

# Paths in Graphs: Dijkstra's Algorithm

Michael Levin

Department of Computer Science and Engineering  
University of California, San Diego

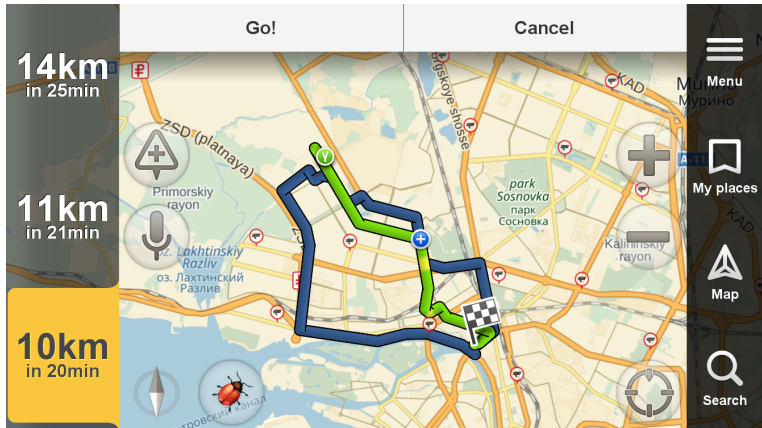
**Graph Algorithms**  
**Algorithms and Data Structures**

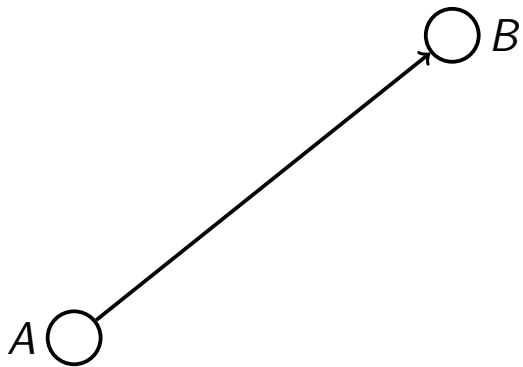
# Outline

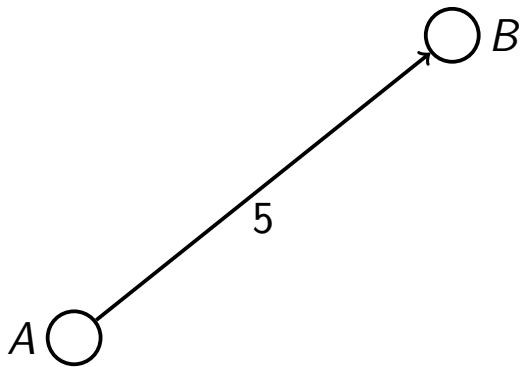
- 1 Fastest Route
- 2 Naive Algorithm
- 3 Dijkstra's Algorithm
- 4 Dijkstra Example
- 5 Implementation
- 6 Proof of Correctness
- 7 Analysis

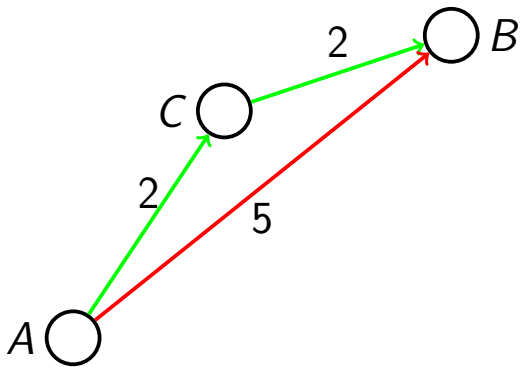
# Fastest Route

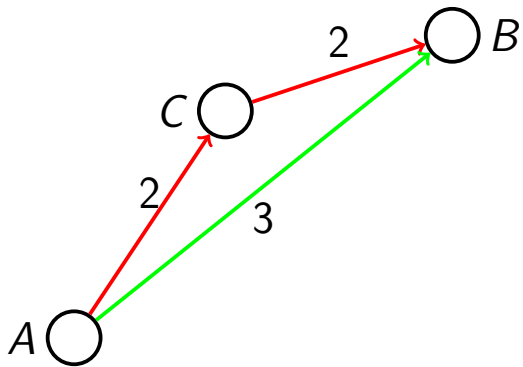
What is the fastest route to get home from work?





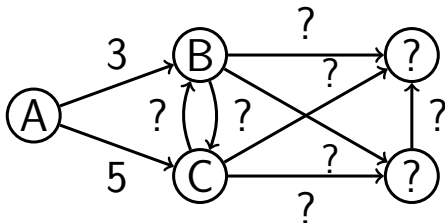






# Intuition

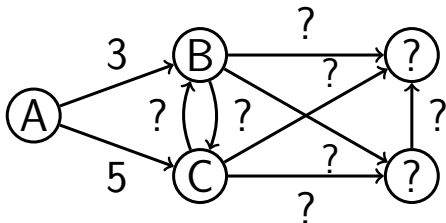
- Assume that we stay at  $A$  and observe two outgoing edges:





# Intuition

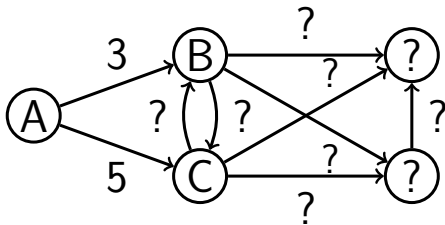
- Assume that we stay at  $A$  and observe two outgoing edges:



- Can we be sure that the distance from  $A$  to  $C$  is 5?

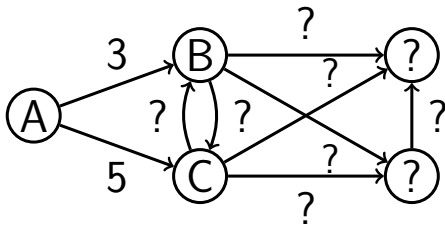
# Intuition

- Can we be sure that the distance from  $A$  to  $C$  is 5?



# Intuition

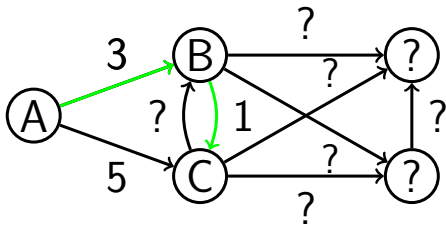
- Can we be sure that the distance from  $A$  to  $C$  is 5?



- No, because the weight of the edge  $(B, C)$  might be equal to, say, 1.

# Intuition

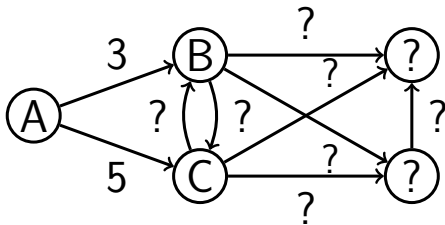
- Can we be sure that the distance from  $A$  to  $C$  is 5?



- No, because the weight of the edge  $(B, C)$  might be equal to, say, 1.

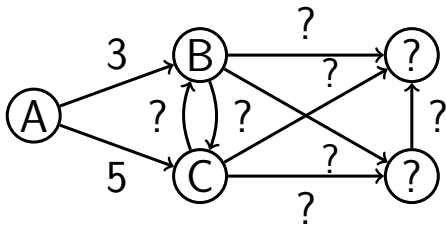
# Intuition

- Can we be sure that the distance from  $A$  to  $B$  is 3?



# Intuition

- Can we be sure that the distance from  $A$  to  $B$  is 3?



- Yes, because there are no negative weight edges.

# Outline

- 1 Fastest Route
- 2 Naive Algorithm
- 3 Dijkstra's Algorithm
- 4 Dijkstra Example
- 5 Implementation
- 6 Proof of Correctness
- 7 Analysis

# Optimal substructure

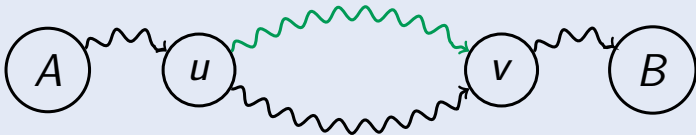
## Observation

Any subpath of an optimal path is also optimal.



## Proof

Consider an optimal path from  $A$  to  $B$  and two vertices  $u$  and  $v$  on this path. If there were a shorter path from  $u$  to  $v$  we would get a shorter path from  $A$  to  $B$ .



# Corollary

If  $A \rightarrow \dots \rightarrow u \rightarrow B$  is a shortest path from  $A$  to  $B$ , then

$$d(A, B) = d(A, u) + w(u, B)$$

# Edge relaxation

- $\text{dist}[v]$  will be an upper bound on the actual distance from  $A$  to  $v$ .

# Edge relaxation

- $\text{dist}[v]$  will be an upper bound on the actual distance from  $A$  to  $v$ .
- The edge relaxation procedure for an edge  $(u, v)$  just checks whether going from  $A$  to  $v$  through  $u$  improves the current value of  $\text{dist}[v]$ .

Relax( $(u, v) \in E$ )

```
if  $dist[v] > dist[u] + w(u, v)$ :  
     $dist[v] \leftarrow dist[u] + w(u, v)$   
     $prev[v] \leftarrow u$ 
```

Relax( $(u, v) \in E$ )

```
if  $dist[v] > dist[u] + w(u, v)$ :  
     $dist[v] \leftarrow dist[u] + w(u, v)$   
     $prev[v] \leftarrow u$ 
```

$\text{Relax}((u, v) \in E)$

```
if  $\text{dist}[v] > \text{dist}[u] + w(u, v)$ :  
     $\text{dist}[v] \leftarrow \text{dist}[u] + w(u, v)$   
     $\text{prev}[v] \leftarrow u$ 
```

$\text{Relax}((u, v) \in E)$

```
if  $\text{dist}[v] > \text{dist}[u] + w(u, v)$ :  
     $\text{dist}[v] \leftarrow \text{dist}[u] + w(u, v)$   
     $\text{prev}[v] \leftarrow u$ 
```



# Naive approach

Naive( $G, A$ )

for all  $u \in V$ :

$dist[u] \leftarrow \infty$

$prev[u] \leftarrow nil$

$dist[A] \leftarrow 0$

do:

relax all the edges

while at least one  $dist$  changes

# Naive approach

Naive( $G, A$ )

for all  $u \in V$ :

$dist[u] \leftarrow \infty$

$prev[u] \leftarrow nil$

$dist[A] \leftarrow 0$

do:

relax all the edges

while at least one  $dist$  changes

# Naive approach

Naive( $G, A$ )

for all  $u \in V$ :

$dist[u] \leftarrow \infty$

$prev[u] \leftarrow nil$

$dist[A] \leftarrow 0$

do:

relax all the edges

while at least one  $dist$  changes

# Naive approach

Naive( $G, A$ )

for all  $u \in V$ :

$dist[u] \leftarrow \infty$

$prev[u] \leftarrow nil$

$dist[A] \leftarrow 0$

do:

relax all the edges

while at least one  $dist$  changes

# Naive approach

Naive( $G, A$ )

for all  $u \in V$ :

$dist[u] \leftarrow \infty$

$prev[u] \leftarrow nil$

$dist[A] \leftarrow 0$

do:

relax all the edges

while at least one  $dist$  changes

# Naive approach

Naive( $G, A$ )

for all  $u \in V$ :

$dist[u] \leftarrow \infty$

$prev[u] \leftarrow nil$

$dist[A] \leftarrow 0$

*do:*

relax all the edges

*while* at least one  $dist$  changes

# Naive approach

Naive( $G, A$ )

for all  $u \in V$ :

$dist[u] \leftarrow \infty$

$prev[u] \leftarrow nil$

$dist[A] \leftarrow 0$

do:

relax all the edges

while at least one  $dist$  changes

# Naive approach

Naive( $G, A$ )

for all  $u \in V$ :

$dist[u] \leftarrow \infty$

$prev[u] \leftarrow nil$

$dist[A] \leftarrow 0$

do:

relax all the edges

while at least one  $dist$  changes



# Correct distances

## Lemma

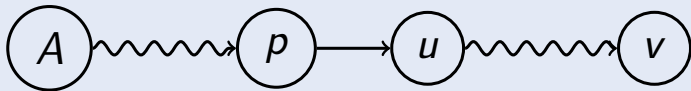
After the call to Naive algorithm all the distances are set correctly.

## Proof

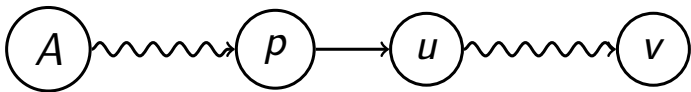
- Assume, for the sake of contradiction, that no edge can be relaxed and there is a vertex  $v$  such that  $\text{dist}[v] > d(A, v)$ .

## Proof

- Assume, for the sake of contradiction, that no edge can be relaxed and there is a vertex  $v$  such that  $\text{dist}[v] > d(A, v)$ .
- Consider a shortest path from  $A$  to  $v$  and let  $u$  be the first vertex on this path with the same property. Let  $p$  be the vertex right before  $u$ .

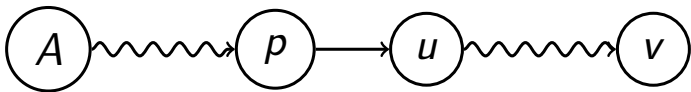


## Proof (continued)



- Then  $d(A, p) = \text{dist}[p]$  and hence  
$$d(A, u) = d(A, p) + w(p, u) = \text{dist}[p] + w(p, u)$$

## Proof (continued)

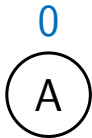


- Then  $d(A, p) = \text{dist}[p]$  and hence
$$d(A, u) = d(A, p) + w(p, u) = \text{dist}[p] + w(p, u)$$
- $\text{dist}[u] > d(A, u) = \text{dist}[p] + w(p, u) \Rightarrow$   
edge  $(p, u)$  can be relaxed —  
a contradiction. □

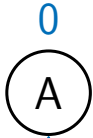
# Outline

- 1 Fastest Route
- 2 Naive Algorithm
- 3 Dijkstra's Algorithm**
- 4 Dijkstra Example
- 5 Implementation
- 6 Proof of Correctness
- 7 Analysis

# Intuition



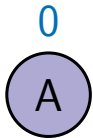
# Intuition



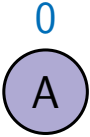
initially, we only know the distance to A



# Intuition

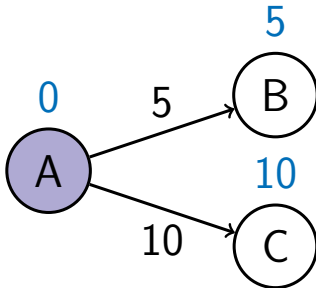


# Intuition



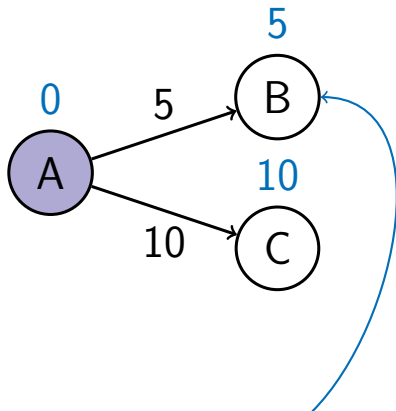
let's relax all the edges from A

# Intuition



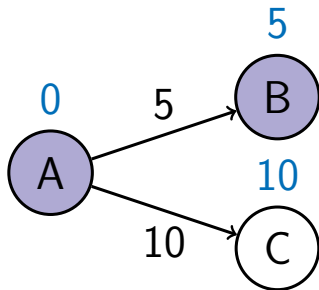
let's relax all the edges from A

# Intuition

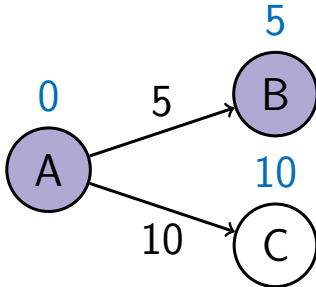


we now know the distance for B

# Intuition

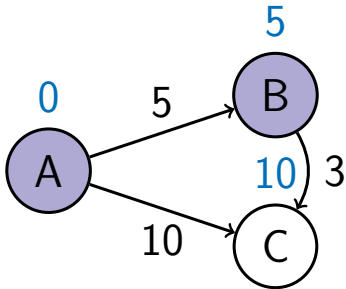


# Intuition



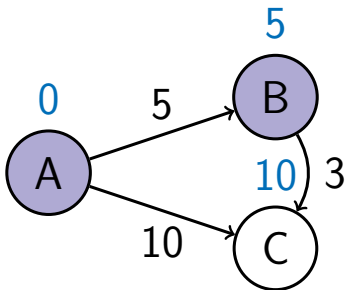
now, let's relax all the edges from B

# Intuition



now, let's relax all the edges from B

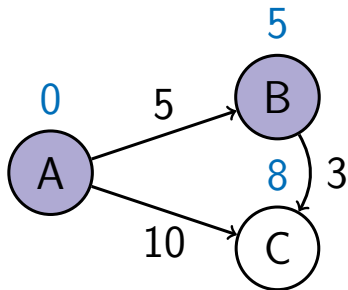
# Intuition



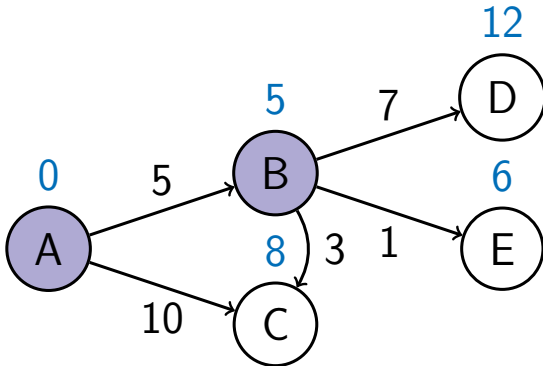
we discover an edge  $(B, C)$  of weight 3  
that updates  $\text{dist}[C]$



# Intuition

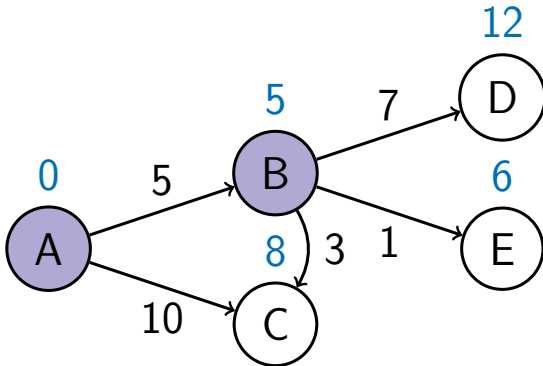


# Intuition



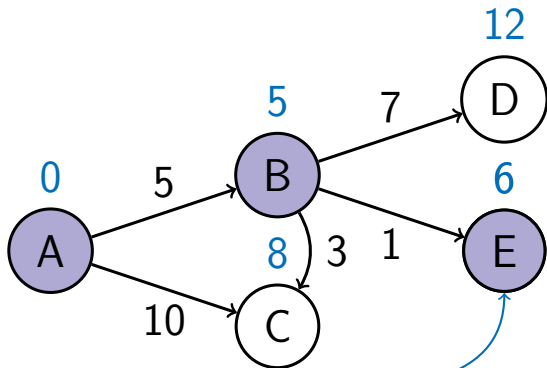
we also discover a few more outgoing edges

# Intuition



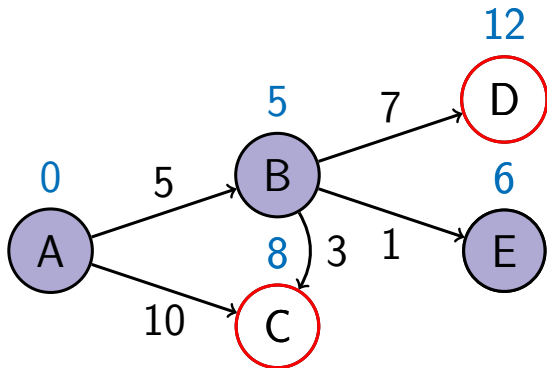
what is the next vertex for which we already know the correct distance?

# Intuition



it is  $E$

# Intuition



while for  $C$  and  $D$  it is possible that their dist values are larger than actual distances

# Main ideas of Dijkstra's Algorithm

- We maintain a set  $R$  of vertices for which `dist` is already set correctly ( “known region” ).

# Main ideas of Dijkstra's Algorithm

- We maintain a set  $R$  of vertices for which `dist` is already set correctly (“known region”).
- The first vertex added to  $R$  is  $A$ .

# Main ideas of Dijkstra's Algorithm

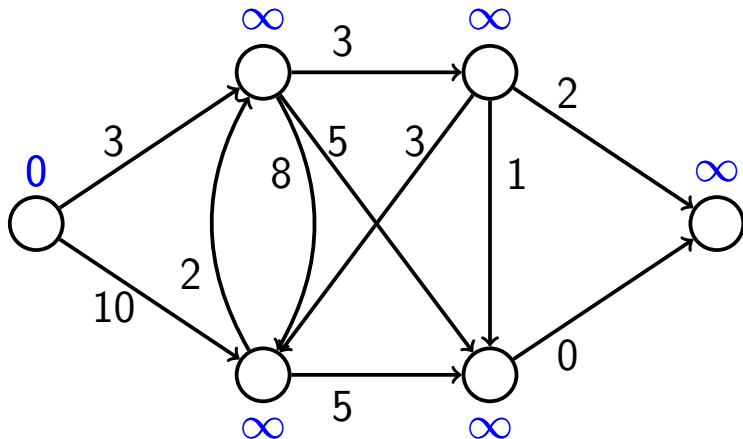
- We maintain a set  $R$  of vertices for which `dist` is already set correctly (“known region”).
- The first vertex added to  $R$  is  $A$ .
- On each iteration we take a vertex outside of  $R$  with the minimal `dist`-value, add it to  $R$ , and relax all its outgoing edges.



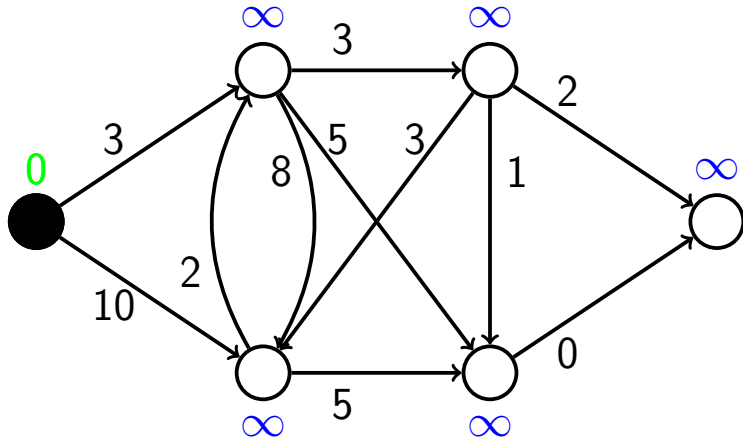
# Outline

- 1 Fastest Route
- 2 Naive Algorithm
- 3 Dijkstra's Algorithm
- 4 Dijkstra Example**
- 5 Implementation
- 6 Proof of Correctness
- 7 Analysis

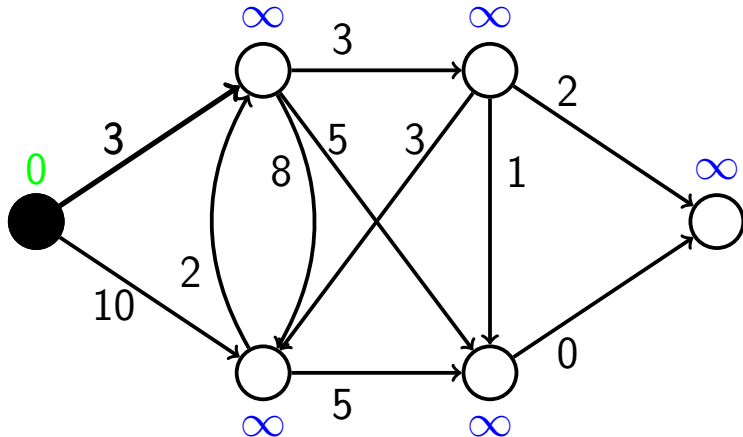
# Example



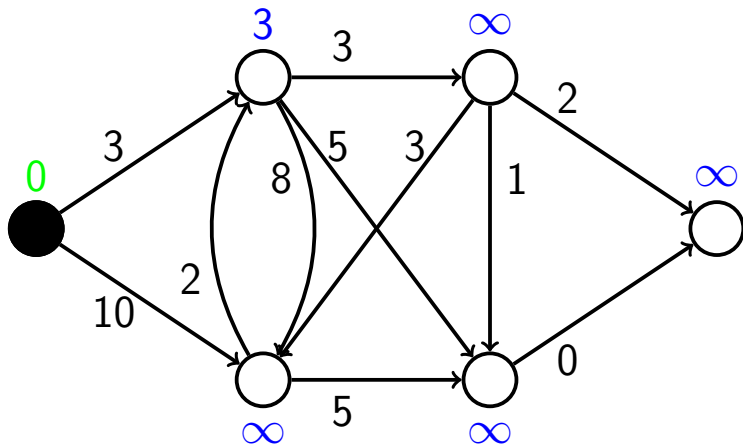
# Example



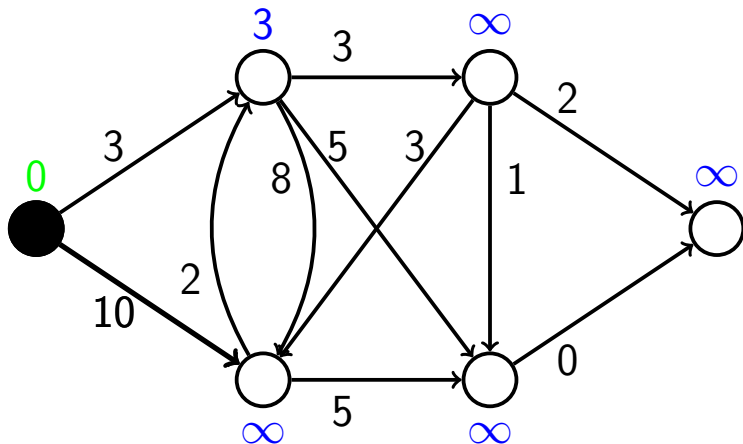
# Example



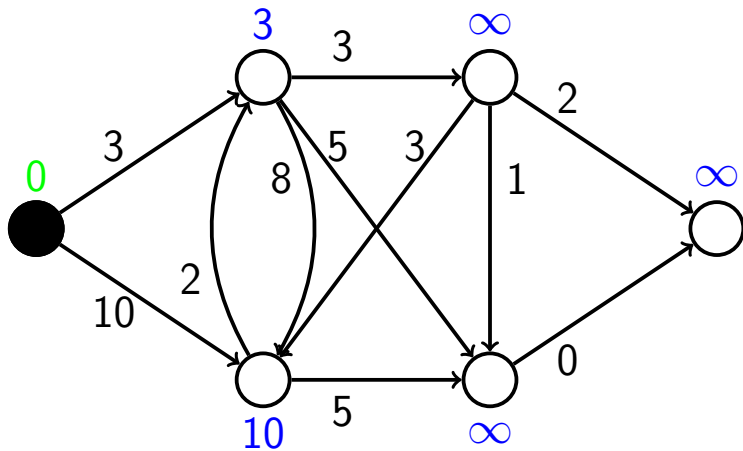
# Example



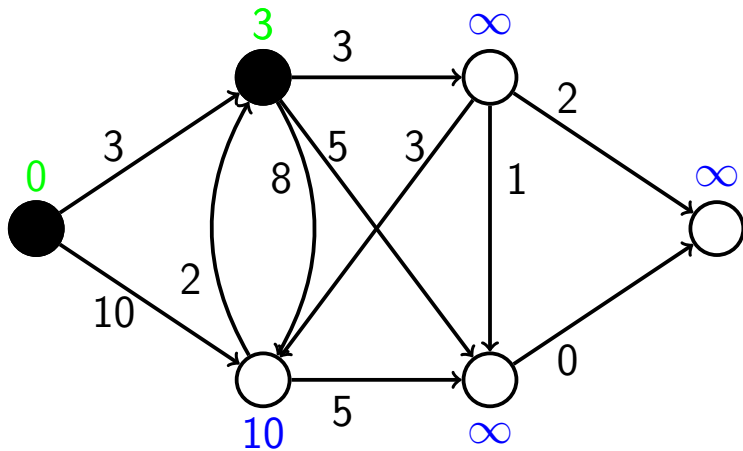
# Example



# Example

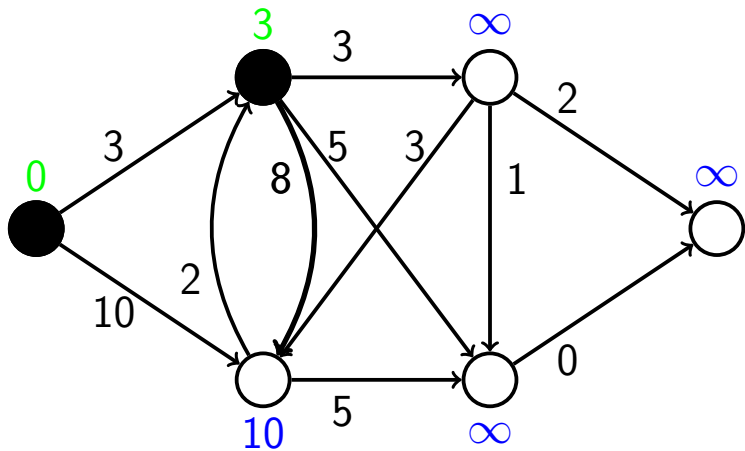


# Example

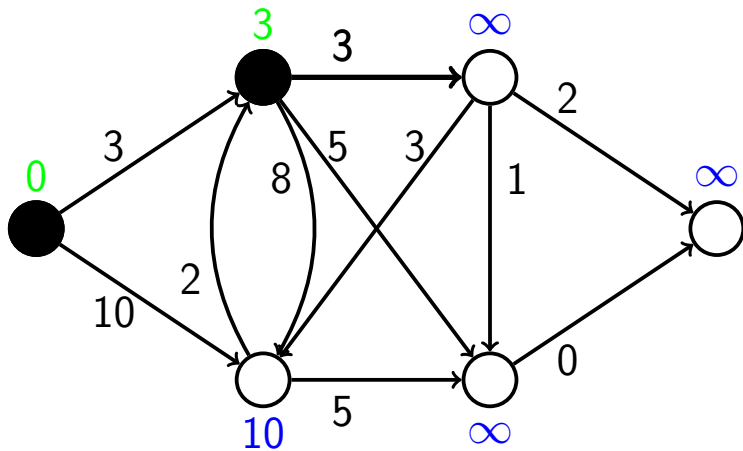




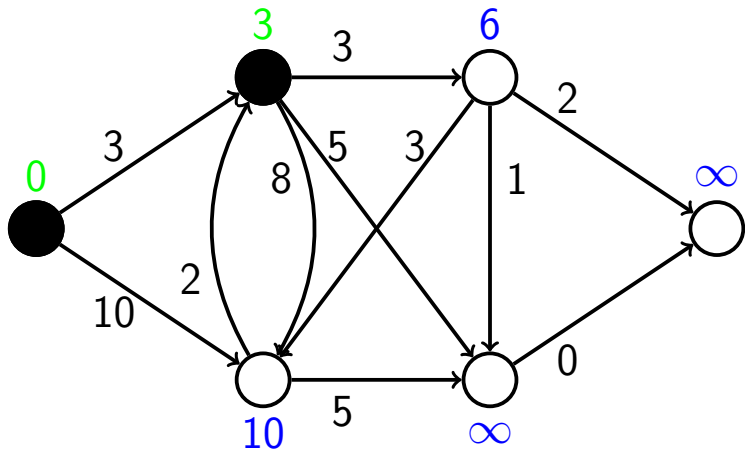
# Example



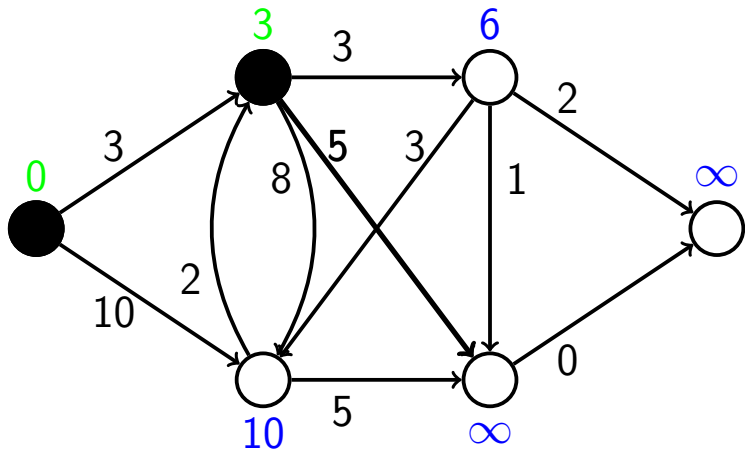
# Example



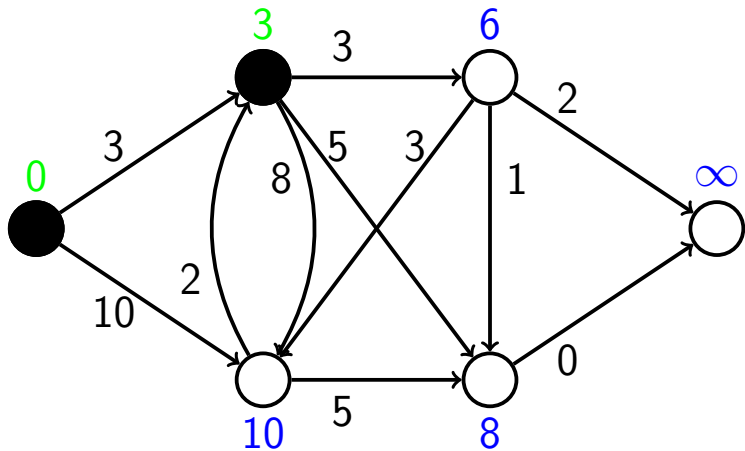
# Example



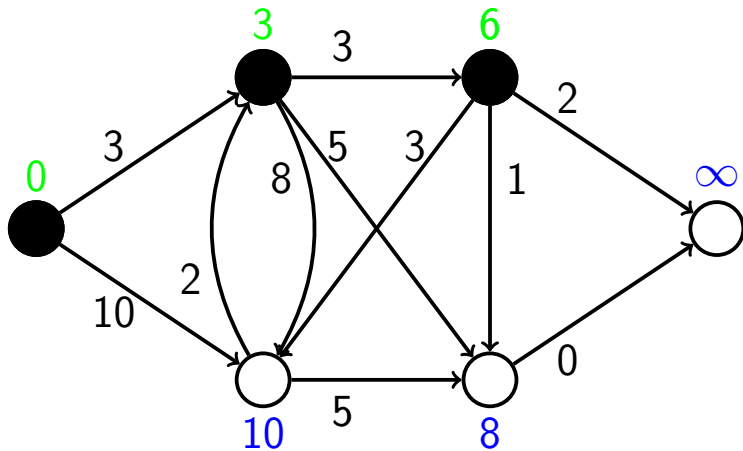
# Example



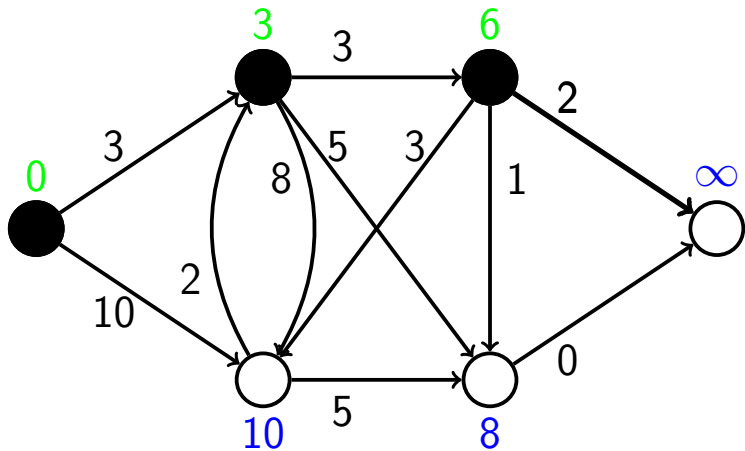
# Example



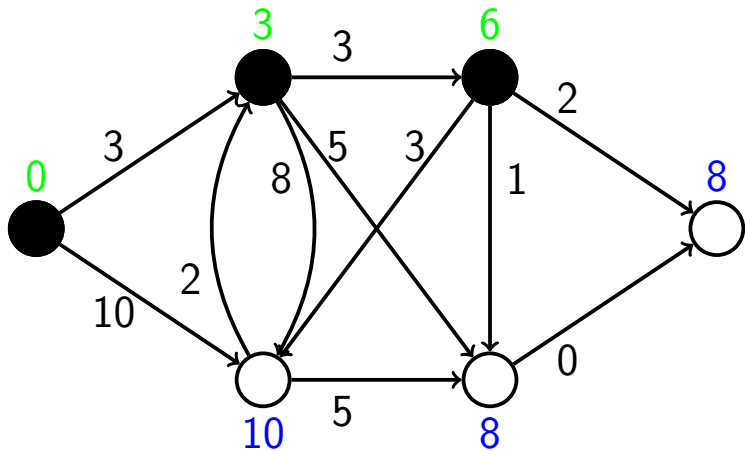
# Example



# Example

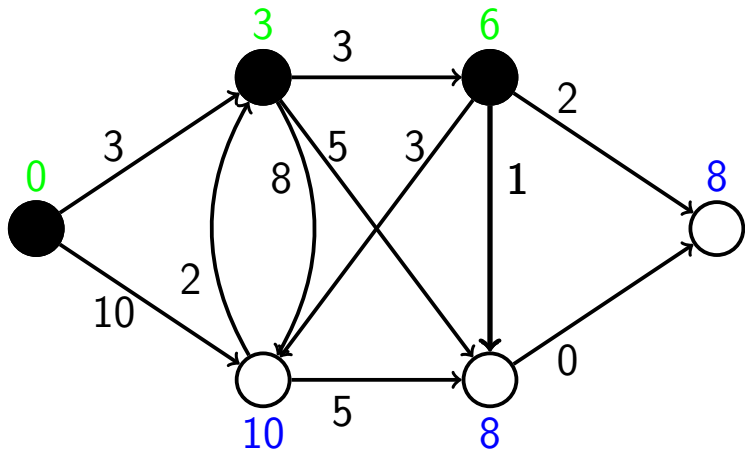


# Example

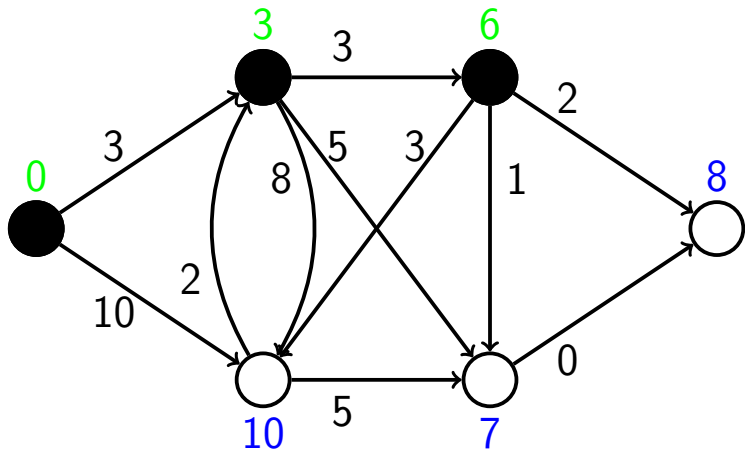




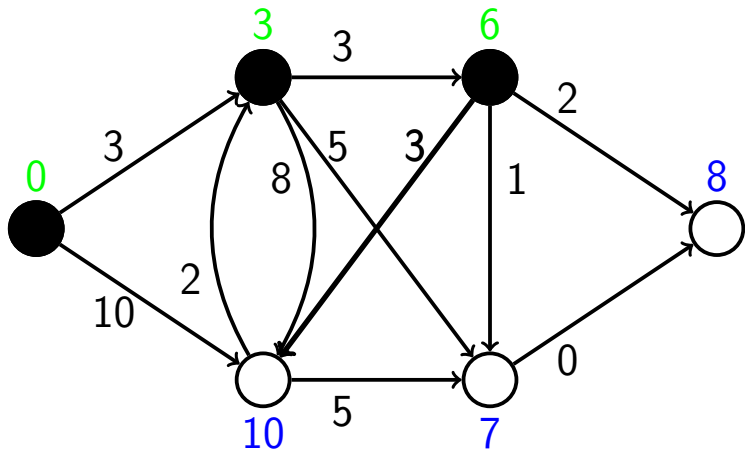
# Example



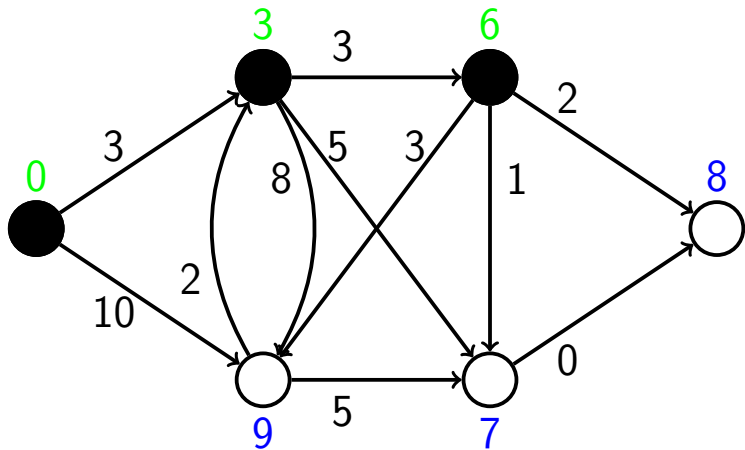
# Example



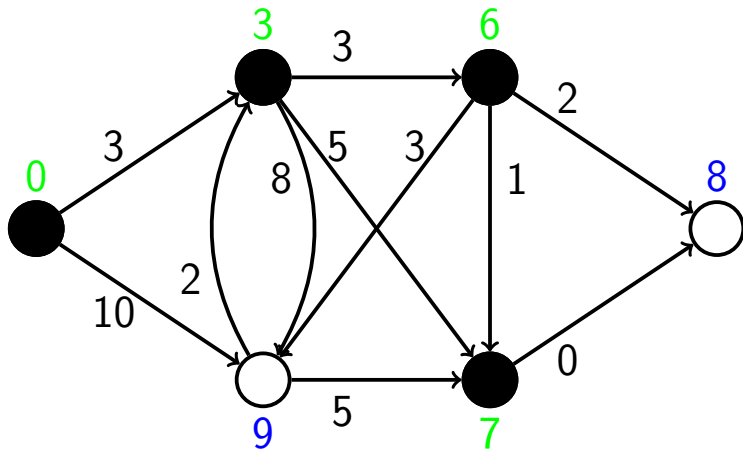
# Example



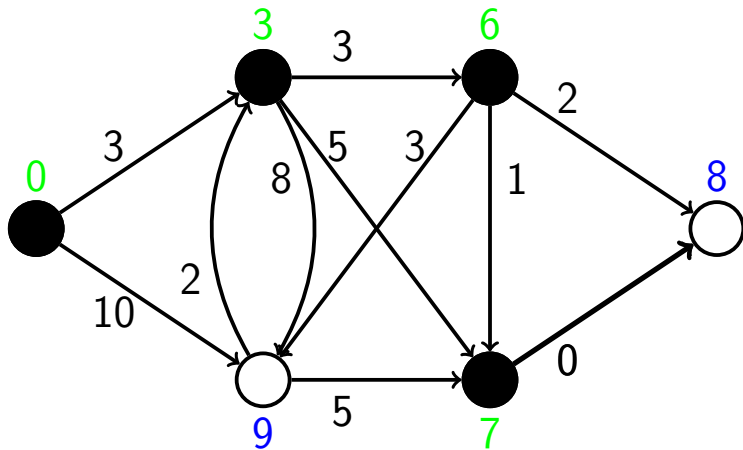
# Example



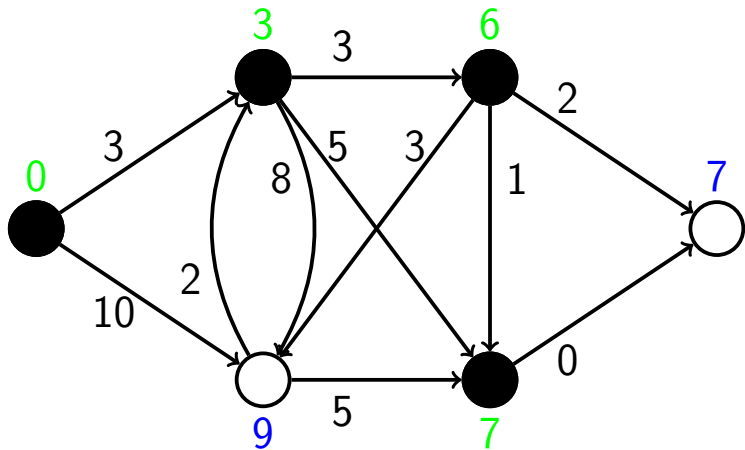
# Example



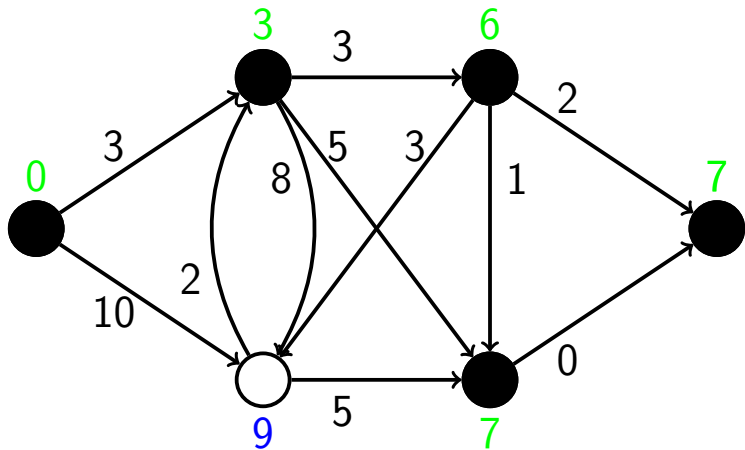
# Example



# Example

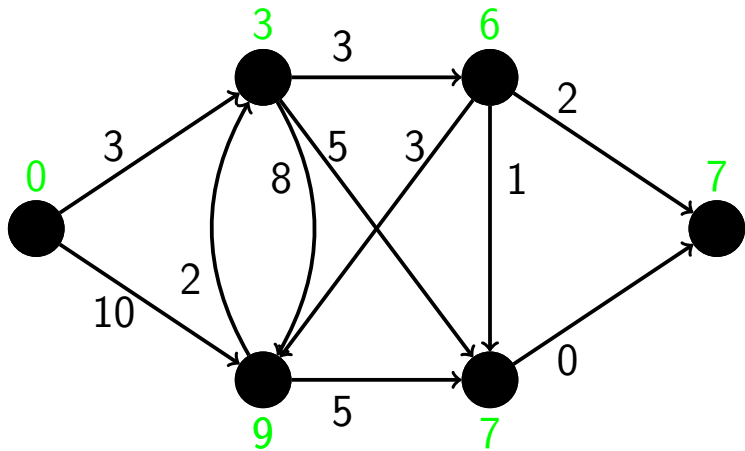


# Example

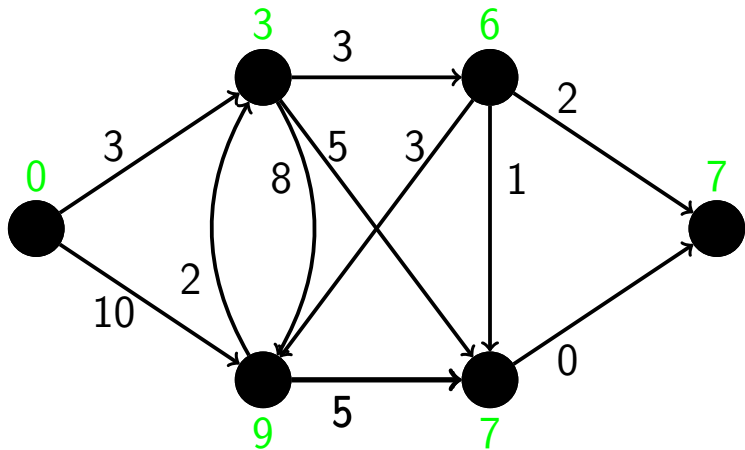




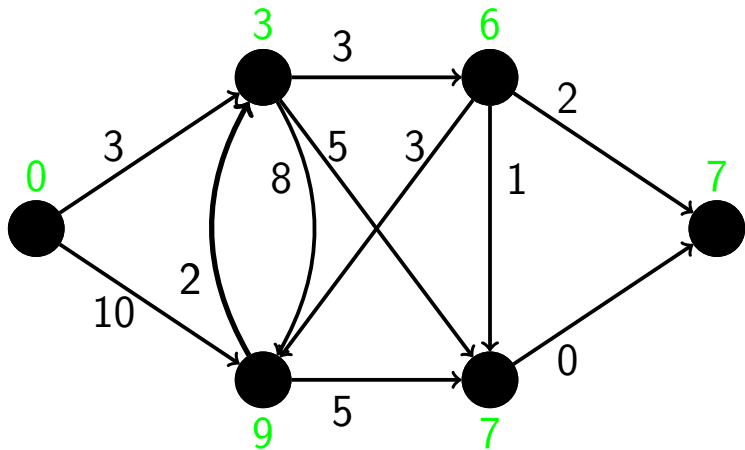
# Example



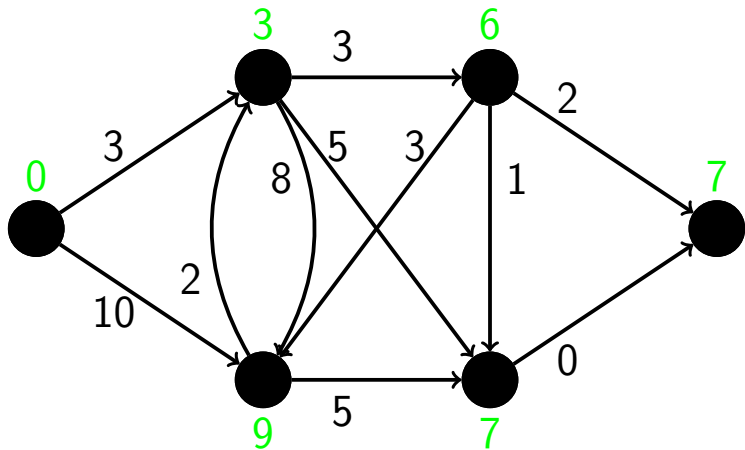
# Example



# Example



# Example



# Outline

- 1 Fastest Route
- 2 Naive Algorithm
- 3 Dijkstra's Algorithm
- 4 Dijkstra Example
- 5 Implementation**
- 6 Proof of Correctness
- 7 Analysis

# Pseudocode

## Dijkstra( $G, A$ )

```
for all  $u \in V$ :  
     $\text{dist}[u] \leftarrow \infty, \text{prev}[u] \leftarrow \text{nil}$   
 $\text{dist}[A] \leftarrow 0$   
 $H \leftarrow \text{MakeQueue}(V)$  {dist-values as keys}  
while  $H$  is not empty:  
     $u \leftarrow \text{ExtractMin}(H)$   
    for all  $(u, v) \in E$ :  
        if  $\text{dist}[v] > \text{dist}[u] + w(u, v)$ :  
             $\text{dist}[v] \leftarrow \text{dist}[u] + w(u, v)$   
             $\text{prev}[v] \leftarrow u$   
             $\text{ChangePriority}(H, v, \text{dist}[v])$ 
```

# Pseudocode

## Dijkstra( $G, A$ )

for all  $u \in V$ :

$\text{dist}[u] \leftarrow \infty, \text{prev}[u] \leftarrow \text{nil}$

$\text{dist}[A] \leftarrow 0$

$H \leftarrow \text{MakeQueue}(V)$  {dist-values as keys}

while  $H$  is not empty:

$u \leftarrow \text{ExtractMin}(H)$

    for all  $(u, v) \in E$ :

        if  $\text{dist}[v] > \text{dist}[u] + w(u, v)$ :

$\text{dist}[v] \leftarrow \text{dist}[u] + w(u, v)$

$\text{prev}[v] \leftarrow u$

$\text{ChangePriority}(H, v, \text{dist}[v])$

# Pseudocode

## Dijkstra( $G, A$ )

for all  $u \in V$ :

$\text{dist}[u] \leftarrow \infty, \text{prev}[u] \leftarrow \text{nil}$

$\text{dist}[A] \leftarrow 0$

$H \leftarrow \text{MakeQueue}(V)$  {dist-values as keys}

while  $H$  is not empty:

$u \leftarrow \text{ExtractMin}(H)$

    for all  $(u, v) \in E$ :

        if  $\text{dist}[v] > \text{dist}[u] + w(u, v)$ :

$\text{dist}[v] \leftarrow \text{dist}[u] + w(u, v)$

$\text{prev}[v] \leftarrow u$

$\text{ChangePriority}(H, v, \text{dist}[v])$



# Pseudocode

## Dijkstra( $G, A$ )

```
for all  $u \in V$ :  
     $\text{dist}[u] \leftarrow \infty, \text{prev}[u] \leftarrow \text{nil}$   
 $\text{dist}[A] \leftarrow 0$   
 $H \leftarrow \text{MakeQueue}(V)$  {dist-values as keys}  
while  $H$  is not empty:  
     $u \leftarrow \text{ExtractMin}(H)$   
    for all  $(u, v) \in E$ :  
        if  $\text{dist}[v] > \text{dist}[u] + w(u, v)$ :  
             $\text{dist}[v] \leftarrow \text{dist}[u] + w(u, v)$   
             $\text{prev}[v] \leftarrow u$   
             $\text{ChangePriority}(H, v, \text{dist}[v])$ 
```

# Pseudocode

## Dijkstra( $G, A$ )

```
for all  $u \in V$ :  
     $\text{dist}[u] \leftarrow \infty, \text{prev}[u] \leftarrow \text{nil}$   
 $\text{dist}[A] \leftarrow 0$   
 $H \leftarrow \text{MakeQueue}(V)$  {dist-values as keys}  
while  $H$  is not empty:  
     $u \leftarrow \text{ExtractMin}(H)$   
    for all  $(u, v) \in E$ :  
        if  $\text{dist}[v] > \text{dist}[u] + w(u, v)$ :  
             $\text{dist}[v] \leftarrow \text{dist}[u] + w(u, v)$   
             $\text{prev}[v] \leftarrow u$   
             $\text{ChangePriority}(H, v, \text{dist}[v])$ 
```

# Pseudocode

## Dijkstra( $G, A$ )

```
for all  $u \in V$ :  
     $\text{dist}[u] \leftarrow \infty, \text{prev}[u] \leftarrow \text{nil}$   
 $\text{dist}[A] \leftarrow 0$   
 $H \leftarrow \text{MakeQueue}(V)$  {dist-values as keys}  
while  $H$  is not empty:  
     $u \leftarrow \text{ExtractMin}(H)$   
    for all  $(u, v) \in E$ :  
        if  $\text{dist}[v] > \text{dist}[u] + w(u, v)$ :  
             $\text{dist}[v] \leftarrow \text{dist}[u] + w(u, v)$   
             $\text{prev}[v] \leftarrow u$   
             $\text{ChangePriority}(H, v, \text{dist}[v])$ 
```

# Pseudocode

## Dijkstra( $G, A$ )

```
for all  $u \in V$ :  
     $\text{dist}[u] \leftarrow \infty, \text{prev}[u] \leftarrow \text{nil}$   
 $\text{dist}[A] \leftarrow 0$   
 $H \leftarrow \text{MakeQueue}(V)$  {dist-values as keys}  
while  $H$  is not empty:  
     $u \leftarrow \text{ExtractMin}(H)$   
    for all  $(u, v) \in E$ :  
        if  $\text{dist}[v] > \text{dist}[u] + w(u, v)$ :  
             $\text{dist}[v] \leftarrow \text{dist}[u] + w(u, v)$   
             $\text{prev}[v] \leftarrow u$   
             $\text{ChangePriority}(H, v, \text{dist}[v])$ 
```

# Pseudocode

## Dijkstra( $G, A$ )

```
for all  $u \in V$ :  
     $\text{dist}[u] \leftarrow \infty, \text{prev}[u] \leftarrow \text{nil}$   
 $\text{dist}[A] \leftarrow 0$   
 $H \leftarrow \text{MakeQueue}(V)$  {dist-values as keys}  
while  $H$  is not empty:  
     $u \leftarrow \text{ExtractMin}(H)$   
    for all  $(u, v) \in E$ :  
        if  $\text{dist}[v] > \text{dist}[u] + w(u, v)$ :  
             $\text{dist}[v] \leftarrow \text{dist}[u] + w(u, v)$   
             $\text{prev}[v] \leftarrow u$   
             $\text{ChangePriority}(H, v, \text{dist}[v])$ 
```

# Pseudocode

## Dijkstra( $G, A$ )

```
for all  $u \in V$ :  
     $\text{dist}[u] \leftarrow \infty, \text{prev}[u] \leftarrow \text{nil}$   
 $\text{dist}[A] \leftarrow 0$   
 $H \leftarrow \text{MakeQueue}(V)$  {dist-values as keys}  
while  $H$  is not empty:  
     $u \leftarrow \text{ExtractMin}(H)$   
    for all  $(u, v) \in E$ :  
        if  $\text{dist}[v] > \text{dist}[u] + w(u, v)$ :  
             $\text{dist}[v] \leftarrow \text{dist}[u] + w(u, v)$   
             $\text{prev}[v] \leftarrow u$   
             $\text{ChangePriority}(H, v, \text{dist}[v])$ 
```

# Pseudocode

## Dijkstra( $G, A$ )

```
for all  $u \in V$ :  
     $\text{dist}[u] \leftarrow \infty, \text{prev}[u] \leftarrow \text{nil}$   
 $\text{dist}[A] \leftarrow 0$   
 $H \leftarrow \text{MakeQueue}(V)$  {dist-values as keys}  
while  $H$  is not empty:  
     $u \leftarrow \text{ExtractMin}(H)$   
    for all  $(u, v) \in E$ :  
        if  $\text{dist}[v] > \text{dist}[u] + w(u, v)$ :  
             $\text{dist}[v] \leftarrow \text{dist}[u] + w(u, v)$   
             $\text{prev}[v] \leftarrow u$   
             $\text{ChangePriority}(H, v, \text{dist}[v])$ 
```

# Pseudocode

## Dijkstra( $G, A$ )

```
for all  $u \in V$ :  
     $\text{dist}[u] \leftarrow \infty, \text{prev}[u] \leftarrow \text{nil}$   
 $\text{dist}[A] \leftarrow 0$   
 $H \leftarrow \text{MakeQueue}(V)$  {dist-values as keys}  
while  $H$  is not empty:  
     $u \leftarrow \text{ExtractMin}(H)$   
    for all  $(u, v) \in E$ :  
        if  $\text{dist}[v] > \text{dist}[u] + w(u, v)$ :  
             $\text{dist}[v] \leftarrow \text{dist}[u] + w(u, v)$   
             $\text{prev}[v] \leftarrow u$   
             $\text{ChangePriority}(H, v, \text{dist}[v])$ 
```



# Pseudocode

## Dijkstra( $G, A$ )

```
for all  $u \in V$ :  
     $\text{dist}[u] \leftarrow \infty, \text{prev}[u] \leftarrow \text{nil}$   
 $\text{dist}[A] \leftarrow 0$   
 $H \leftarrow \text{MakeQueue}(V)$  {dist-values as keys}  
while  $H$  is not empty:  
     $u \leftarrow \text{ExtractMin}(H)$   
    for all  $(u, v) \in E$ :  
        if  $\text{dist}[v] > \text{dist}[u] + w(u, v)$ :  
             $\text{dist}[v] \leftarrow \text{dist}[u] + w(u, v)$   
             $\text{prev}[v] \leftarrow u$   
             $\text{ChangePriority}(H, v, \text{dist}[v])$ 
```

# Outline

- 1 Fastest Route
- 2 Naive Algorithm
- 3 Dijkstra's Algorithm
- 4 Dijkstra Example
- 5 Implementation
- 6 Proof of Correctness**
- 7 Analysis

# Pseudocode

## Dijkstra( $G, A$ )

```
for all  $u \in V$ :  
     $\text{dist}[u] \leftarrow \infty, \text{prev}[u] \leftarrow \text{nil}$   
 $\text{dist}[A] \leftarrow 0$   
 $H \leftarrow \text{MakeQueue}(V)$  {dist-values as keys}  
while  $H$  is not empty:  
     $u \leftarrow \text{ExtractMin}(H)$   
    for all  $(u, v) \in E$ :  
        if  $\text{dist}[v] > \text{dist}[u] + w(u, v)$ :  
             $\text{dist}[v] \leftarrow \text{dist}[u] + w(u, v)$   
             $\text{prev}[v] \leftarrow u$   
             $\text{ChangePriority}(H, v, \text{dist}[v])$ 
```

# Pseudocode

## Dijkstra( $G, A$ )

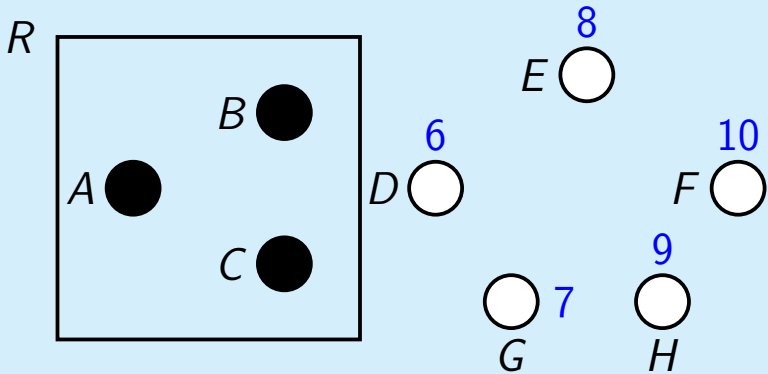
```
for all  $u \in V$ :  
     $\text{dist}[u] \leftarrow \infty, \text{prev}[u] \leftarrow \text{nil}$   
 $\text{dist}[A] \leftarrow 0$   
 $H \leftarrow \text{MakeQueue}(V)$  {dist-values as keys}  
while  $H$  is not empty:  
     $u \leftarrow \text{ExtractMin}(H)$   
    for all  $(u, v) \in E$ :  
        if  $\text{dist}[v] > \text{dist}[u] + w(u, v)$ :  
             $\text{dist}[v] \leftarrow \text{dist}[u] + w(u, v)$   
             $\text{prev}[v] \leftarrow u$   
             $\text{ChangePriority}(H, v, \text{dist}[v])$ 
```

# Correct distances

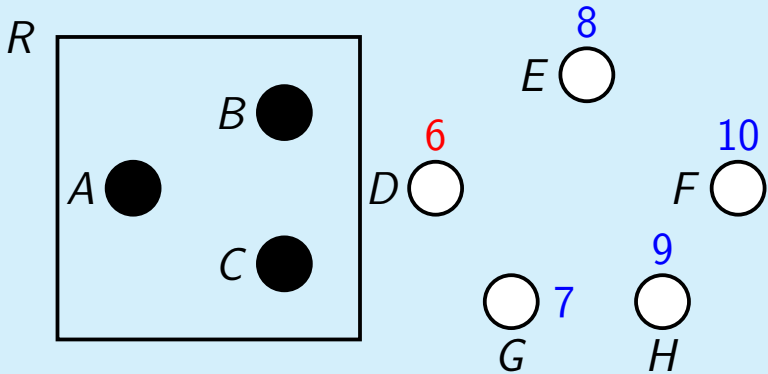
## Lemma

When a node  $u$  is selected via `ExtractMin`,  
 $\text{dist}[u] = d(A, u)$ .

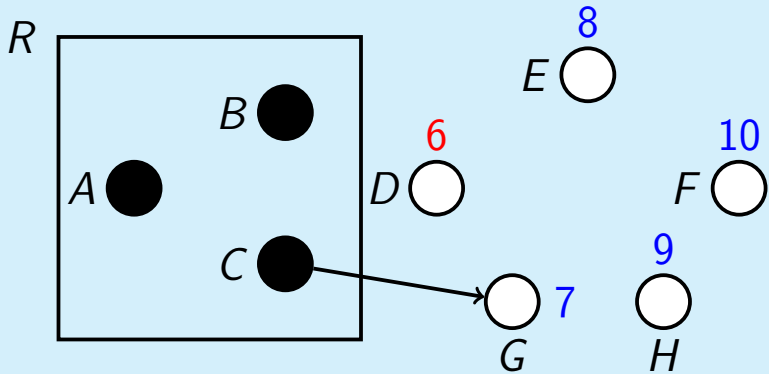
# Proof



# Proof

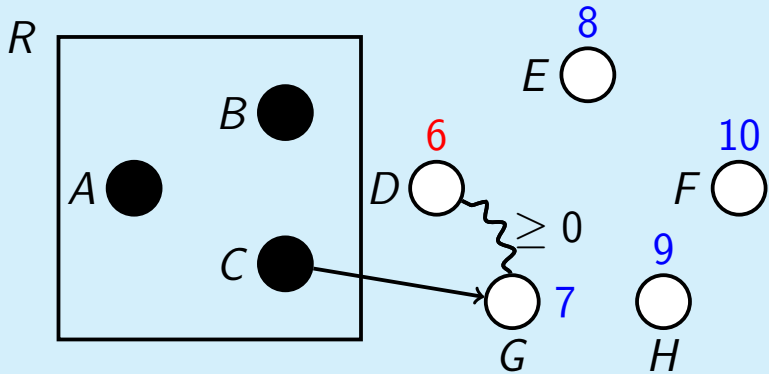


# Proof

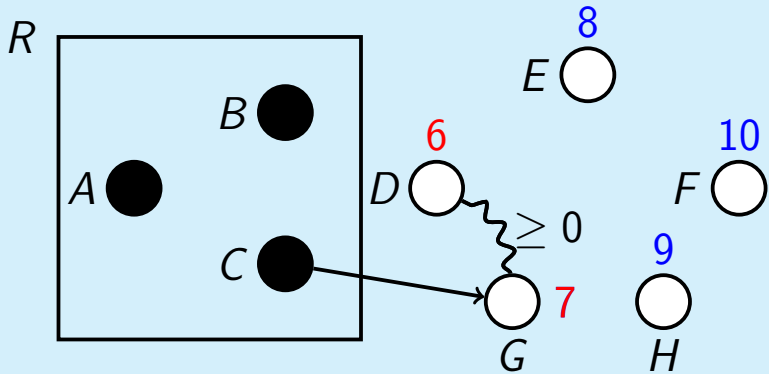




# Proof



# Proof



# Outline

- 1 Fastest Route
- 2 Naive Algorithm
- 3 Dijkstra's Algorithm
- 4 Dijkstra Example
- 5 Implementation
- 6 Proof of Correctness
- 7 Analysis

# Pseudocode

## Dijkstra( $G, A$ )

```
for all  $u \in V$ :  
     $\text{dist}[u] \leftarrow \infty, \text{prev}[u] \leftarrow \text{nil}$   
 $\text{dist}[A] \leftarrow 0$   
 $H \leftarrow \text{MakeQueue}(V)$  {dist-values as keys}  
while  $H$  is not empty:  
     $u \leftarrow \text{ExtractMin}(H)$   
    for all  $(u, v) \in E$ :  
        if  $\text{dist}[v] > \text{dist}[u] + w(u, v)$ :  
             $\text{dist}[v] \leftarrow \text{dist}[u] + w(u, v)$   
             $\text{prev}[v] \leftarrow u$   
             $\text{ChangePriority}(H, v, \text{dist}[v])$ 
```

# Pseudocode

## Dijkstra( $G, A$ )

```
for all  $u \in V$ :  
     $\text{dist}[u] \leftarrow \infty, \text{prev}[u] \leftarrow \text{nil}$   
 $\text{dist}[A] \leftarrow 0$   
 $H \leftarrow \text{MakeQueue}(V)$  {dist-values as keys}  
while  $H$  is not empty:  
     $u \leftarrow \text{ExtractMin}(H)$   
    for all  $(u, v) \in E$ :  
        if  $\text{dist}[v] > \text{dist}[u] + w(u, v)$ :  
             $\text{dist}[v] \leftarrow \text{dist}[u] + w(u, v)$   
             $\text{prev}[v] \leftarrow u$   
             $\text{ChangePriority}(H, v, \text{dist}[v])$ 
```

# Pseudocode

## Dijkstra( $G, A$ )

```
for all  $u \in V$ :  
     $\text{dist}[u] \leftarrow \infty, \text{prev}[u] \leftarrow \text{nil}$   
 $\text{dist}[A] \leftarrow 0$   
 $H \leftarrow \text{MakeQueue}(V)$  {dist-values as keys}  
while  $H$  is not empty:  
     $u \leftarrow \text{ExtractMin}(H)$   
    for all  $(u, v) \in E$ :  
        if  $\text{dist}[v] > \text{dist}[u] + w(u, v)$ :  
             $\text{dist}[v] \leftarrow \text{dist}[u] + w(u, v)$   
             $\text{prev}[v] \leftarrow u$   
             $\text{ChangePriority}(H, v, \text{dist}[v])$ 
```

# Pseudocode

## Dijkstra( $G, A$ )

```
for all  $u \in V$ :  
     $\text{dist}[u] \leftarrow \infty, \text{prev}[u] \leftarrow \text{nil}$   
 $\text{dist}[A] \leftarrow 0$   
 $H \leftarrow \text{MakeQueue}(V)$  {dist-values as keys}  
while  $H$  is not empty:  
     $u \leftarrow \text{ExtractMin}(H)$   
    for all  $(u, v) \in E$ :  
        if  $\text{dist}[v] > \text{dist}[u] + w(u, v)$ :  
             $\text{dist}[v] \leftarrow \text{dist}[u] + w(u, v)$   
             $\text{prev}[v] \leftarrow u$   
             $\text{ChangePriority}(H, v, \text{dist}[v])$ 
```

# Pseudocode

## Dijkstra( $G, A$ )

```
for all  $u \in V$ :  
     $\text{dist}[u] \leftarrow \infty, \text{prev}[u] \leftarrow \text{nil}$   
 $\text{dist}[A] \leftarrow 0$   
 $H \leftarrow \text{MakeQueue}(V)$  {dist-values as keys}  
while  $H$  is not empty:  
     $u \leftarrow \text{ExtractMin}(H)$   
    for all  $(u, v) \in E$ :  
        if  $\text{dist}[v] > \text{dist}[u] + w(u, v)$ :  
             $\text{dist}[v] \leftarrow \text{dist}[u] + w(u, v)$   
             $\text{prev}[v] \leftarrow u$   
             $\text{ChangePriority}(H, v, \text{dist}[v])$ 
```



# Running time

Total running time:

$$\begin{aligned} O(V) &+ T(\text{MakeQueue}) \\ &+ |V| \cdot T(\text{ExtractMin}) \\ &+ |E| \cdot T(\text{ChangePriority}) \end{aligned}$$

Priority queue implemented as array:

$$O(|V| + |V| + |V|^2 + |E|) = O(|V|^2)$$

# Running time

Total running time:

$$\begin{aligned} O(V) &+ T(\text{MakeQueue}) \\ &+ |V| \cdot T(\text{ExtractMin}) \\ &+ |E| \cdot T(\text{ChangePriority}) \end{aligned}$$

Priority queue implemented as binary heap:

$$\begin{aligned} O(|V| + |V| + |V| \log |V| + |E| \log |V|) &= \\ O((|V| + |E|) \log |V|) \end{aligned}$$

# Conclusion

- Can find the minimum time to get from work to home
- Can find the fastest route from work to home
- Works for any graph with non-negative edge weights
- Works in  $O(|V|^2)$  or  $O((|V| + |E|) \log(|V|))$  depending on the implementation