# Paths in Graphs: Fastest Route

Michael Levin

Higher School of Economics

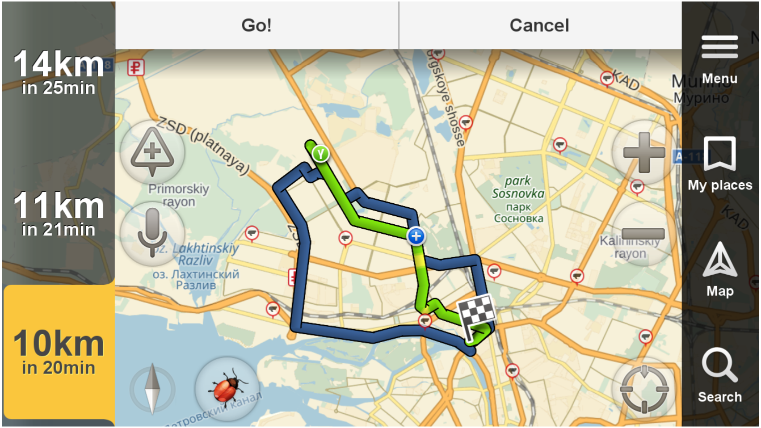
## Graph Algorithms
## Data Structures and Algorithms

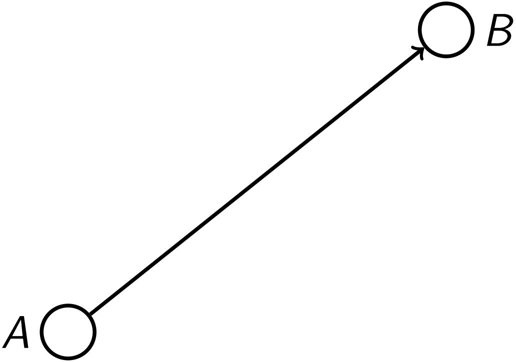
# Outline

1. Fastest Route
2. Naive Algorithm
3. Dijkstra's Algorithm

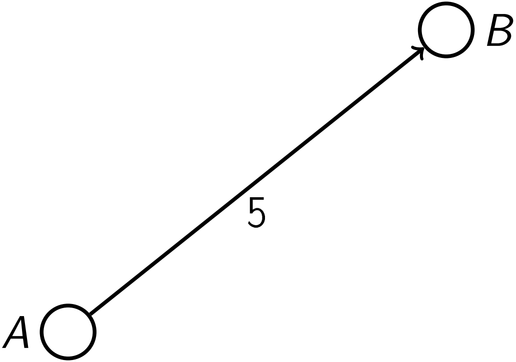
# Fastest Route

What is the fastest route to get home from work?

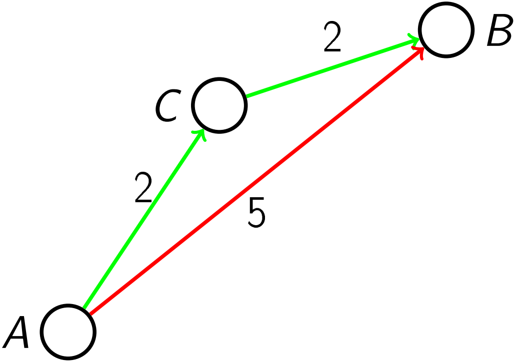


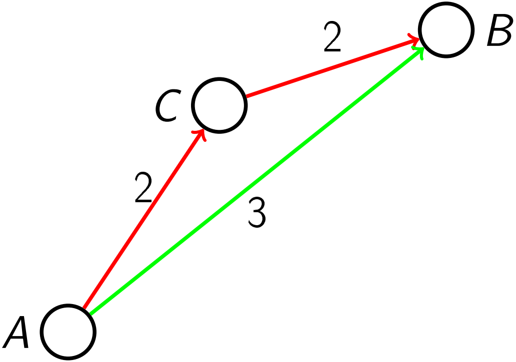


A
B



A
B
5

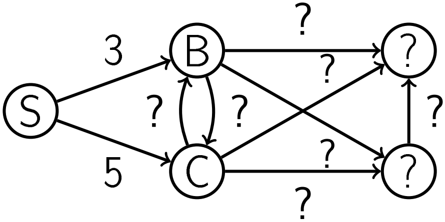

A
B
C
2
2
5



A
B
C
2
2
3

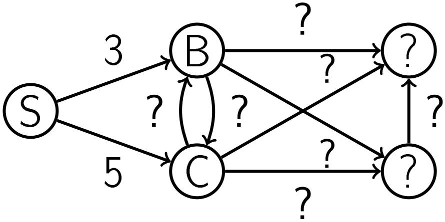
# Intuition

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



# Intuition

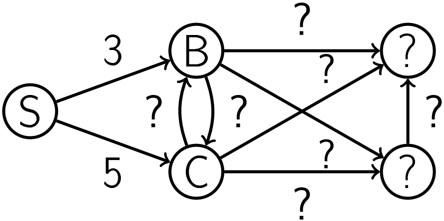
- Assume that we stay at $S$ and observe two outgoing edges:




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

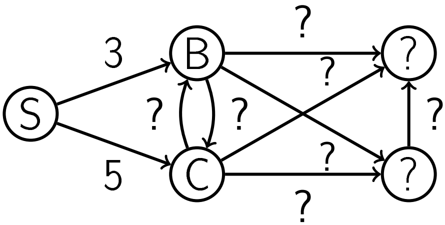
# Intuition

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



# Intuition

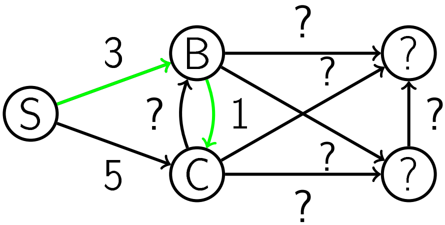
- Can we be sure that the distance from $S$ to $C$ is 5?




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

# Intuition

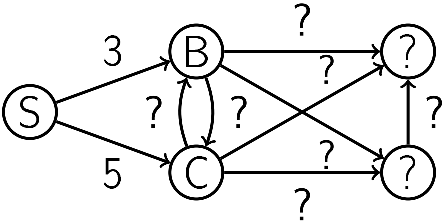
- Can we be sure that the distance from $S$ to $C$ is 5?




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

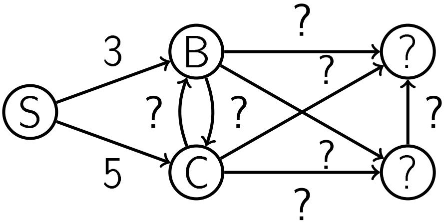
# Intuition

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



# Intuition

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




- Yes, because there are no negative weight edges.

# Outline

1. Fastest Route
2. Naive Algorithm
3. Dijkstra's Algorithm

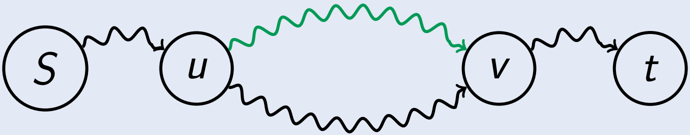
# Optimal substructure

## Observation

Any subpath of an optimal path is also optimal.

## Proof

Consider an optimal path from $S$ to $t$ 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 $S$ to $t$.



# Corollary

If $S \to \ldots \to u \to t$ is a shortest path from $S$ to $t$, then

$$d(S,t) = d(S,u) + w(u,t)$$

# Edge relaxation

- `dist`[$v$] will be an upper bound on the actual distance from $S$ to $v$.

# Edge relaxation

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

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

```
if  dist[v] > dist[u] + w(u, v):
    dist[v] ← dist[u] + w(u, v)
    prev[v] ← u
```

# Naive approach

**Naive$(G, S)$**

```
for all u ∈ V:
    dist[u] ← ∞
    prev[u] ← nil
dist[S] ← 0
do:
    relax all the edges
while at least one dist changes
```

# Correct distances

## Lemma

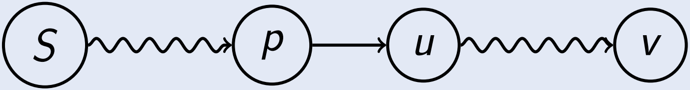
After the call to `Naive` algorithm all the distances are set correctly.

## Proof

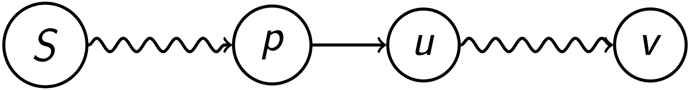
- Assume, for the sake of contradiction, that no edge can be relaxed and there is a vertex $v$ such that $\text{dist}[v] > d(S, v)$.

# Proof

- Assume, for the sake of contradiction, that no edge can be relaxed and there is a vertex $v$ such that $\text{dist}[v] > d(S, v)$.
- Consider a shortest path from $S$ 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 $d(S,p) = \text{dist}[p]$ and hence $d(S,u) = d(S,p) + w(p,u) = \text{dist}[p] + w(p,u)$

# Proof (continued)

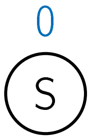
$$S \rightsquigarrow p \longrightarrow u \rightsquigarrow v$$

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

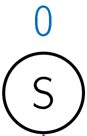
# Outline

1. Fastest Route
2. Naive Algorithm
3. Dijkstra's Algorithm

# Intuition

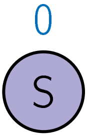


# Intuition

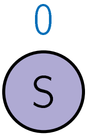



initially, we only know the distance to S

# Intuition

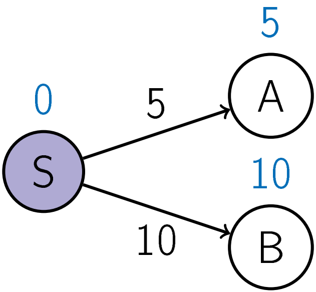


# Intuition



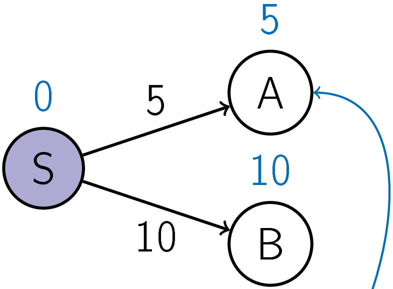

let's relax all the edges from S

# Intuition



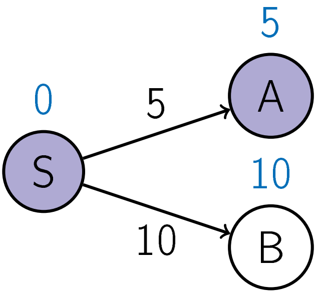

let's relax all the edges from S

# Intuition

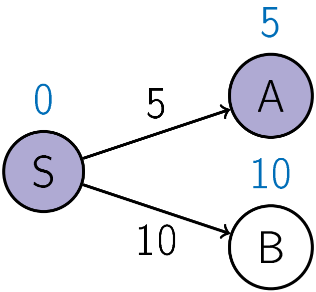



we now know the distance for A

# Intuition

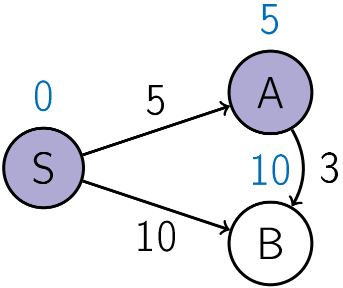


# Intuition



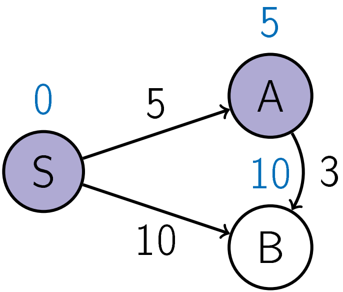

now, let's relax all the edges from A

# Intuition



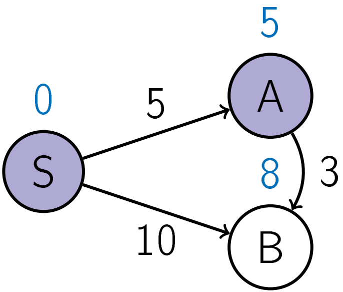

now, let's relax all the edges from A

# Intuition

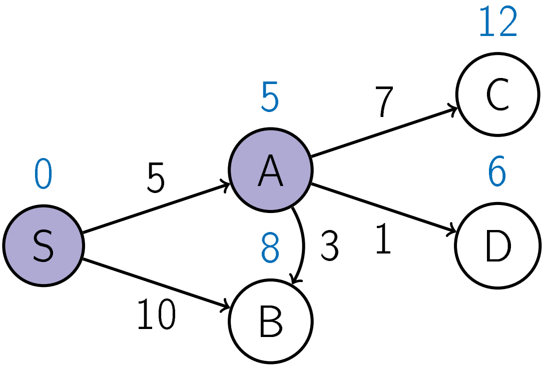



we discover an edge $(A, B)$ of weight 3 that updates dist$[B]$

# Intuition

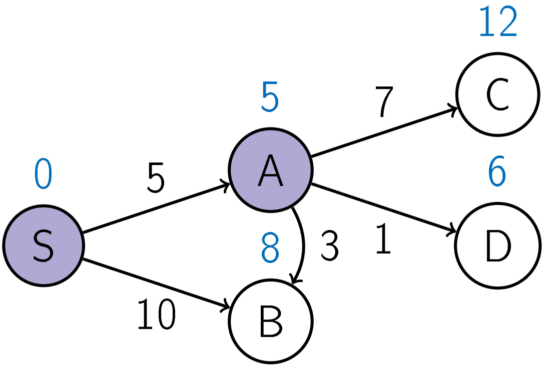


# Intuition



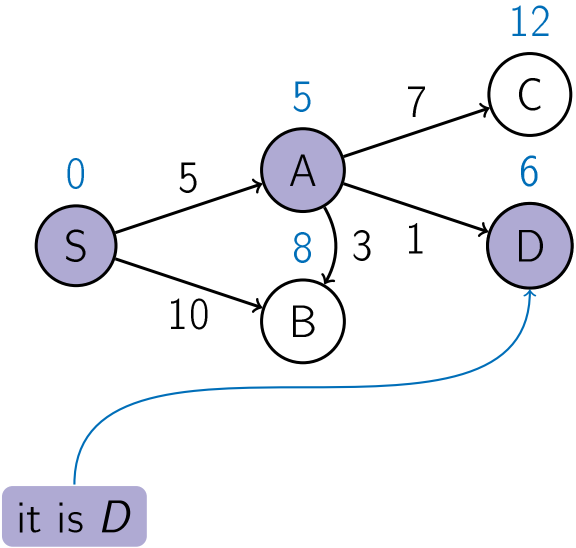

we also discover a few more outgoing edges

# Intuition

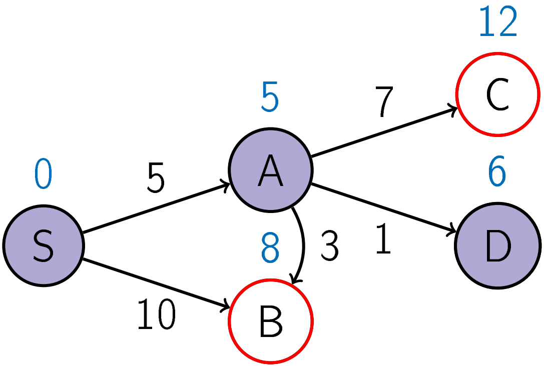



what is the next vertex for which we already know the correct distance?

# Intuition



# Intuition




while for $B$ and $C$ 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").

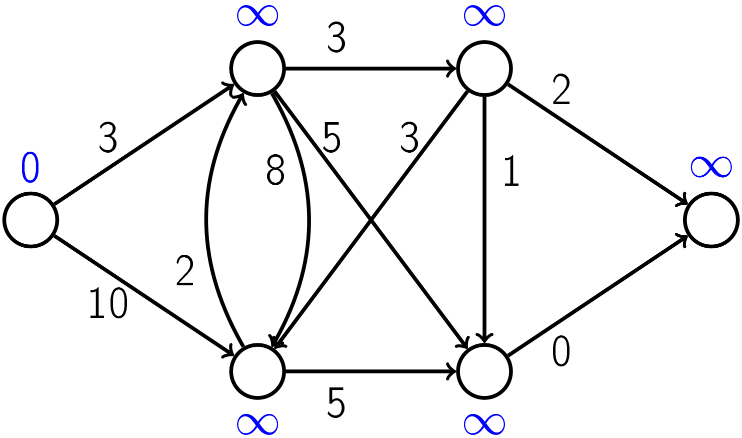
# Main ideas of Dijkstra's Algorithm

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

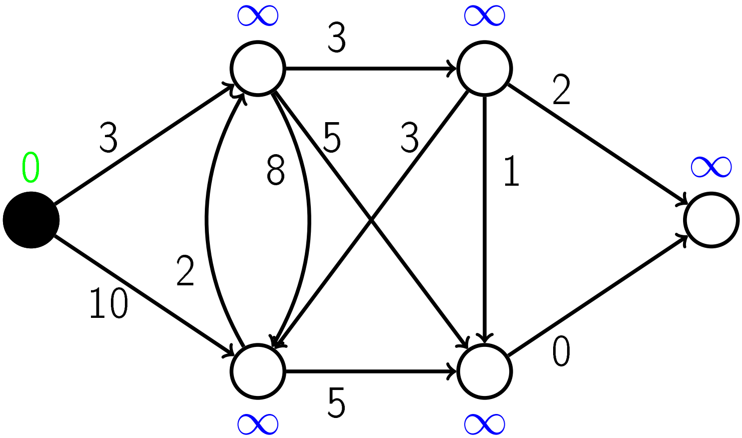
# Main ideas of Dijkstra's Algorithm

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

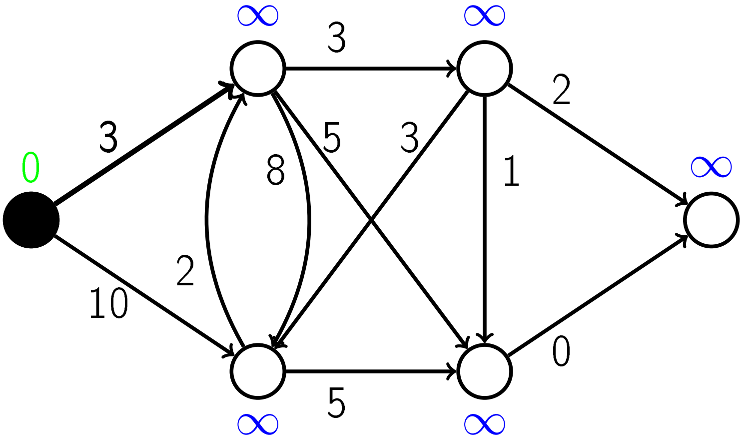
# Example



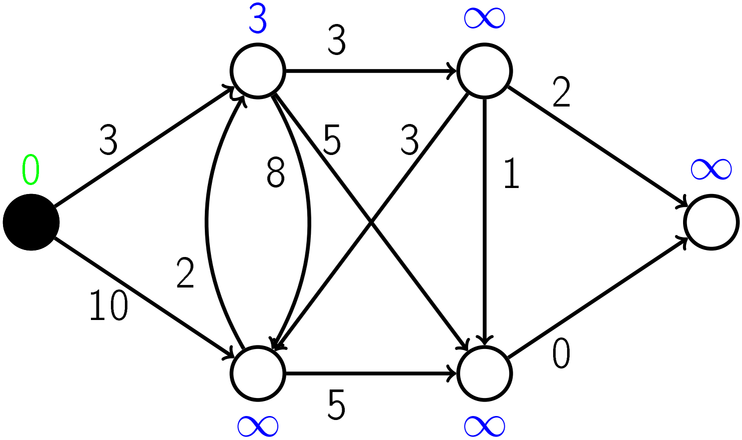
# Example



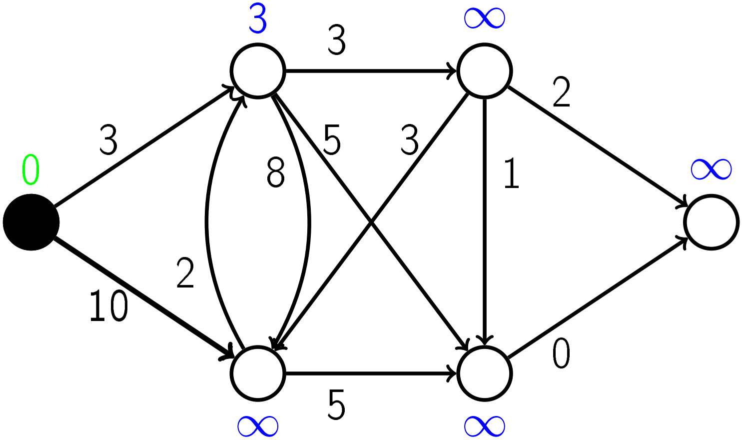
# Example



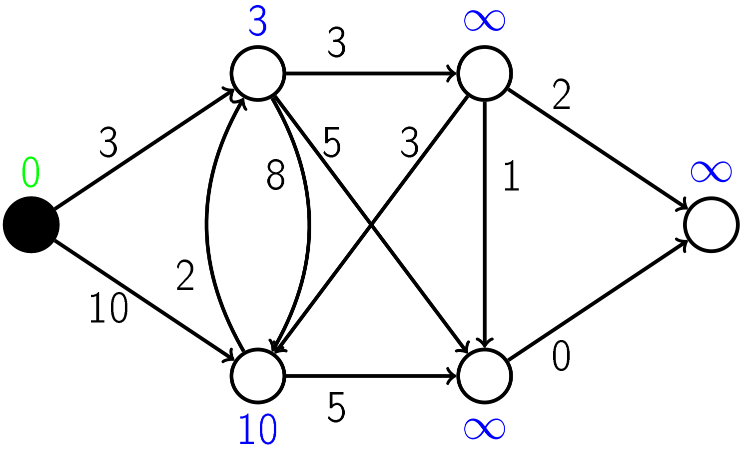
# Example



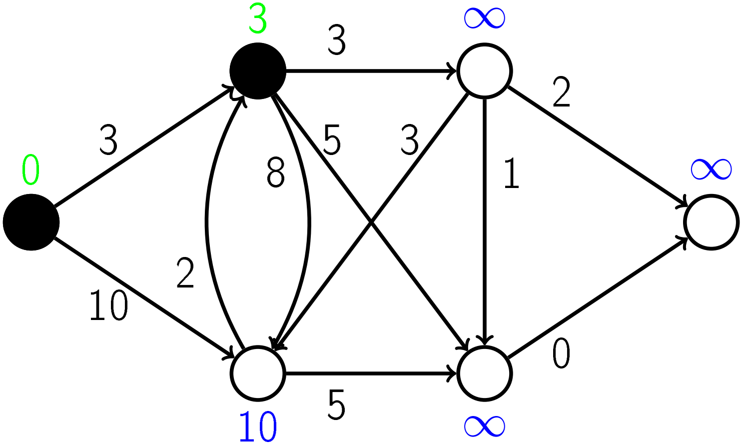
# Example



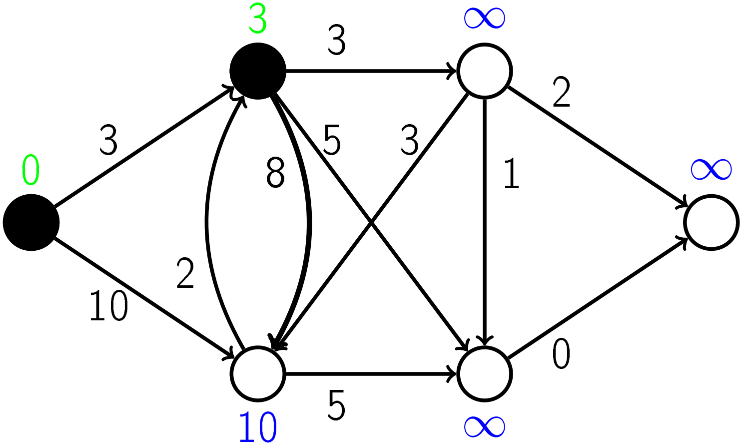
# Example



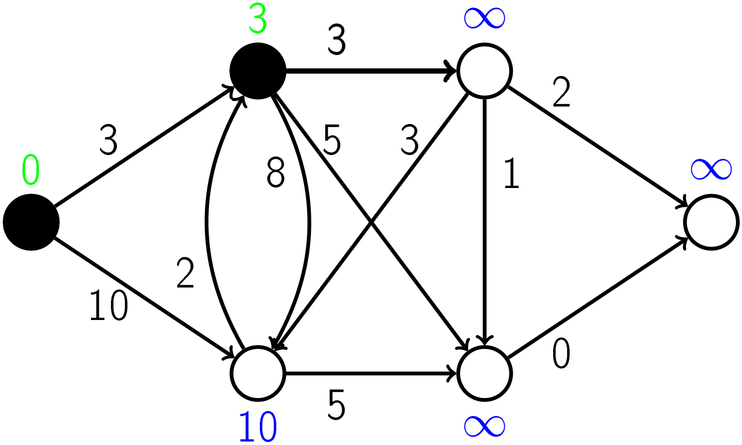
# Example



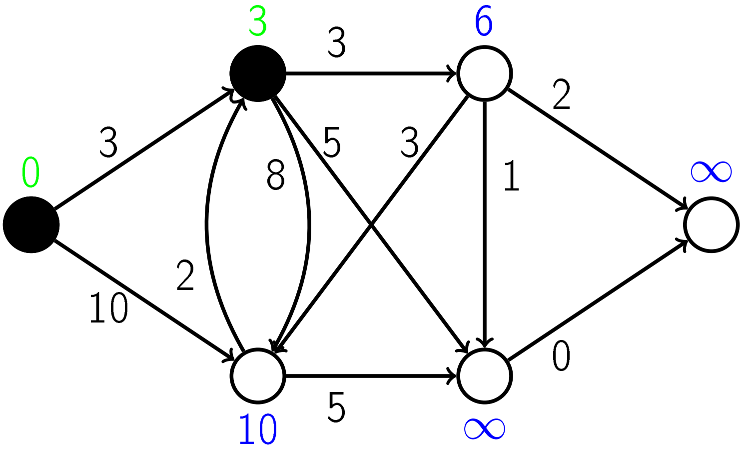
# Example



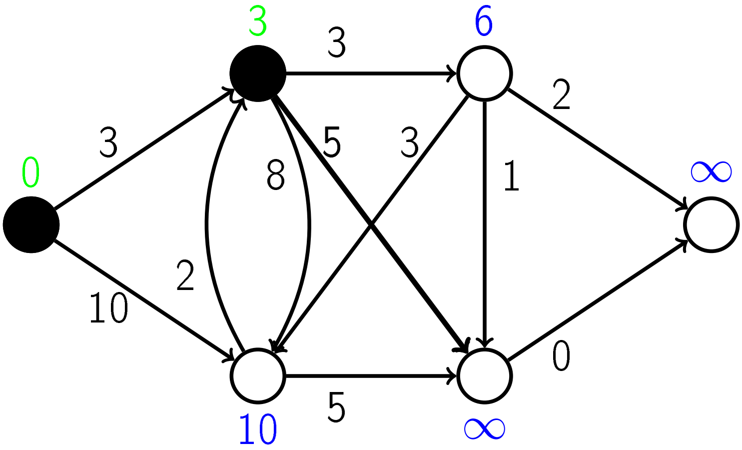
# Example



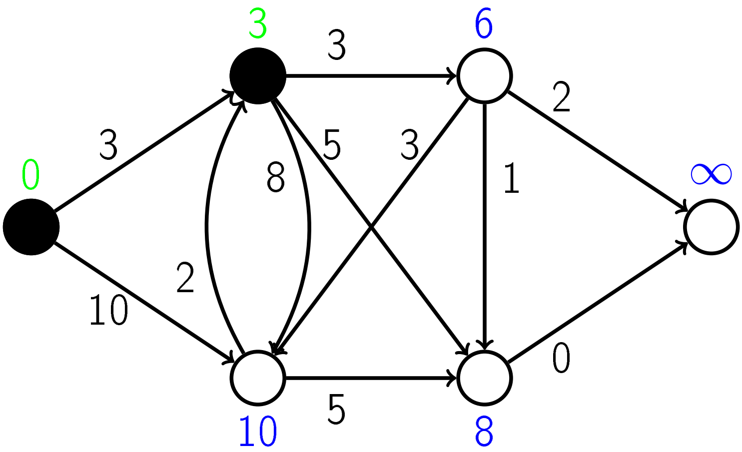
# Example



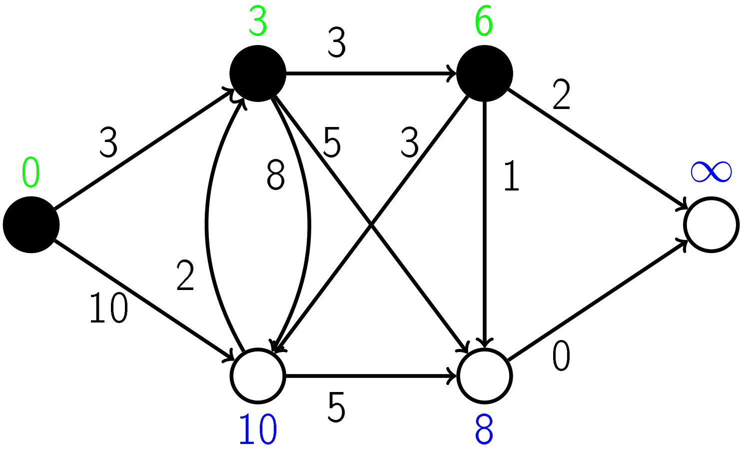
# Example



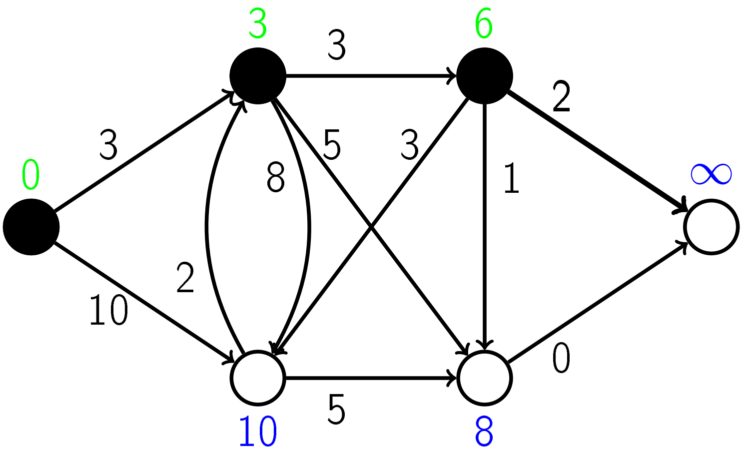
# Example



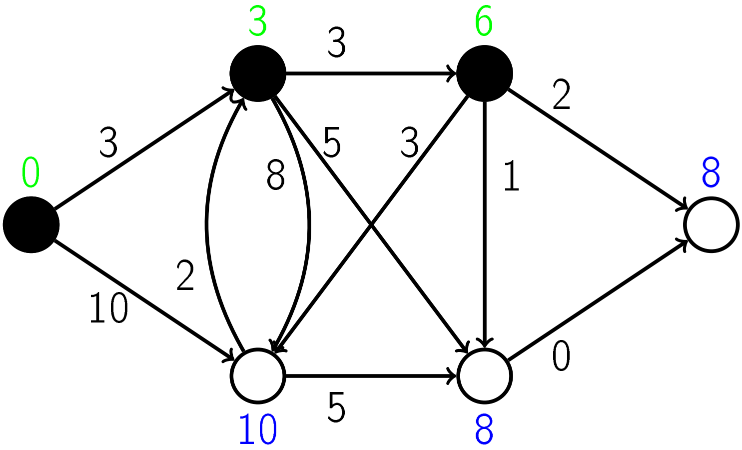
# Example



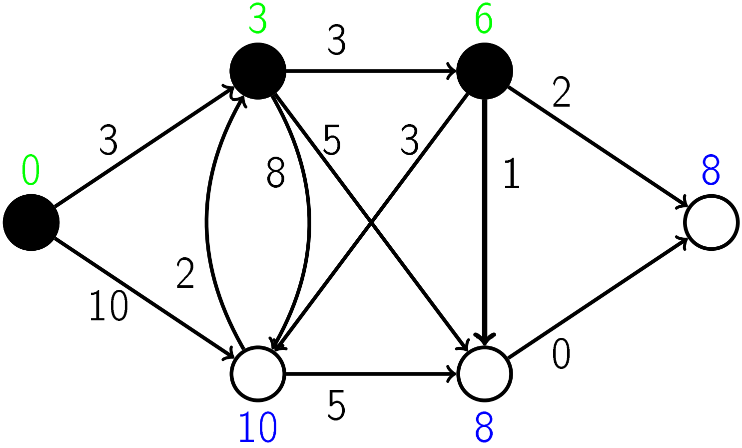
# Example



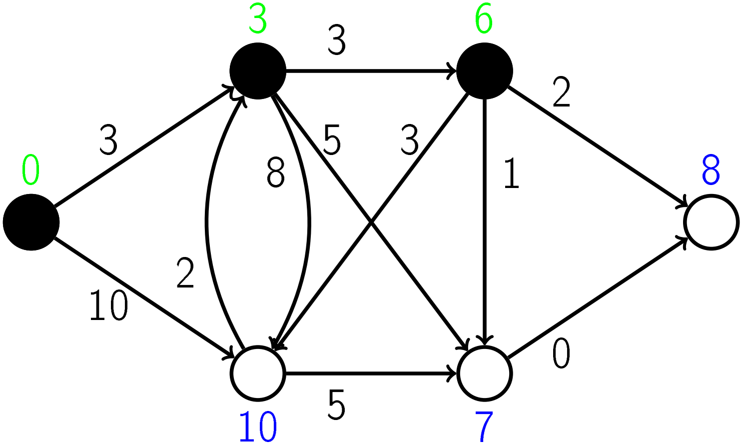
# Example



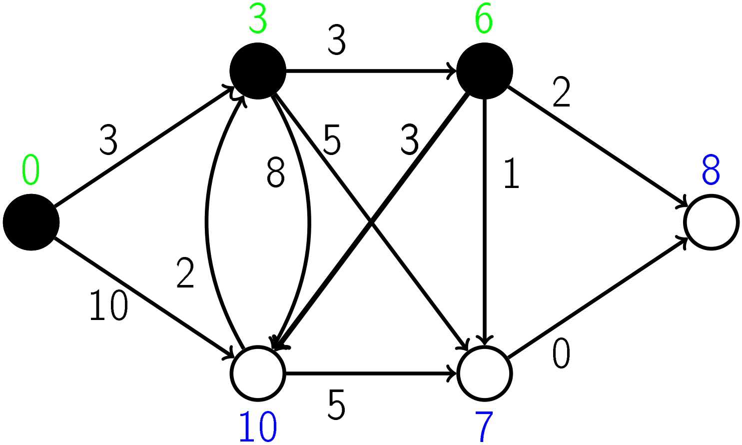
# Example



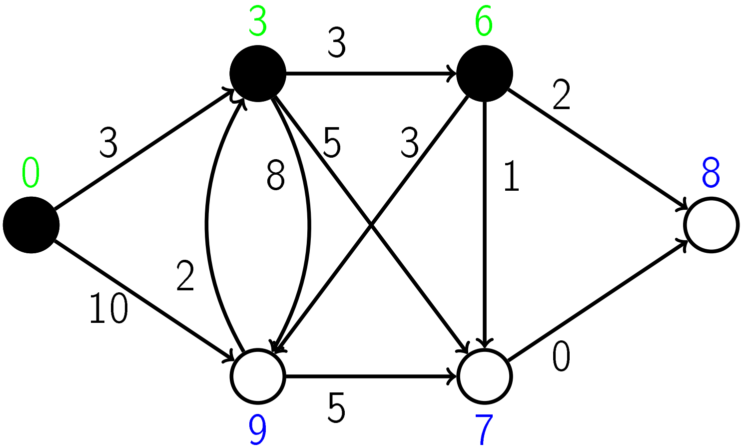
# Example



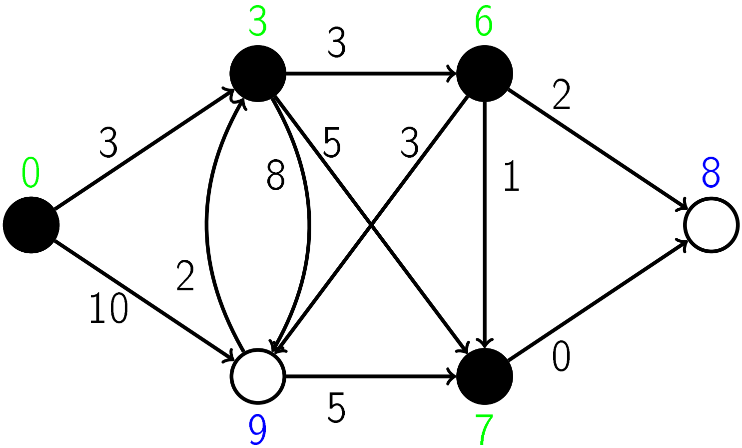
# Example



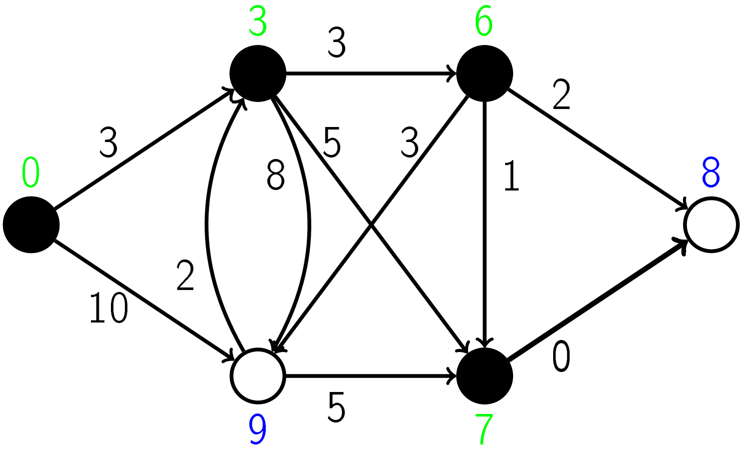
# Example



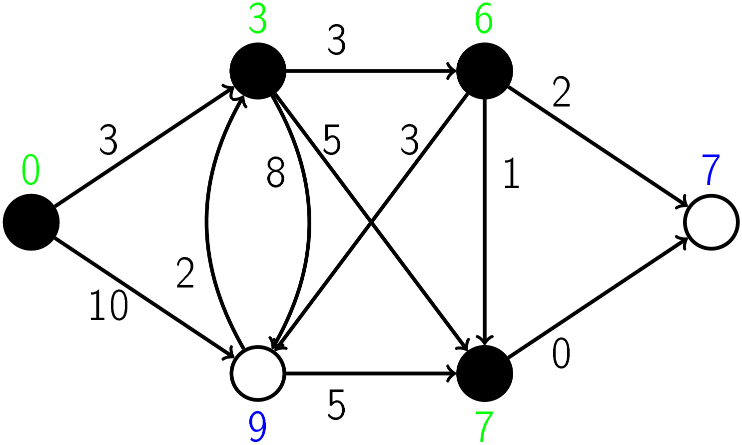
# Example



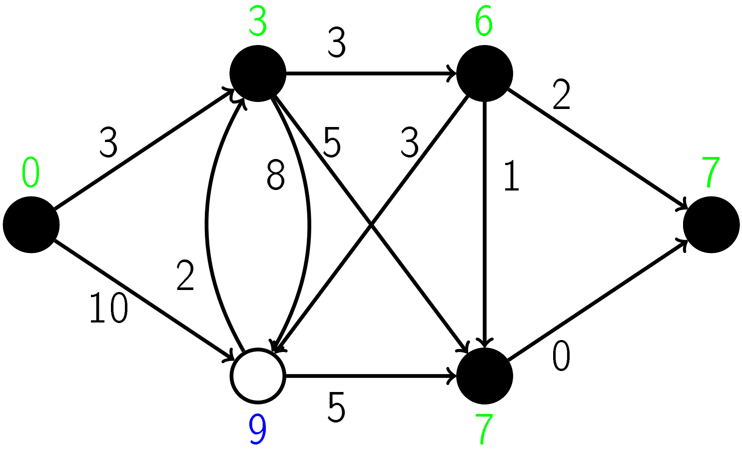
# Example



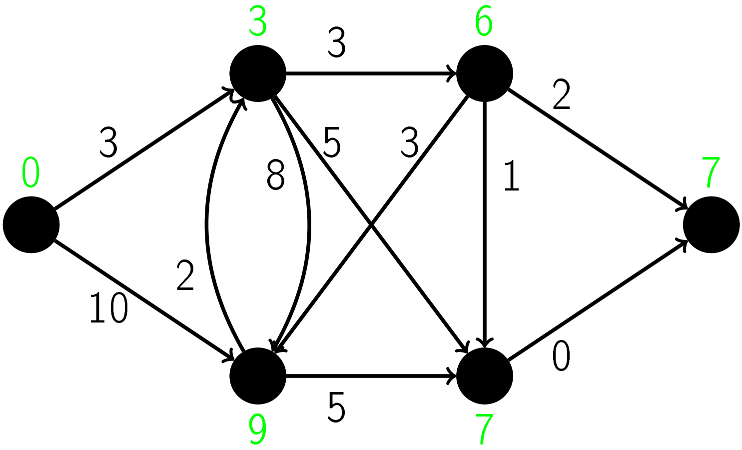
# Example



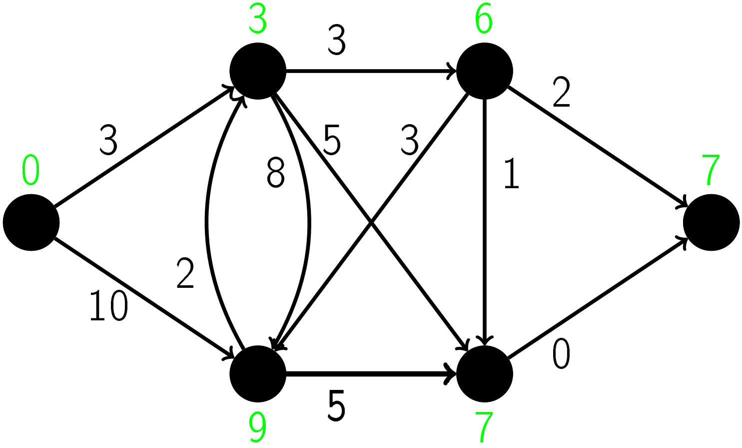
# Example



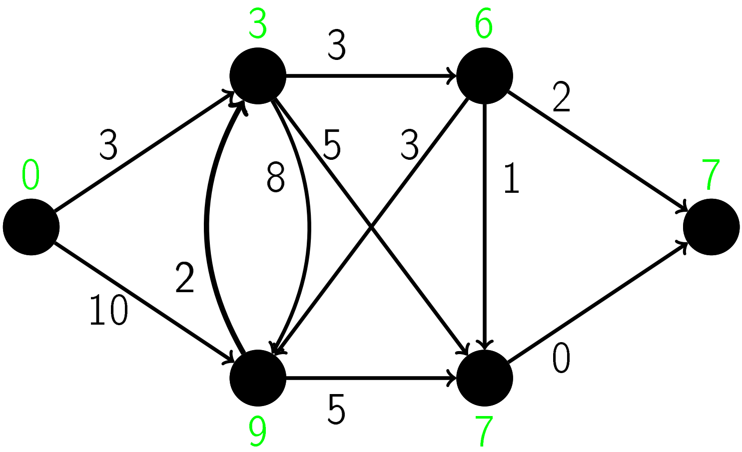
# Example



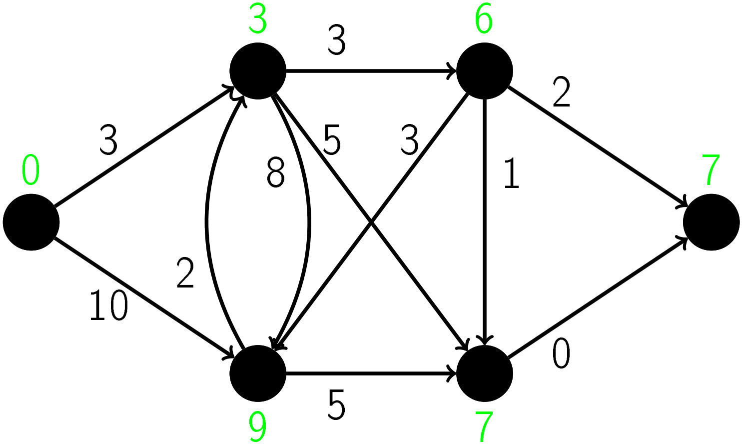
# Example



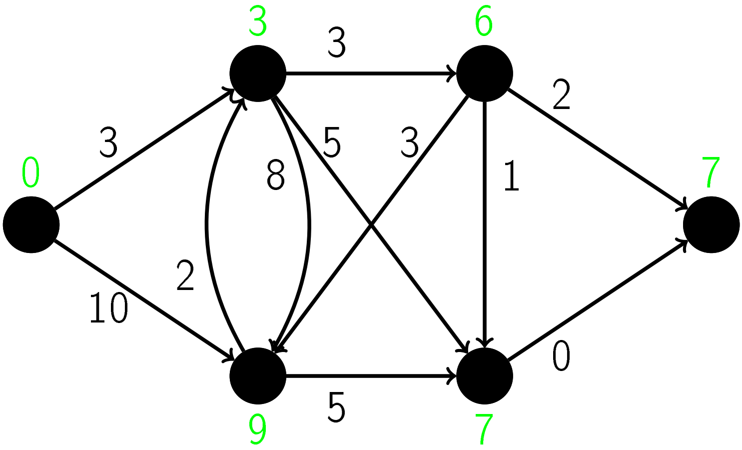
# Example



# Example



# Example



# Pseudocode

**Dijkstra($G, S$)**

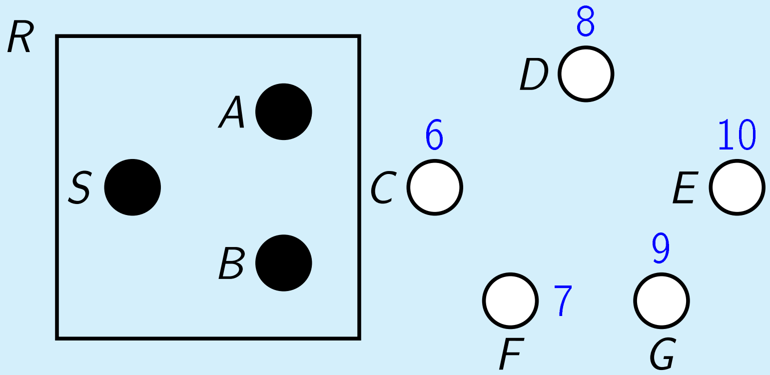
for all $u \in V$:
  dist[$u$] $\leftarrow \infty$, prev[$u$] $\leftarrow$ nil
dist[$S$] $\leftarrow$ 0
$H \leftarrow$ MakeQueue($V$) {dist-values as keys}
while $H$ is not empty:
  $u \leftarrow$ ExtractMin($H$)
  for all $(u, v) \in E$:
    if $dist[v] > dist[u] + w(u, v)$:
      $dist[v] \leftarrow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

# Correct distances

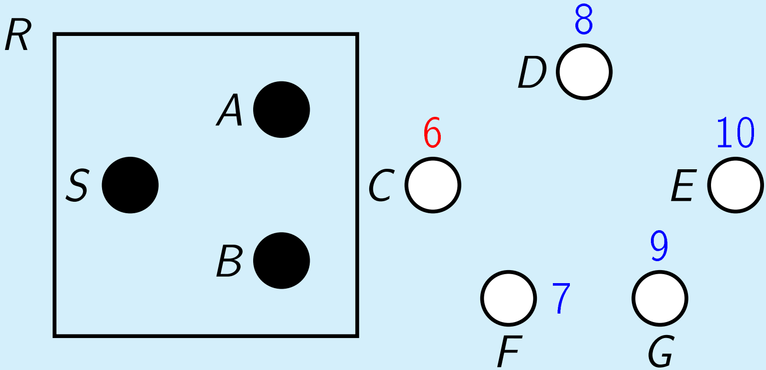
**Lemma**

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

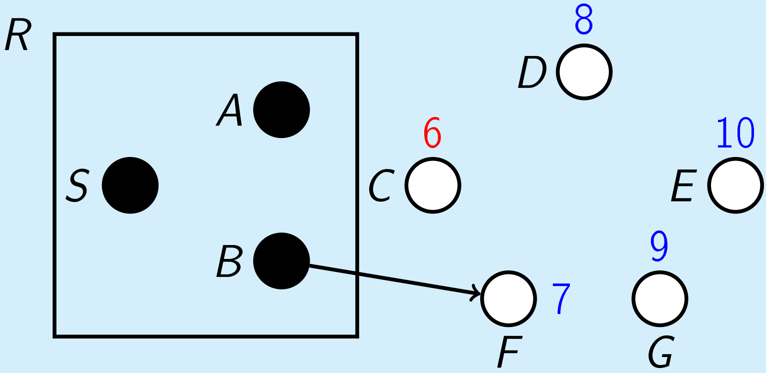
## Proof



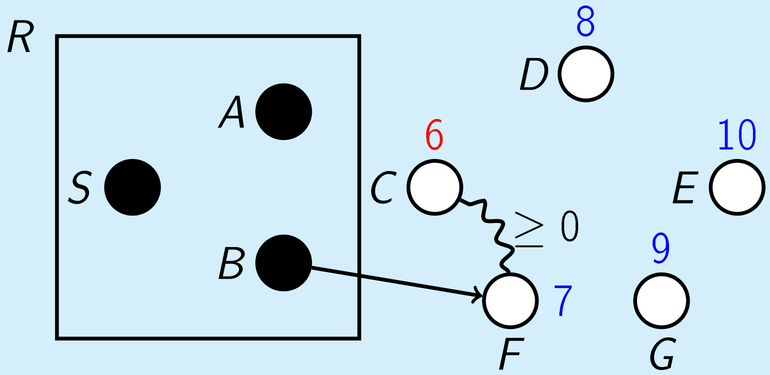
## Proof



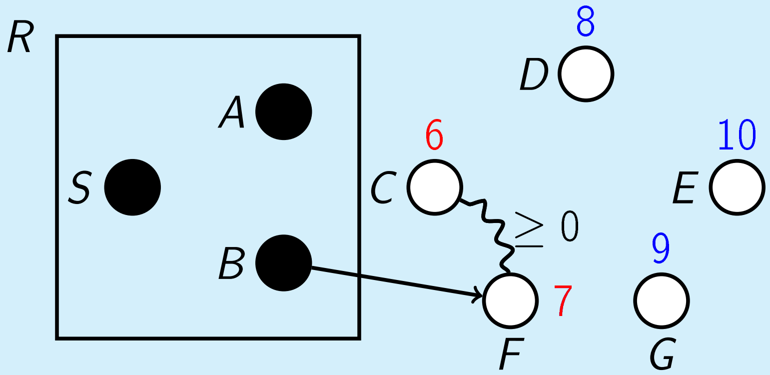
Proof



## Proof



## Proof



# Running time

Total running time:

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

# Running time

Total running time:

$$T(\texttt{MakeQueue}) + |V| \cdot T(\texttt{ExtractMin}) + |E| \cdot T(\texttt{ChangePriority})$$

Priority queue implementations:

- array:

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

# Running time

Total running time:

$$T(\texttt{MakeQueue}) + |V| \cdot T(\texttt{ExtractMin}) + |E| \cdot T(\texttt{ChangePriority})$$

Priority queue implementations:

- array:

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

- binary heap:
  $O(|V| + |V| \log |V| + |E| \log |V|) = O((|V| + |E|) \log |V|)$

# 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