# Single-Source Shortest Paths

Chapter 24 of Cormen's book

Giulia Bernardini
*giulia.bernardini@units.it*

Algorithmic Design
a.y. 2024/2025

# Does BFS work for weighted graphs too?

BFS($G$,$s$) - $G$ is represented by the adjacency lists $\mathrm{Adj}[\cdot]$ of its vertices

```
    for each $u \in V \setminus \{s\}$
        $u.color \leftarrow$ white;
        $u.distance \leftarrow \infty$;
    $s.color \leftarrow$ gray;
    $s.distance \leftarrow 0$;
    $Q \leftarrow \emptyset$;
    enqueue($Q$,$s$);
    while $Q \neq \emptyset$
        $u \leftarrow$ dequeue($Q$);

        for each $v \in$ Adj[$u$]
            if $v.color$ = white
                $v.color \leftarrow$ gray;
                $\boxed{v.distance \leftarrow u.distance + 1;}$
                enqueue($Q$,$v$);
        $u.color \leftarrow$ black;
```

BFS assigns to each $v$ value $v.distance$, the least possible number of edges on any source-to-$v$ path.

# Does BFS work for weighted graphs too?

BFS($G$,$s$) - $G$ is represented by the adjacency lists Adj[·] of its vertices

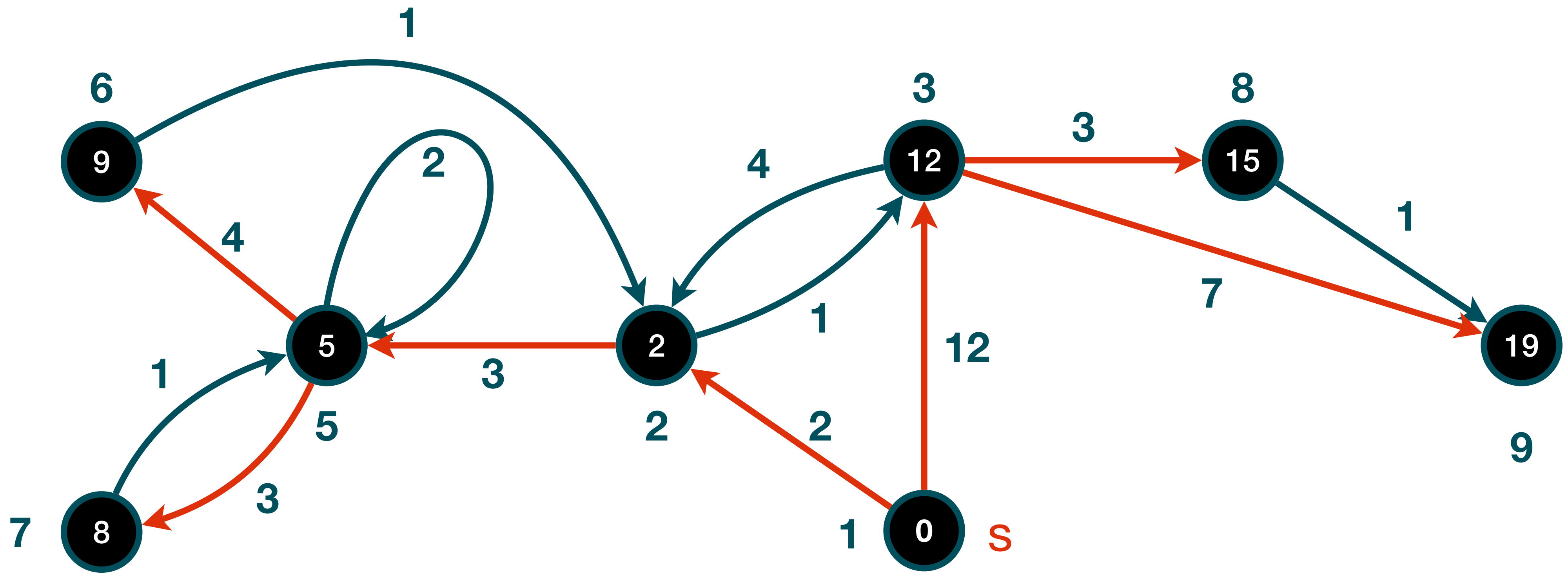
```
for each $u \in V \setminus \{s\}$
    $u.color \leftarrow$ white;
    $u.distance \leftarrow \infty$;
$s.color \leftarrow$ gray;
$s.distance \leftarrow 0$;
$Q \leftarrow \emptyset$;
enqueue($Q$,$s$);
while $Q \neq \emptyset$
    $u \leftarrow$ dequeue($Q$);
    for each $v \in$ Adj[$u$]
        if $v.color$ = white
            $v.color \leftarrow$ gray;
            $\boxed{v.distance \leftarrow u.distance + w(u,v)}$
            enqueue($Q$,$v$);
    $u.color \leftarrow$ black;
```

BFS assigns to each $v$ value $v.distance$, the least possible number of edges on any source-to-$v$ path.

Can't we just modify this instruction to make it work for weighted graphs?

# Why does BFS not work for weighted graphs?

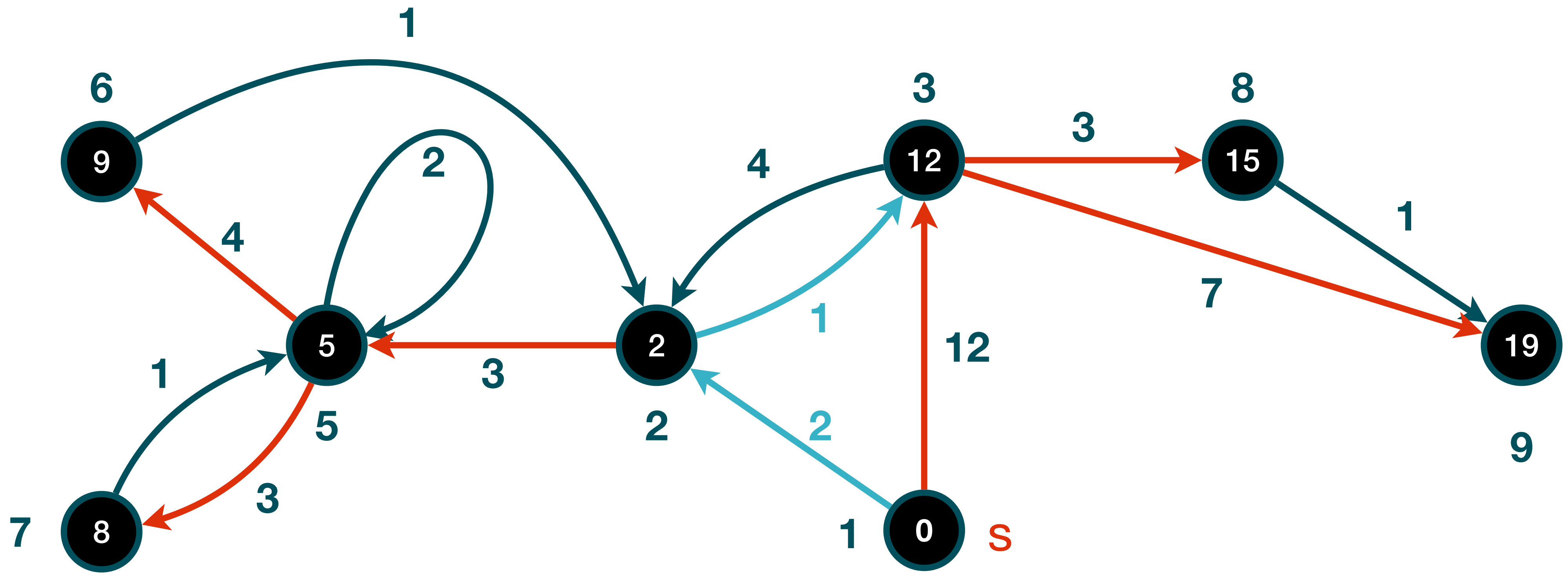
The shortest weighted path between two vertices may not be the one with the least number of edges!



# Why does BFS not work for weighted graphs?

The shortest weighted path between two vertices may not be the one with the least number of edges!

The shortest path from S to vertex number 3 would be through vertex 2: the length of 1 2 3 is 2+1=3<12, even if this path has two edges in place of one.



# Dijkstra’s algorithm

If all the weights are nonnegative, we can use Dijkstra’s (pronounced “Deikstra”) algorithm.

Recall:

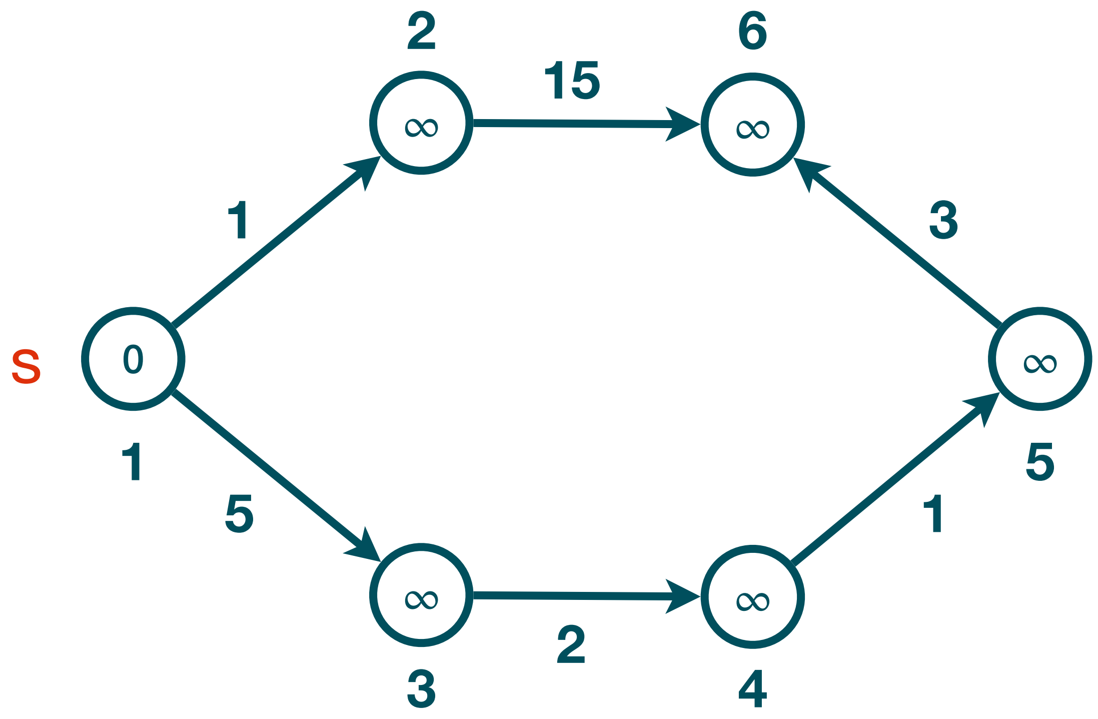
```
RELAX($u$,$v$,$w$)
    if $v.d > u.d + w(u,v)$
        $v.d = u.d + w(u,v);$
        $v.p \leftarrow u;$
```

# Dijkstra’s algorithm

At each step, one edge is relaxed. The vertices that are still to be finalised are maintained in a min-priority queue (many different implementations are possible). S is the set of finalised vertices.

S={}

$Q=\{\overset{1}{0},\overset{2}{\infty},\overset{3}{\infty},\overset{4}{\infty},\overset{5}{\infty},\overset{6}{\infty}\}$



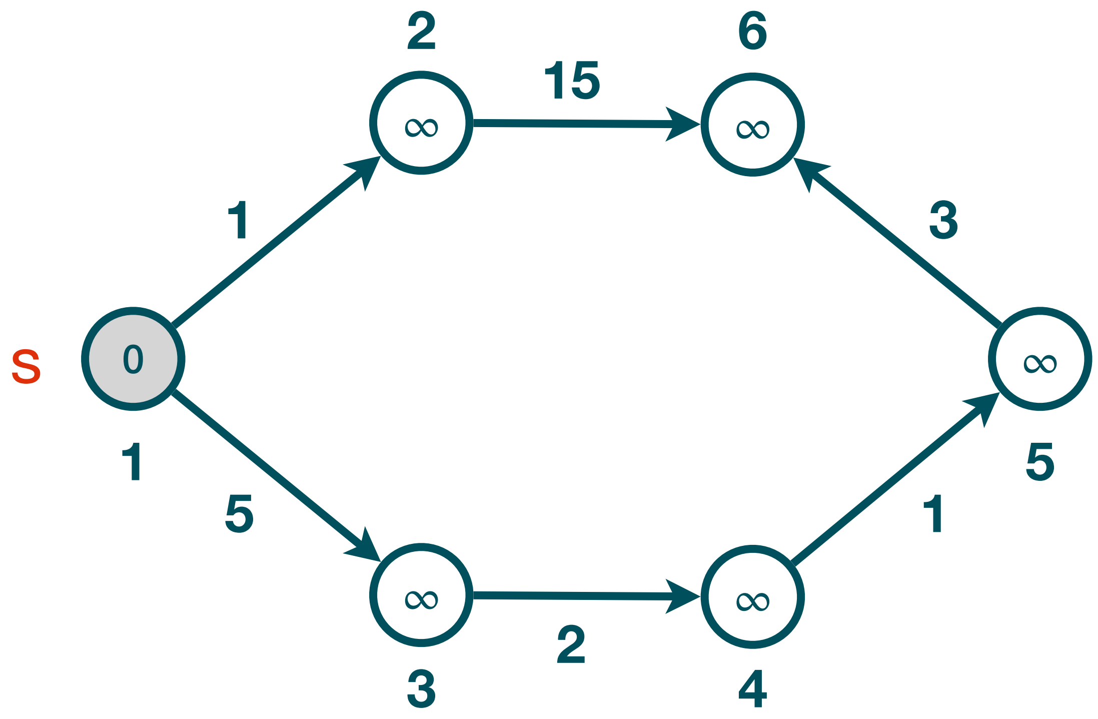
```
DIJKSTRA($G$,$w$,$s$)
    INITIALISE($G$,$s$);
    S ← ∅;
    Q ← V;
    while Q ≠ ∅
        $u$ ← EXTRACTMIN(Q);
        S ← SU{$u$};
        for each $v \in$ Adj[$u$]
            RELAX($u$,$v$,$w$);
```

# Dijkstra’s algorithm

At each step, one edge is relaxed. The vertices that are still to be finalised are maintained in a min-priority queue (many different implementations are possible). S is the set of finalised vertices.

S={}

$$\begin{array}{lcccccc} & \boxed{1} & 2 & 3 & 4 & 5 & 6 \\ Q=\{ & \boxed{0}, & \infty, & \infty, & \infty, & \infty, & \infty\} \end{array}$$



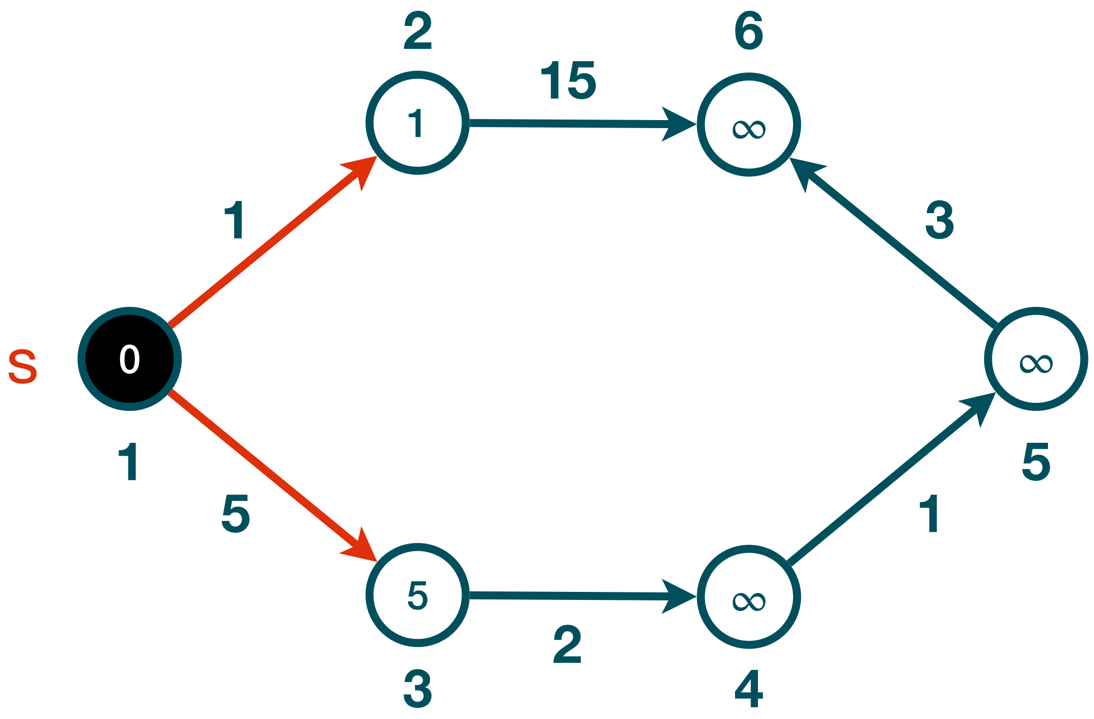
```
DIJKSTRA($G$,$w$,$s$)
    INITIALISE($G$,$s$);
    S $\leftarrow \varnothing$;
    Q $\leftarrow$ V;
    while Q $\neq \varnothing$
        $u \leftarrow$ EXTRACTMIN(Q);
        S $\leftarrow$ S∪{$u$};
        for each $v \in$ Adj[$u$]
            RELAX($u$,$v$,$w$);
```

# Dijkstra’s algorithm

At each step, one edge is relaxed. The vertices that are still to be finalised are maintained in a min-priority queue (many different implementations are possible). S is the set of finalised vertices.

$S=\{1\}$

$Q=\{\overset{2}{1},\overset{3}{5},\overset{4}{\infty},\overset{5}{\infty},\overset{6}{\infty}\}$



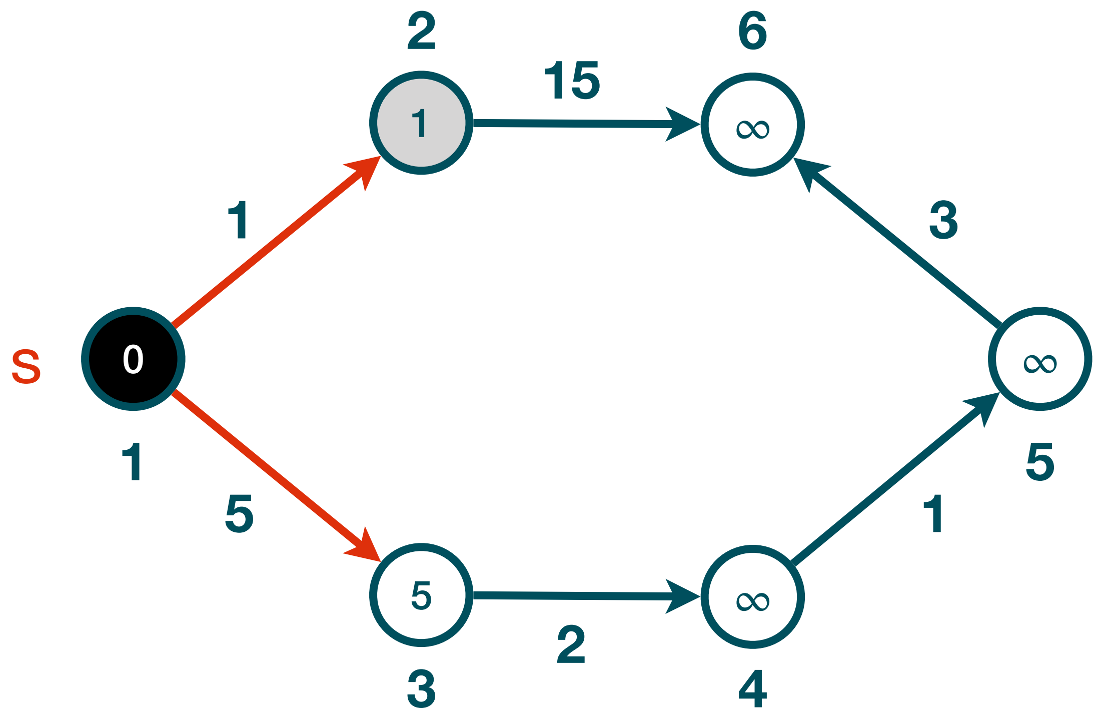
```
DIJKSTRA($G$,$w$,$s$)
    INITIALISE($G$,$s$);
    $S \leftarrow \emptyset$;
    $Q \leftarrow V$;
    while $Q \neq \emptyset$
        $u \leftarrow$ EXTRACTMIN(Q);
        $S \leftarrow S \cup \{u\}$;
        for each $v \in \mathrm{Adj}[u]$
            RELAX($u$,$v$,$w$);
```

# Dijkstra’s algorithm

At each step, one edge is relaxed. The vertices that are still to be finalised are maintained in a min-priority queue (many different implementations are possible). S is the set of finalised vertices.

S={1}

$$Q=\left\{\begin{array}{ccccc} \boxed{\begin{array}{c}2\\1\end{array}} & \begin{array}{c}3\\5\end{array} & \begin{array}{c}4\\\infty\end{array} & \begin{array}{c}5\\\infty\end{array} & \begin{array}{c}6\\\infty\end{array}\end{array}\right\}$$



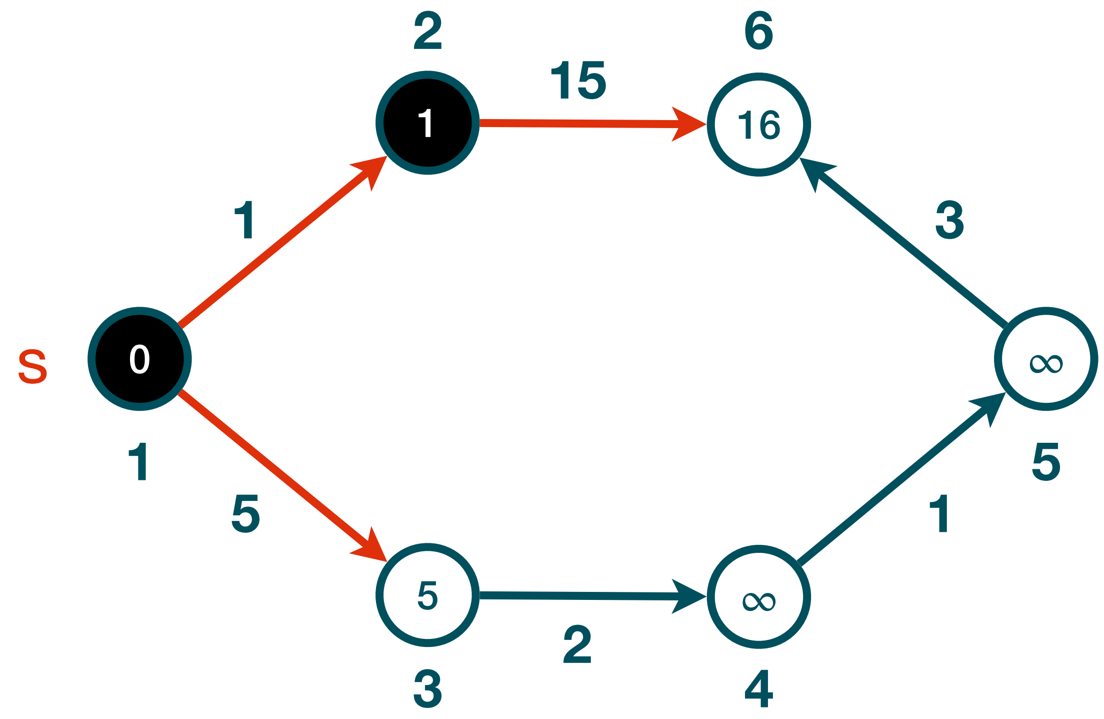
```
DIJKSTRA($G$,$w$,$s$)
    INITIALISE($G$,$s$);
    S $\leftarrow \emptyset$;
    Q $\leftarrow$ V;
    **while** Q $\neq \emptyset$
        $u \leftarrow$ EXTRACTMIN(Q);
        S $\leftarrow$ S$\cup\{u\}$;
        **for each** $v \in$ Adj[$u$]
            RELAX($u$,$v$,$w$);
```

# Dijkstra’s algorithm

At each step, one edge is relaxed. The vertices that are still to be finalised are maintained in a min-priority queue (many different implementations are possible). S is the set of finalised vertices.

$S=\{1,2\}$

$Q=\{\overset{3}{5},\overset{4}{\infty},\overset{5}{\infty},\overset{6}{16}\}$



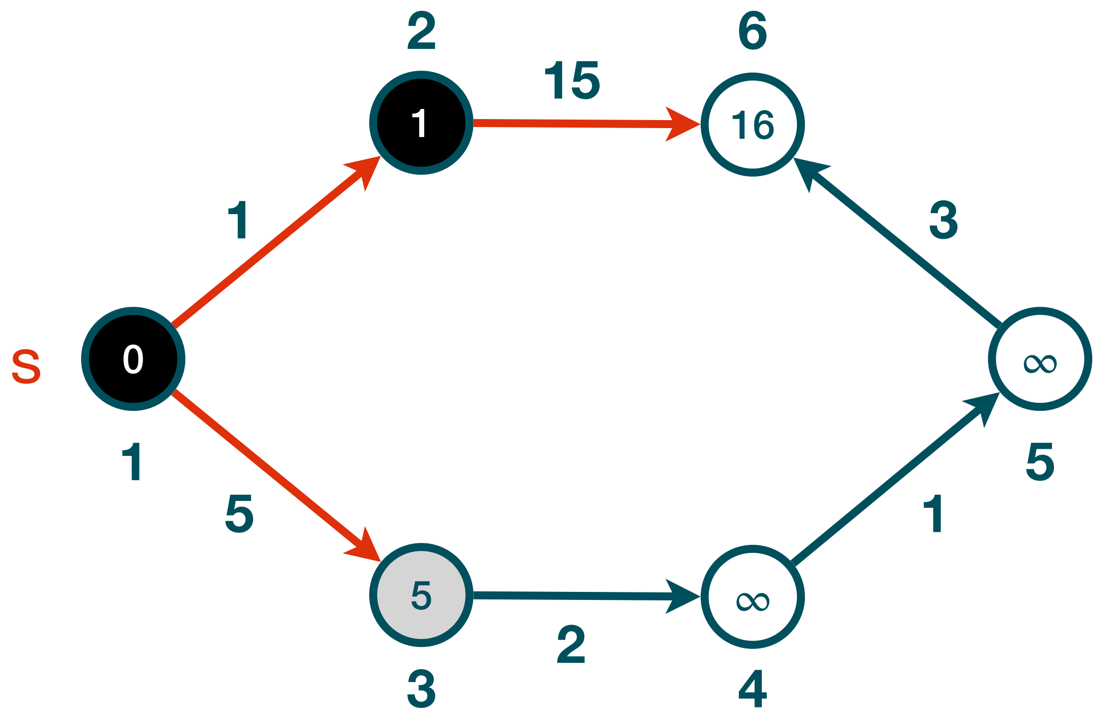
```
DIJKSTRA($G,w,s$)
    INITIALISE($G,s$);
    $S \leftarrow \emptyset$;
    $Q \leftarrow V$;
    while $Q \neq \emptyset$
        $u \leftarrow$ EXTRACTMIN($Q$);
        $S \leftarrow S \cup \{u\}$;
        for each $v \in \mathrm{Adj}[u]$
            RELAX($u,v,w$);
```

# Dijkstra’s algorithm

At each step, one edge is relaxed. The vertices that are still to be finalised are maintained in a min-priority queue (many different implementations are possible). S is the set of finalised vertices.

$S=\{1,2\}$

$Q=\{\boxed{\overset{3}{5}},\overset{4}{\infty},\overset{5}{\infty},\overset{6}{16}\}$



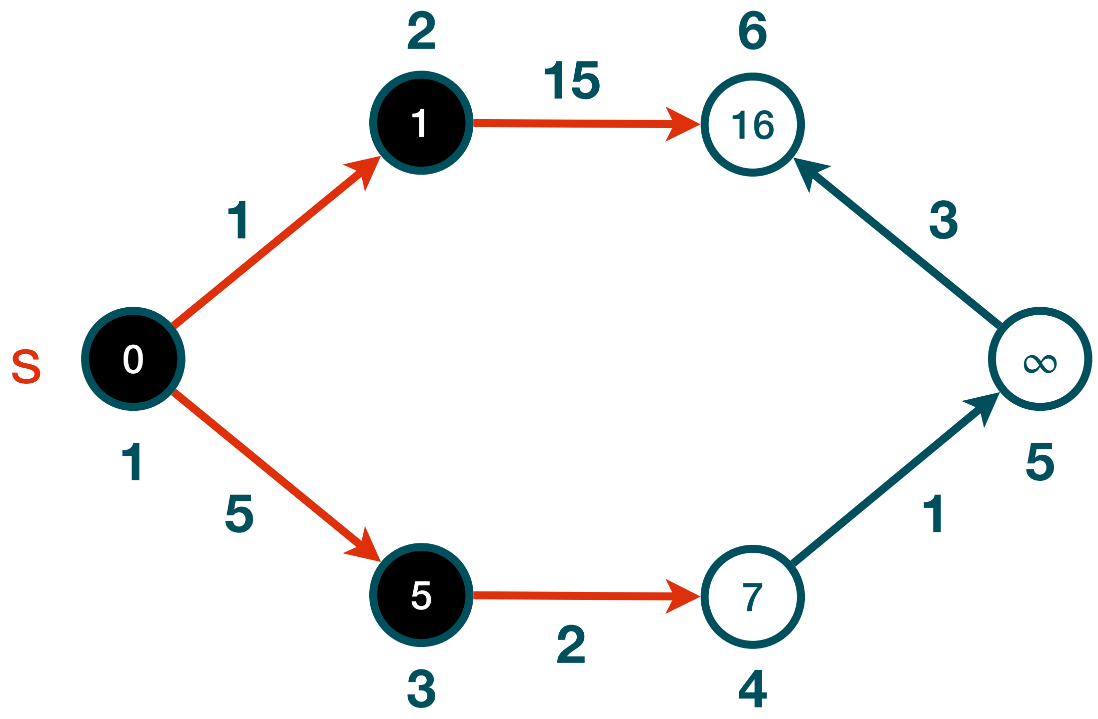
```
DIJKSTRA($G,w,s$)
    INITIALISE($G,s$);
    S $\leftarrow \varnothing$;
    Q $\leftarrow$ V;
    while Q $\neq \varnothing$
        $u \leftarrow$ EXTRACTMIN(Q);
        S $\leftarrow$ SU{$u$};
        for each $v \in \mathrm{Adj}[u]$
            RELAX($u,v,w$);
```

# Dijkstra’s algorithm

At each step, one edge is relaxed. The vertices that are still to be finalised are maintained in a min-priority queue (many different implementations are possible). S is the set of finalised vertices.

S={1,2,3}

$Q=\{\overset{4}{7},\overset{5}{\infty},\overset{6}{16}\}$



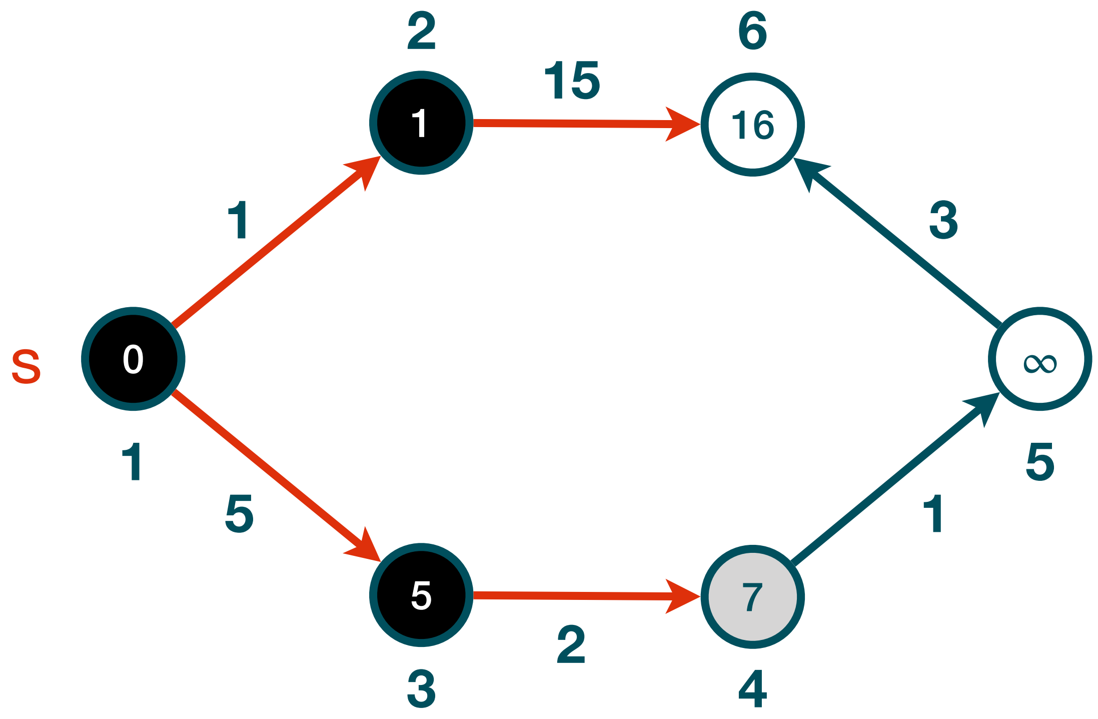
```
DIJKSTRA($G$,$w$,$s$)
    INITIALISE($G$,$s$);
    $S \leftarrow \varnothing$;
    $Q \leftarrow V$;
    while $Q \neq \varnothing$
        $u \leftarrow$ EXTRACTMIN(Q);
        $S \leftarrow S \cup \{u\}$;
        for each $v \in \mathrm{Adj}[u]$
            RELAX($u$,$v$,$w$);
```

# Dijkstra’s algorithm

At each step, one edge is relaxed. The vertices that are still to be finalised are maintained in a min-priority queue (many different implementations are possible). S is the set of finalised vertices.

S={1,2,3}

$Q=\{\boxed{\overset{4}{7}},\overset{5}{\infty},\overset{6}{16}\}$



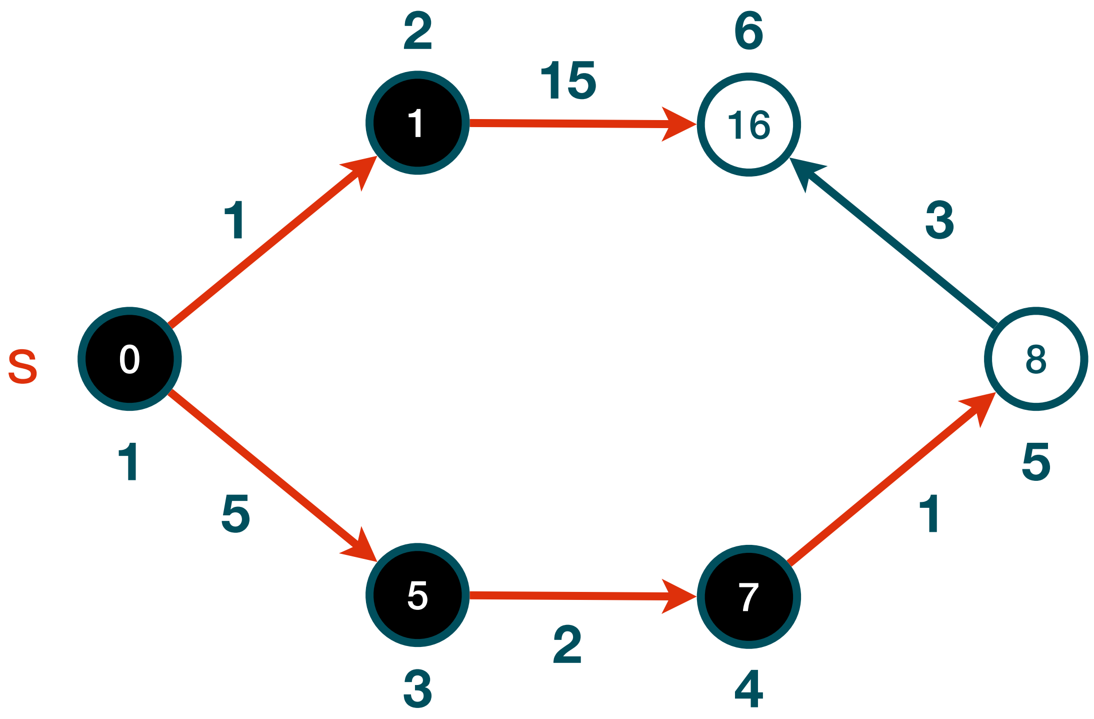
```
DIJKSTRA($G$,$w$,$s$)
    INITIALISE($G$,$s$);
    S ← ∅;
    Q ← V;
    while Q ≠ ∅
        $u$ ← EXTRACTMIN(Q);
        S ← SU{$u$};
        for each $v \in$ Adj[$u$]
            RELAX($u$,$v$,$w$);
```

# Dijkstra’s algorithm

At each step, one edge is relaxed. The vertices that are still to be finalised are maintained in a min-priority queue (many different implementations are possible). S is the set of finalised vertices.

$S=\{1,2,3,4\}$

$Q=\{\overset{5}{8},\overset{6}{16}\}$



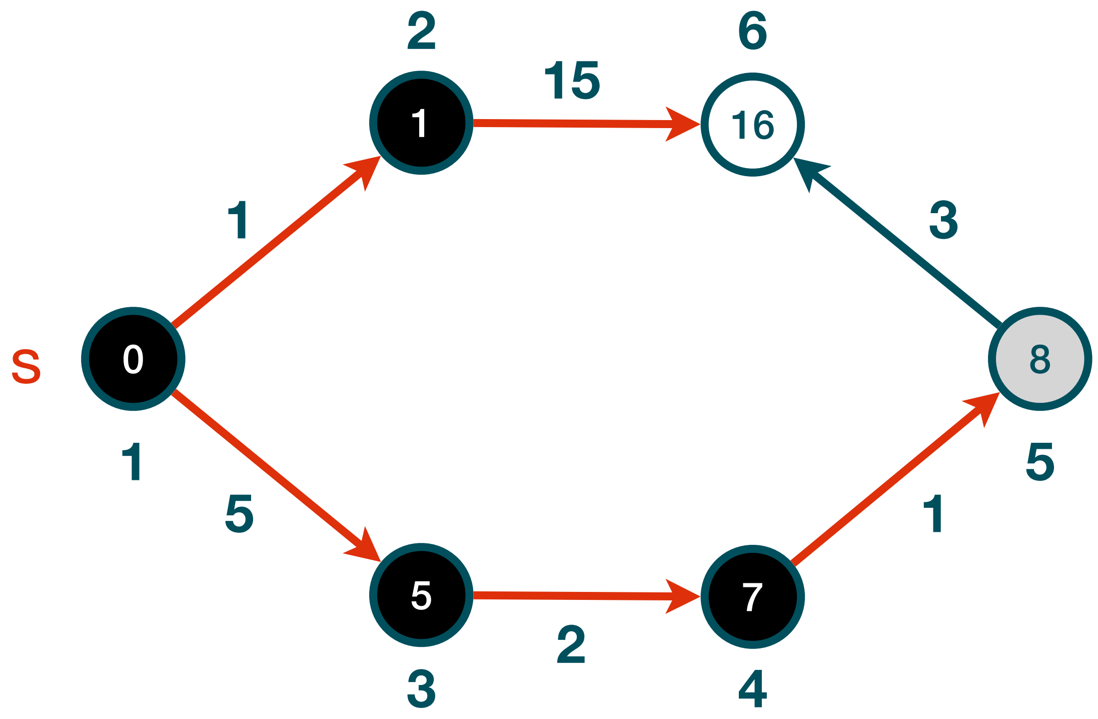
```
DIJKSTRA($G,w,s$)
    INITIALISE($G,s$);
    $S \leftarrow \emptyset$;
    $Q \leftarrow V$;
    while $Q \neq \emptyset$
        $u \leftarrow$ EXTRACTMIN(Q);
        $S \leftarrow S \cup \{u\}$;
        for each $v \in \mathrm{Adj}[u]$
            RELAX($u,v,w$);
```

# Dijkstra’s algorithm

At each step, one edge is relaxed. The vertices that are still to be finalised are maintained in a min-priority queue (many different implementations are possible). S is the set of finalised vertices.

$S=\{1,2,3,4\}$

$Q=\{\overset{5}{\boxed{8}},\overset{6}{16}\}$



```
DIJKSTRA($G$,$w$,$s$)
    INITIALISE($G$,$s$);
    $S \leftarrow \varnothing$;
    $Q \leftarrow V$;
    while $Q \neq \varnothing$
        $u \leftarrow$ EXTRACTMIN(Q);
        $S \leftarrow S \cup \{u\}$;
        for each $v \in \mathrm{Adj}[u]$
            RELAX($u$,$v$,$w$);
```

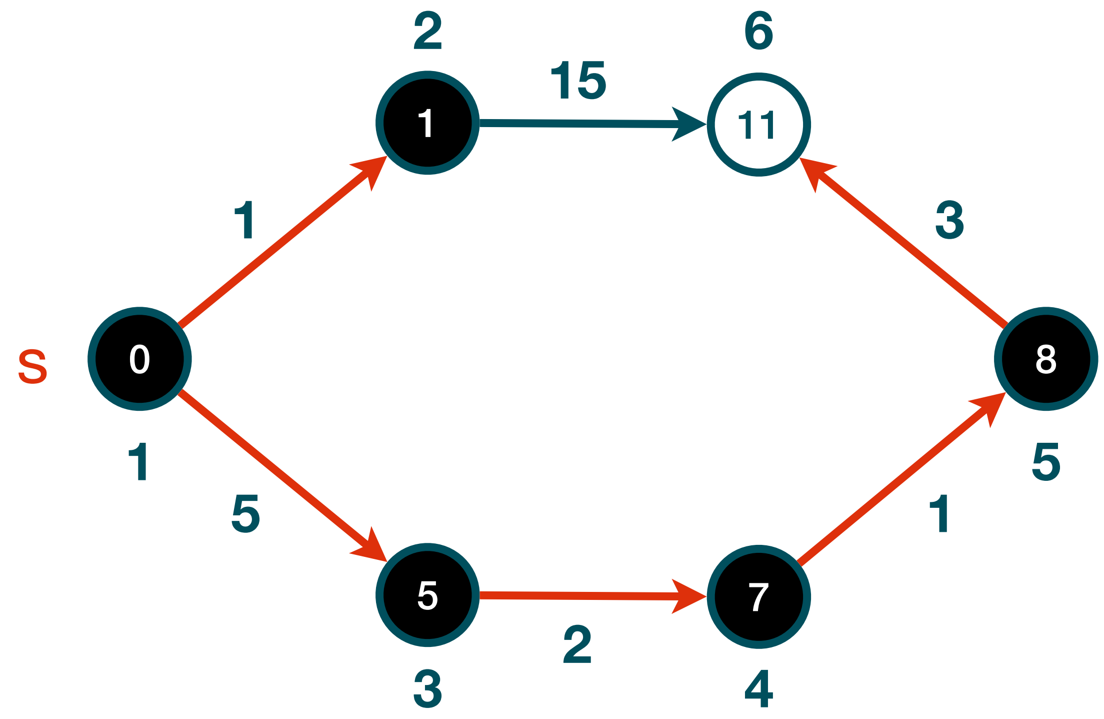
# Dijkstra's algorithm

At each step, one edge is relaxed. The vertices that are still to be finalised are maintained in a min-priority queue (many different implementations are possible). S is the set of finalised vertices.

S={1,2,3,4,5}

$Q=\{\overset{6}{11}\}$

RELAX makes 6.*d* change from 16 to 11!



```
DIJKSTRA($G,w,s$)
    INITIALISE($G,s$);
    $S \leftarrow \varnothing$;
    $Q \leftarrow V$;
    while $Q \neq \varnothing$
        $u \leftarrow$ EXTRACTMIN(Q);
        $S \leftarrow S \cup \{u\}$;
        for each $v \in \mathrm{Adj}[u]$
            RELAX($u,v,w$);
```

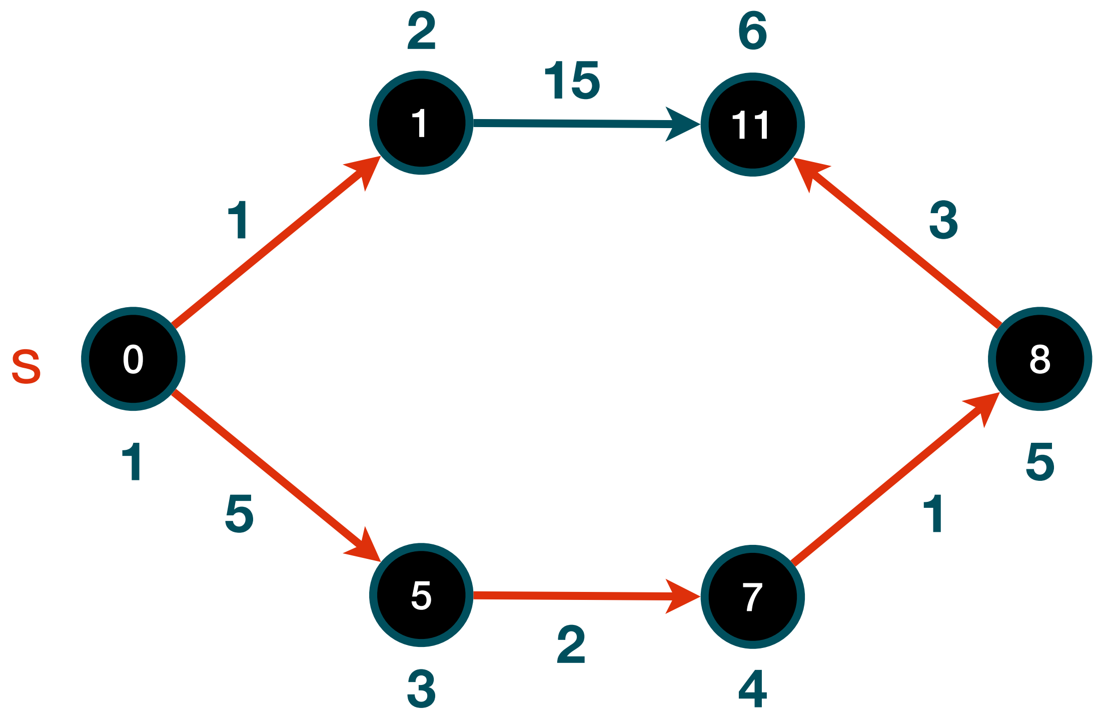
# Dijkstra's algorithm

At each step, one edge is relaxed. The vertices that are still to be finalised are maintained in a min-priority queue (many different implementations are possible). S is the set of finalised vertices.

S={1,2,3,4,5,6}

Q={}

RELAX makes 6.*d* change from 16 to 11!



```
DIJKSTRA($G$,$w$,$s$)
    INITIALISE($G$,$s$);
    $S \leftarrow \varnothing$;
    $Q \leftarrow V$;
    while $Q \neq \varnothing$
        $u \leftarrow$ EXTRACTMIN(Q);
        $S \leftarrow S \cup \{u\}$;
        for each $v \in \mathrm{Adj}[u]$
            RELAX($u$,$v$,$w$);
```

# Dijkstra’s algorithm: complexity

Time complexity: $\Theta(|V|) + T_B(|V|) + |V| \cdot T_E(|V|) + |E| \cdot T_R(|V|)$

| Queue data structure | $T_B(n)$ | $T_E(n)$ | $T_R(n)$ | $T_D(G)$ |
|---|---|---|---|---|
| Arrays | $\Theta(n)$ | $\Theta(n)$ | $\Theta(1)$ | $\Theta(\|E\|+\|V\|^2)$ |
| Binary Heaps | $\Theta(n)$ | $O(\log n)$ | $O(\log n)$ | $O((\|E\|+\|V\|)\log \|V\|)$ |
| Fibonacci Heaps | $\Theta(n)$ | $O(\log n)$ | $\Theta(1)$ | $O(\|E\|+\|V\|\log \|V\|)$ |

# Exercises

**Cormen 24.3-6:** We are given a directed graph G which each edge (u,v) has an associated value r(u,v), which is a real number in the range [0,1] that represents the reliability of a communication channel from vertex u to vertex v. We interpret r(u,v) as the probability that the channel from u to v will not fail, and we assume that these probabilities are independent. Give an efficient algorithm to find the most reliable path between two given vertices. *(Hint: either modify Dijkstra or transform the weights…)*