COMP 1201 Algorithmics

Shortest Path

Hikmat Farhat

Single Source Shortest Path

- Given a graph $G = (V, E)$ with a real-valued weight function $w : E \to \mathbb{R}$ we often ask the question:
- What is the minimal cost (shortest) path from $s \in V$ to all other vertices of the graph.
- We will look at two algorithms that perform that task
  1. Bellman-Ford.
  2. Dijkstra.
- First we need some definitions and theorems.

- Let $G = (V, E)$ with the associated weight function $w : E \to \mathbb{R}$.
- The total weight of a path $p = (v_0, \ldots, v_k)$ is the sum of weights of individual edges:

$$w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i)$$

- The shortest path cost δ

$$\delta(u, v) = \begin{cases} \min_p w(p) & \text{if there at least one path from u to v} \\ \infty & \text{otherwise} \end{cases}$$

Negative weights and cycles

- Even if a path contains edges with negative weight a shortest path can still be defined.
- It is undefined if the path contains a negative weight **cycle**.
- This is because we can "cross" the cycle as many times as we want, every time lower the cost.
- Therefore in the case when there is a negative cycle on a path from u to v then we set $\delta(u, v) = -\infty$ where $\delta(a, b)$ is the shortest path cost from a to b.

University of Southampton

Example of Negative Cycles

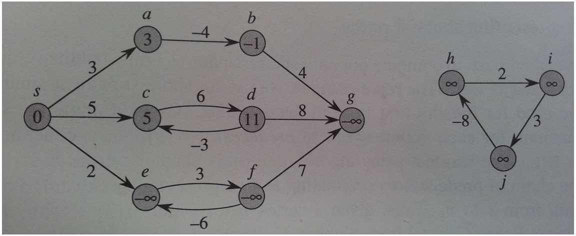


Figure: from CLRS

- $\delta(s,a) = 3$,$\delta(s,b) = -1$,$\delta(s,c) = 5$,$\delta(s,d) = 11$.
- (e,f) form a negative cycle therefore any node reachable from s through this cycle has $\delta = -\infty$
  $\delta(s,e) = \delta(s,f) = \delta(s,g) = -\infty$
- h, i, j are not reachable from s thus
  $\delta(s,h) = \delta(s,i) = \delta(s,j) = \infty$

Representation of Shortest Paths

- In all the algorithms that we will deal with, we maintain for every vertex v its predecessor $v.p$ (which could be NULL)
- At **termination** $v.p$ will be the predecessor of v on a shortest path from source s to v.
- We also maintain a value $v.d$ which at termination will be the value of the shortest path cost from source s to v.
- During the execution of the algorithm $v.d$ will be **an upper bound** on the value of the shortest path cost.

Relaxation

- **Relaxing** an edge (u, v) means testing if we can improve the shortest path cost of v by using the edge (u, v).
- If we can then we update $v.d$ and $v.p$.

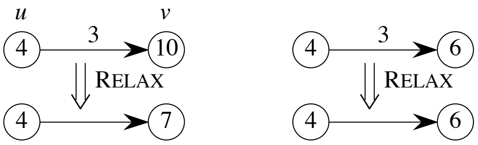


Figure: from CLRS

- In the figure to the left the cost of v was changed to the new cost (7) whereas to the right it was not changed since the new cost (7) is bigger than the current (6).
- What is NOT shown is the change to $v.p$ in the first case.

- Initially all vertices (except the source) have cost ∞ and no predecessors (including the source).

```
INITIALIZE(G,s)
```

foreach $v \in V$ **do**
    $v.d \leftarrow \infty$
    $v.p \leftarrow NULL$
$s.d \leftarrow 0$

```
RELAX(u,v)
```

if $v.d > u.d + w(u,v)$ **then**
    $v.d \leftarrow u.d + w(u,v)$
    $v.p \leftarrow u$

Bellman-Ford Algorithm

- The Bellman-Ford algorithm computes the shortest path from a given source to all other nodes in the graph.
- It works with negative weights.
- It can detect negative cycles.
- It uses the previously defined procedure RELAX to compute the shortest path.

Bellman-Ford Pseudo Code

```
BELLMAN-FORD(G,s);
INITIALIZE(G,s)
for i ← 1 To V − 1 do
    foreach (u, v) ∈ E do
        RELAX(u,v)

foreach (u, v) ∈ E do
    if v.d > u.d + w(u, v) then
        return FALSE
return TRUE
```

Example

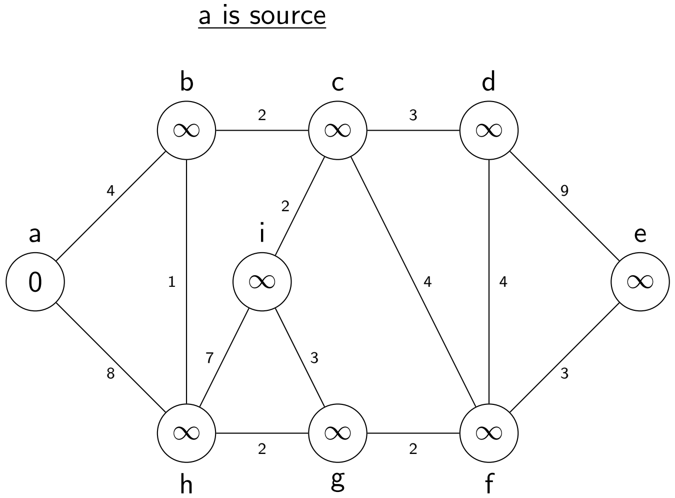
Complexity of Bellman-Ford

- The initialization is $O(|V|)$.
- the double loop is $O(|V| \cdot |E|)$.
- Therefore the total cost of the Bellman-Ford is $O(|V| \cdot |E|)$.

Dijkstra's Algorithm

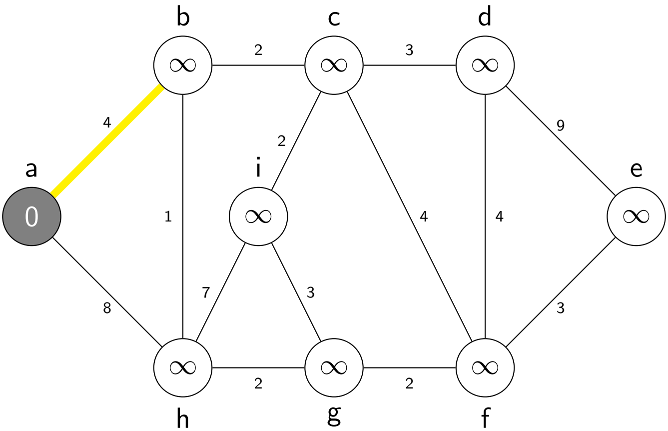
- Dijkstra's algorithm is another single source shortest path.
- It works when all weights are **positive**.
- We will see that it is faster than the Bellman-Ford algorithm.
- It maintains a set S of nodes whose shortest paths have been determined
- All other nodes are kept in a min-priority queue to keep track of the next node to process.

Example


a is source
b
c
d
a
i
e
h
g
f
0
∞
4
2
3
9
2
1
4
4
8
7
3
3
2
2

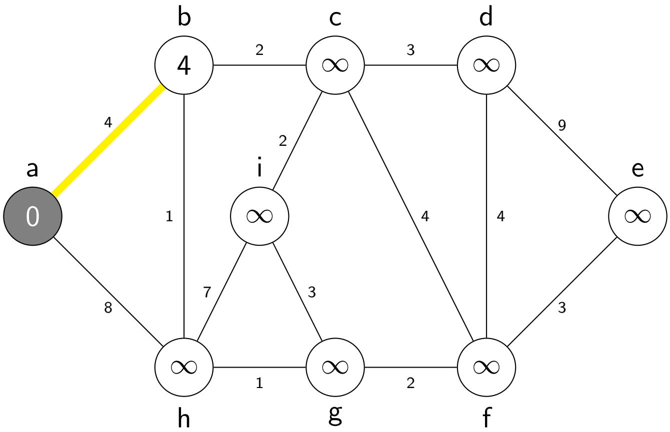
Example

neighbors of a

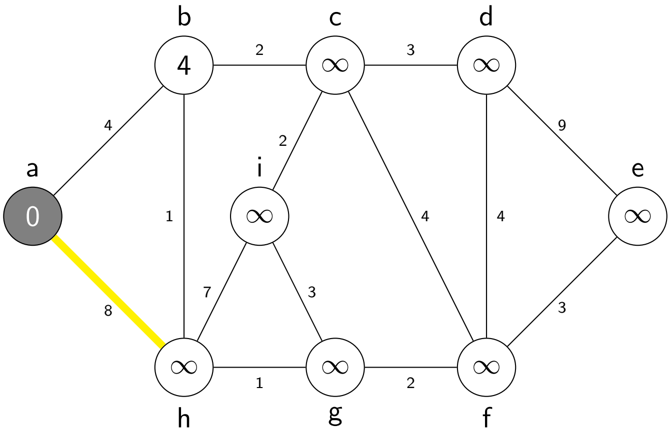


Example

neighbors of a

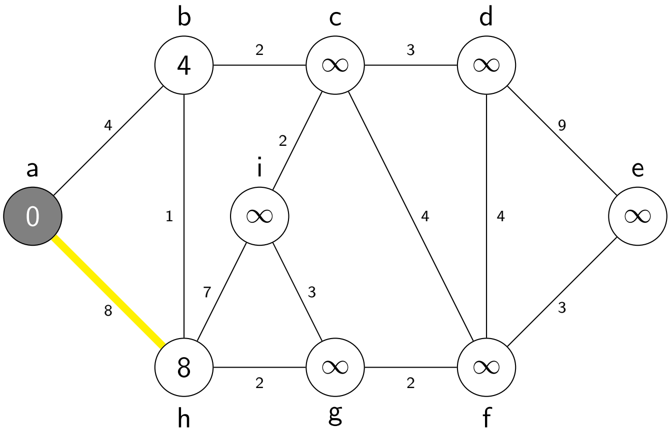


neighbors of a



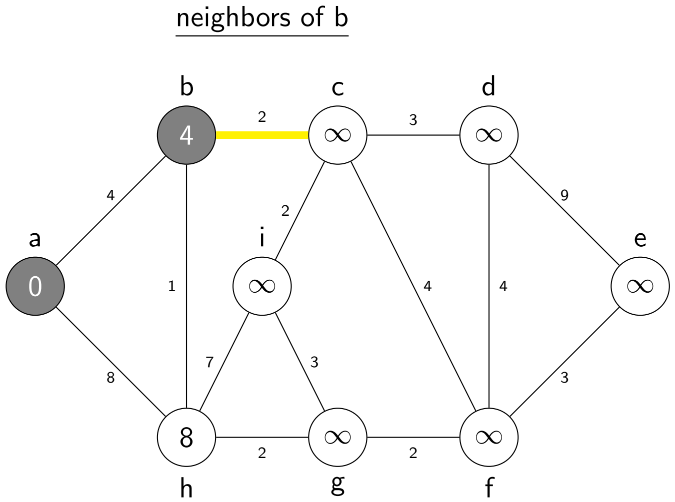
Example

neighbors of a



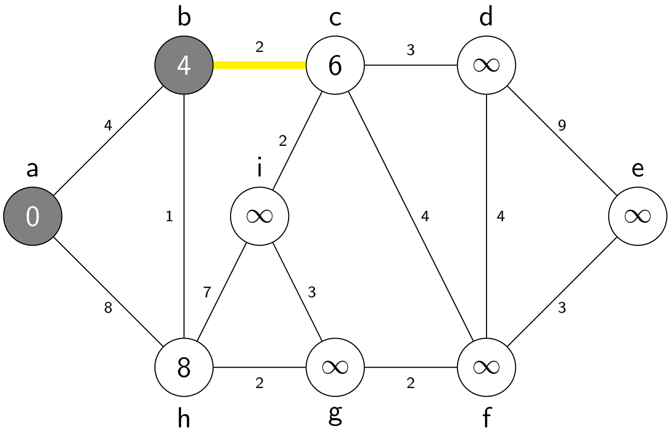
Southampton

Example



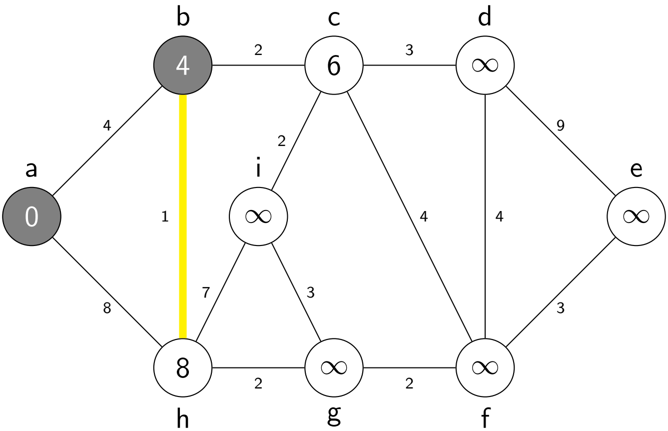
Example

neighbors of b



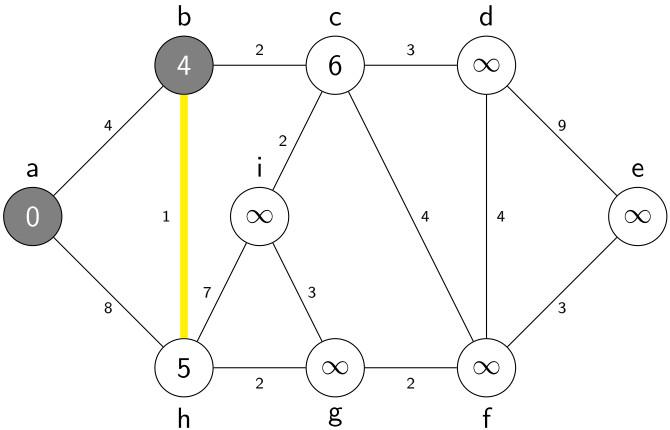
Example

neighbors of b

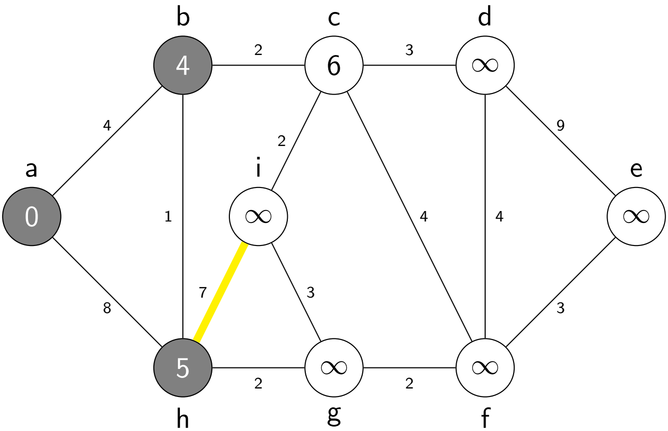


Example

neighbors of b

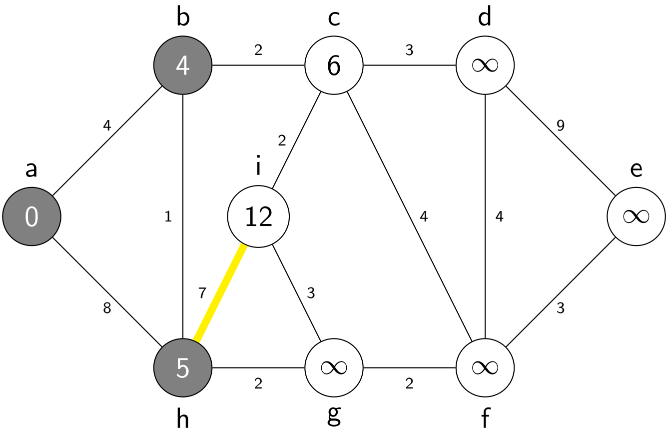


neighbors of h



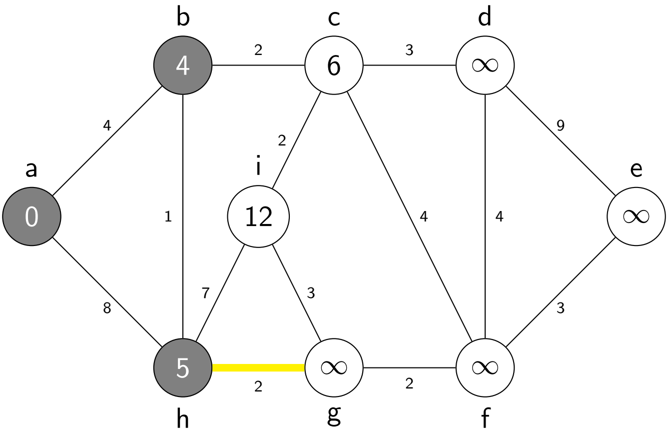
Example

neighbors of h



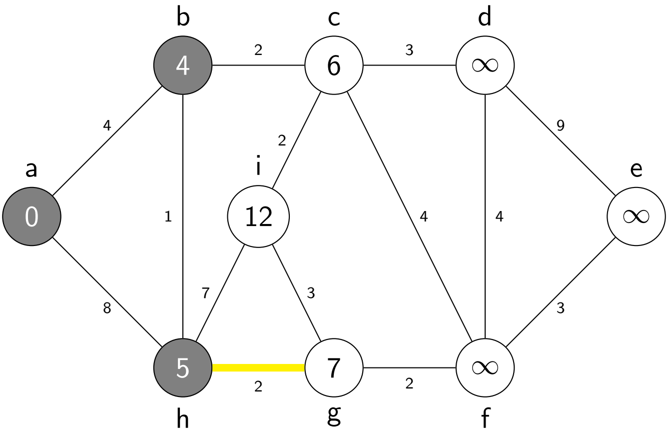
Example

neighbors of h



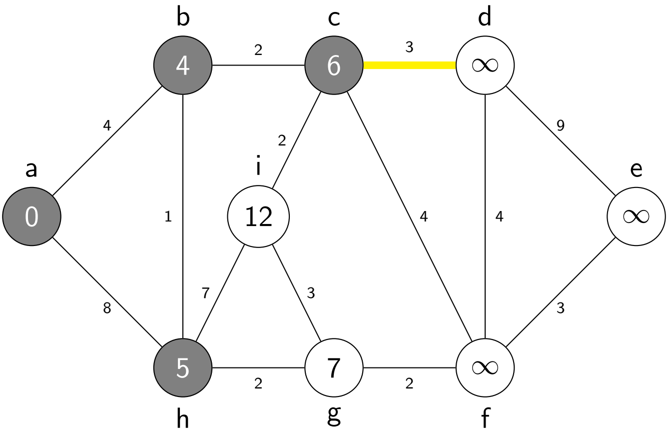
Example

neighbors of h



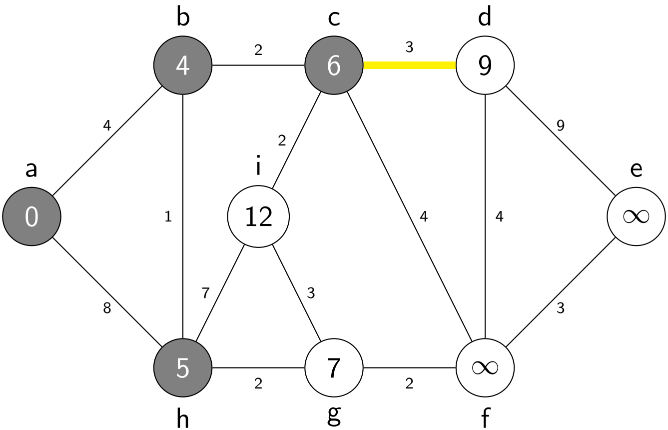
Example

neighbors of c



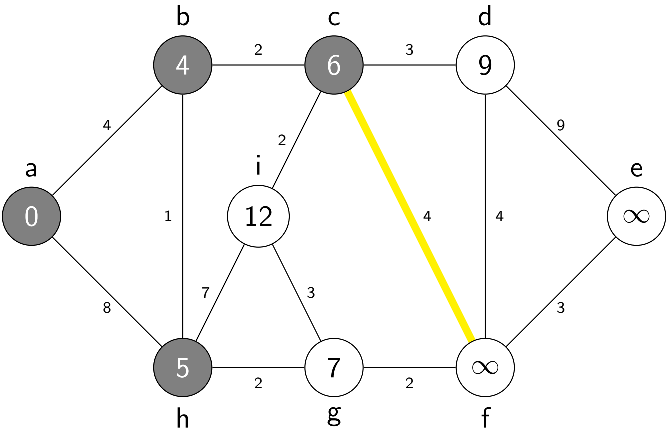
Example

neighbors of c



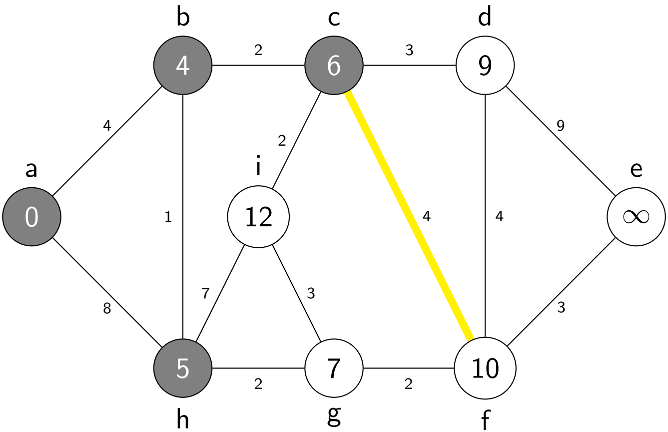
Example

neighbors of c



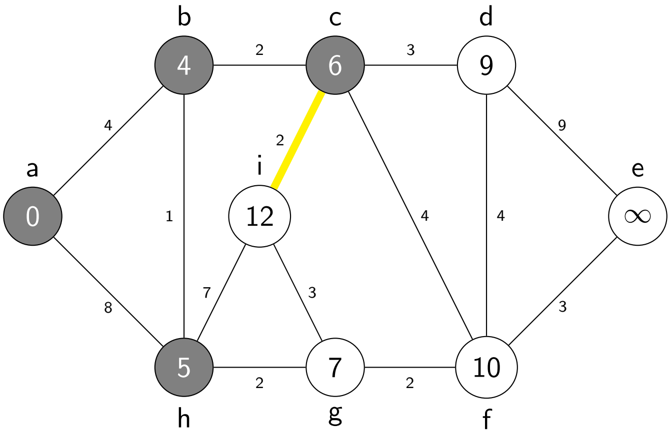
Example

neighbors of c



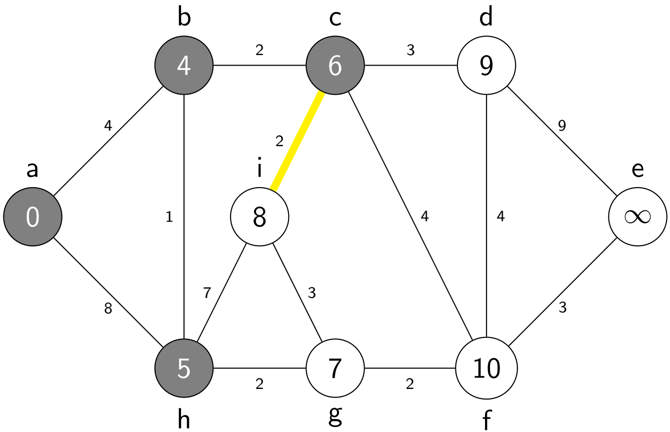
Example

neighbors of c



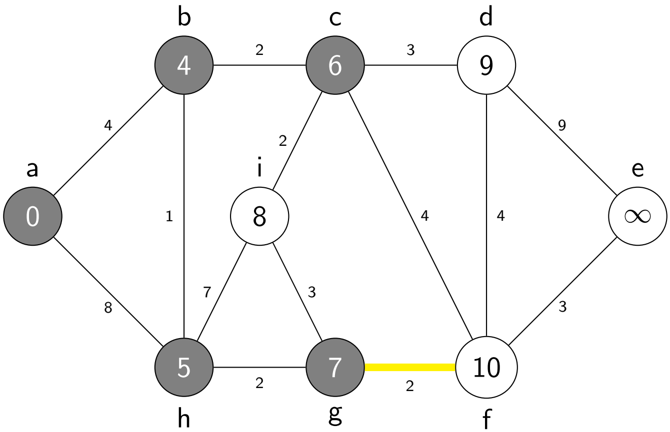
Example

neighbors of c

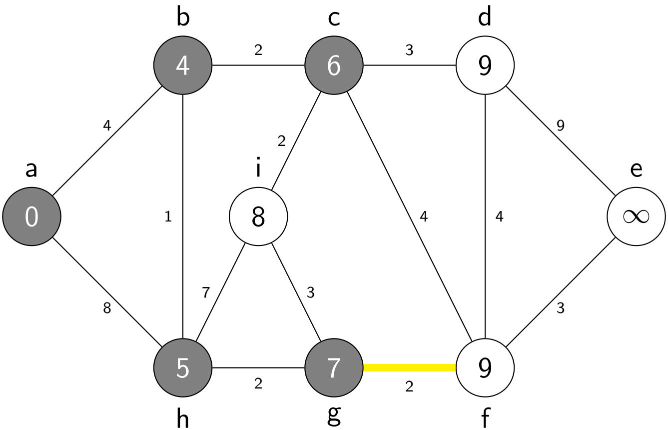


Example

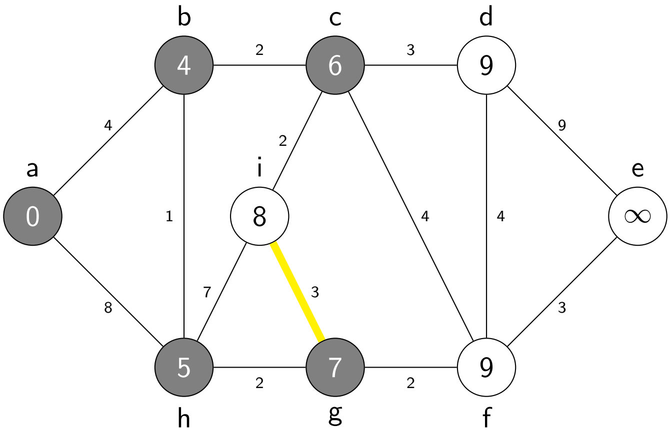
neighbors of g



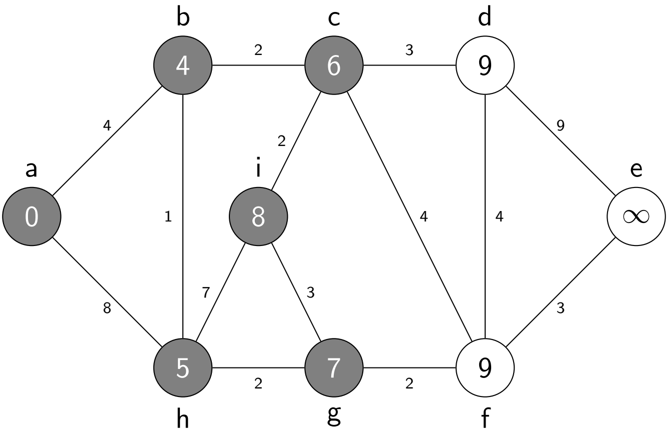
neighbors of g



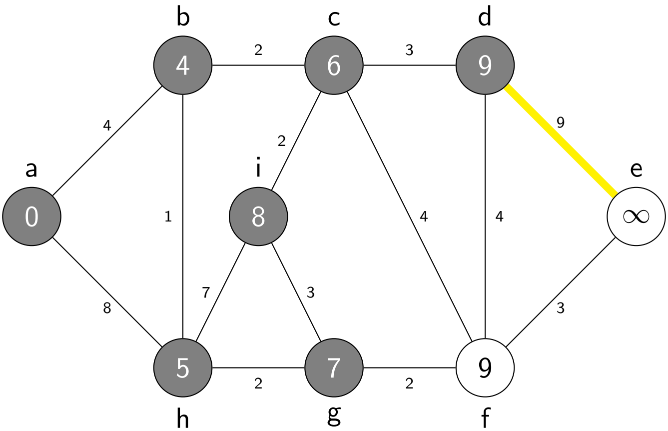
neighbors of g



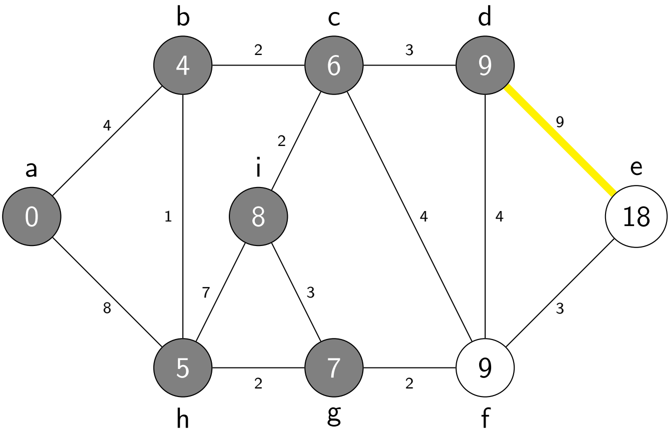
neighbors of i



neighbors of d

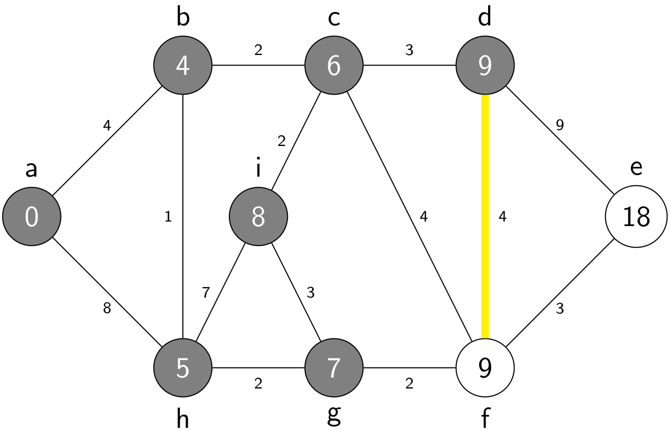


neighbors of d



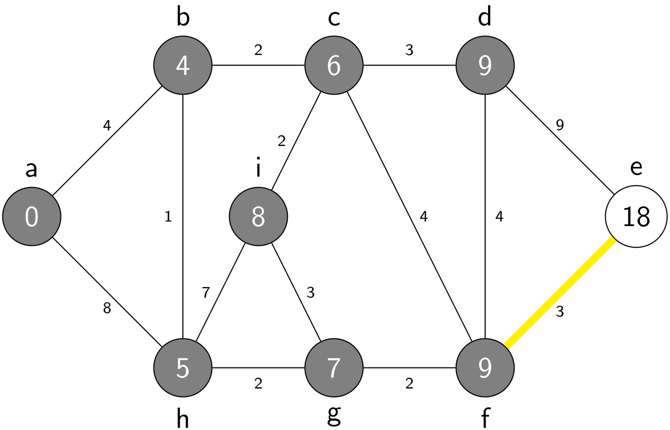
Example

neighbors of d

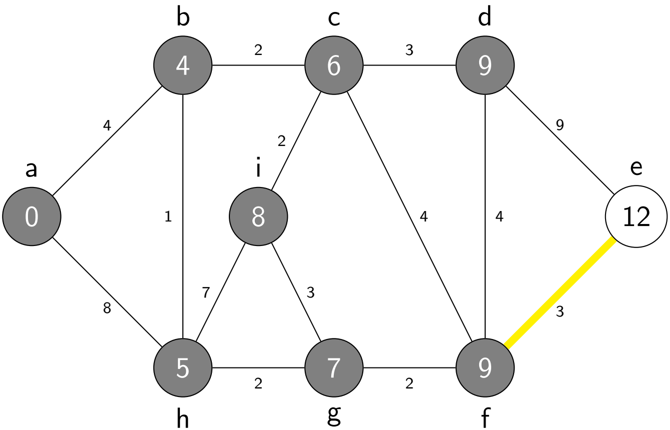


Example

neighbors of f

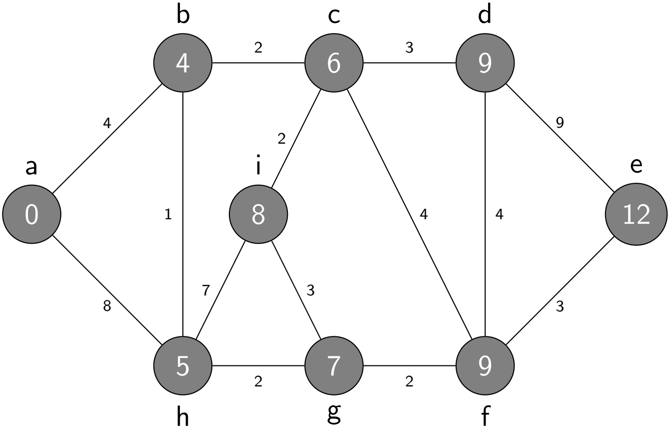


neighbors of f



Example

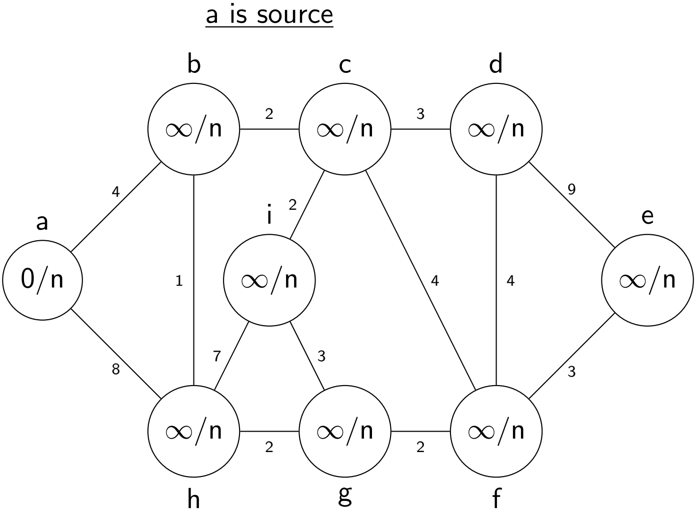
Done



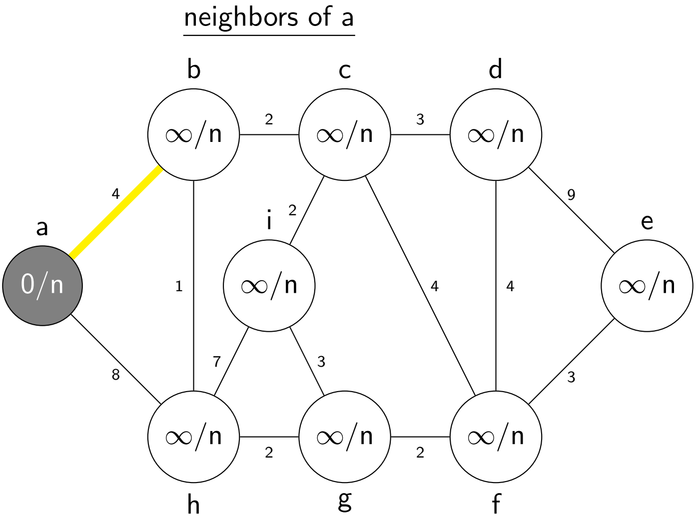
- The running time of Dijkstra's algorithm depends on the implementation of the queue.
- Using a min-heap on a sparse graph gives complexity of $O((V + E) \log V)$.
- This is because the while loop executes V times. The extract-min is $O(\log V)$ for a cost of $V \log V$. The relax includes an key update which means $\log V$. Since each edge is relaxed at most once then the total is E with a cost of $E \log V$.

And the shortest PATH?

- The algorithm computes the shortest distance from the source to all other nodes.
- But in most situations we are interested in actual shortest path from the source to the destinations.
- This can be done by updating and saving the predecessor of each node.

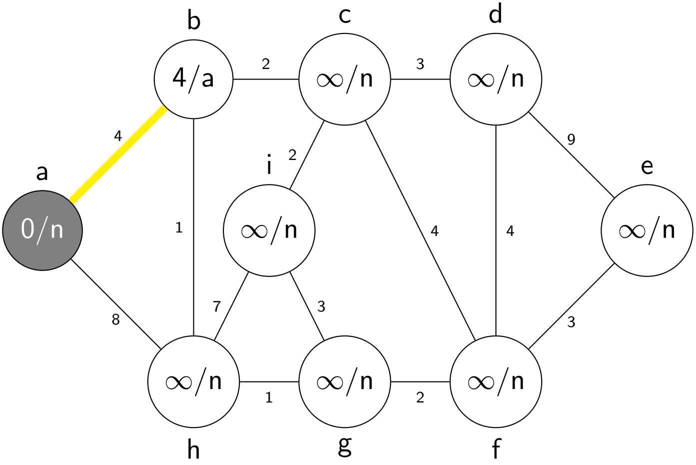

a is source
a
0/n
b
∞/n
c
∞/n
d
∞/n
e
∞/n
f
∞/n
g
∞/n
h
∞/n
i
∞/n
4
8
1
2
3
2
7
3
4
4
9
3
2
2

Example



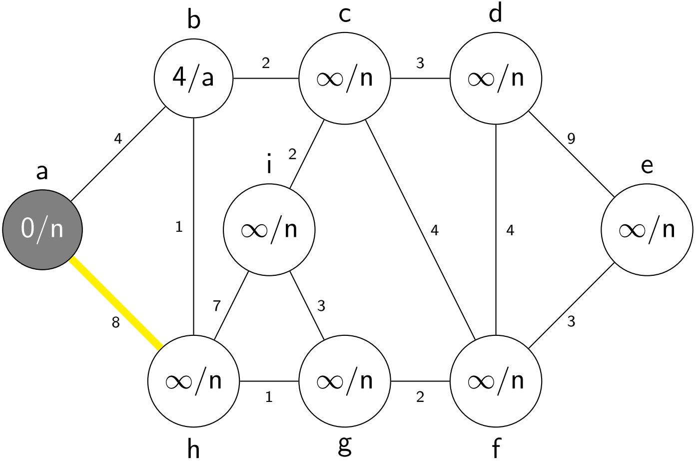
Example

neighbors of a

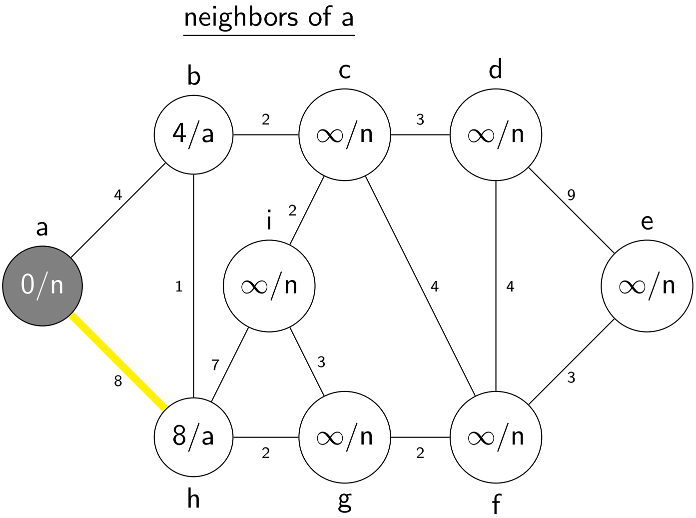


Example

neighbors of a



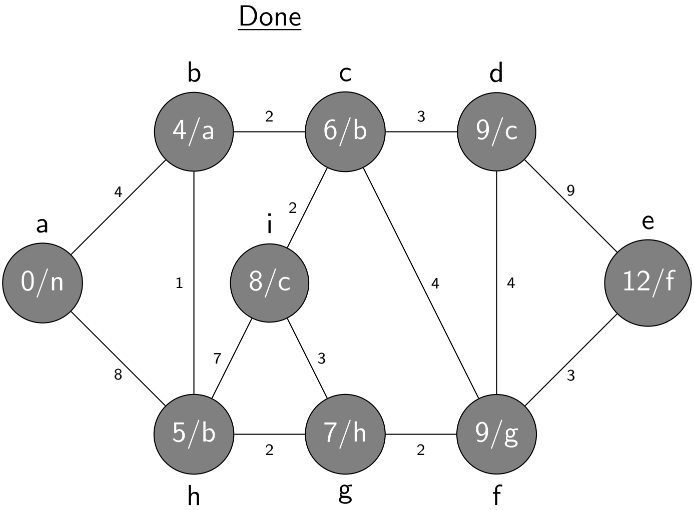
Example



Example

ETC...

Example



Recovering the shortest path

- By iterating backwards over the predecessors we can recover the shortest path.
- For example, consider node e
- $e.p = f, f.p = g, g.p = h, h.p = b, b.p = a$
- So the shortest path from a to e is a, b, h, g, f, e

Dijkstra Pseudo Code

DIJKSTRA(G,s);
INITIALIZE(G,s)
$S \leftarrow \emptyset$
$Q \leftarrow V$
while $Q \neq \emptyset$ **do**
    $u \leftarrow$ EXTRACT-MIN(Q)
    $S \leftarrow S \cup \{u\}$
    foreach $v \in adj[u]$ **do**
        RELAX(u,v)

Complexity

Let $n = |V|$ and $m = |E|$ and Q is a min-heap.

	`DIJKSTRA(G,s);`	
	`INITIALIZE(G,s)`	$O(n)$
1	$S \leftarrow \emptyset$	$O(1)$
2	$Q \leftarrow V$	$O(n)$
3	**while** $Q \neq \emptyset$ **do**	$O(n)$
4	$u \leftarrow$ `EXTRACT-MIN(Q)`	$O(\log n)$
5	$S \leftarrow S \cup \{u\}$	$O(1)$
6	**foreach** $v \in adj[u]$ **do**	$O(\|adj[u]\|)$
7	`RELAX(u,v)`	$O(\log n)$

The dominant term comes from the execution of line 7 $O(m)$ times(sum of adjacencies) for a total of $O(m \log n)$.