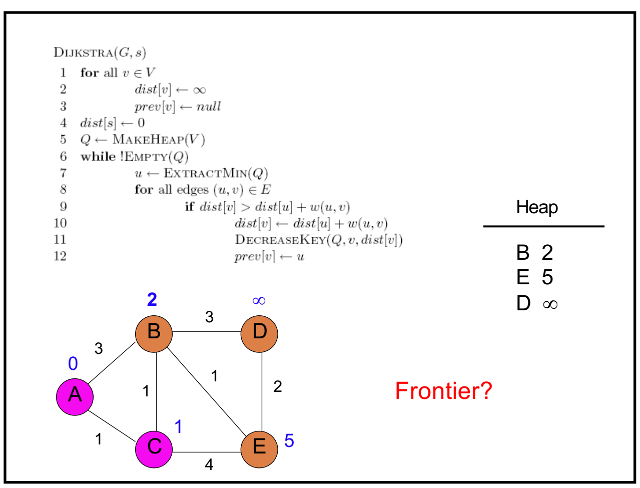
32


DIJKSTRA(G, s)
for all v ∈ V
dist[v] ← ∞
prev[v] ← null
dist[s] ← 0
Q ← MAKEHEAP(V)
while !EMPTY(Q)
u ← EXTRACTMIN(Q)
for all edges (u, v) ∈ E
if dist[v] > dist[u] + w(u, v)
dist[v] ← dist[u] + w(u, v)
DECREASEKEY(Q, v, dist[v])
prev[v] ← u
Heap
B 2
D ∞
E ∞
A B C D E
0 2 1 ∞ ∞
3 1 1 3 1 2 4

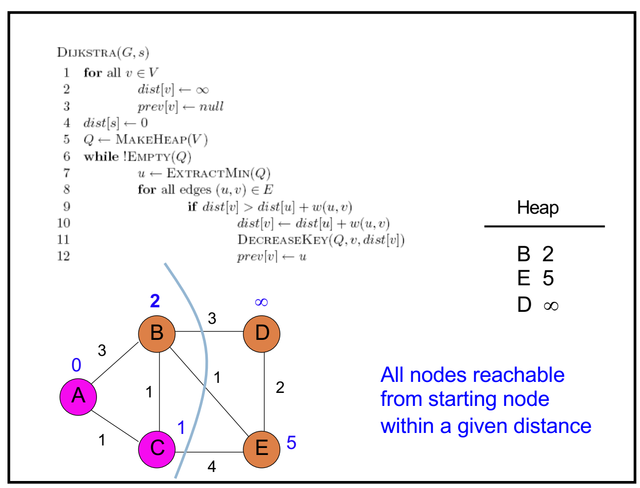
33


DIJKSTRA(G, s)
for all v ∈ V
dist[v] ← ∞
prev[v] ← null
dist[s] ← 0
Q ← MAKEHEAP(V)
while !EMPTY(Q)
u ← EXTRACTMIN(Q)
for all edges (u, v) ∈ E
if dist[v] > dist[u] + w(u, v)
dist[v] ← dist[u] + w(u, v)
DECREASEKEY(Q, v, dist[v])
prev[v] ← u
Heap
B 2
E 5
D ∞
A B C D E
0 2 1 ∞ 5
3 1 1 3 1 2 4

34


DIJKSTRA(G, s)
for all v ∈ V
dist[v] ← ∞
prev[v] ← null
dist[s] ← 0
Q ← MAKEHEAP(V)
while !EMPTY(Q)
u ← EXTRACTMIN(Q)
for all edges (u, v) ∈ E
if dist[v] > dist[u] + w(u, v)
dist[v] ← dist[u] + w(u, v)
DECREASEKEY(Q, v, dist[v])
prev[v] ← u
Heap
B 2
E 5
D ∞
Frontier?
A B C D E
0 2 1 ∞ 5
3 1 1 3 1 2 4

35

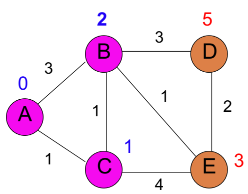

DIJKSTRA(G, s)
for all v ∈ V
dist[v] ← ∞
prev[v] ← null
dist[s] ← 0
Q ← MAKEHEAP(V)
while !EMPTY(Q)
u ← EXTRACTMIN(Q)
for all edges (u, v) ∈ E
if dist[v] > dist[u] + w(u, v)
dist[v] ← dist[u] + w(u, v)
DECREASEKEY(Q, v, dist[v])
prev[v] ← u
Heap
B 2
E 5
D ∞
All nodes reachable
from starting node
within a given distance
A B C D E
0 2 1 ∞ 5
3 1 1 3 1 2 4

36

```
DIJKSTRA(G, s)
for all v ∈ V
        dist[v] ← ∞
        prev[v] ← null
dist[s] ← 0
Q ← MAKEHEAP(V)
while !EMPTY(Q)
        u ← EXTRACTMIN(Q)
        for all edges (u, v) ∈ E
                if dist[v] > dist[u] + w(u, v)
                        dist[v] ← dist[u] + w(u, v)
                        DECREASEKEY(Q, v, dist[v])
                        prev[v] ← u
```

Heap

E 3
D 5

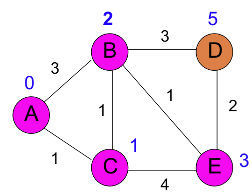


37

```
DIJKSTRA(G, s)
for all v ∈ V
        dist[v] ← ∞
        prev[v] ← null
dist[s] ← 0
Q ← MAKEHEAP(V)
while !EMPTY(Q)
        u ← EXTRACTMIN(Q)
        for all edges (u, v) ∈ E
                if dist[v] > dist[u] + w(u, v)
                        dist[v] ← dist[u] + w(u, v)
                        DECREASEKEY(Q, v, dist[v])
                        prev[v] ← u
```

Heap

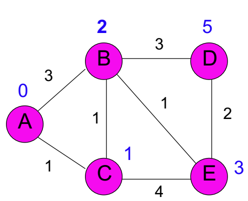
D 5



38

```
DIJKSTRA(G, s)
for all v ∈ V
        dist[v] ← ∞
        prev[v] ← null
dist[s] ← 0
Q ← MAKEHEAP(V)
while !EMPTY(Q)
        u ← EXTRACTMIN(Q)
        for all edges (u, v) ∈ E
                if dist[v] > dist[u] + w(u, v)
                        dist[v] ← dist[u] + w(u, v)
                        DECREASEKEY(Q, v, dist[v])
                        prev[v] ← u
```

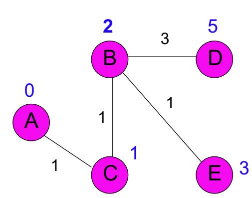
Heap



39

```
DIJKSTRA(G, s)
for all v ∈ V
        dist[v] ← ∞
        prev[v] ← null
dist[s] ← 0
Q ← MAKEHEAP(V)
while !EMPTY(Q)
        u ← EXTRACTMIN(Q)
        for all edges (u, v) ∈ E
                if dist[v] > dist[u] + w(u, v)
                        dist[v] ← dist[u] + w(u, v)
                        DECREASEKEY(Q, v, dist[v])
                        prev[v] ← u
```

Heap



40

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)
for all v ∈ V
        dist[v] ← ∞
        prev[v] ← null
dist[s] ← 0
Q ← MakeHeap(V)
while !Empty(Q)
        u ← ExtractMin(Q)
        for all edges (u, v) ∈ E
                if dist[v] > dist[u] + w(u, v)
                        dist[v] ← dist[u] + w(u, v)
                        DecreaseKey(Q, v, dist[v])
                        prev[v] ← u
```

proof?

41

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

42

Running time?

```
Dijkstra(G, s)
for all v ∈ V
        dist[v] ← ∞
        prev[v] ← null
dist[s] ← 0
Q ← MakeHeap(V)
while !Empty(Q)
        u ← ExtractMin(Q)
        for all edges (u, v) ∈ E
                if dist[v] > dist[u] + w(u, v)
                        dist[v] ← dist[u] + w(u, v)
                        DecreaseKey(Q, v, dist[v])
                        prev[v] ← u
```

43

Running time?

```
Dijkstra(G, s)
for all v ∈ V
        dist[v] ← ∞
        prev[v] ← null
dist[s] ← 0
Q ← MakeHeap(V)                          1 call to MakeHeap
while !Empty(Q)
        u ← ExtractMin(Q)
        for all edges (u, v) ∈ E
                if dist[v] > dist[u] + w(u, v)
                        dist[v] ← dist[u] + w(u, v)
                        DecreaseKey(Q, v, dist[v])
                        prev[v] ← u
```

44

Running time?

```
Dijkstra(G, s)
for all v ∈ V
        dist[v] ← ∞
        prev[v] ← null
dist[s] ← 0
Q ← MakeHeap(V)
while !Empty(Q)                          |V| iterations
        u ← ExtractMin(Q)
        for all edges (u, v) ∈ E
                if dist[v] > dist[u] + w(u, v)
                        dist[v] ← dist[u] + w(u, v)
                        DecreaseKey(Q, v, dist[v])
                        prev[v] ← u
```

45

Running time?

```
Dijkstra(G, s)
for all v ∈ V
        dist[v] ← ∞
        prev[v] ← null
dist[s] ← 0
Q ← MakeHeap(V)
while !Empty(Q)
        u ← ExtractMin(Q)                |V| calls
        for all edges (u, v) ∈ E
                if dist[v] > dist[u] + w(u, v)
                        dist[v] ← dist[u] + w(u, v)
                        DecreaseKey(Q, v, dist[v])
                        prev[v] ← u
```

46

Running time?

```
Dijkstra(G, s)
for all v ∈ V
        dist[v] ← ∞
        prev[v] ← null
dist[s] ← 0
Q ← MakeHeap(V)
while !Empty(Q)
        u ← ExtractMin(Q)
        for all edges (u, v) ∈ E
                if dist[v] > dist[u] + w(u, v)
                        dist[v] ← dist[u] + w(u, v)
                        DecreaseKey(Q, v, dist[v])   O(|E|) calls
                        prev[v] ← u
```

47

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|) O(|E| log |V|)

48

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|) O(|E| log |V|)

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

49

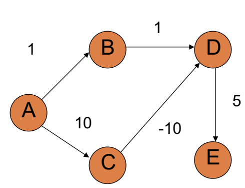
Running time?

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|) O(|E| log |V|)
Fib heap	O(|V|)	O(|V| log |V|)	O(|E|)	O(|V| log |V| + |E|)

50

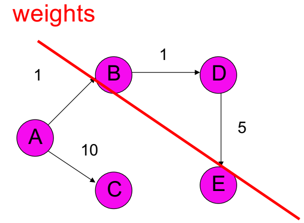
What about Dijkstra's on…?



51

What about Dijkstra's on…?

Dijkstra's algorithm only works for positive edge weights



52

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!

53

Bounding the distance

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

```
DIJKSTRA(G, s)
for all v ∈ V
    dist[v] ← ∞
    prev[v] ← null
dist[s] ← 0
Q ← MAKEHEAP(V)
while !EMPTY(Q)
    u ← EXTRACTMIN(Q)
    for all edges (u, v) ∈ E
        if dist[v] > dist[u] + w(u, v)
            dist[v] ← dist[u] + w(u, v)
            DECREASEKEY(Q, v, dist[v])
            prev[v] ← u
```

Is this a valid invariant?

54

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

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

55

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

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?

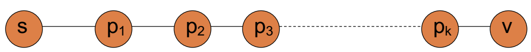
- If u is along the shortest path to v and dist[u] is correct

56

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

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

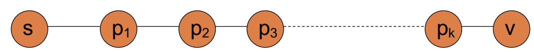


57

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

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?

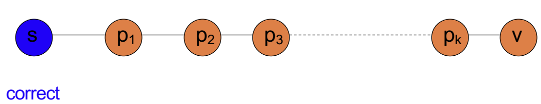


58

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

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?

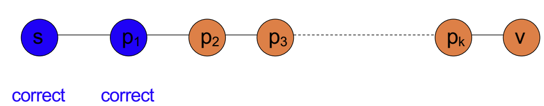


59

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

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?

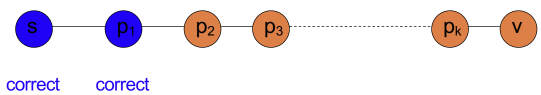


60

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

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?



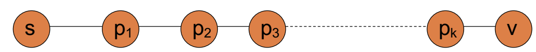
61

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

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

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

- i times



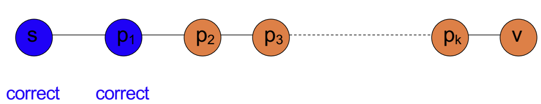
62

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

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

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

- i times



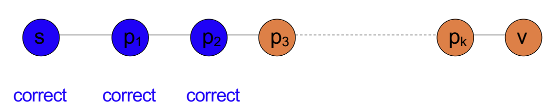
63

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

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

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

- i times



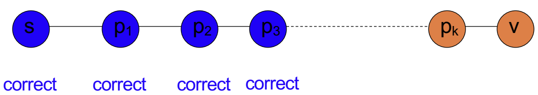
64

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

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

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

- i times



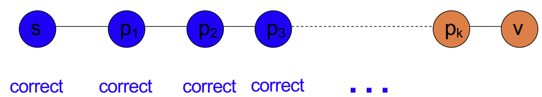
65

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

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

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

- i times



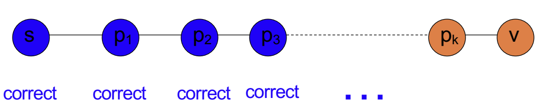
66

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

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

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

- |V| - 1 edges/vertices



67

Bellman-Ford algorithm

```
BELLMAN-FORD(G, s)
for all v ∈ V
        dist[v] ← ∞
        prev[v] ← null
dist[s] ← 0
for i ← 1 to |V| − 1
        for all edges (u, v) ∈ E
                if dist[v] > dist[u] + w(u, v)
                        dist[v] ← dist[u] + w(u, v)
                        prev[v] ← u
for all edges (u, v) ∈ E
        if dist[v] > dist[u] + w(u, v)
                return false
```

68

Bellman-Ford algorithm

```
BELLMAN-FORD(G, s)
for all v ∈ V
        dist[v] ← ∞
        prev[v] ← null
dist[s] ← 0
for i ← 1 to |V| − 1
        for all edges (u, v) ∈ E
                if dist[v] > dist[u] + w(u, v)
                        dist[v] ← dist[u] + w(u, v)
                        prev[v] ← u
for all edges (u, v) ∈ E
        if dist[v] > dist[u] + w(u, v)
                return false
```

Initialize all the distances

do it |V| -1 times

iterate over all edges/vertices and apply update rule

69

Bellman-Ford algorithm

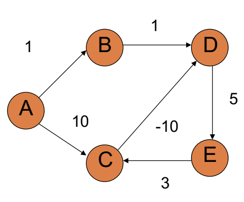
```
BELLMAN-FORD(G, s)
for all v ∈ V
        dist[v] ← ∞
        prev[v] ← null
dist[s] ← 0
for i ← 1 to |V| − 1
        for all edges (u, v) ∈ E
                if dist[v] > dist[u] + w(u, v)
                        dist[v] ← dist[u] + w(u, v)
                        prev[v] ← u
for all edges (u, v) ∈ E
        if dist[v] > dist[u] + w(u, v)
                return false
```

check for negative cycles

70

Negative cycles

What is the shortest path from a to e?



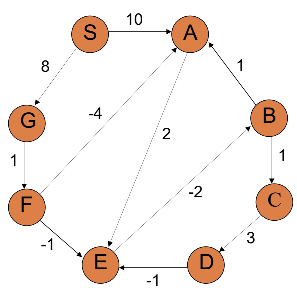
71

Bellman-Ford algorithm

```
BELLMAN-FORD(G, s)
for all v ∈ V
        dist[v] ← ∞
        prev[v] ← null
dist[s] ← 0
for i ← 1 to |V| − 1
        for all edges (u, v) ∈ E
                if dist[v] > dist[u] + w(u, v)
                        dist[v] ← dist[u] + w(u, v)
                        prev[v] ← u
for all edges (u, v) ∈ E
        if dist[v] > dist[u] + w(u, v)
                return false
```

72

Bellman-Ford algorithm



How many edges is the shortest path from s to:

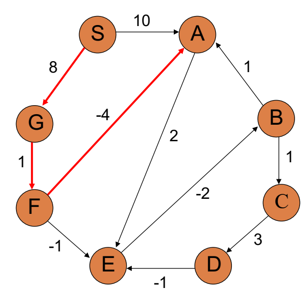
A:

B:

D:

73

Bellman-Ford algorithm

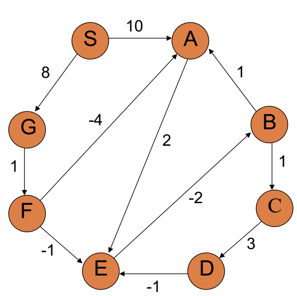


How many edges is the shortest path from s to:

A: 3

74

Bellman-Ford algorithm



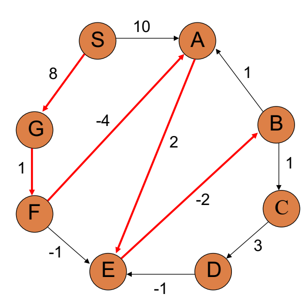
How many edges is the shortest path from s to:

A: 3

B:

75

Bellman-Ford algorithm



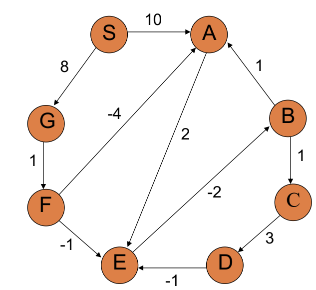
How many edges is the shortest path from s to:

A: 3

B: 5

76

Bellman-Ford algorithm



How many edges is the shortest path from s to:

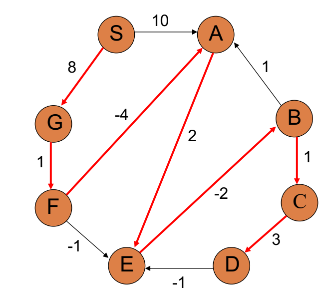
A: 3

B: 5

D:

77

Bellman-Ford algorithm



How many edges is the shortest path from s to:

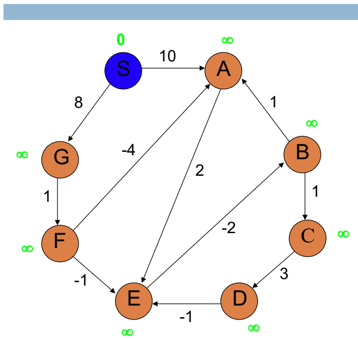
A: 3

B: 5

D: 7

78

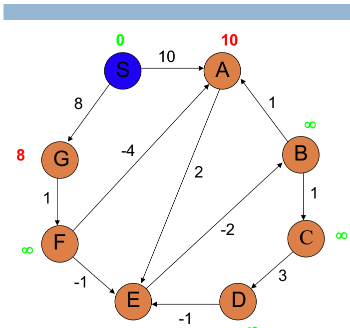
Bellman-Ford algorithm



Iteration: 0

79

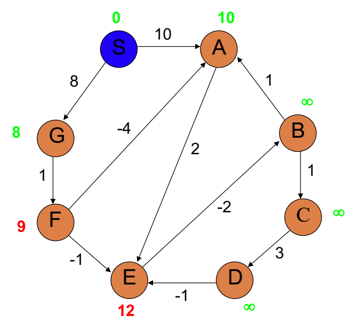
Bellman-Ford algorithm



Iteration: 1

80

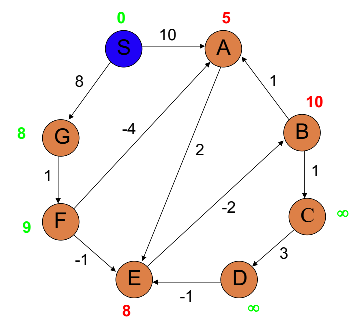
Bellman-Ford algorithm



Iteration: 2

81

Bellman-Ford algorithm

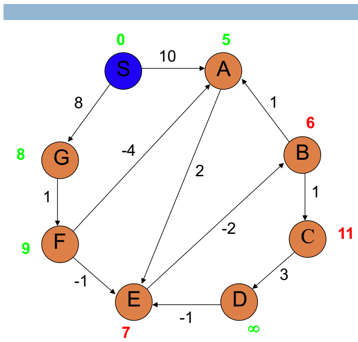


Iteration: 3

A has the correct distance and path

82

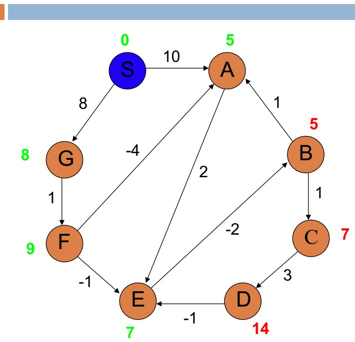
Bellman-Ford algorithm



Iteration: 4

83

Bellman-Ford algorithm

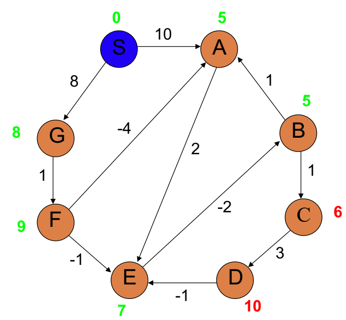


Iteration: 5

B has the correct distance and path

84

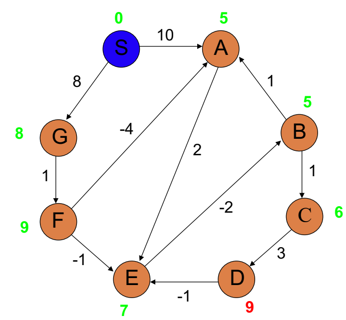
Bellman-Ford algorithm



Iteration: 6

85

Bellman-Ford algorithm



Iteration: 7

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

86

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)
for all v ∈ V
        dist[v] ← ∞
        prev[v] ← null
dist[s] ← 0
for i ← 1 to |V| − 1
        for all edges (u, v) ∈ E
                if dist[v] > dist[u] + w(u, v)
                        dist[v] ← dist[u] + w(u, v)
                        prev[v] ← u
for all edges (u, v) ∈ E
        if dist[v] > dist[u] + w(u, v)
                return false
```

87

Runtime of Bellman-Ford

```
BELLMAN-FORD(G, s)
for all v ∈ V
        dist[v] ← ∞
        prev[v] ← null
dist[s] ← 0
for i ← 1 to |V| − 1
        for all edges (u, v) ∈ E
                if dist[v] > dist[u] + w(u, v)
                        dist[v] ← dist[u] + w(u, v)
                        prev[v] ← u
for all edges (u, v) ∈ E
        if dist[v] > dist[u] + w(u, v)
                return false
```

$O(|V|\,|E|)$

88

Runtime of Bellman-Ford

```
BELLMAN-FORD(G, s)
for all v ∈ V
        dist[v] ← ∞
        prev[v] ← null
dist[s] ← 0
for i ← 1 to |V| − 1
        for all edges (u, v) ∈ E
                if dist[v] > dist[u] + w(u, v)
                        dist[v] ← dist[u] + w(u, v)
                        prev[v] ← u
for all edges (u, v) ∈ E
        if dist[v] > dist[u] + w(u, v)
                return false
```

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

89

Single source shortest paths

All of the shortest path algorithms we've looked at today are call "single source shortest paths" algorithms

Why?

90