CMPSC 465: LECTURE XVI

Revisit Dijkstra's Algorithm

Ke Chen

October 06, 2025

Dijkstra is weighted BFS

	BFS	Dijkstra
What		
Why		
How		

Dijkstra is weighted BFS

	BFS	Dijkstra
What	Visit the unvisited node with the smallest number of links from s	
Why		
How		

Dijkstra is weighted BFS

	BFS	Dijkstra
What	Visit the unvisited node with the smallest number of links from s	Add the unvisited node with the shortest distance from s to R
Why		
How		

Dijkstra is weighted BFS

	BFS	Dijkstra
What	Visit the unvisited node with the smallest number of links from s	Add the unvisited node with the shortest distance from s to R
Why	With more iterations, the number of links from s can only increase	
How		

Dijkstra is weighted BFS

	BFS	Dijkstra
What	Visit the unvisited node with the smallest number of links from s	Add the unvisited node with the shortest distance from s to R
Why	With more iterations, the number of links from s can only increase	With more iterations, the shortest distance from s can only increase
How		

Dijkstra is weighted BFS

	BFS	Dijkstra
What	Visit the unvisited node with the smallest number of links from s	Add the unvisited node with the shortest distance from s to R
Why	With more iterations, the number of links from s can only increase	With more iterations, the shortest distance from s can only increase (positive edge weights!)
How		

Dijkstra is weighted BFS

	BFS	Dijkstra
What	Visit the unvisited node with the smallest number of links from s	Add the unvisited node with the shortest distance from s to R
Why	With more iterations, the number of links from s can only increase	With more iterations, the shortest distance from s can only increase (positive edge weights!)
How	Use queue (FIFO, earlier in queue, fewer links needed)	

Dijkstra is weighted BFS

	BFS	Dijkstra
What	Visit the unvisited node with the smallest number of links from s	Add the unvisited node with the shortest distance from s to R
Why	With more iterations, the number of links from s can only increase	With more iterations, the shortest distance from s can only increase (positive edge weights!)
How	Use queue (FIFO, earlier in queue, fewer links needed)	Use priority queue

Dijkstra with priority queue

```
Dijkstra(G = (V, E, ℓ), s)
    // dist stores distances from s
    // prev can be used to reconstruct paths
    foreach v ∈ V do  dist[v] = ∞, prev[v] = null
    dist[s] = 0, Insert(Q, {dist[s], s}) // Q is a priority queue

    while Q is not empty do
        v = GetMin(Q), Delete(Q, 0)
        foreach (v, w) ∈ E do
            if dist[w] > dist[v] + ℓ(v, w) then
                dist[w] = dist[v] + ℓ(v, w)
                prev[w] = v

                // pos[w] returns the index of w in Q
                if w in Q then  DecreaseKey(Q, pos[w], dist[w])
                else  Insert(Q, {dist[w], w})
```

Dijkstra with priority queue

```
Dijkstra(G = (V, E, ℓ), s)
    // dist stores distances from s
    // prev can be used to reconstruct paths
    foreach v ∈ V do  dist[v] = ∞, prev[v] = null
    dist[s] = 0, Insert(Q, {dist[s], s}) // Q is a priority queue

    while Q is not empty do
        v = GetMin(Q), Delete(Q, 0)
        foreach (v, w) ∈ E do
            if dist[w] > dist[v] + ℓ(v, w) then
                dist[w] = dist[v] + ℓ(v, w)
                prev[w] = v

                // pos[w] returns the index of w in Q
                if w in Q then  DecreaseKey(Q, pos[w], dist[w])
                else  Insert(Q, {dist[w], w})
```

Time complexity?

Dijkstra with priority queue

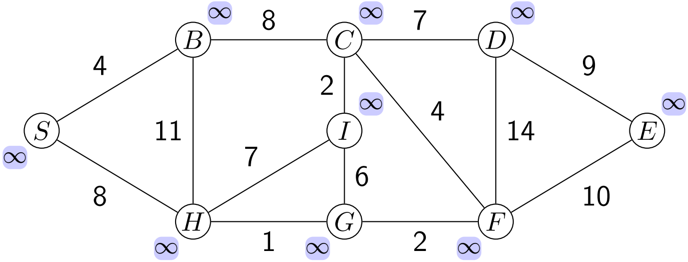
Dijkstra($G = (V, E, \ell)$, s)

```
// dist stores distances from s
// prev can be used to reconstruct paths
foreach v ∈ V do  dist[v] = ∞, prev[v] = null
dist[s] = 0, Insert(Q, {dist[s], s}) // Q is a priority queue

while Q is not empty do
    v = GetMin(Q), Delete(Q, 0)
    foreach (v, w) ∈ E do
        if dist[w] > dist[v] + ℓ(v, w) then
            dist[w] = dist[v] + ℓ(v, w)
            prev[w] = v

            // pos[w] returns the index of w in Q
            if w in Q then  DecreaseKey(Q, pos[w], dist[w])
            else  Insert(Q, {dist[w], w})
```

Time complexity? Calls at most $|V|$ Insert/Delete, and at most $|E|$ DecreaseKey, with a binary min-heap, $O\left((|V| + |E|)\log|V|\right)$.

One more example

G :

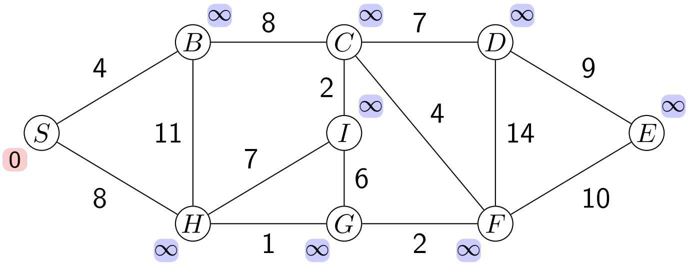


Q :

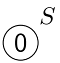
[empty]

One more example

G :

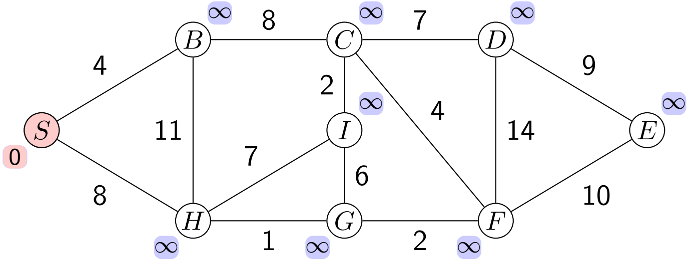


Q :



One more example

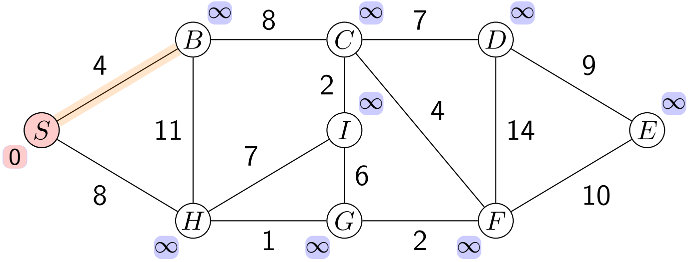
G :



Q :

One more example

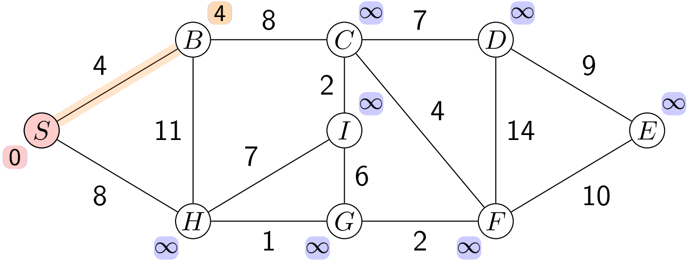
G :



Q :

One more example

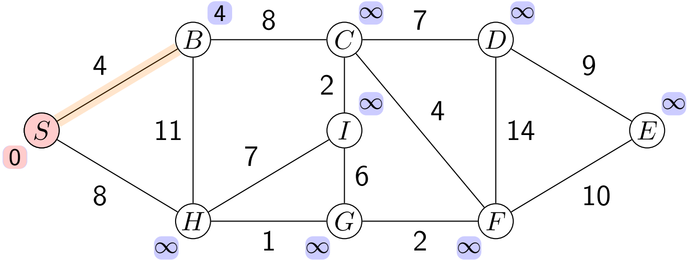
G :



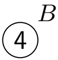
Q :

One more example

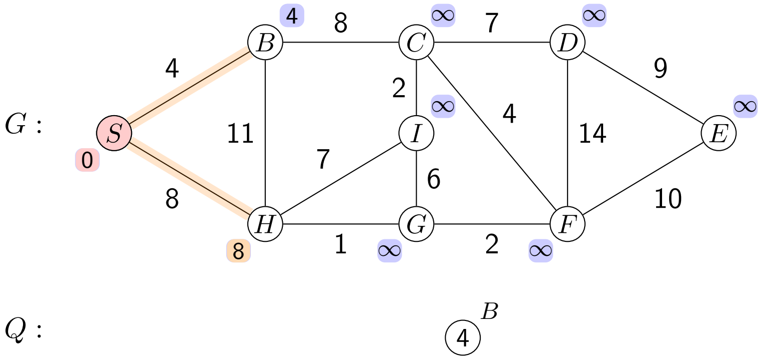
G :



Q :

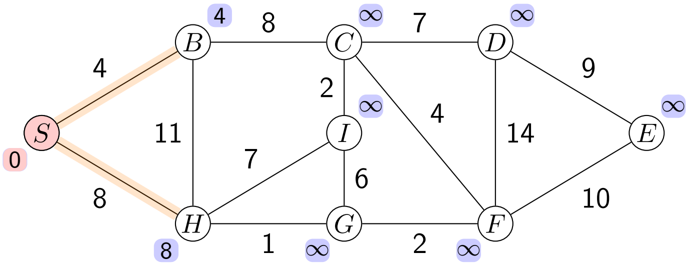


One more example

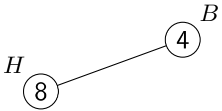


One more example

$G:$

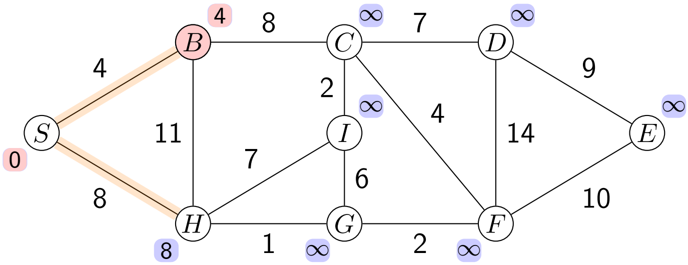


$Q:$

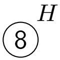


One more example

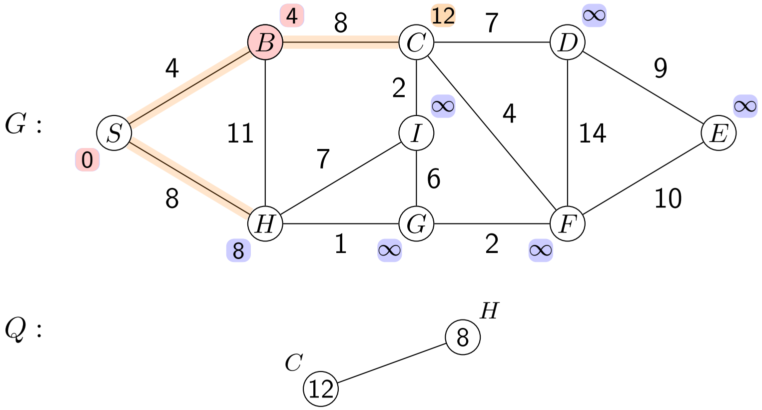
$G:$



$Q:$

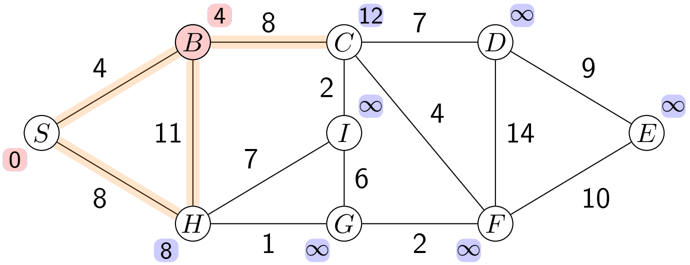


One more example

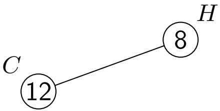


One more example

G :

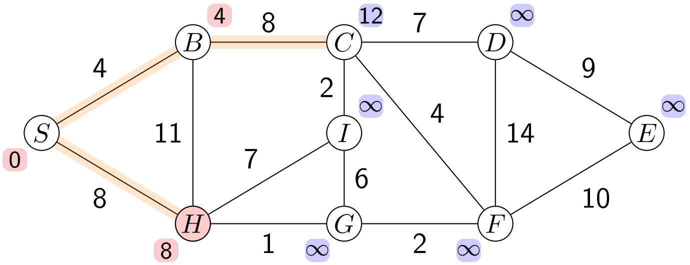


Q :

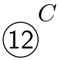


One more example

G :

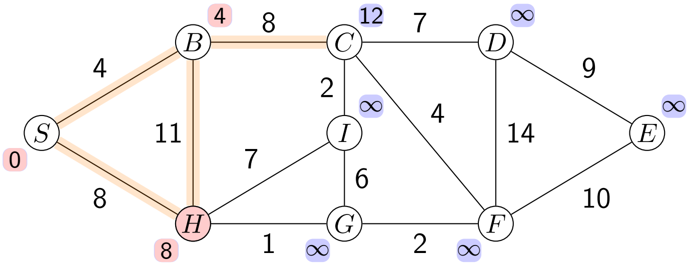


Q :

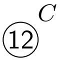


One more example

G :

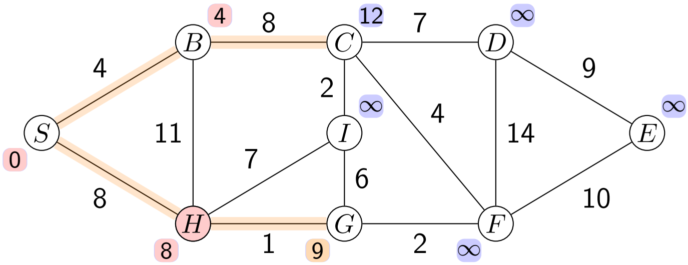


Q :

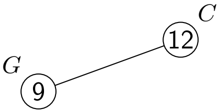


One more example

G :

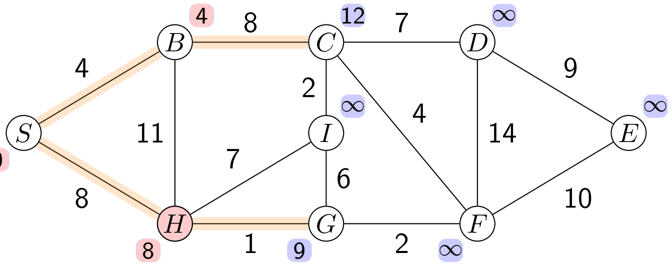


Q :

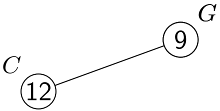


One more example

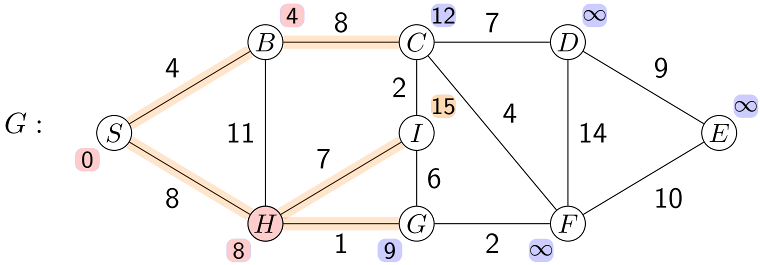
$G:$

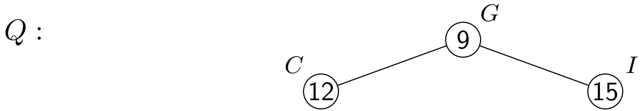


$Q:$



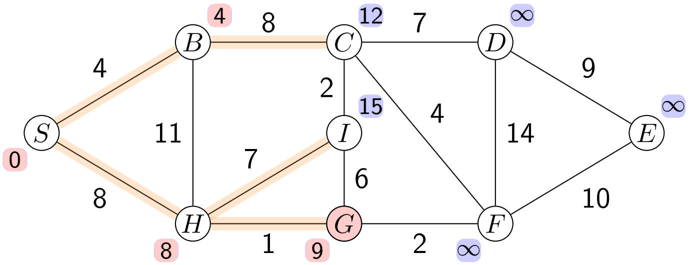
One more example



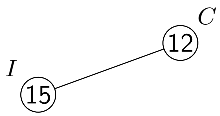


One more example

$G:$

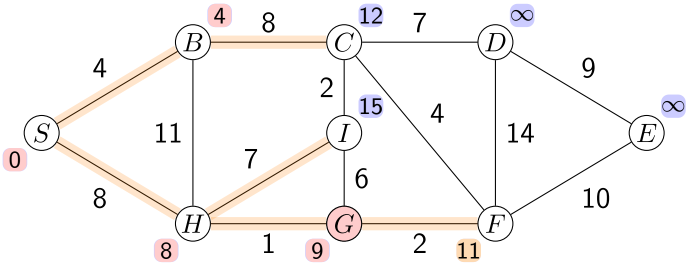


$Q:$

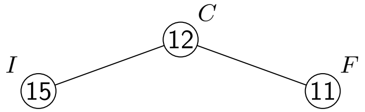


One more example

G :

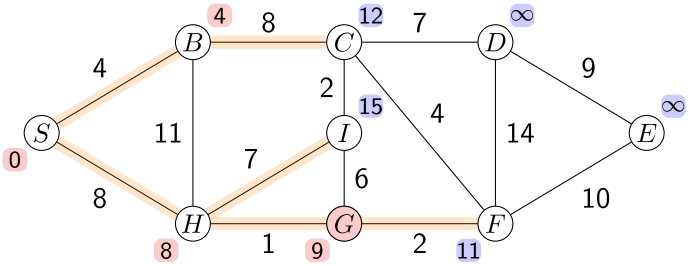


Q :

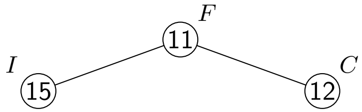


One more example

$G:$

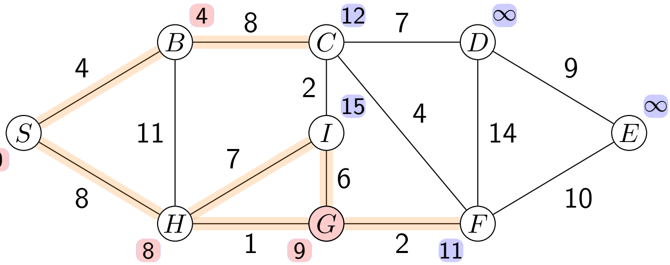


$Q:$

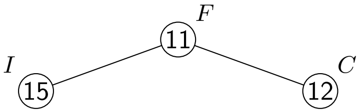


One more example

$G:$

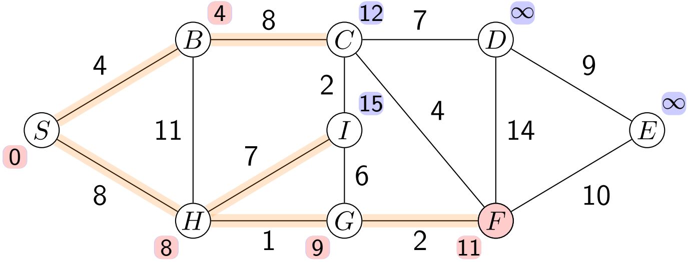


$Q:$

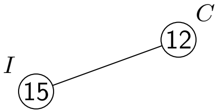


One more example

$G:$

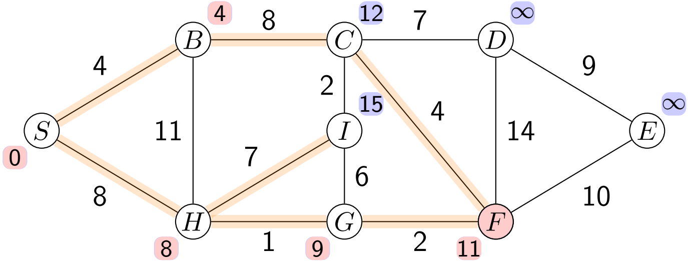


$Q:$

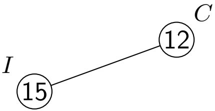


One more example

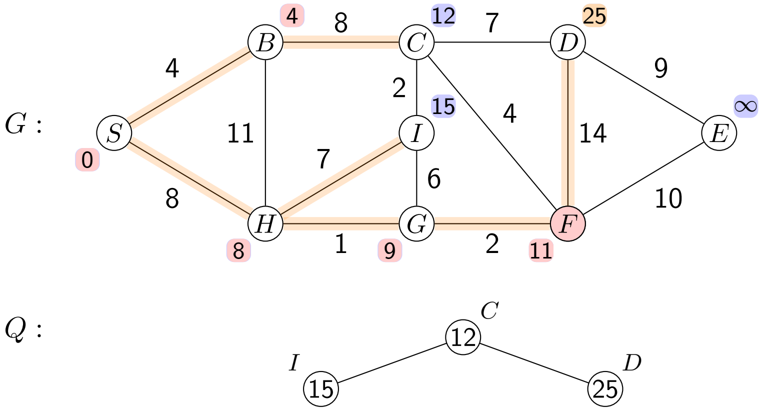
G :



Q :

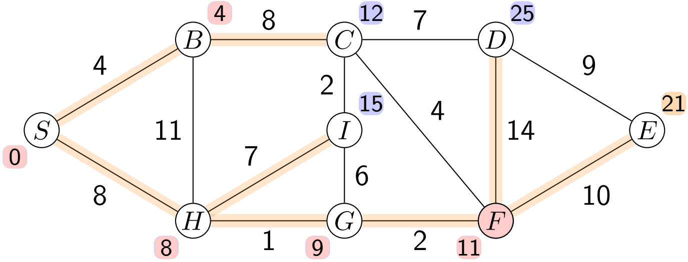


One more example

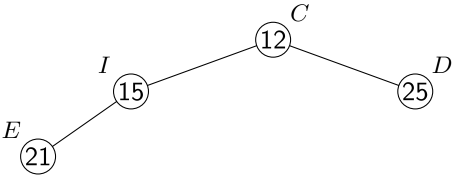


One more example

G :

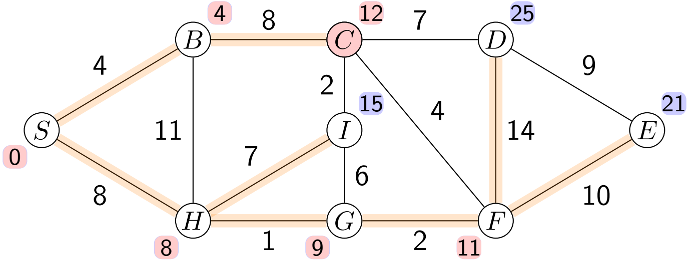


Q :

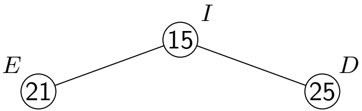


One more example

$G:$

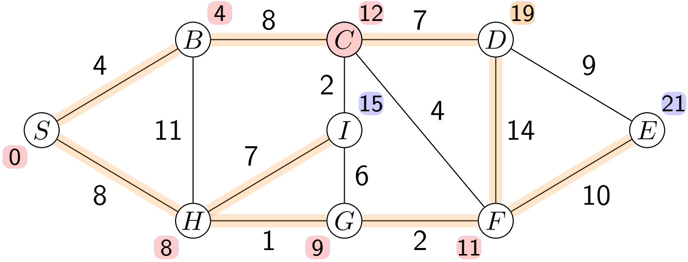


$Q:$

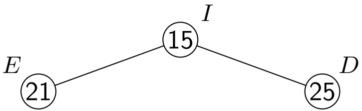


One more example

$G:$

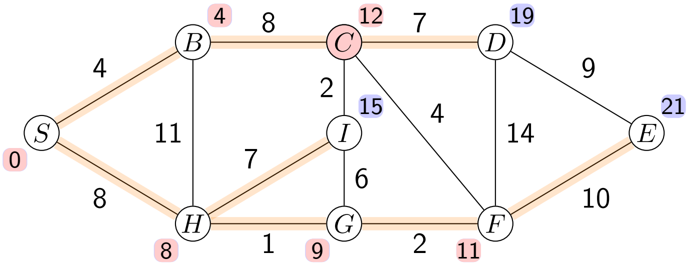


$Q:$

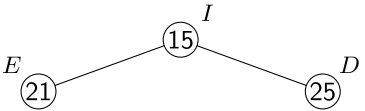


One more example

$G:$

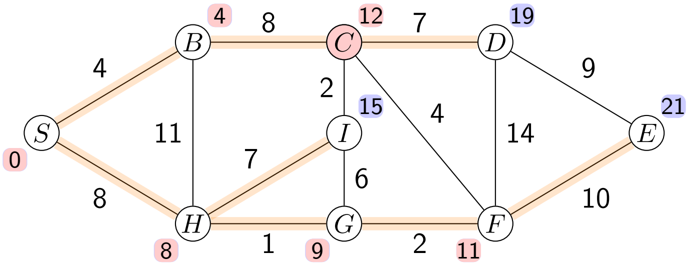


$Q:$

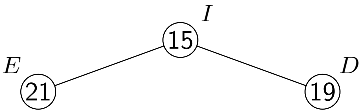


One more example

G :

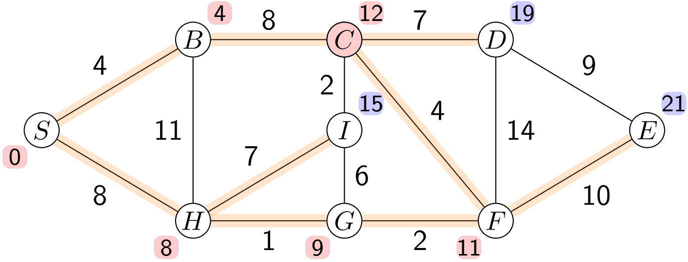


Q :

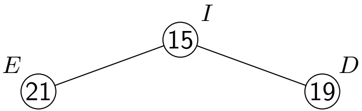


One more example

$G:$

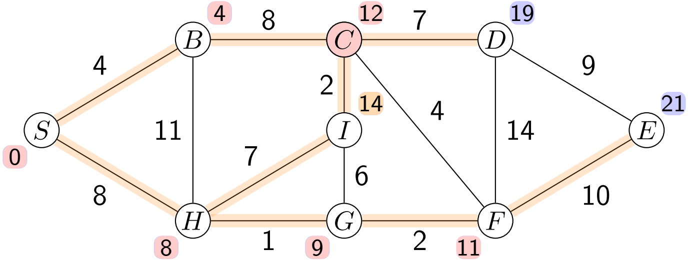


$Q:$

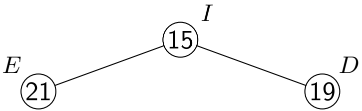


One more example

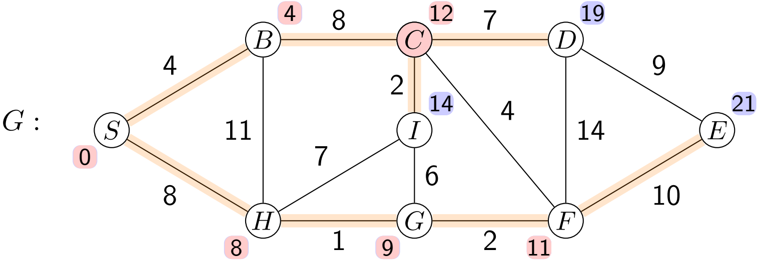
$G:$

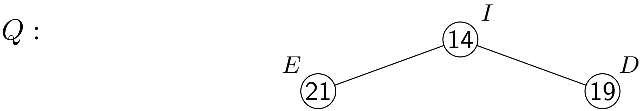


$Q:$



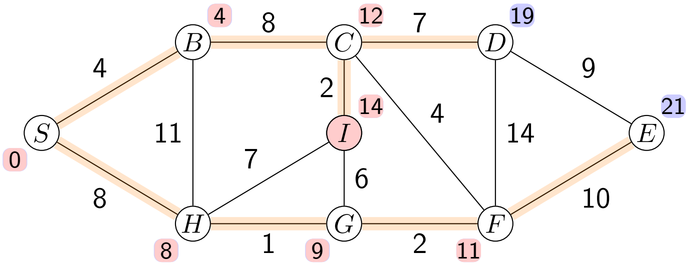
One more example



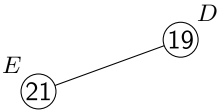


One more example

$G:$

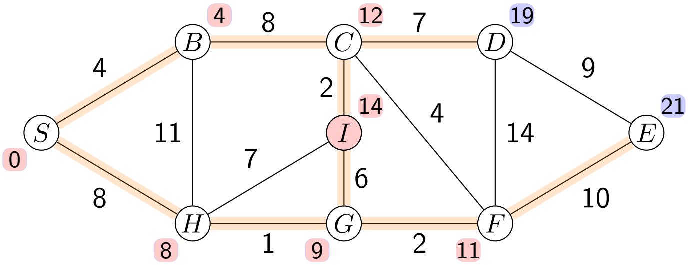


$Q:$

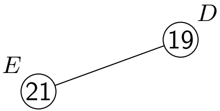


One more example

$G:$

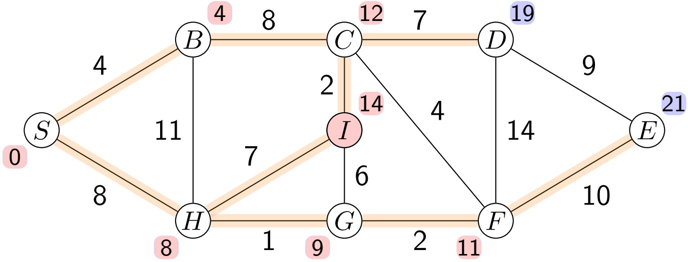


$Q:$

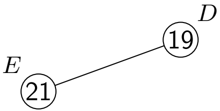


One more example

$G:$

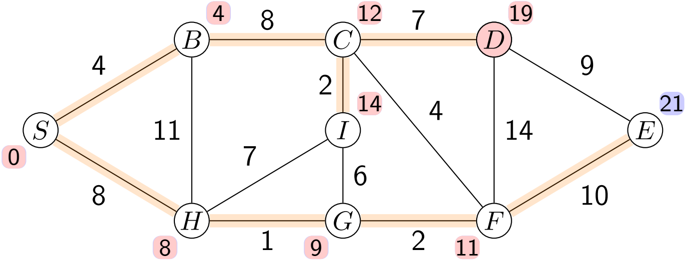


$Q:$

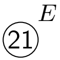


One more example

G :

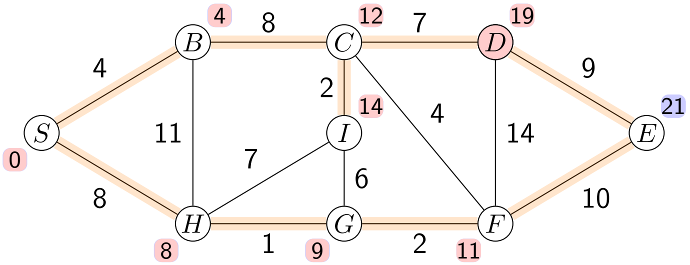


Q :

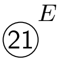


One more example

G :

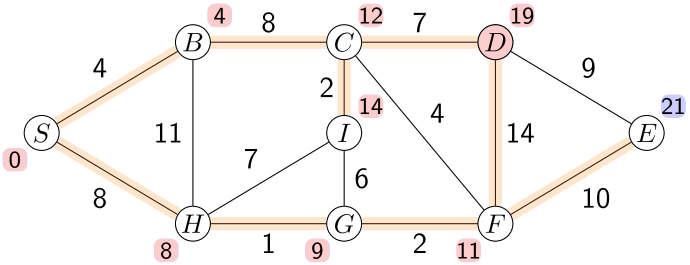


Q :

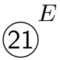


One more example

G :

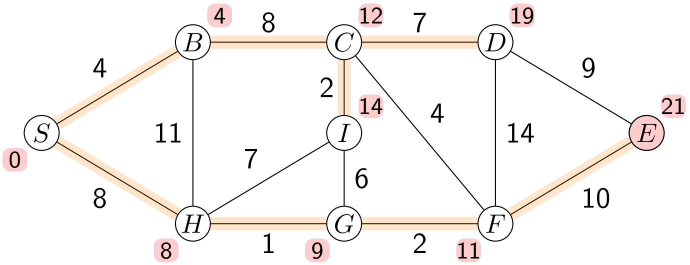


Q :



One more example

G :

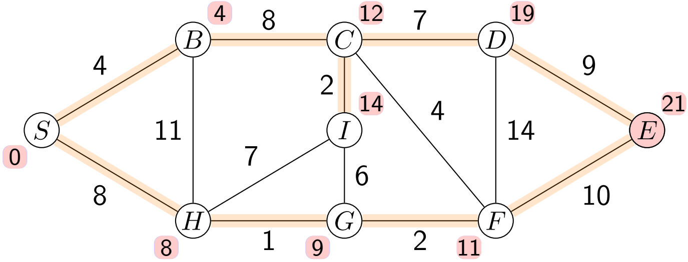


Q :

[empty]

One more example

G :

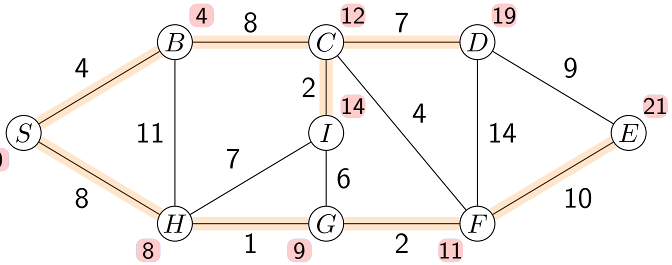


Q :

[empty]

One more example

G :



Q :

[empty]

Dijkstra with priority queue

Time complexity?

- Calls at most $|V|$ Insert and Delete, and at most $|E|$ DecreaseKey, with a binary min-heap, $O\left((|V| + |E|) \log |V|\right)$.

Dijkstra with priority queue

Time complexity?

- Calls at most $|V|$ Insert and Delete, and at most $|E|$ DecreaseKey, with a binary min-heap, $O\left((|V| + |E|) \log |V|\right)$.

Can we do better?

- With more fancy heaps (Fibonacci Heaps), we can achieve $O(|V| \log |V| + |E|)$.

Dijkstra with priority queue

Time complexity?

- Calls at most $|V|$ Insert and Delete, and at most $|E|$ DecreaseKey, with a binary min-heap, $O\left((|V| + |E|) \log |V|\right)$.

Can we do better?

- With more fancy heaps (Fibonacci Heaps), we can achieve $O(|V| \log |V| + |E|)$.
- A breakthrough this year by Duan et al. won the best paper award at STOC 2025.

A third interpretation of Dijkstra

- In Dijkstra, we maintain an array of upper bounds (overestimates) of distances.

A third interpretation of Dijkstra

- In Dijkstra, we maintain an array of upper bounds (overestimates) of distances.
- The bounds are tight for all vertices in R and the smallest one not in R.

A third interpretation of Dijkstra

- In Dijkstra, we maintain an array of upper bounds (overestimates) of distances.
- The bounds are tight for all vertices in R and the smallest one not in R.
- The only way an entry in $dist[\cdot]$ is reduced is by the Update operation:

Update($(v, w) \in E$)

if $dist[w] > dist[w] + \ell(v, w)$ **then**

  $dist[w] = dist[w] + \ell(v, w)$

A third interpretation of Dijkstra

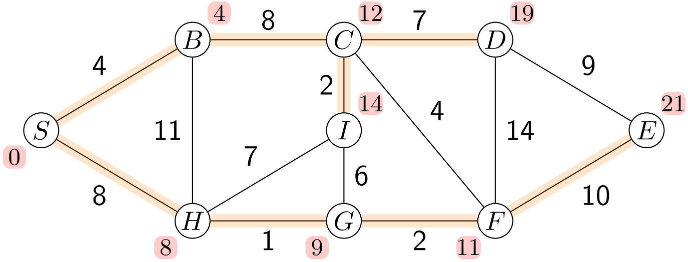
- In Dijkstra, we maintain an array of upper bounds (overestimates) of distances.
- The bounds are tight for all vertices in R and the smallest one not in R.
- The only way an entry in $dist[\cdot]$ is reduced is by the Update operation:

  Update($(v, w) \in E$)

  if $dist[w] > dist[w] + \ell(v, w)$ **then**

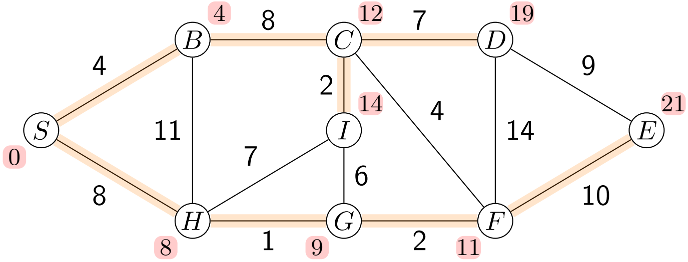
  $\quad dist[w] = dist[w] + \ell(v, w)$
- Dijkstra's algorithm can be viewed as a clever sequence of Update operations.

A third interpretation of Dijkstra



(S, B) (S, H) (B, C) (H, G) (G, F) (F, E) (C, D) (C, I)

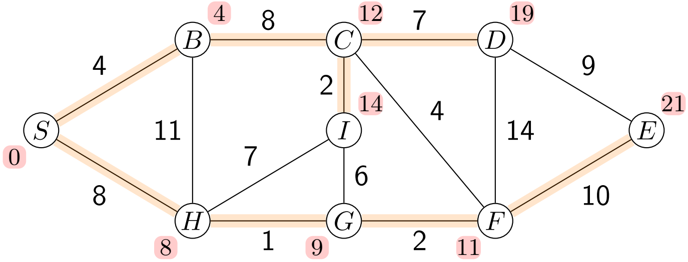
A third interpretation of Dijkstra



(S, B) (S, H) (B, C) (H, G) (G, F) (F, E) (C, D) (C, I)

- The computation is correct for a node as long as the sequence includes all edges on its shortest path in order.

A third interpretation of Dijkstra



(S, B) (S, H) (B, C) (H, G) (G, F) (F, E) (C, D) (C, I)

- The computation is correct for a node as long as the sequence includes all edges on its shortest path in order.
- Note that having additional Update calls doesn't hurt – Update is safe.