SHORTEST PATHS

David Kauchak
CS 140 – Spring 2024

1

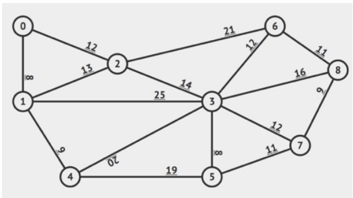
Admin

Assignment 8

No mentor hours Friday

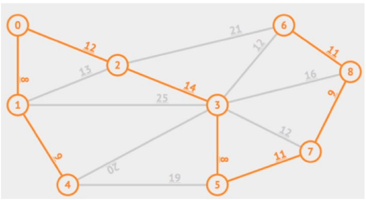
2

Practice



3

Solution



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

4

Minimum spanning trees

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

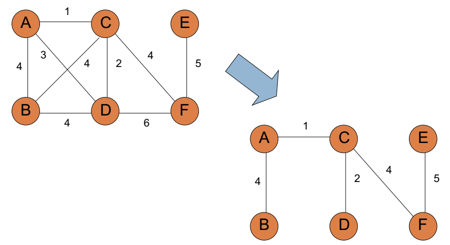
Input: An undirected, positive weight graph, G=(V,E)

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

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

5

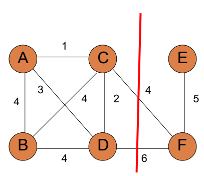
MST example



6

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.



7

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)
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
```

8

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.

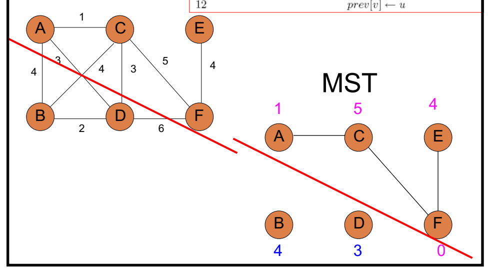
Let S be the set of vertices visited so far

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

9

Prim's

```
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
```



10

Running time of 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
```

11

Running time of Prim's

```
Prim(G, r)
for all v ∈ V                                   Θ(|V|)
    key[v] ← ∞
    prev[v] ← null
key[r] ← 0
H ← MakeHeap(key)                               1 call to MakeHeap
while !Empty(H)
    u ← Extract-Min(H)                          |V| calls to Extract-Min
    visited[u] ← true
    for each edge (u, v) ∈ E
        if !visited[v] and w(u, v) < key(v)
            Decrease-Key(v, w(u, v))            |E| calls to Decrease-Key
            prev[v] ← u
```

12

Running time of Prim's

	1 MakeHeap	|V| ExtractMin	|E| DecreaseKey	Total
Array	$\theta(\|V\|)$	$O(\|V\|^2)$	$O(\|E\|)$	$O(\|V\|^2)$
Bin heap	$\theta(\|V\|)$	$O(\|V\| \log \|V\|)$	$O(\|E\| \log \|V\|)$	$O((\|V\|+\|E\|) \log \|V\|)$ $O(\|E\| \log \|V\|)$
Fib heap	$\theta(\|V\|)$	$O(\|V\| \log \|V\|)$	$O(\|E\|)$	**$O(\|V\| \log \|V\| + \|E\|)$**

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

13

Running time of Prim's

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

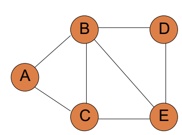
	1 MakeHeap	|V| ExtractMin	|E| DecreaseKey	Total
Array	$\theta(\|V\|)$	$O(\|V\|^2)$	$O(\|E\|)$	$O(\|V\|^2)$
Bin heap	$\theta(\|V\|)$	$O(\|V\| \log \|V\|)$	$O(\|E\| \log \|V\|)$	$O((\|V\|+\|E\|) \log \|V\|)$ $O(\|E\| \log \|V\|)$
Fib heap	$\theta(\|V\|)$	$O(\|V\| \log \|V\|)$	$O(\|E\|)$	**$O(\|V\| \log \|V\| + \|E\|)$**

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

14

Shortest paths

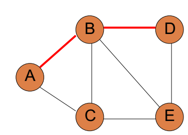
What is the shortest path from a to d?



15

Shortest paths

How can we find this?

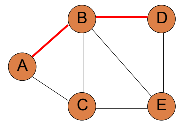


16

Shortest paths

BFS

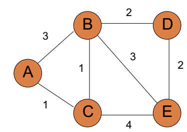
```
BFS(G, s)
for each v ∈ V
        dist[v] = ∞
dist[s] = 0
ENQUEUE(Q, s)
while !EMPTY(Q)
        u ← DEQUEUE(Q)
        VISIT(u)
        for each edge (u, v) ∈ E
                if dist[v] = ∞
                        ENQUEUE(Q, v)
                        dist[v] ← dist[u] + 1
```



17

Shortest paths

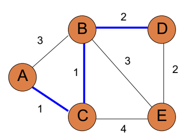
What is the shortest path from a to d?



18

Shortest paths

What is the shortest path from a to d?



19

Shortest path algorithms?

20

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
```

21

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
```

22

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
```

23

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
```

24

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
```

25

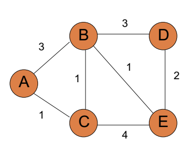
Dijkstra's algorithm

```
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
```

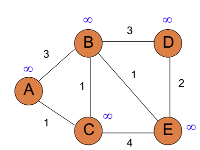
26

```
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
```

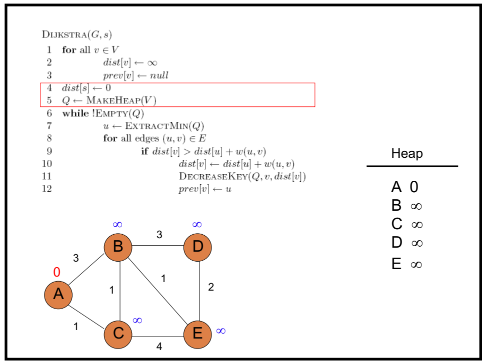


27

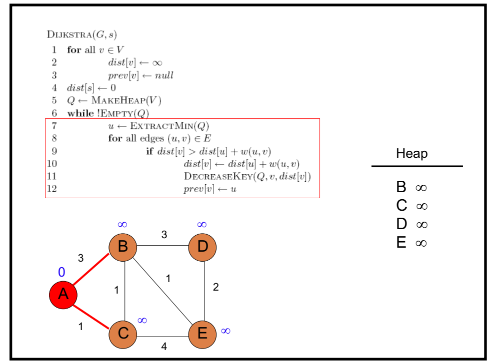
```
DIJKSTRA(G, s)
for all v ∈ V
        dist[v] ← ∞
        prev[v] ← null
dist[s] ← 0
Q ← MAKEHEAP(V)
while !EMPTY(Q)
        u ← EXTRACTMIN(Q)
        for all edges (u, v) ∈ E
                if dist[v] > dist[u] + w(u, v)
                        dist[v] ← dist[u] + w(u, v)
                      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



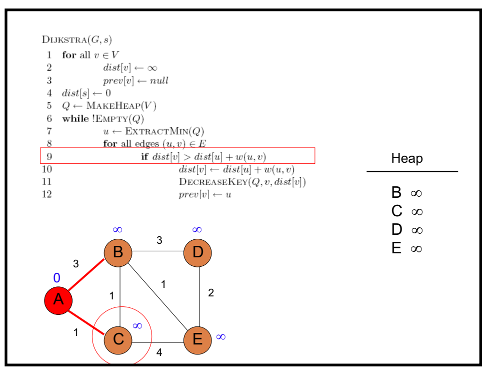
28


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
A 0
B ∞
C ∞
D ∞
E ∞

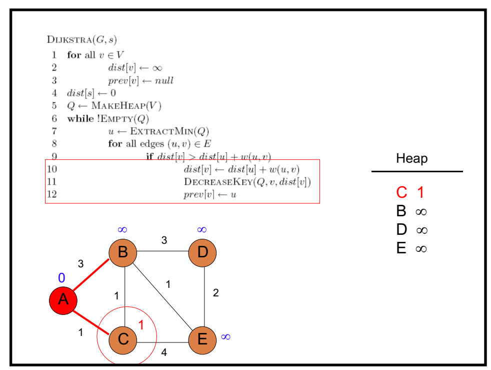
29


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 ∞
C ∞
D ∞
E ∞

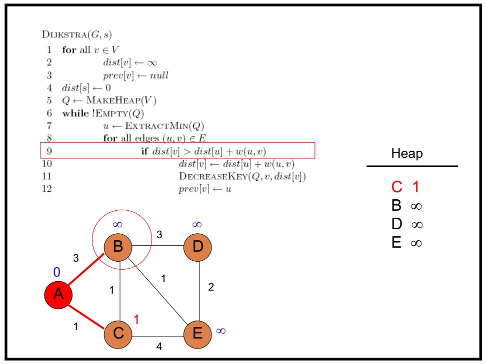
30


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 ∞
C ∞
D ∞
E ∞

31

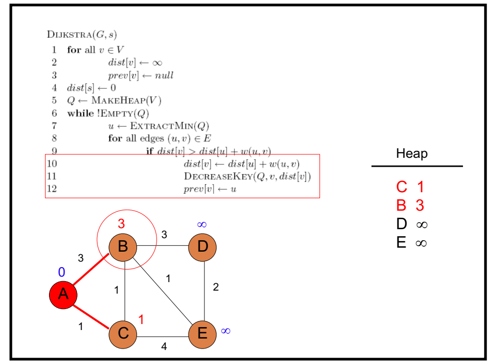

Dijkstra(G, s)
for all v ∈ V
dist[v] ← ∞
prev[v] ← null
dist[s] ← 0
Q ← MakeHeap(V)
while !Empty(Q)
u ← ExtractMin(Q)
for all edges (u, v) ∈ E
if dist[v] > dist[u] + w(u, v)
dist[v] ← dist[u] + w(u, v)
DecreaseKey(Q, v, dist[v])
prev[v] ← u
Heap
C 1
B ∞
D ∞
E ∞

32



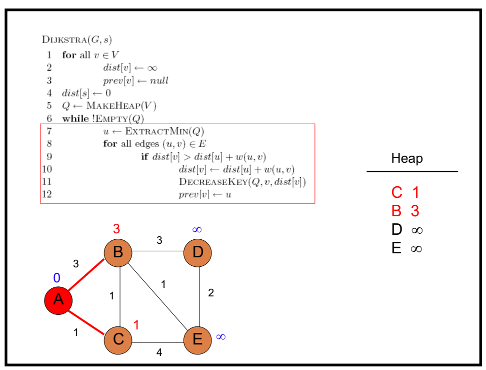
DIJKSTRA(G, s)
for all v ∈ V
dist[v] ← ∞
prev[v] ← null
dist[s] ← 0
Q ← MAKEHEAP(V)
while !EMPTY(Q)
u ← EXTRACTMIN(Q)
for all edges (u, v) ∈ E
if dist[v] > dist[u] + w(u, v)
dist[v] ← dist[u] + w(u, v)
DECREASEKEY(Q, v, dist[v])
prev[v] ← u
Heap
C 1
B ∞
D ∞
E ∞

33



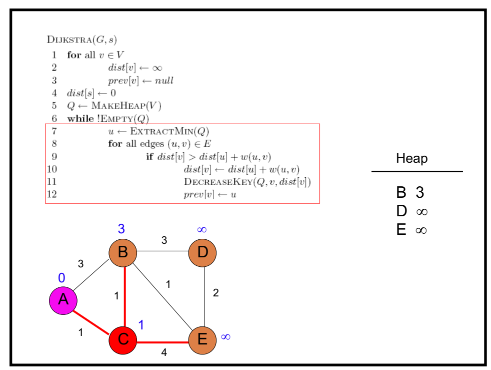
DIJKSTRA(G, s)
for all v ∈ V
dist[v] ← ∞
prev[v] ← null
dist[s] ← 0
Q ← MAKEHEAP(V)
while !EMPTY(Q)
u ← EXTRACTMIN(Q)
for all edges (u, v) ∈ E
if dist[v] > dist[u] + w(u, v)
dist[v] ← dist[u] + w(u, v)
DECREASEKEY(Q, v, dist[v])
prev[v] ← u
Heap
C 1
B 3
D ∞
E ∞

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
C 1
B 3
D ∞
E ∞

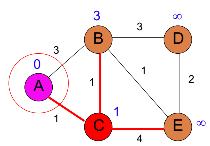
35



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 3
D ∞
E ∞

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
```

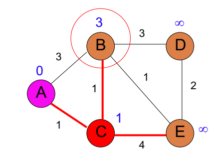


Heap

B 3
D ∞
E ∞

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
```

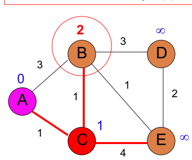


Heap

B 3
D ∞
E ∞

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
```

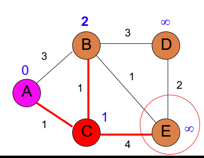


Heap

B 2
D ∞
E ∞

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

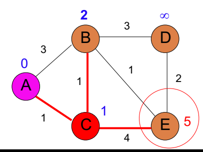
B 2
D ∞
E ∞

40

```
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 ∞

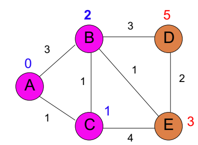


41

```
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

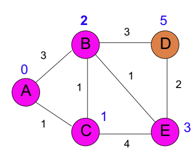


42

```
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

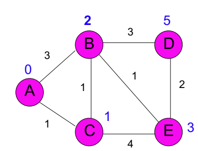
D 5



43

```
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
```

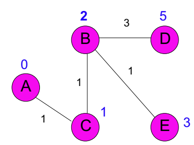
Heap



44

```
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

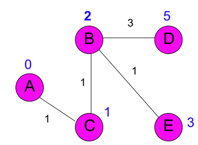


Prev

45

```
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

How do we get the actual paths?

46

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?

47

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

48

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
```

49

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
```

Θ(|V|)

1 call to MakeHeap

|V| calls to Extract-M

|E| calls to Decrease-K

50

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|)

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

Is this an improvement? If \|E\| < \|V\|^2 / 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|) O(|E| log |V|)
Fib heap	O(|V|)	O(|V| log |V|)	O(|E|)	O(|V| log |V| + |E|)

53

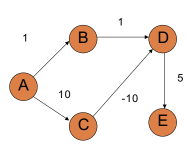
Dijkstra's vs Prim's

```
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
```

54

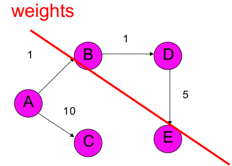
What about Dijkstra's on…?



55

What about Dijkstra's on…?

Dijkstra's algorithm only works for positive edge weights



56

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!

57

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?

58

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)

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

59

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

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?

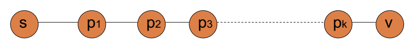
- If u is along the shortest path to v and dist[u] is correct

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

Consider the shortest path from s to v

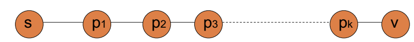


61

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

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

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

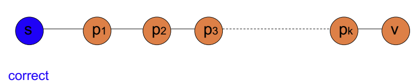


62

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

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?

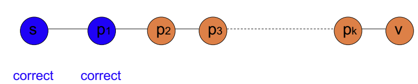


63

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

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?

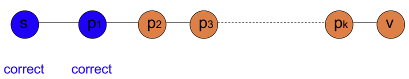


64

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

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

Does the order that we update the vertices matter?



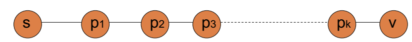
65

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

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



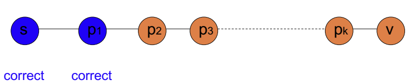
66

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

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

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

- i times



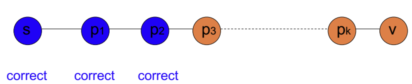
67

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

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



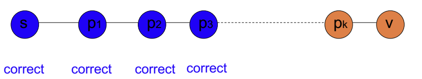
68

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

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



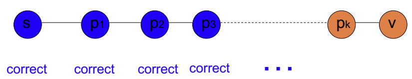
69

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

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



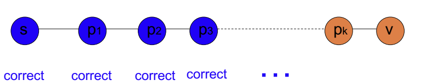
70

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

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

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

- |V| - 1 edges/vertices



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

```
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

73

Bellman-Ford algorithm

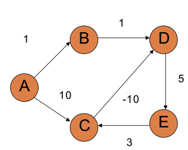
```
BELLMAN-FORD(G, s)
for all v ∈ V
    dist[v] ← ∞
    prev[v] ← null
dist[s] ← 0
for i ← 1 to |V| − 1
    for all edges (u, v) ∈ E
        if dist[v] > dist[u] + w(u, v)
            dist[v] ← dist[u] + w(u, v)
            prev[v] ← u
for all edges (u, v) ∈ E
    if dist[v] > dist[u] + w(u, v)
   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

check for negative cycles

74

Negative cycles

What is the shortest path from a to e?



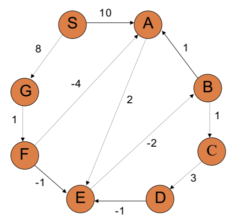
75

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
```

76

Bellman-Ford algorithm



How many edges is the shortest path from s to:

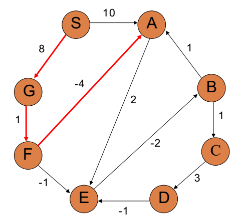
A:

B:

D:

77

Bellman-Ford algorithm

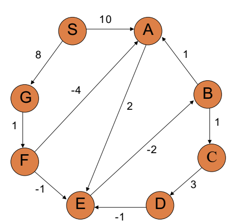


How many edges is the shortest path from s to:

A: 3

78

Bellman-Ford algorithm



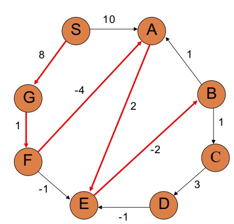
How many edges is the shortest path from s to:

A: 3

B:

79

Bellman-Ford algorithm



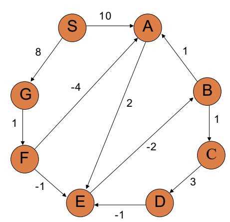
How many edges is the shortest path from s to:

A: 3

B: 5

80

Bellman-Ford algorithm



How many edges is the shortest path from s to:

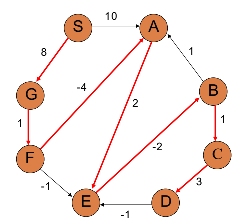
A: 3

B: 5

D:

81

Bellman-Ford algorithm



How many edges is the shortest path from s to:

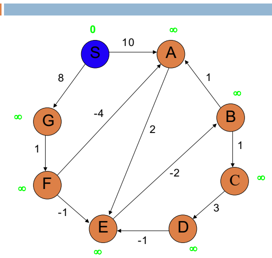
A: 3

B: 5

D: 7

82

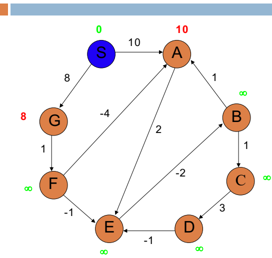
Bellman-Ford algorithm



Iteration: 0

83

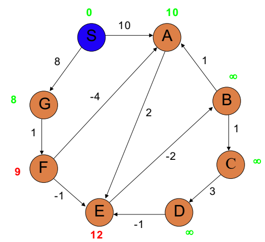
Bellman-Ford algorithm



Iteration: 1

84

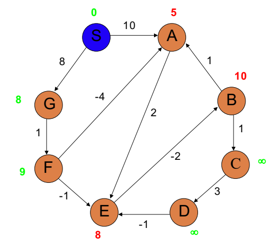
Bellman-Ford algorithm



Iteration: 2

85

Bellman-Ford algorithm

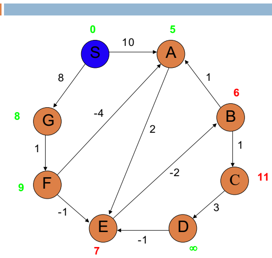


Iteration: 3

A has the correct distance and path

86

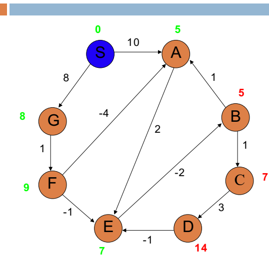
Bellman-Ford algorithm



Iteration: 4

87

Bellman-Ford algorithm

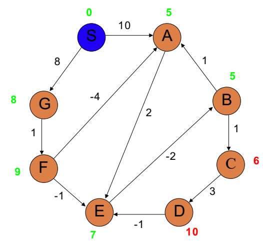


Iteration: 5

B has the correct distance and path

88

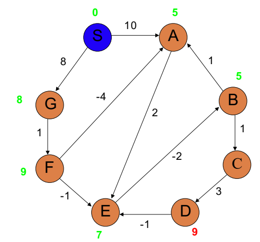
Bellman-Ford algorithm



Iteration: 6

89

Bellman-Ford algorithm



Iteration: 7

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

90

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
```

91

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|)

92

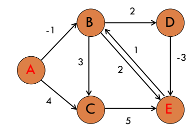
Runtime of Bellman-Ford

```
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)?

93

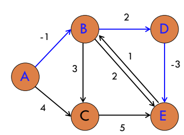
Shortest Paths



What is the shortest path from A to E?

94

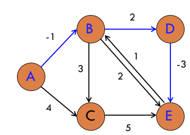
Shortest Paths



-2

95

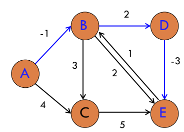
Shortest Paths



What algorithm would we use to calculate this?

96

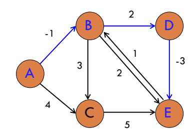
Shortest Paths



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

97

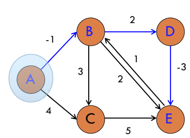
Shortest Paths



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

98

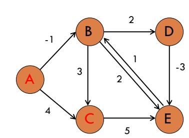
Shortest Paths



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

99

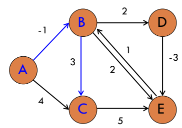
Shortest Paths



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?

100

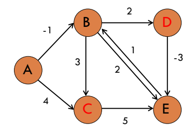
Shortest Paths



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

101

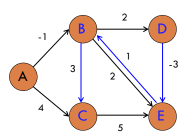
Shortest Paths



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?

102

Shortest Paths

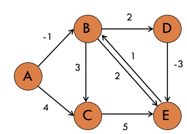


Different source.
Have to run Bellman-Ford again!

103

All pairs shortest paths

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



104

All pairs shortest paths

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

Easy solution?

105

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)?

106

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

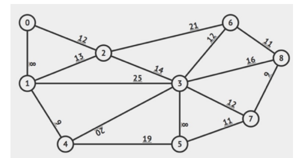
107

DAGs?

108

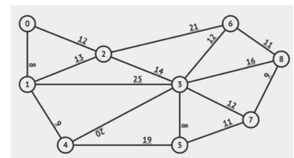
Handout

109



Kruskals

Add smallest edge that connects two sets not already connected

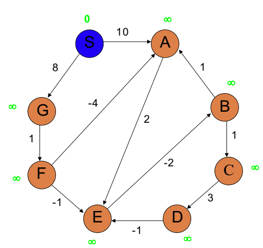


Prims

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

110

Bellman-Ford algorithm



111