SPOOKY SHORTEST PATHS

David Kauchak
CS 140 –Fall 2024

1

2

3

Admin

Assignment 8

Group 8

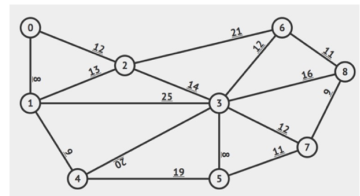
4

Prim's

```
Prim(G, r)
for all v ∈ V
        key[v] ← ∞
        prev[v] ← null
key[r] ← 0
H ← MakeHeap(key)
while !Empty(H)
        u ← Extract-Min(H)
        visited[u] ← true
        for each edge (u, v) ∈ E
                if !visited[v] and w(u, v) < key(v)
                        Decrease-Key(v, w(u, v))
                        prev[v] ← u
```

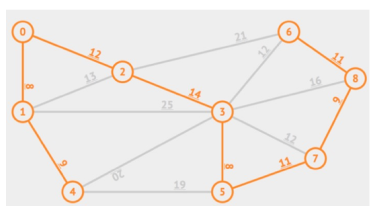
5

MST Practice



6

Solution

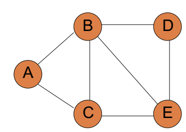


Sum = 8 + 8 + 9 + 9 + 11 + 11 + 12 + 14 = 82

7

Shortest paths

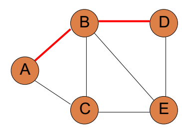
What is the shortest path from a to d?



8

Shortest paths

How can we find this?

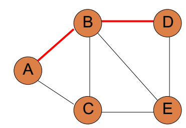


9

Shortest paths

BFS

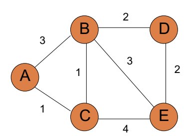
```
BFS(G, s)
for each v ∈ V
        dist[v] = ∞
dist[s] = 0
Enqueue(Q, s)
while !Empty(Q)
        u ← Dequeue(Q)
        Visit(u)
        for each edge (u, v) ∈ E
                if dist[v] = ∞
                        Enqueue(Q, v)
                        dist[v] ← dist[u] + 1
```



10

Shortest paths

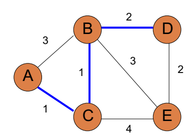
What is the shortest path from a to d?



11

Shortest paths

What is the shortest path from a to d?



12

Shortest path algorithms?

13

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
```

What is dist?

What is prev?

How does it work?

What is the run-time?

How do we get the shortest path?

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

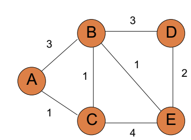
Dijkstra's algorithm

```
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
```

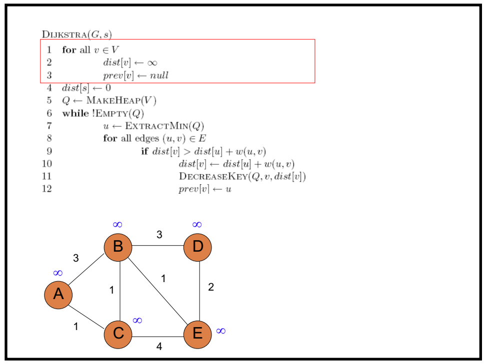
```
BFS(G, s)
for each v ∈ V
    dist[v] = ∞
dist[s] = 0
ENQUEUE(Q, s)
while !EMPTY(Q)
    u ← DEQUEUE(Q)
    VISIT(U)
    for each edge (u, v) ∈ E
        if dist[v] = ∞
            ENQUEUE(Q, v)
            dist[v] ← dist[u] + 1
```

19

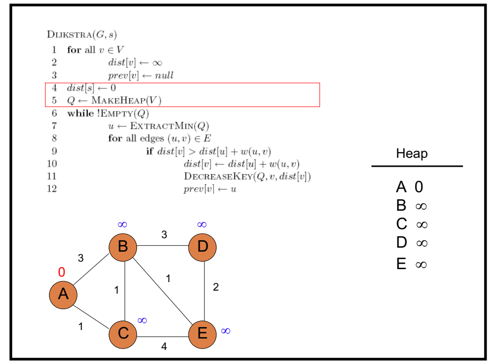
```
DIJKSTRA(G, s)
for all v ∈ V
    dist[v] ← ∞
    prev[v] ← null
dist[s] ← 0
Q ← MAKEHEAP(V)
while !EMPTY(Q)
    u ← EXTRACTMIN(Q)
    for all edges (u, v) ∈ E
        if dist[v] > dist[u] + w(u, v)
            dist[v] ← dist[u] + w(u, v)
            DECREASEKEY(Q, v, dist[v])
            prev[v]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



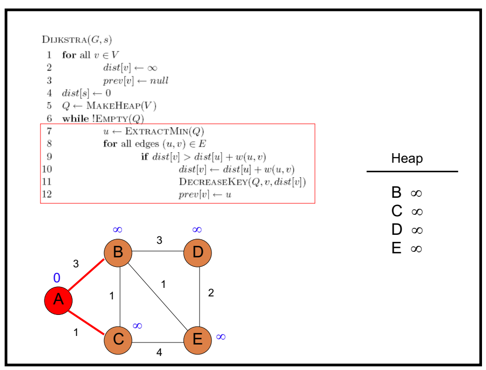
20


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

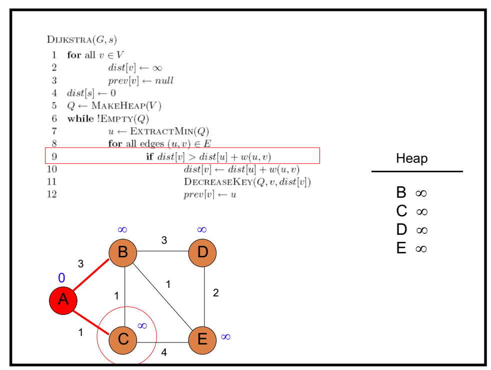
21


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 ∞

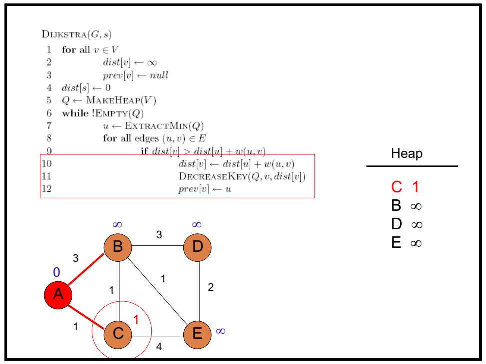
22


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 ∞

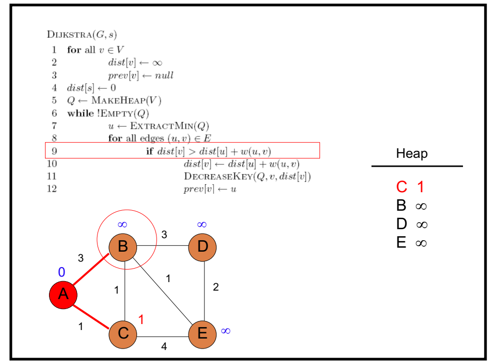
23


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 ∞

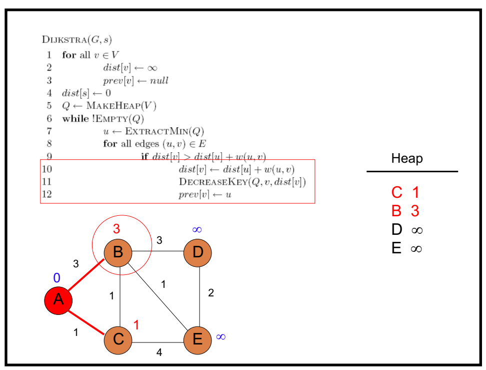
24


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 ∞
A
B
C
D
E
0
1
∞
∞
∞
3
3
1
1
1
2
4

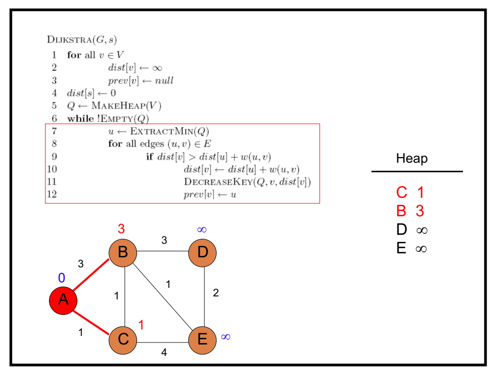
25


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 ∞
A
B
C
D
E
0
1
∞
∞
∞
3
3
1
1
1
2
4

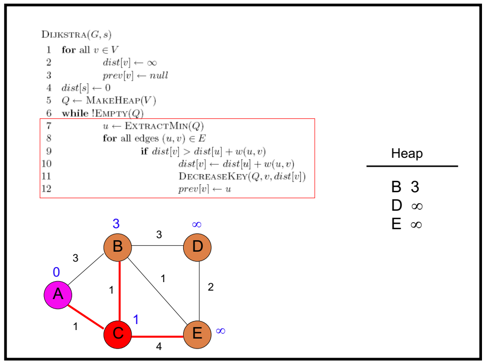
26


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 ∞
A
B
C
D
E
0
1
3
∞
∞
3
3
1
1
1
2
4

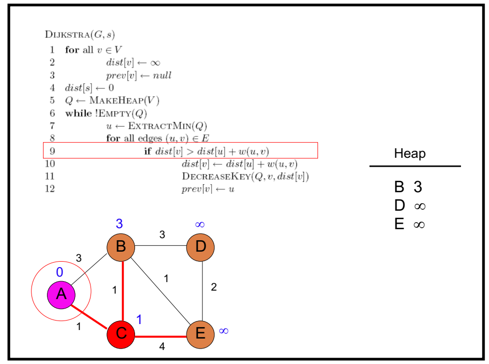
27


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 ∞
A
B
C
D
E
0
1
3
∞
∞
3
3
1
1
1
2
4

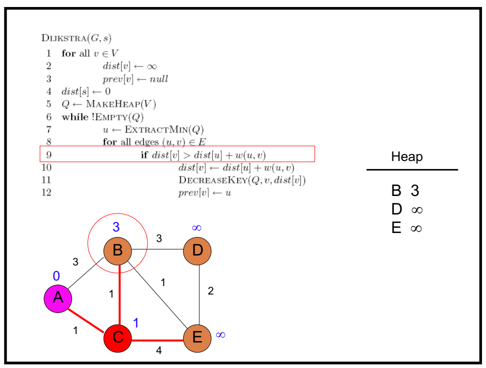
28


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

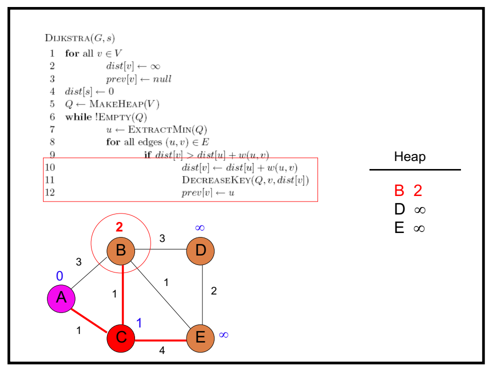
29


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

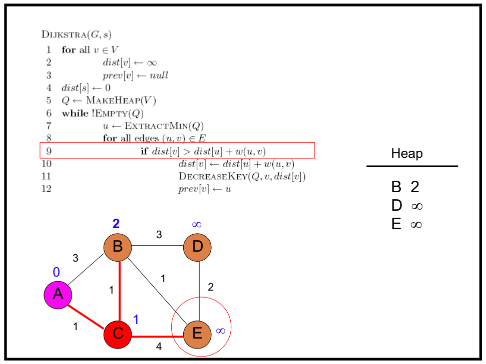
30


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

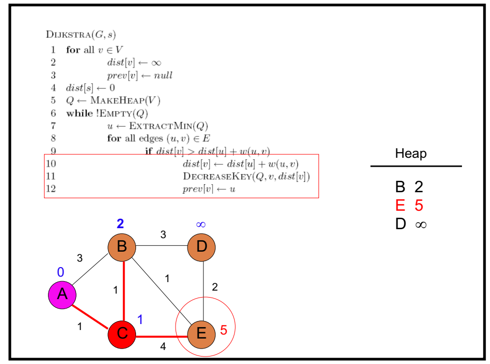
31


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

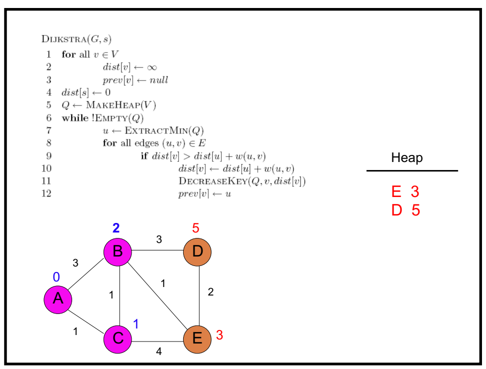
32


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 0
B 2
C 1
D ∞
E ∞

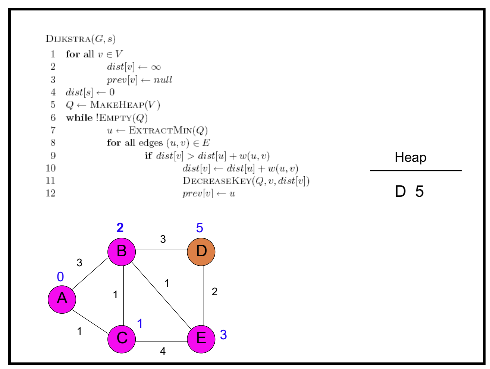
33


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 0
B 2
C 1
D ∞
E 5

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
E 3
D 5
A 0
B 2
C 1
D 5
E 3

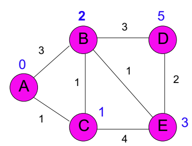
35


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
D 5
A 0
B 2
C 1
D 5
E 3

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
```

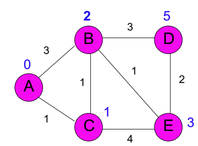
Heap



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



Prev

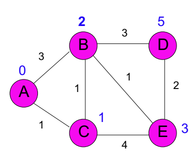
A: null
B: C
C: A
D: B
E: B

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

How do we get the actual paths?



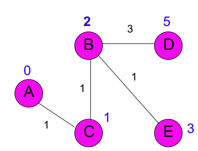
Prev

A: null
B: C
C: A
D: B
E: B

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



Prev

A: null
B: C
C: A
D: B
E: B

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                              Θ(|V|)
    dist[v] ← ∞
    prev[v] ← null
dist[s] ← 0
Q ← MakeHeap(V)                            1 call to MakeHeap
while !Empty(Q)
    u ← ExtractMin(Q)                      |V| calls to Extract-M
    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])     |E| calls to Decrease-K
            prev[v] ← u
```

44

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|)

47

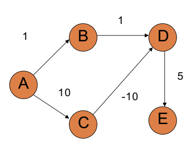
Dijkstra's vs Prim's

```
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
```

```
Prim(G, r)
for all v ∈ V
        key[v] ← ∞
        prev[v] ← null
key[r] ← 0
H ← MakeHeap(key)
while !Empty(H)
        u ← Extract-Min(H)
        visited[u] ← true
        for each edge (u, v) ∈ E
                if !visited[v] and w(u, v) < key(v)
                        Decrease-Key(v, w(u, v))
                        prev[v] ← u
```

48

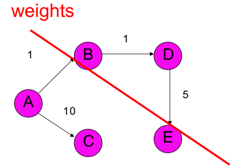
What about Dijkstra's on…?



49

What about Dijkstra's on…?

Dijkstra's algorithm only works for positive edge weights



50

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!

51

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?

52

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: for an edge (u,v)

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

53

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

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?

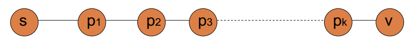
- If u is along the shortest path to v and dist[u] is correct

54

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

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

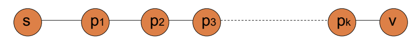


55

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

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?

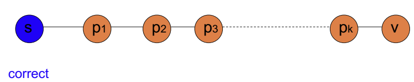


56

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

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

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

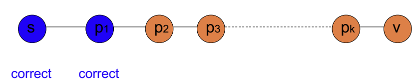


57

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

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?

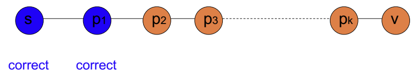


58

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

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

Does the order that we update the vertices matter?



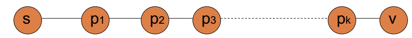
59

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

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

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

- i times



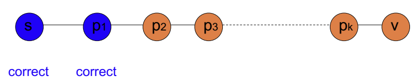
60

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

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

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

- i times



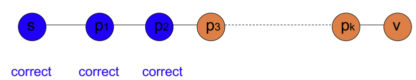
61

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

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



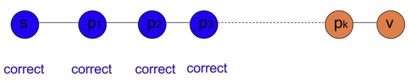
62

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

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



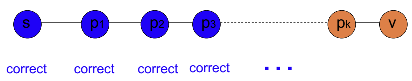
63

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

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



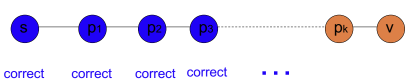
64

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

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

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

- |V| - 1 edges/vertices



65

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
```

66

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

67

Bellman-Ford algorithm

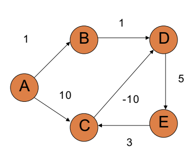
```
Bellman-Ford(G, s)
for all v ∈ V
        dist[v] ← ∞
        prev[v] ← null
dist[s] ← 0
for i ← 1 to |V| − 1
        for all edges (u, v) ∈ E
                if dist[v] > dist[u] + w(u, v)
                        dist[v] ← dist[u] + w(u, v)
                        prev[v] ← u
for all edges (u, v)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

check for negative cycles

68

Negative cycles

What is the shortest path from a to e?



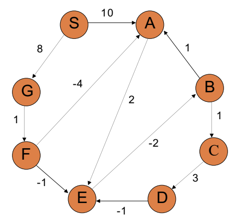
69

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
```

70

Bellman-Ford algorithm



How many edges is the shortest path from s to:

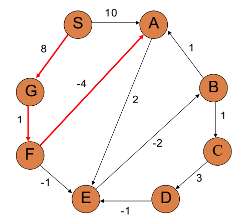
A:

B:

D:

71

Bellman-Ford algorithm

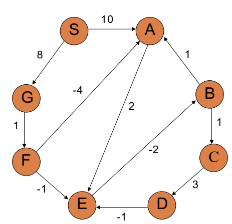


How many edges is the shortest path from s to:

A: 3

72

Bellman-Ford algorithm



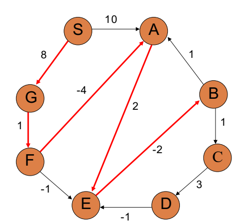
How many edges is the shortest path from s to:

A: 3

B:

73

Bellman-Ford algorithm



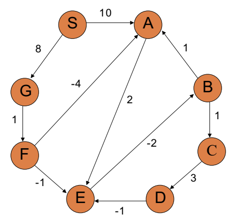
How many edges is the shortest path from s to:

A: 3

B: 5

74

Bellman-Ford algorithm



How many edges is the shortest path from s to:

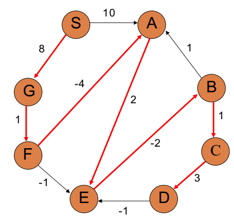
A: 3

B: 5

D:

75

Bellman-Ford algorithm



How many edges is the shortest path from s to:

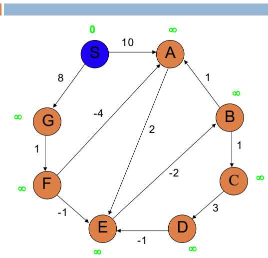
A: 3

B: 5

D: 7

76

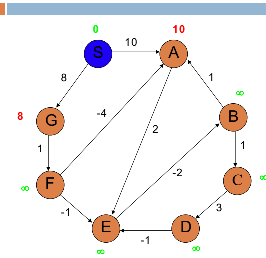
Bellman-Ford algorithm



Iteration: 0

77

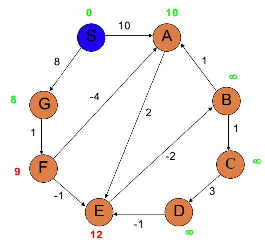
Bellman-Ford algorithm



Iteration: 1

78

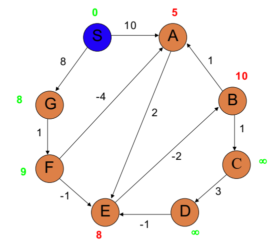
Bellman-Ford algorithm



Iteration: 2

79

Bellman-Ford algorithm

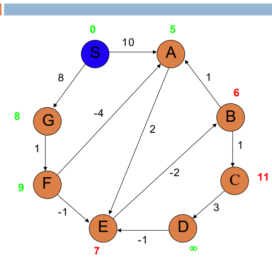


Iteration: 3

A has the correct distance and path

80

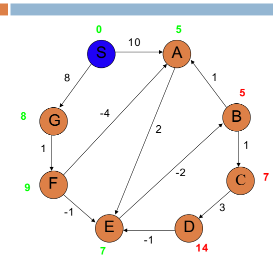
Bellman-Ford algorithm



Iteration: 4

81

Bellman-Ford algorithm

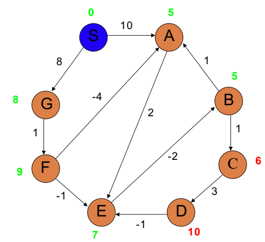


Iteration: 5

B has the correct distance and path

82

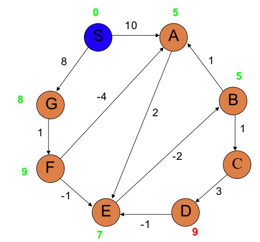
Bellman-Ford algorithm



Iteration: 6

83

Bellman-Ford algorithm



Iteration: 7

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

84

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
```

85

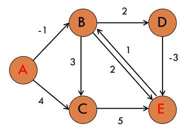
Runtime of Bellman-Ford

```
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|)

86

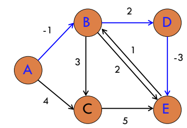
Shortest Paths



What is the shortest path from A to E?

88

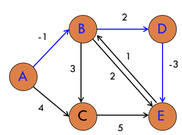
Shortest Paths



-2

89

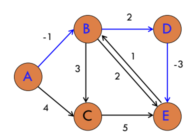
Shortest Paths



What algorithm would we use to calculate this?

90

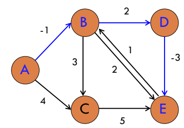
Shortest Paths



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

91

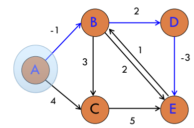
Shortest Paths



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

92

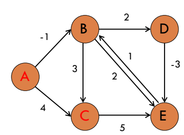
Shortest Paths



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

93

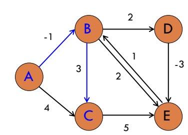
Shortest Paths



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?

94

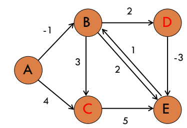
Shortest Paths



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

95

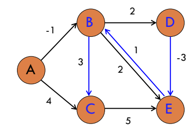
Shortest Paths



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?

96

Shortest Paths

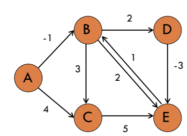


Different source.
Have to run Bellman-Ford again!

97

All pairs shortest paths

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



98

All pairs shortest paths

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

Easy solution?

99

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)?

100

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

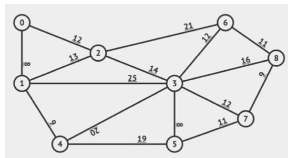
101

DAGs?

102

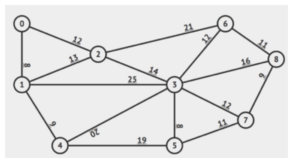
Handout

103



Kruskals

Add smallest edge that connects two sets not already connected

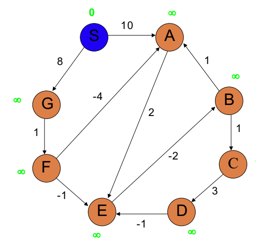


Prims

Starting at 0, grow MST by adding lowest weight edge out of current MST

104

Bellman-Ford algorithm



105