# CS 344: Design and Analysis of Computer Algorithms

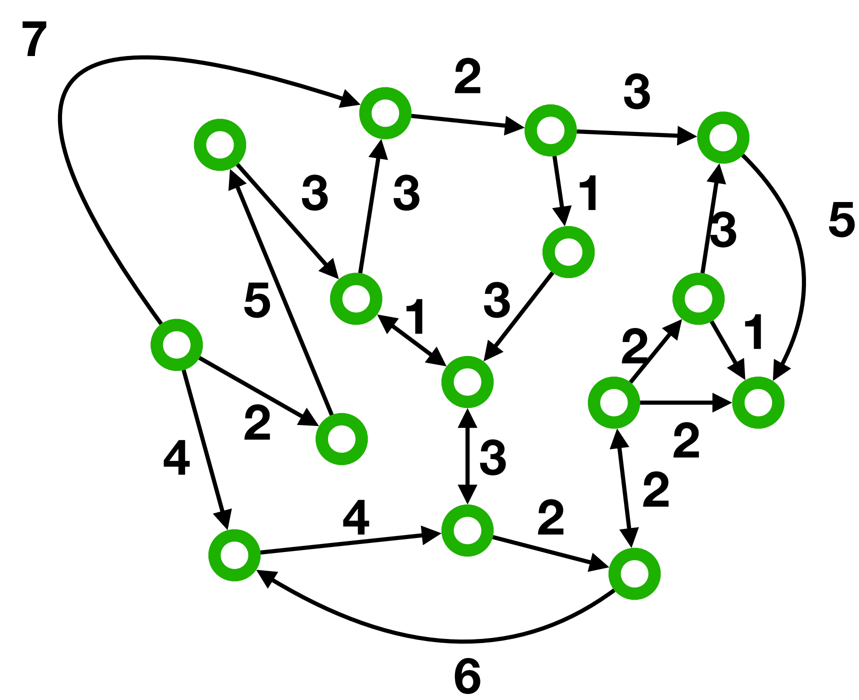
(Spring 2022 — Sections 5,6,7,8)

## Lecture 20: Single-Source Shortest Path: Bellman-Ford, Dijkstra

# The Single-Source Shortest Path Problem

# Shortest Paths

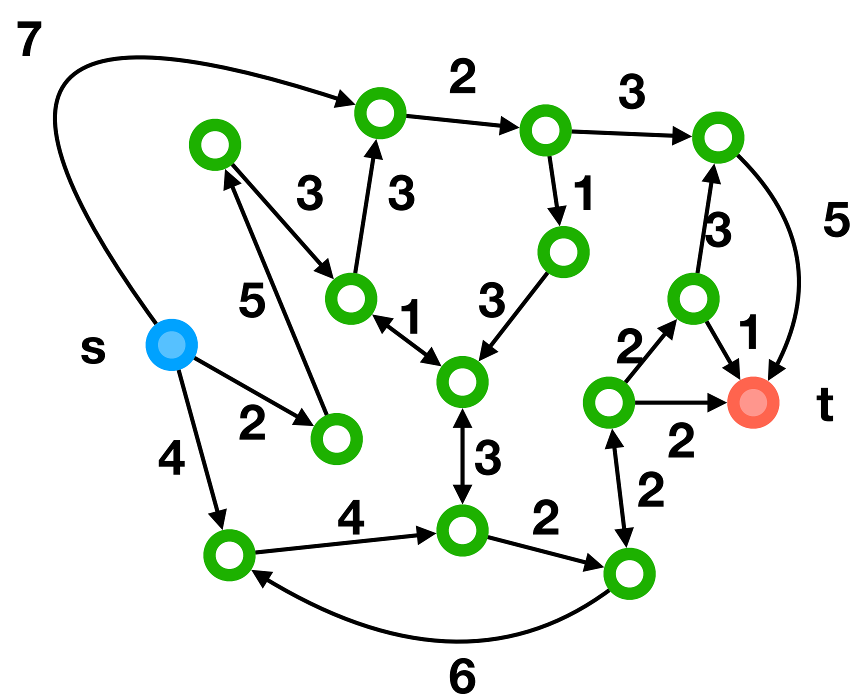
- For a graph G=(V,E) (directed or undirected) with weights $w_e$ over each edge e



# Shortest Paths

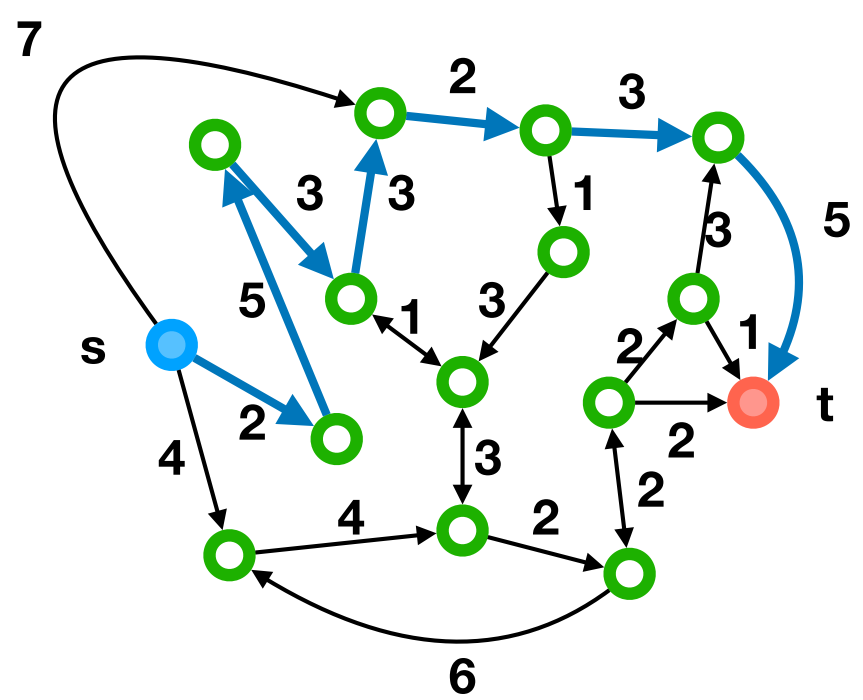
- For a graph G=(V,E) (directed or undirected) with weights $w_e$ over each edge e

- Let $P_{st}$ be any path between vertices s and t



# Shortest Paths

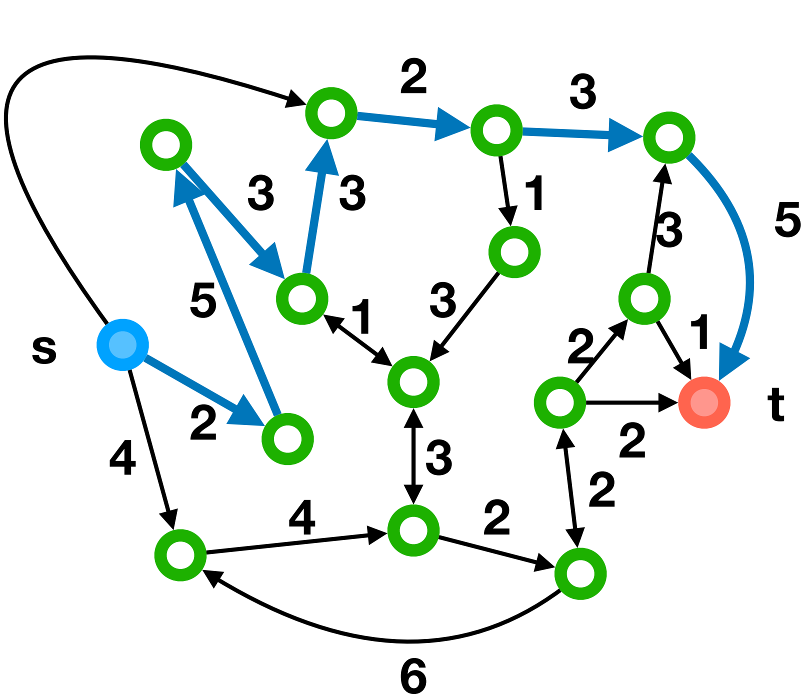
- For a graph G=(V,E) (directed or undirected) with weights $w_e$ over each edge e
- Let $P_{st}$ be any path between vertices s and t



# Shortest Paths

- For a graph G=(V,E) (directed or undirected) with weights $w_e$ over each edge e
- Let $P_{st}$ be any path between vertices s and t
- Weight of the path:

  - $w(P_{st}) = \sum_{e \in P} w_e$



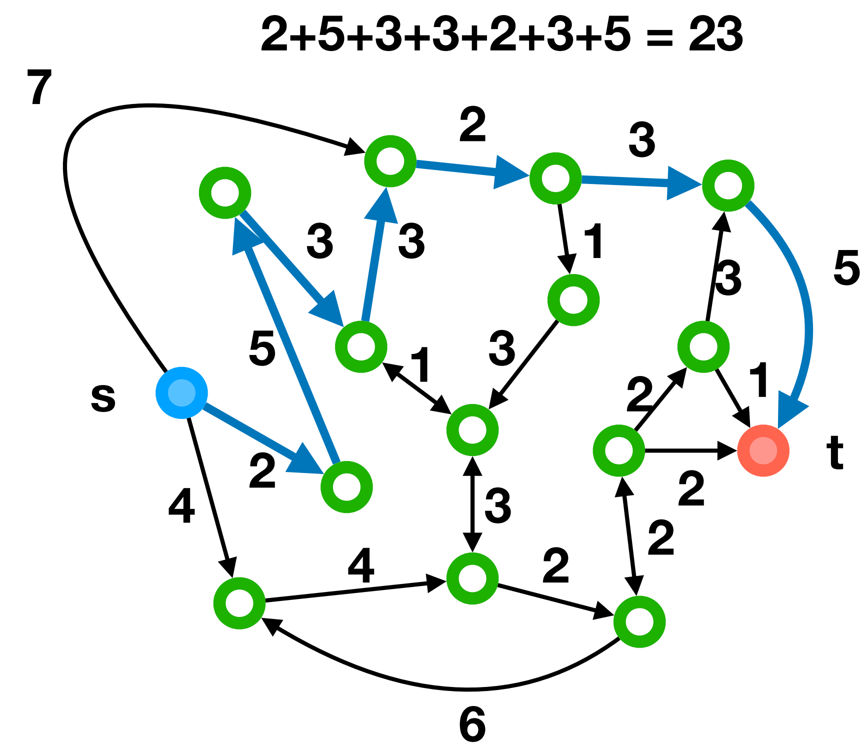
# Shortest Paths

- For a graph G=(V,E) (directed or undirected) with weights $w_e$ over each edge e

- Let $P_{st}$ be any path between vertices s and t

- Weight of the path:

$$w(P_{st}) = \sum_{e \in P} w_e$$

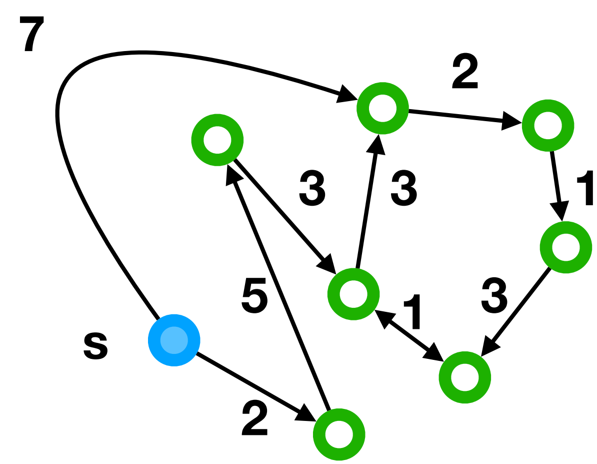


# Shortest Paths

- Shortest s-t Path:
  - The path with minimum weight
  - $P_{st}$ that minimizes $w(P_{st})$
- Distance of s to t, $dist(s, t)$:
  - Weight of shortest path from s to t

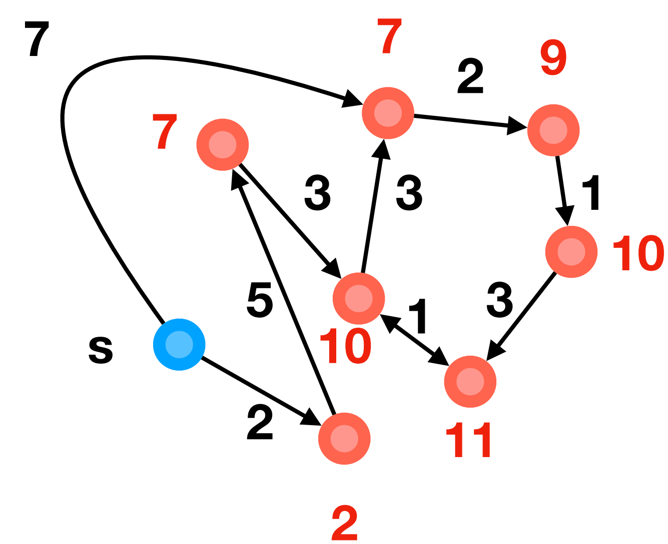
# Shortest Paths

- Shortest s-t Path:
  - The path with minimum weight
  - $P_{st}$ that minimizes $w(P_{st})$
- Distance of s to t, $dist(s, t)$:
  - Weight of shortest path from s to t



# Shortest Paths

- Shortest s-t Path:
  - The path with minimum weight
  - $P_{st}$ that minimizes $w(P_{st})$
- Distance of s to t, $dist(s, t)$:
  - Weight of shortest path from s to t



# Single-Source Shortest Path Problem

- **Input:**
  - A graph G=(V,E) (undirected or directed)
  - Weights $w_e$ on each edge $e$
  - A single vertex s called source
- **Output:**
  - The distance of s to all other vertices: $dist(s, v)$ for all $v \in V$

# From SSSP to Finding Shortest Paths

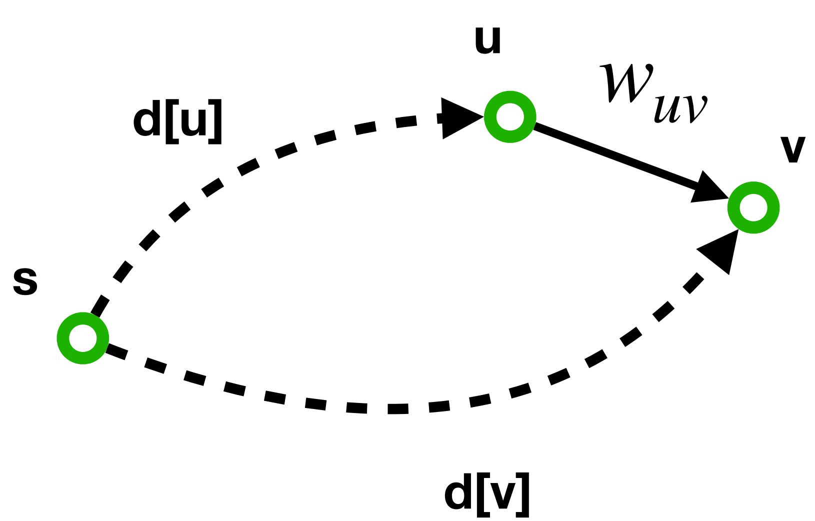
# Bellman-Ford Algorithm

# Bellman-Ford Algorithm

- Let d[s] = 0 and d[v] = $+\infty$ for $v \in V - \{s\}$
- For every edge (u,v) in the graph:
  - If $d[v] > d[u] + w_{uv}$ **update** $d[v] \leftarrow d[u] + w_{uv}$
- If no update happened in the for-loop terminate, otherwise run the for-loop again.

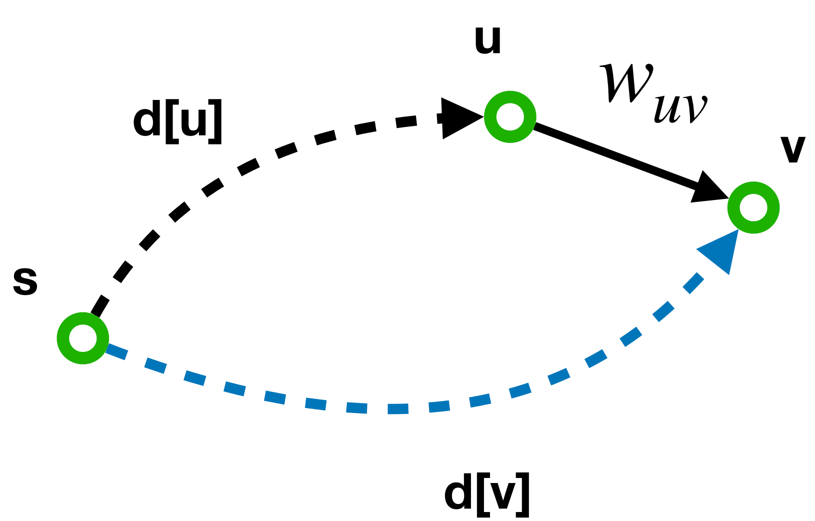
# Bellman-Ford Algorithm

- We update values like this:
  - for any edge (u,v), if $d[v] > d[u] + w_{uv}$ set $d[v] \leftarrow d[u] + w_{uv}$



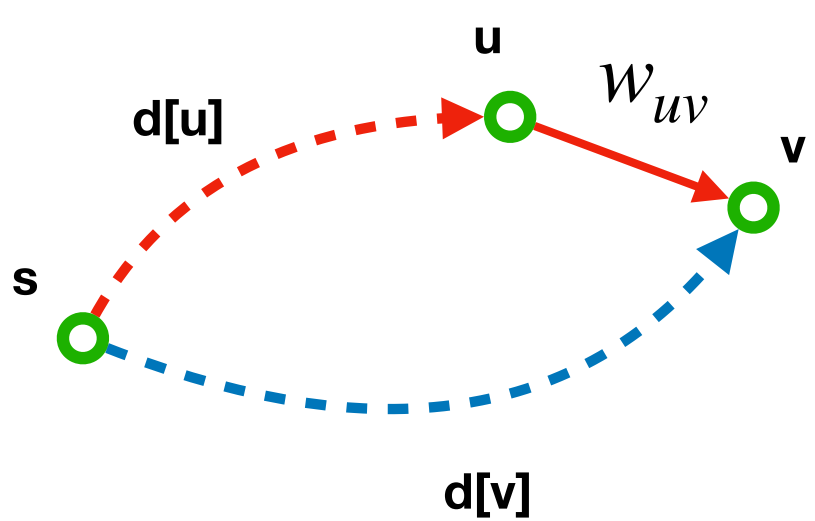
# Bellman-Ford Algorithm

- We update values like this:
  - for any edge (u,v), if $d[v] > d[u] + w_{uv}$ set $d[v] \leftarrow d[u] + w_{uv}$



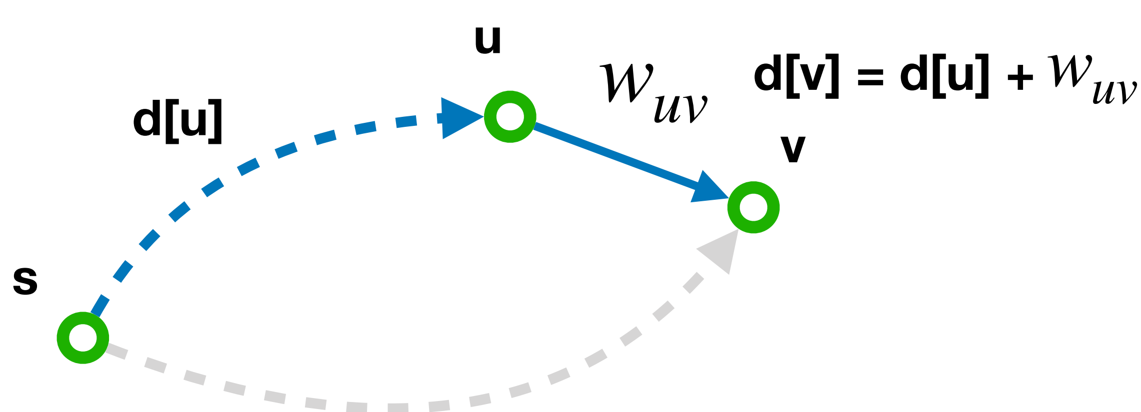
# Bellman-Ford Algorithm

- We update values like this:
  - for any edge (u,v), if $d[v] > d[u] + w_{uv}$ set $d[v] \leftarrow d[u] + w_{uv}$



# Bellman-Ford Algorithm

- We update values like this:
  - for any edge (u,v), if $d[v] > d[u] + w_{uv}$ set $d[v] \leftarrow d[u] + w_{uv}$



# Proof of Correctness

- Let d[s] = 0 and d[v] = $+\infty$ for $v \in V - \{s\}$

- For every edge (u,v) in the graph:

    - If $d[v] > d[u] + w_{uv}$ **update** $d[v] \leftarrow d[u] + w_{uv}$

- If no update happened in the for-loop terminate, otherwise run the for-loop again.

- Define $dist_i(s, v)$ as the weight of shortest path from s to v using at most i edges

- **Inductive statement:** For any i, after i-th run of the for-loop entirely, $d[v] \leq dist_i(s, v)$

# Runtime Analysis

- Let d[s] = 0 and d[v] = $+\infty$ for $v \in V - \{s\}$

- For every edge (u,v) in the graph:
  - If $d[v] > d[u] + w_{uv}$ **update** $d[v] \leftarrow d[u] + w_{uv}$

- If no update happened in the for-loop terminate, otherwise run the for-loop again.

- We can have at most n iterations of the for-loop

- (By the proof of correctness)

- Each iteration takes O(m) time

- So total runtime is O(mn)

# Bellman-Ford Algorithm

- Is extremely simple
- Is extremely agile and can be used in different settings:
  - Distributed algorithms or computer networks
- But its runtime is too slow
- We will see another algorithm with much faster runtime

# Bellman-Ford Algorithm

- Is extremely simple
- Is extremely agile and can be used in different settings:
  - Distributed algorithms or computer networks
- But its runtime is too slow
- We will see another algorithm with much faster runtime

- Btw, Bellman-Ford is a dynamic programming algorithm — in fact, perhaps the first serious one ever invented!

# Dijkstra's Algorithm

# Dijkstra’s Algorithm

- Is a much faster algorithm for SSSP than Bellman-Ford
- Is almost, but not quite, the same as Prim’s algorithm for MST
  - But note that SSSP is an entirely different problem than MST

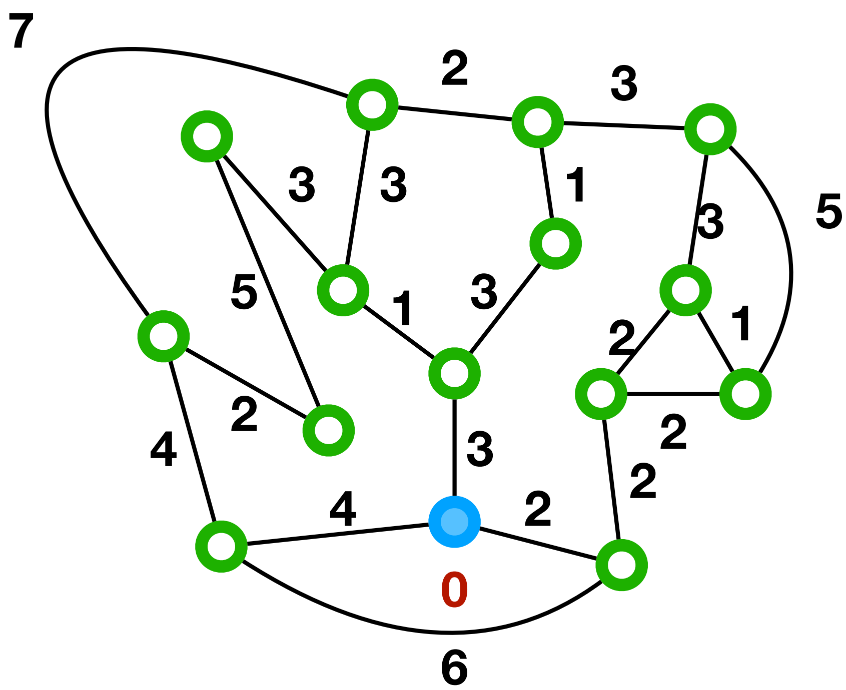
# Dijkstra's Algorithm

- Let mark[1:n] = false for all vertices
- Let mark[s] = true, d[s]=0 and H be edges incident on s
- While H is not empty:
  - Remove edge e=(u,v) with minimum value of $d[u] + w_{uv}$ from H
  - If mark[v] = true, ignore this edge and go to next iteration of the while-loop
  - Otherwise, set $d[v] = d[u] + w_{uv}$ and mark[v] = true, and insert all edges e incident on v with value $d[v] + w_e$ to H

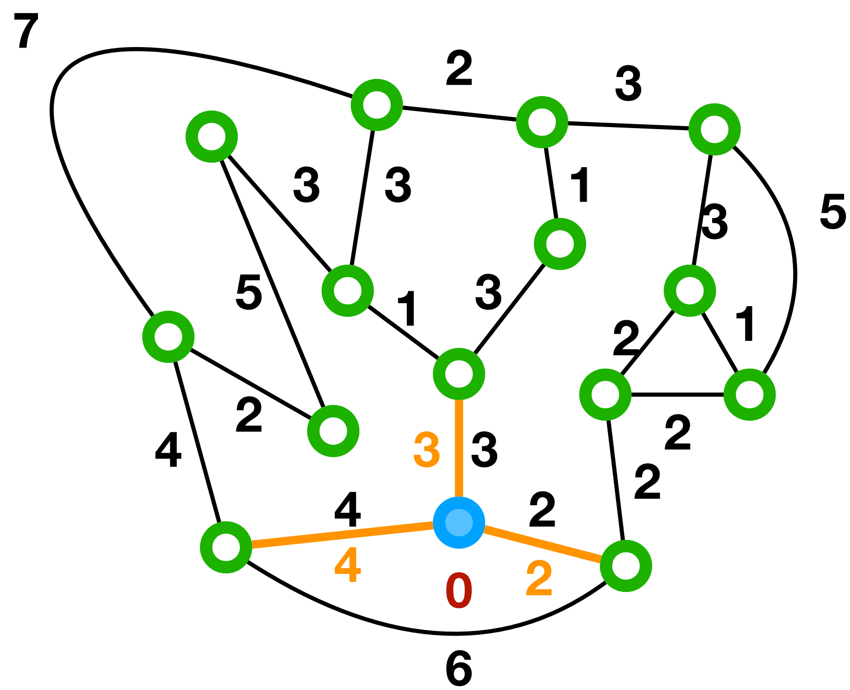
# Dijkstra's Algorithm

- Let mark[1:n] = false for all vertices
- Let mark[s] = true, d[s]=0 and H be edges incident on s
- While H is not empty:
  - **Remove e=(u,v) with minimum value of $d[u] + w_{uv}$ from H**
  - If mark[v] = true, ignore this edge and go to next iteration of the while-loop
  - Otherwise, set $d[v] = d[u] + w_{uv}$ and mark[v] = true, and insert all edges e incident on v with value $d[v] + w_e$ to H

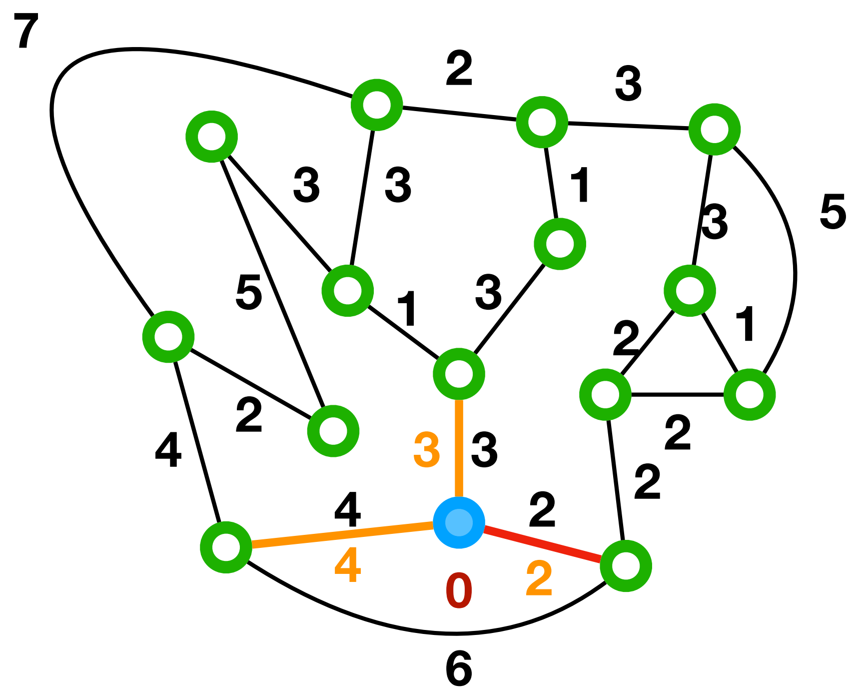
# Example



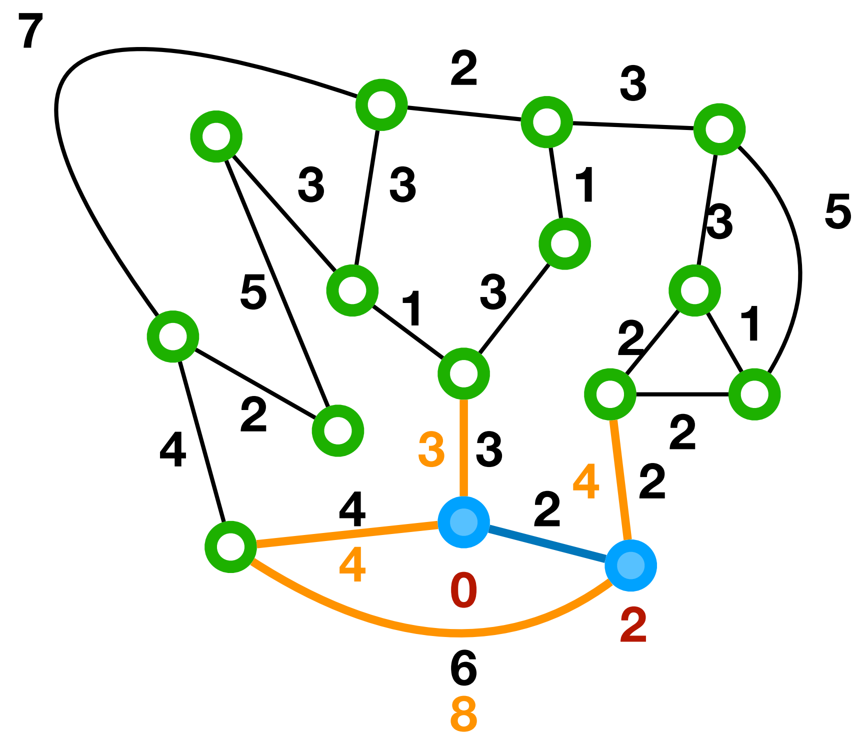
# Example



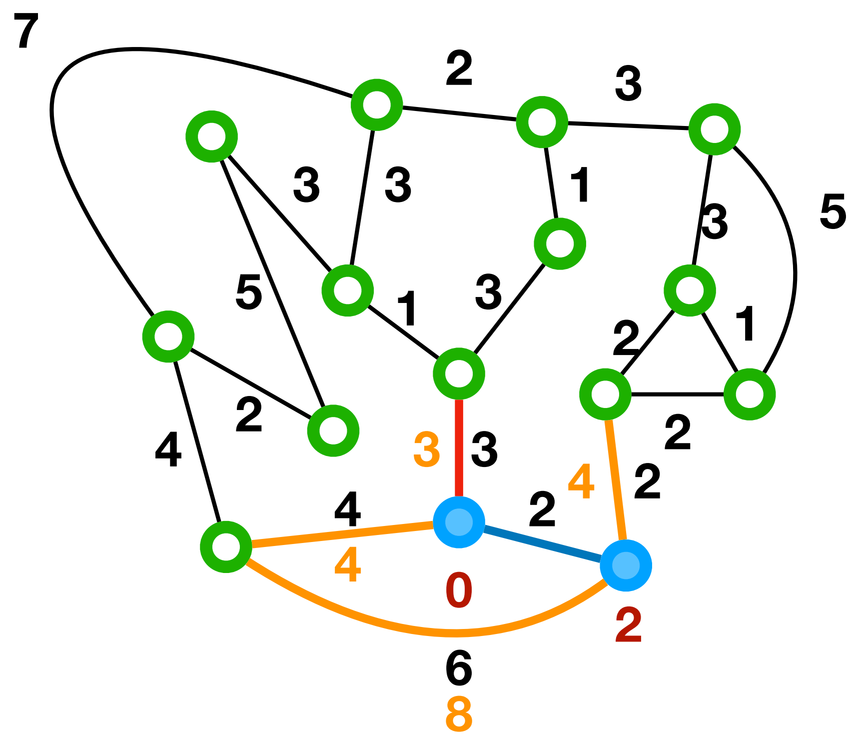
# Example



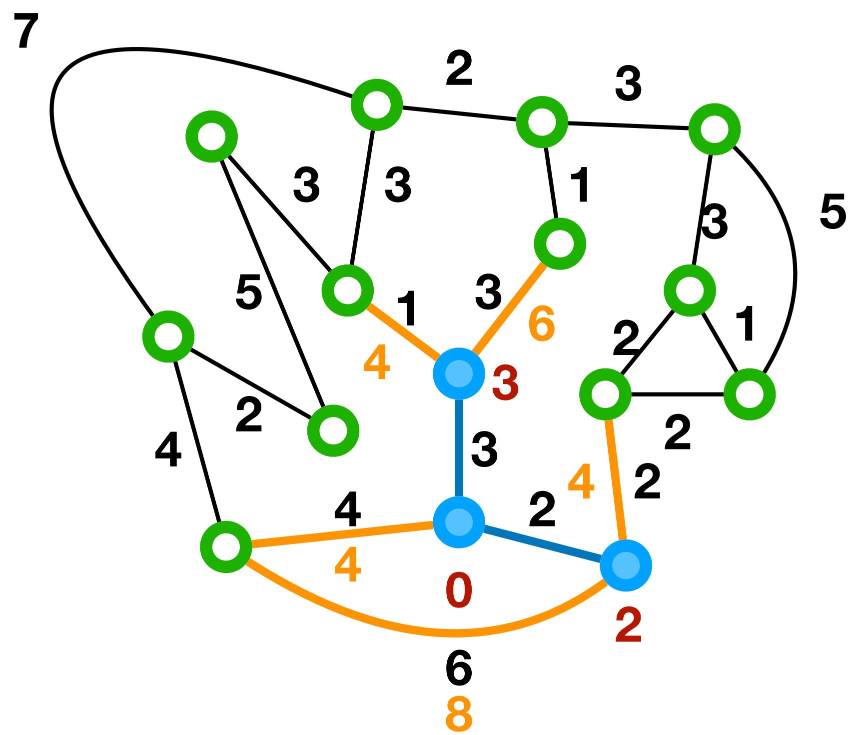
# Example



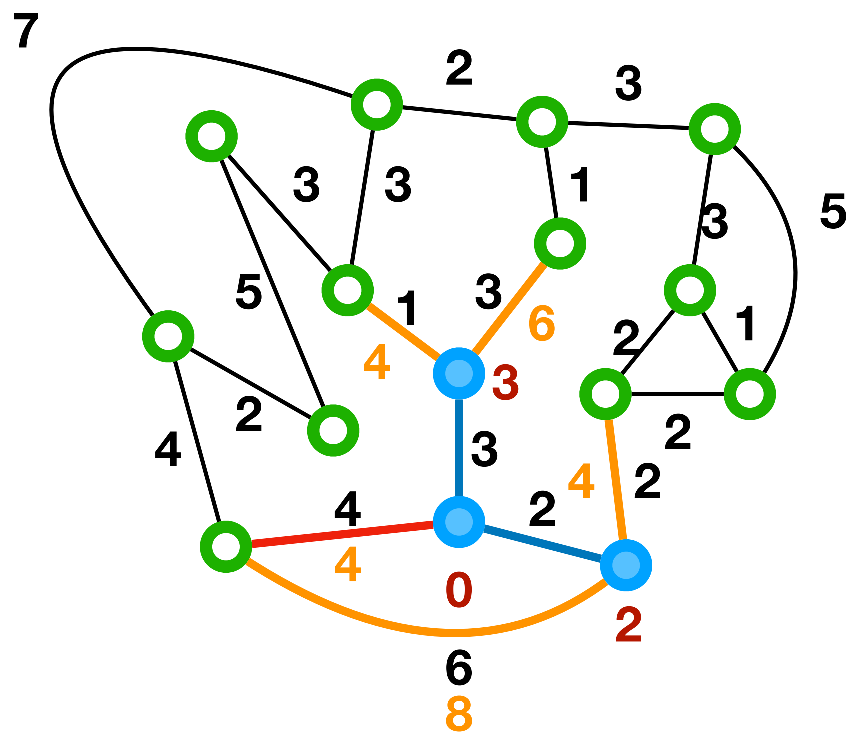
# Example



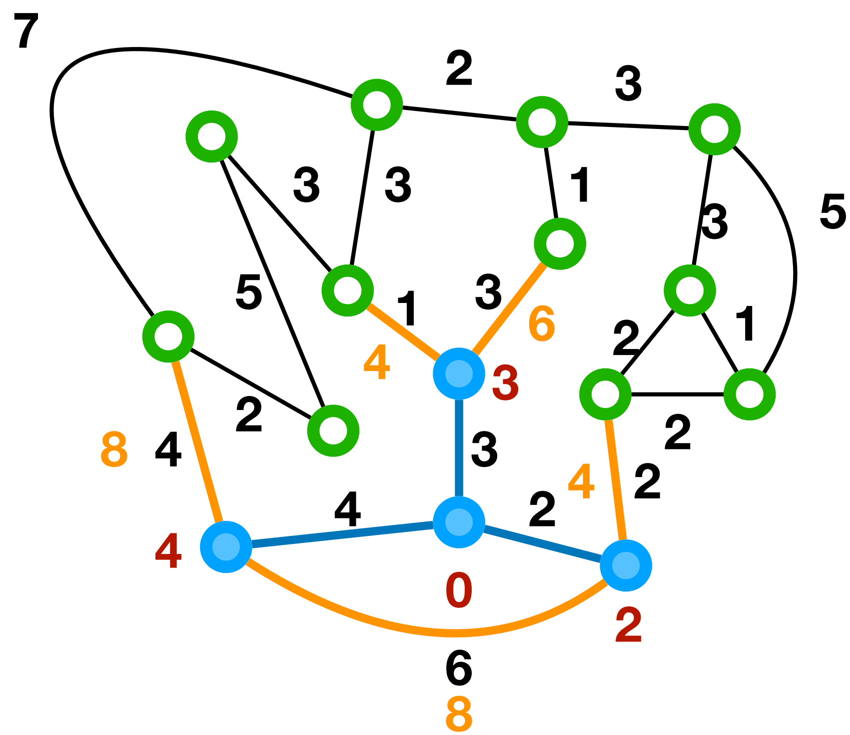
# Example



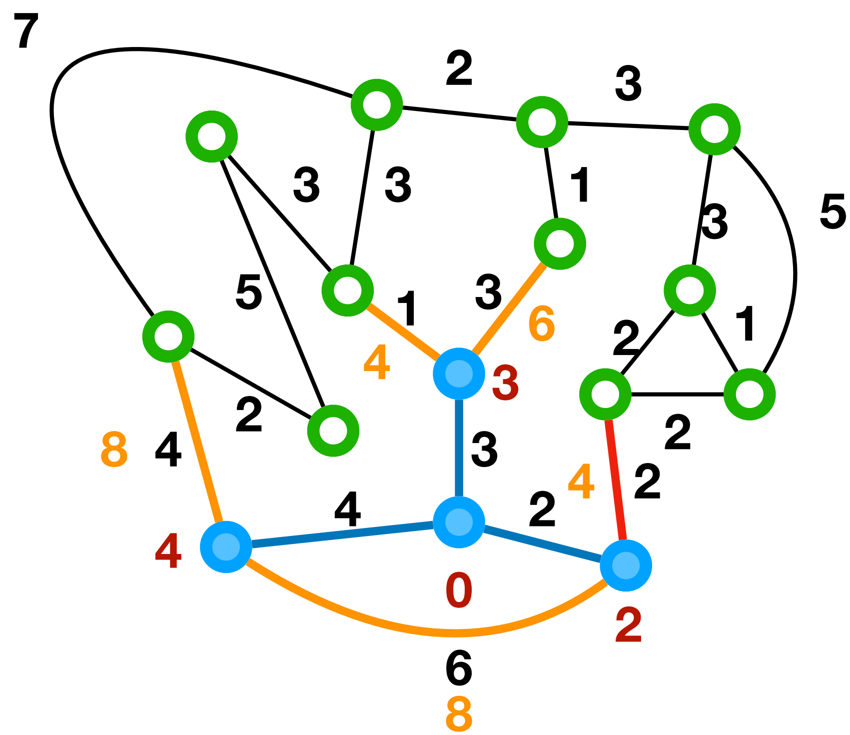
# Example



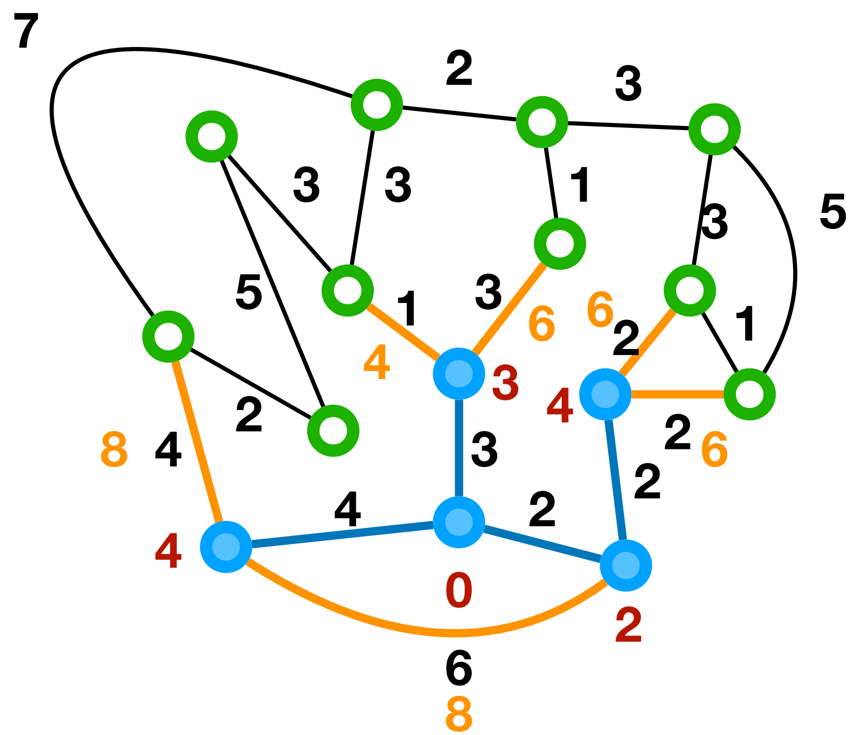
# Example



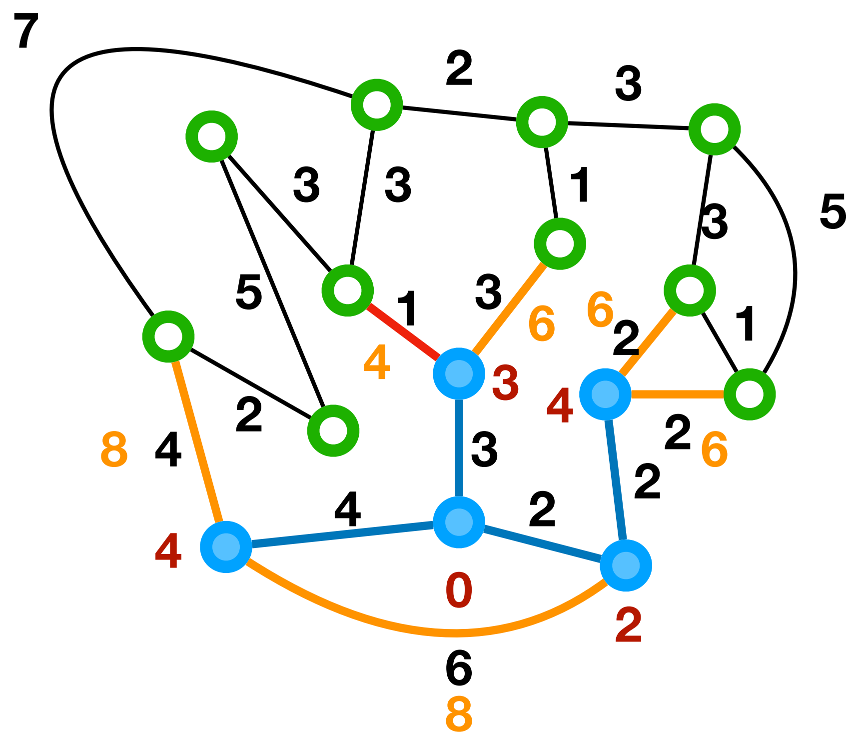
# Example



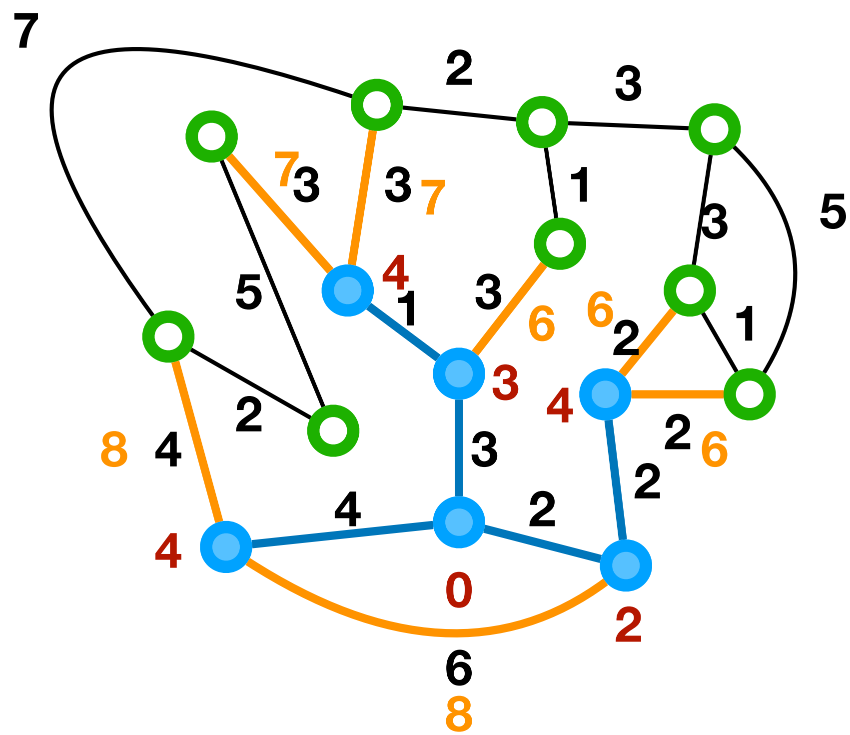
# Example



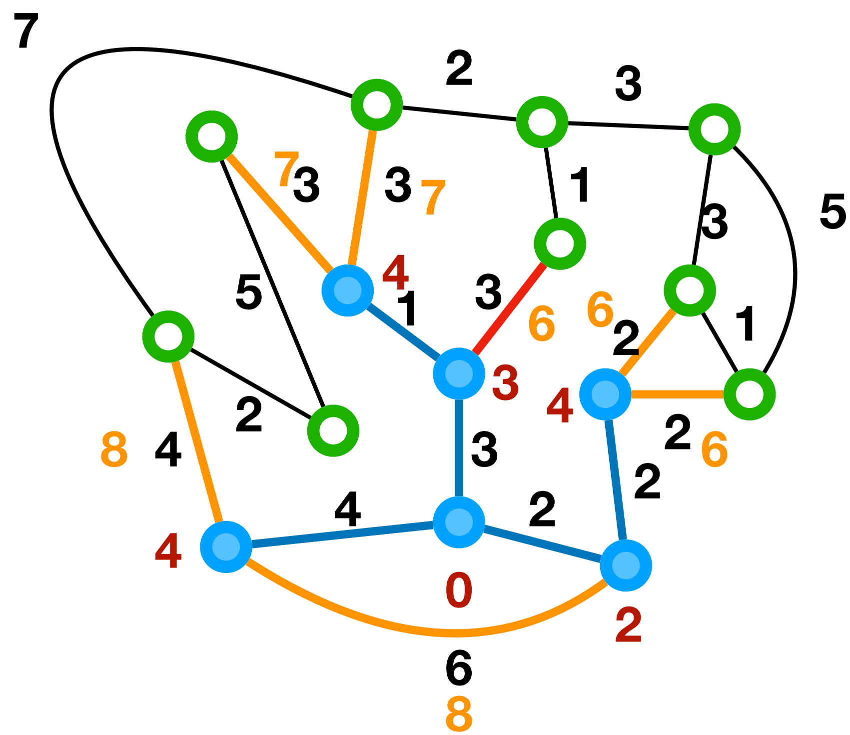
# Example



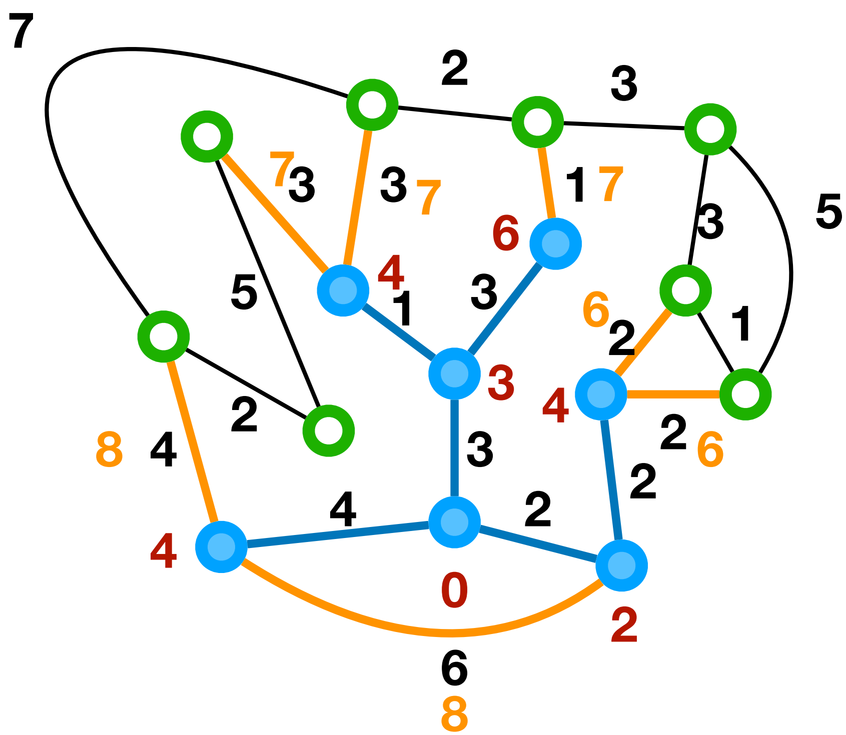
# Example



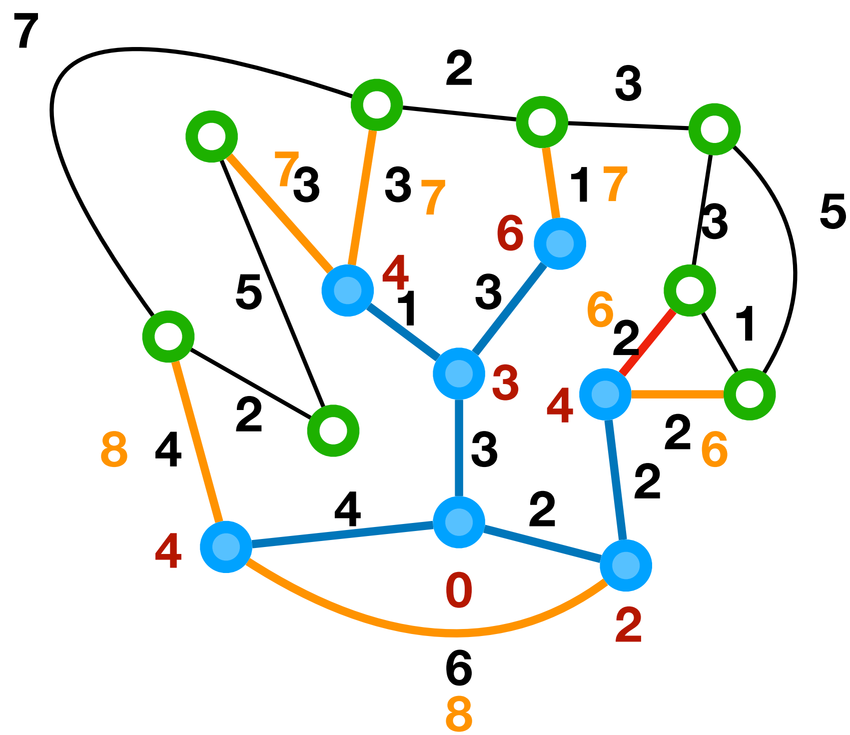
# Example



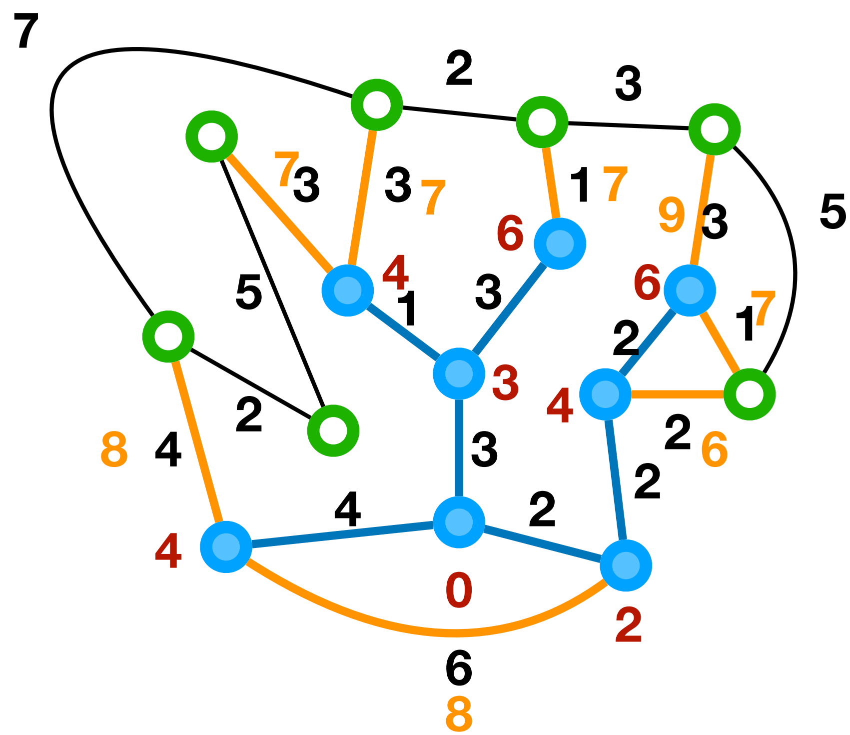
# Example



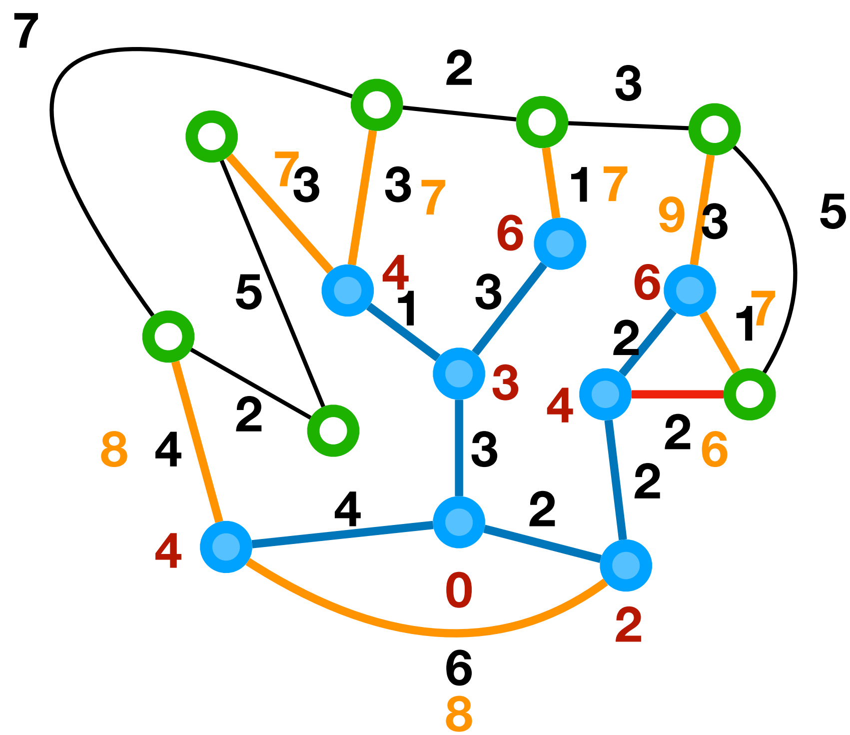
# Example



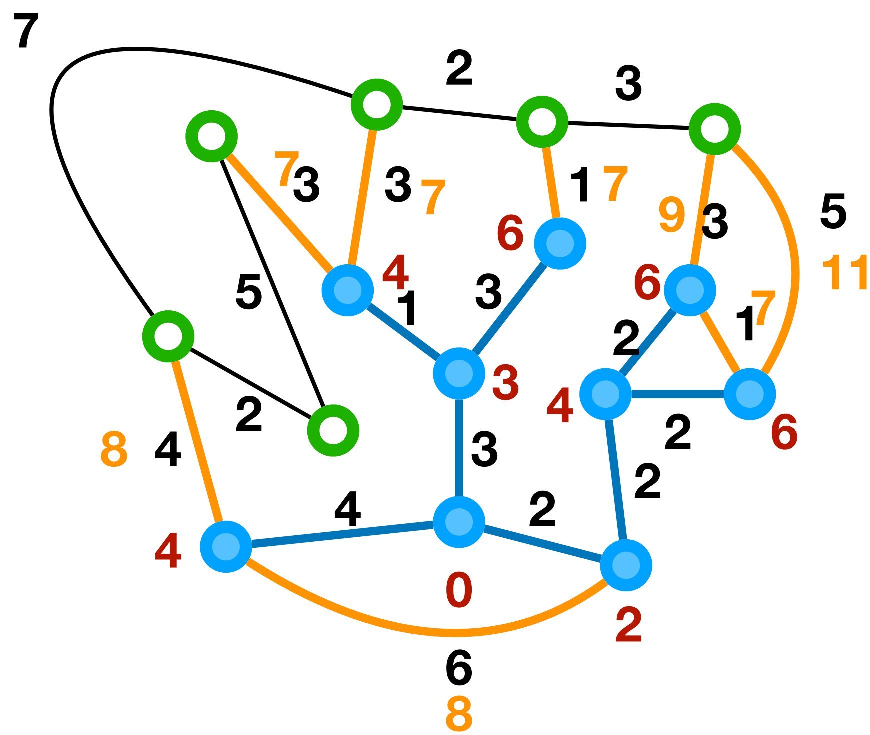
# Example



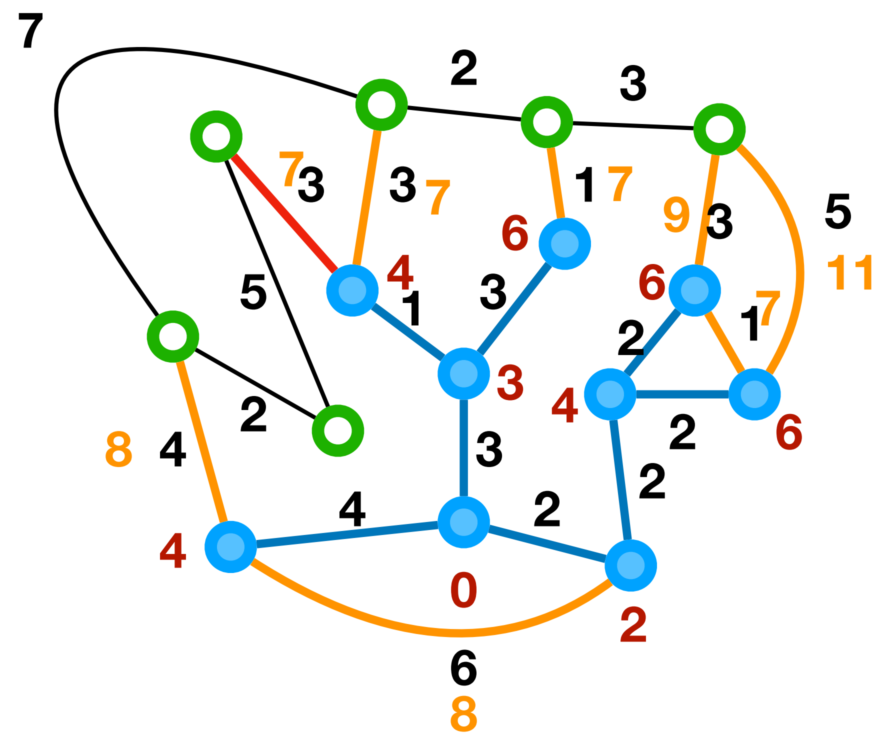
# Example



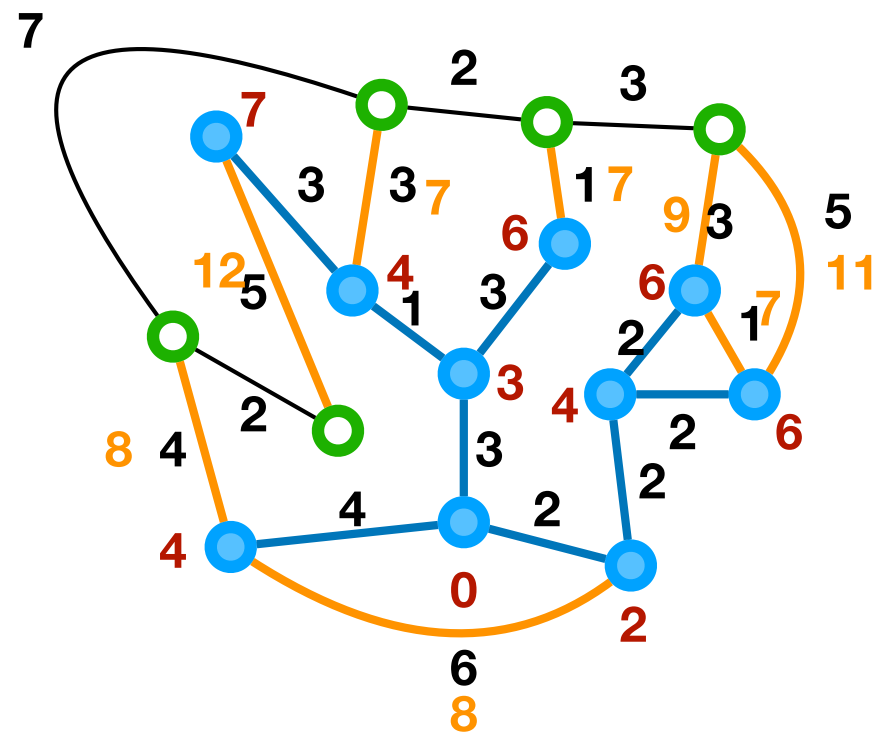
# Example



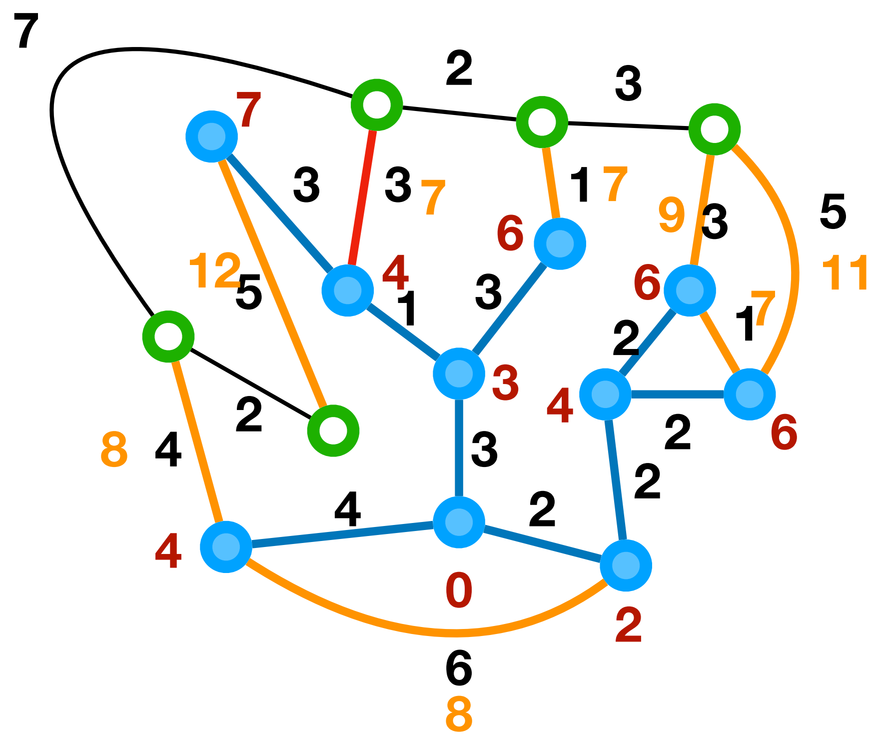
# Example



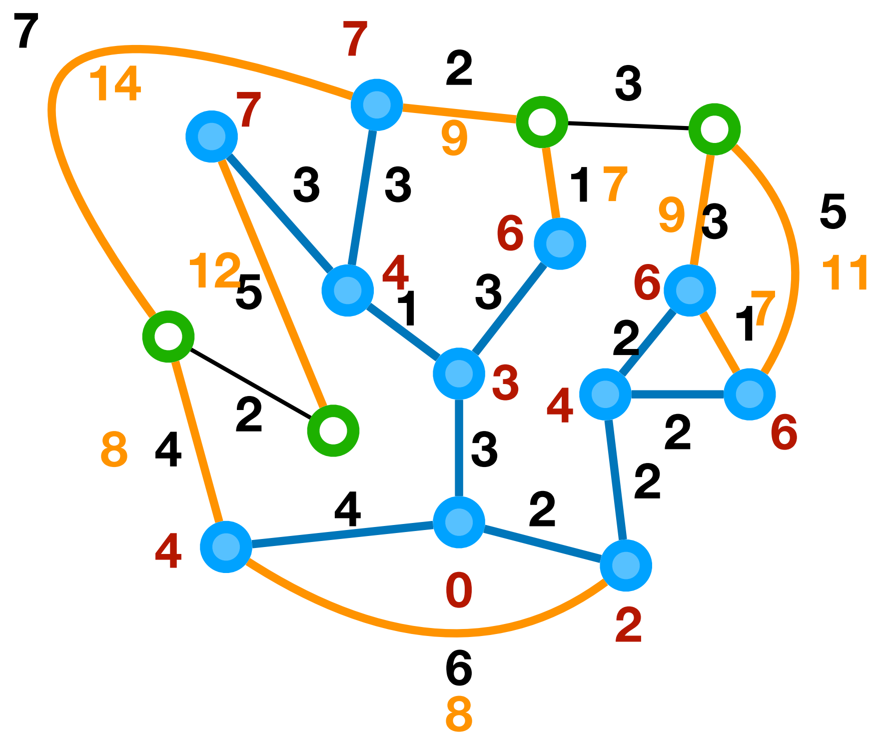
# Example



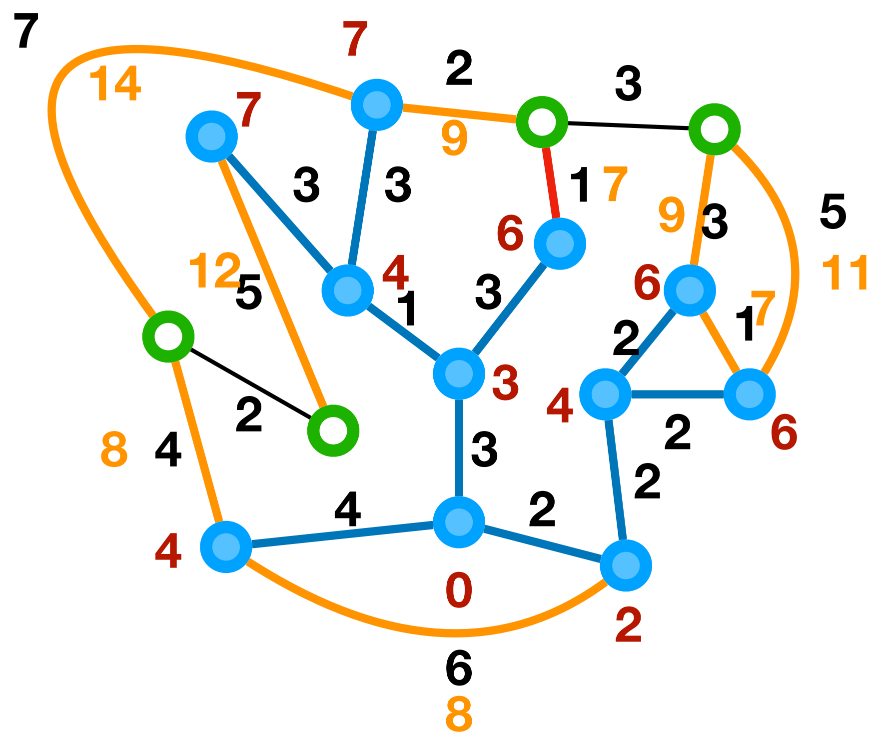
# Example



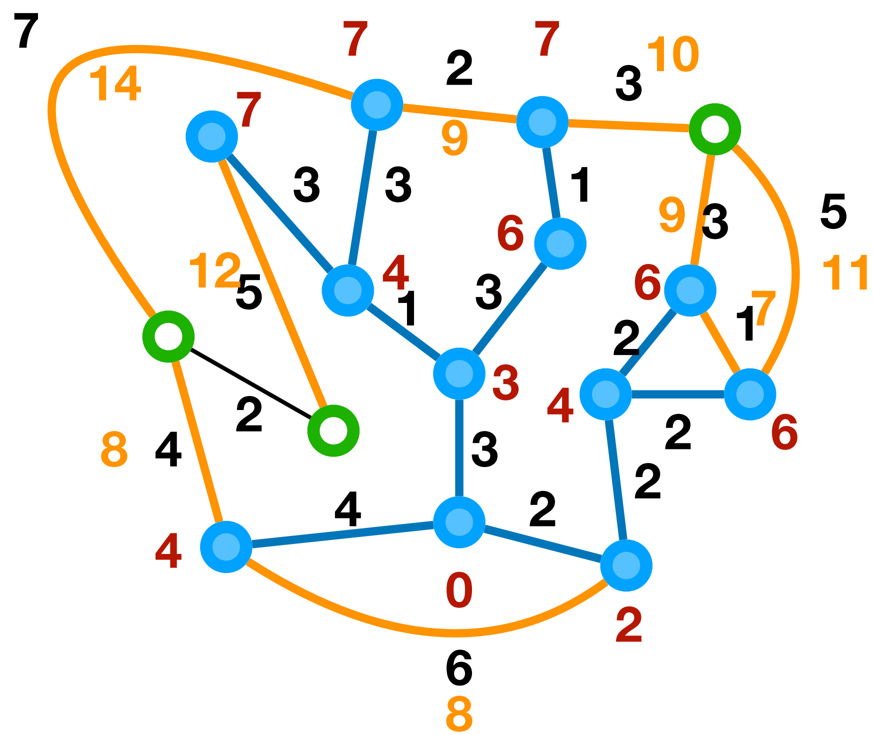
# Example



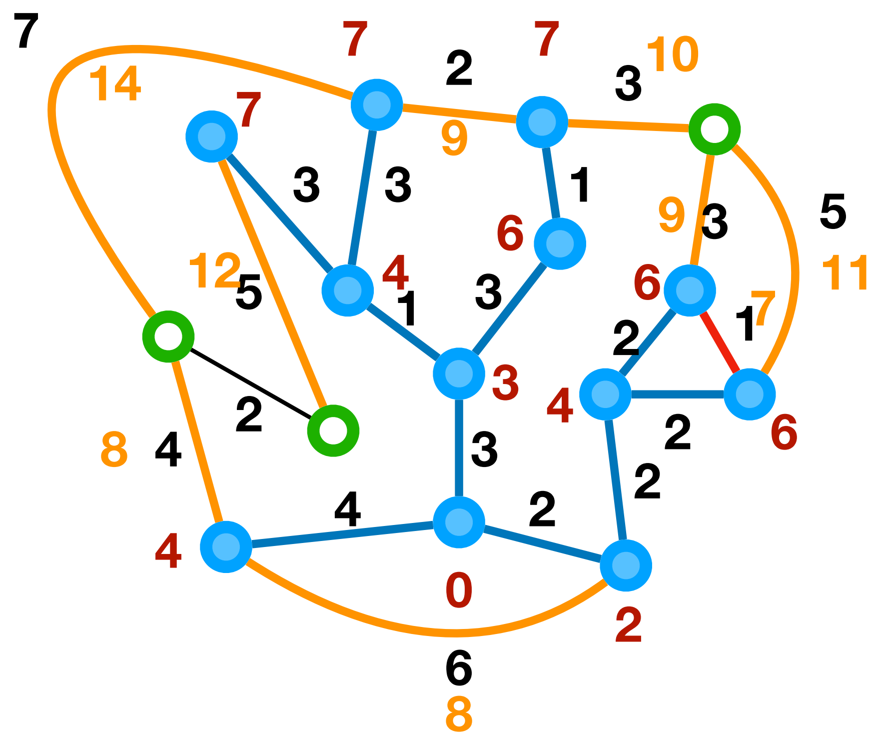
# Example



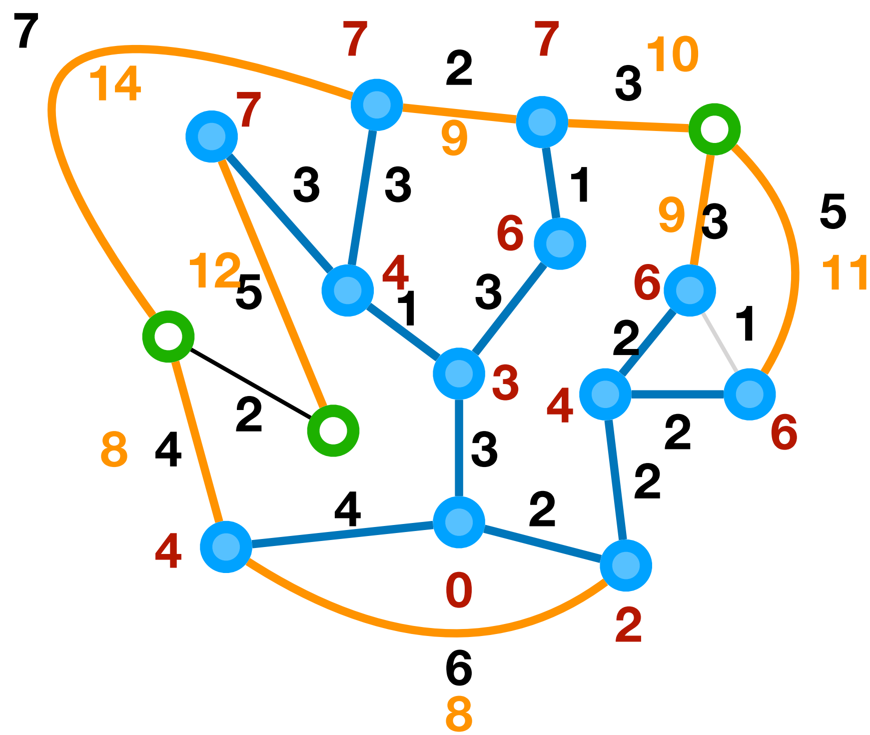
# Example



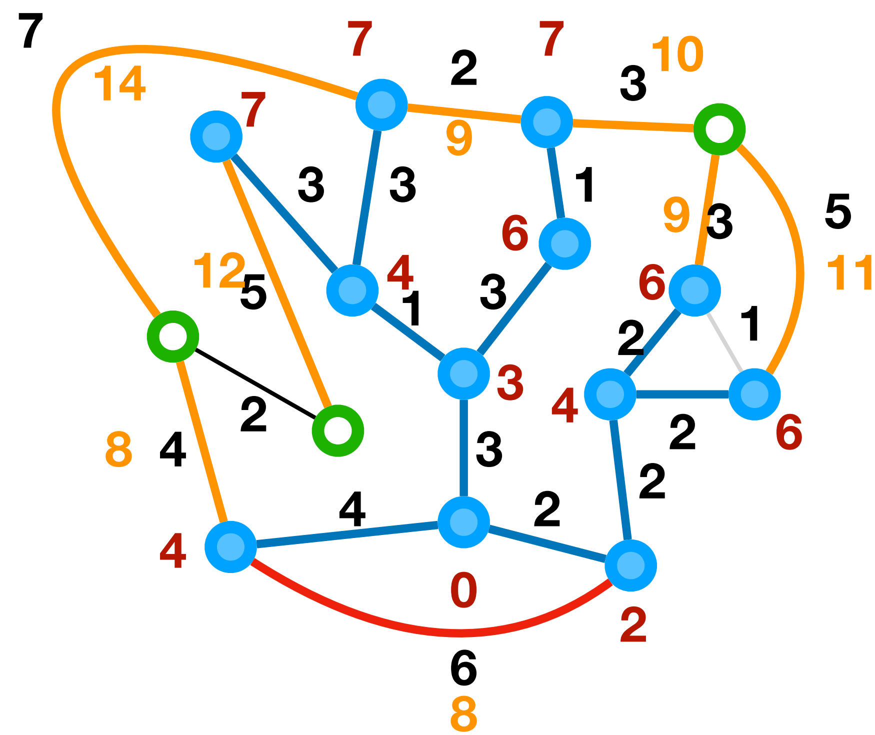
# Example



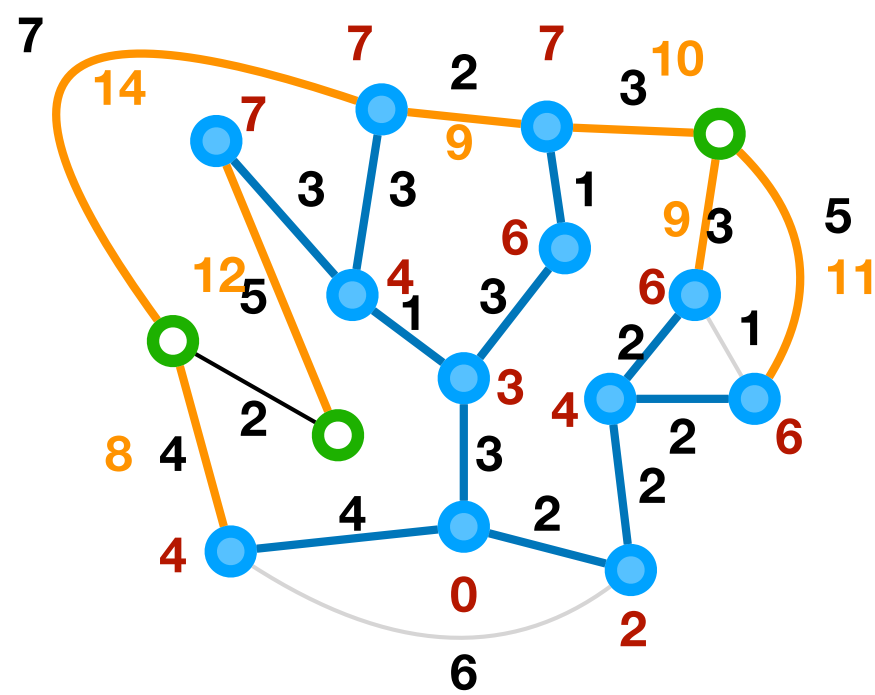
# Example



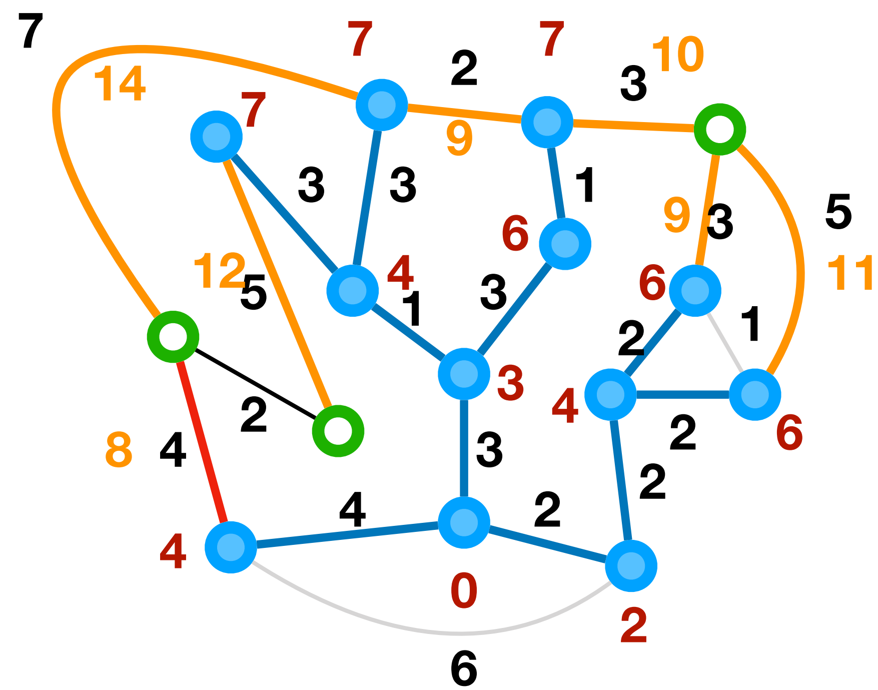
# Example



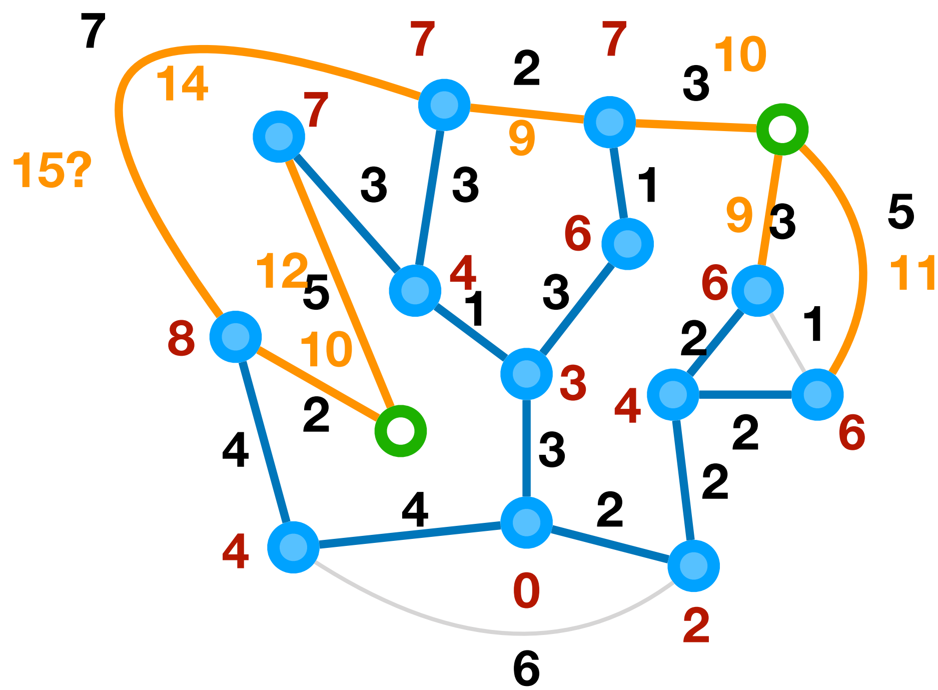
# Example



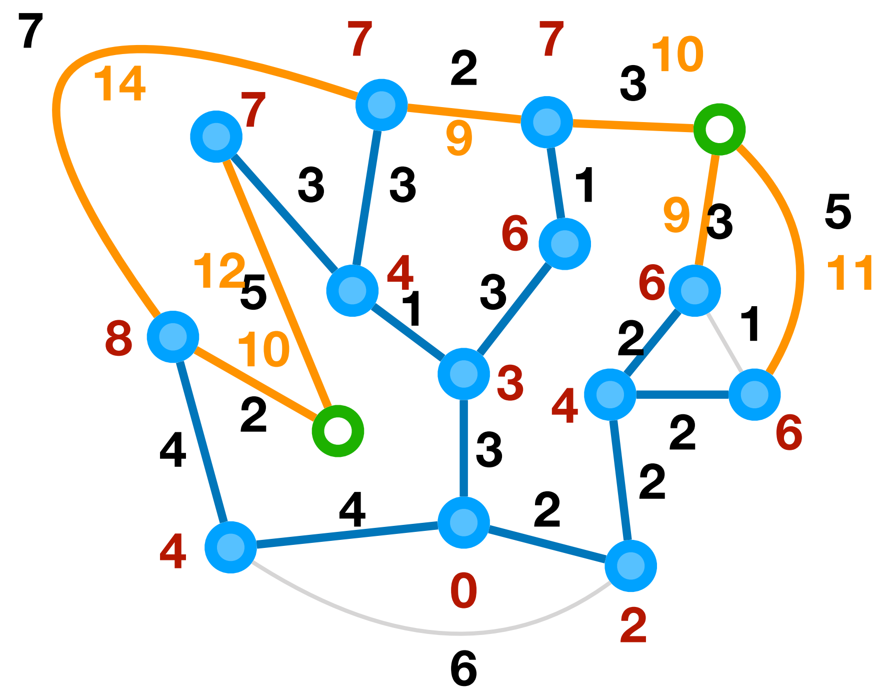
# Example



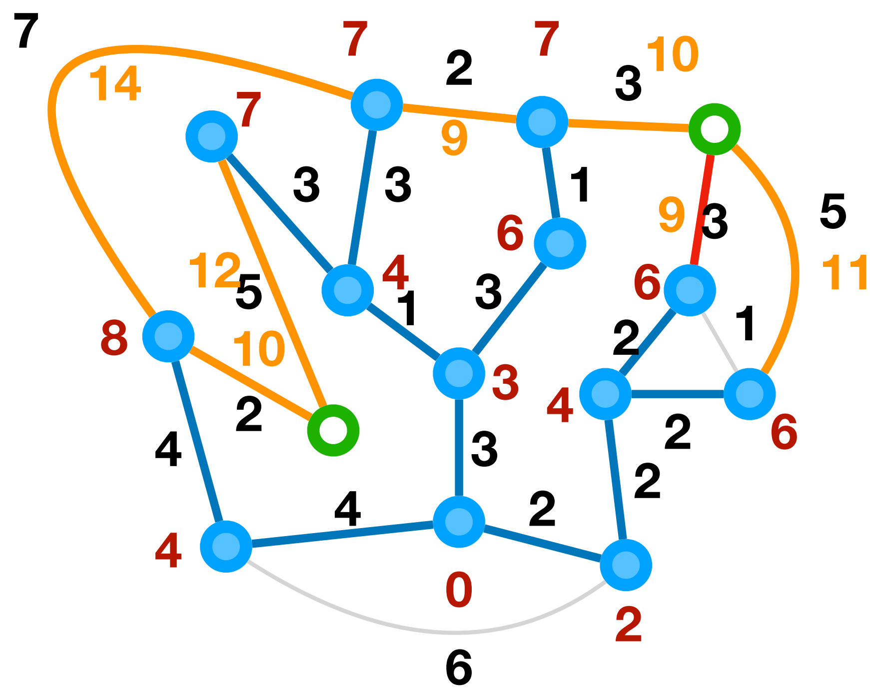
# Example



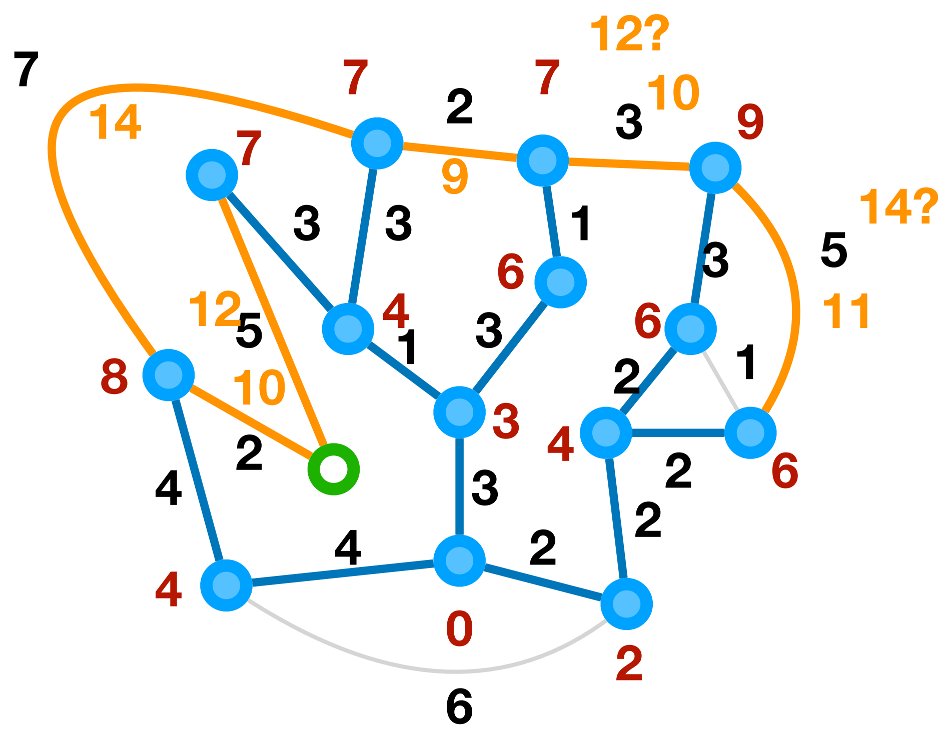
# Example



# Example

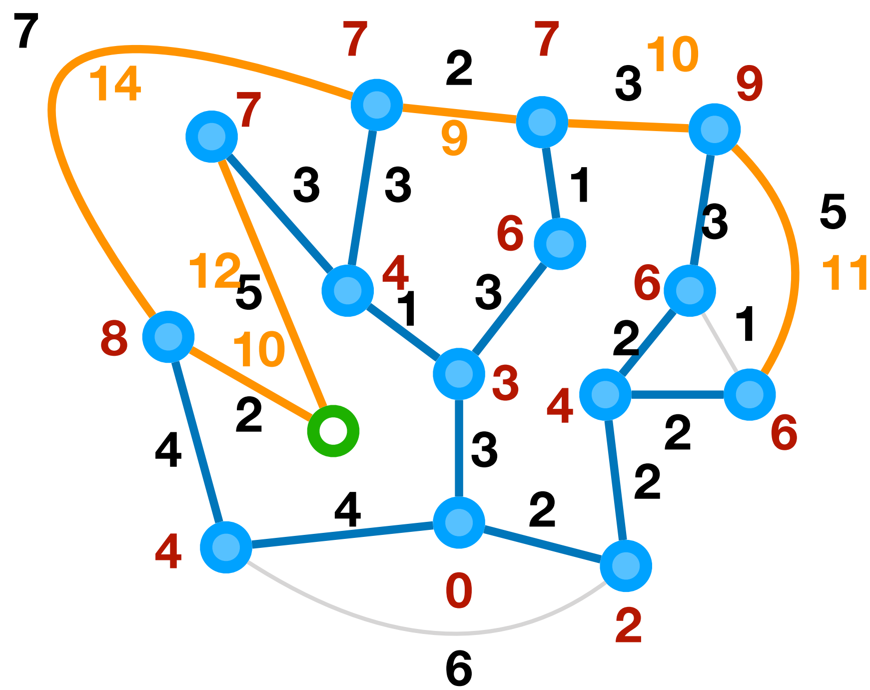


# Example

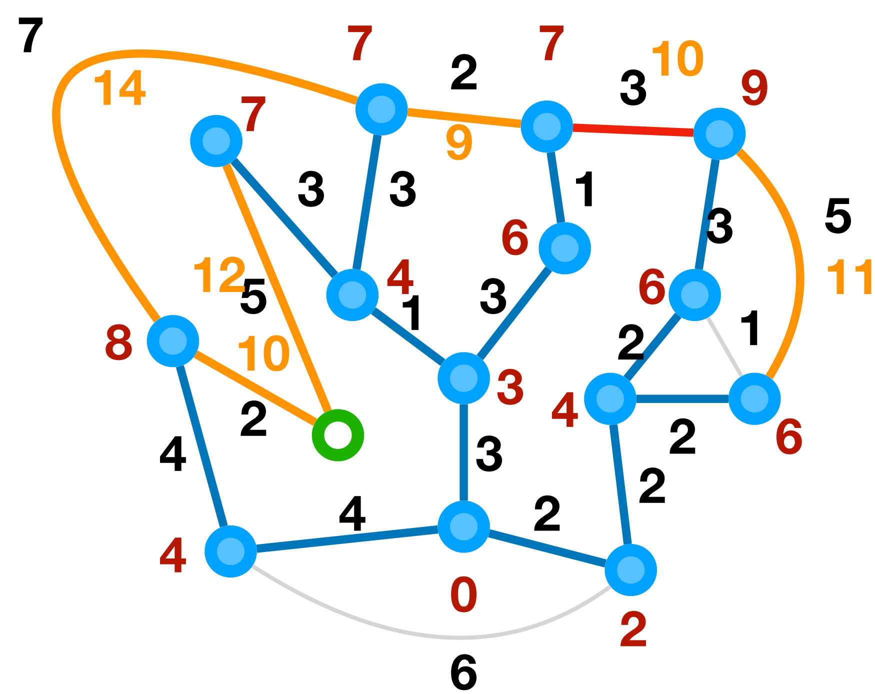


7
14
7
7
2
7
12?
3
10
9
9
3
3
1
14?
5
6
3
12
5
4
3
6
1
11
1
8
10
2
3
4
4
2
2
6
4
3
2
4
2
0
2
6

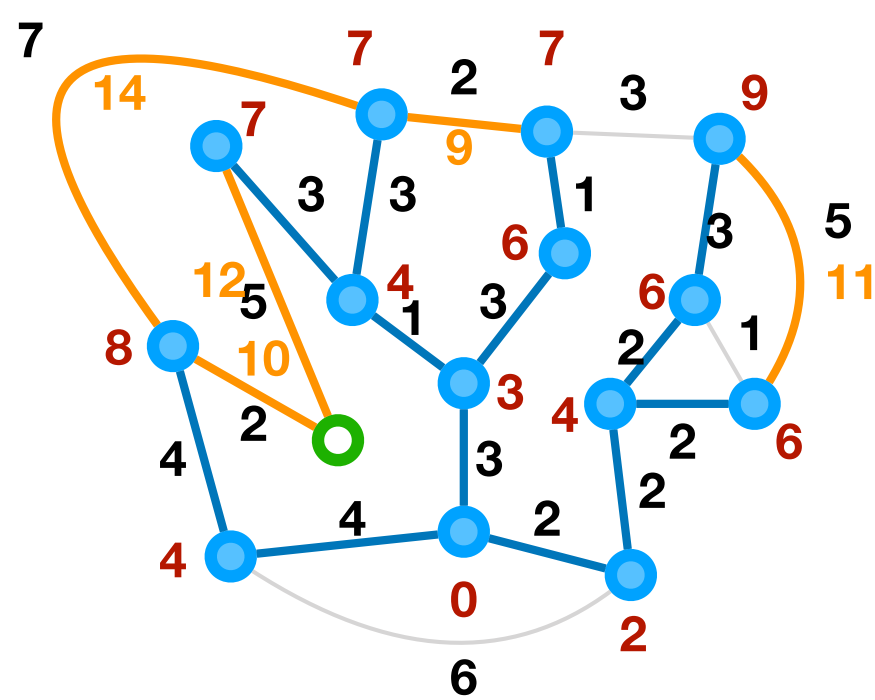
# Example



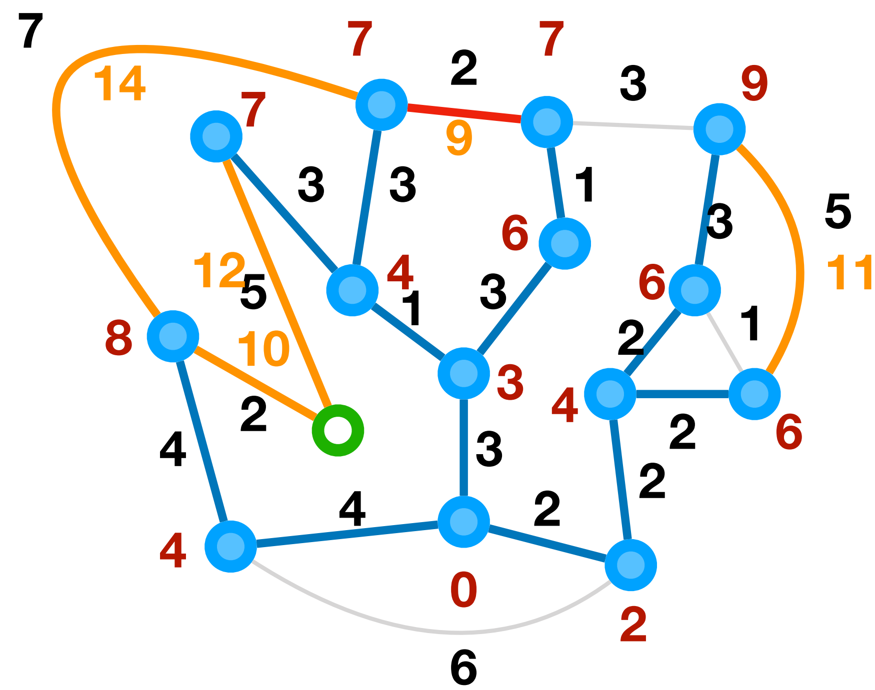
# Example



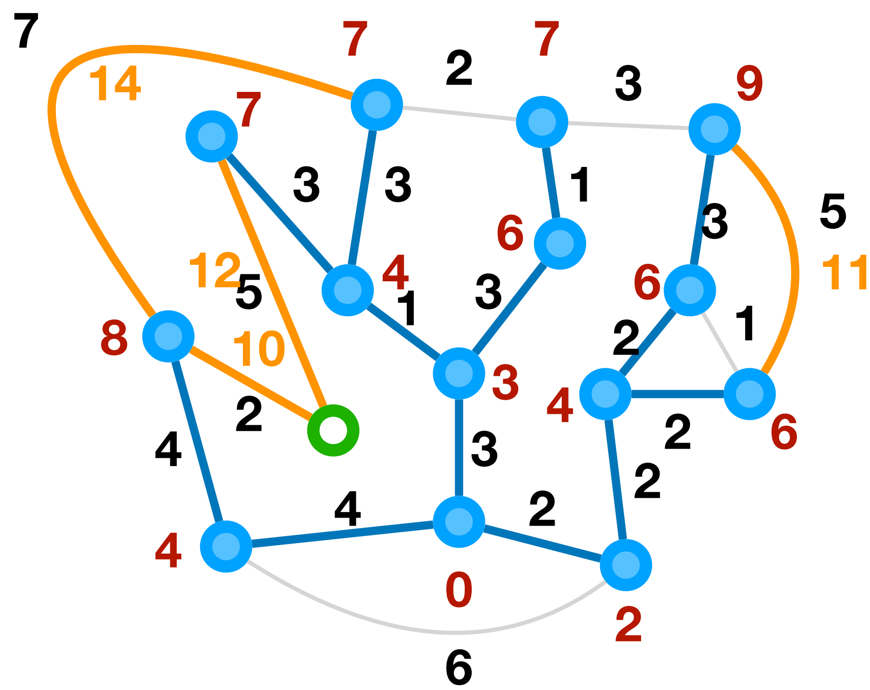
# Example



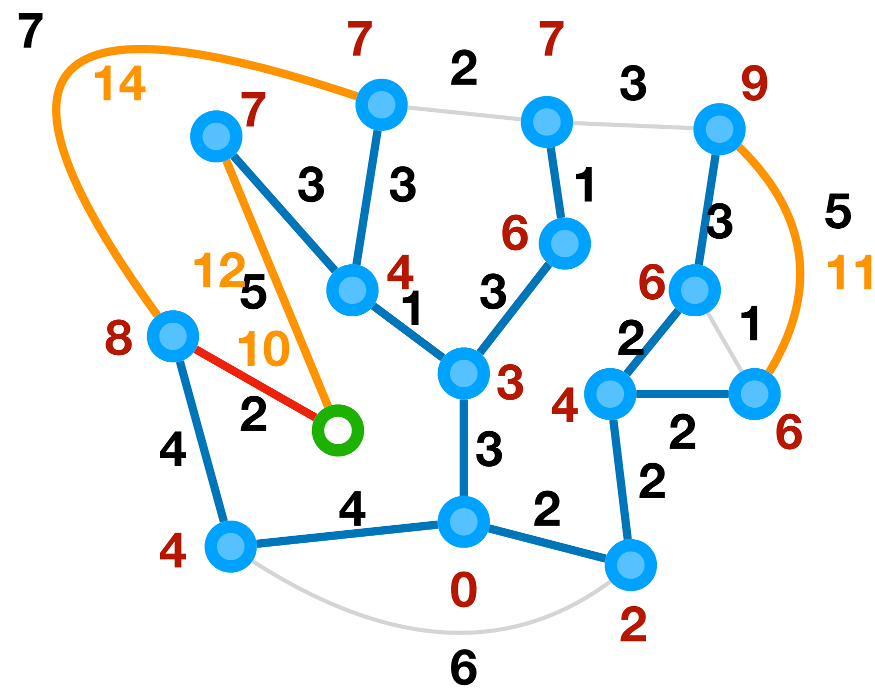
# Example



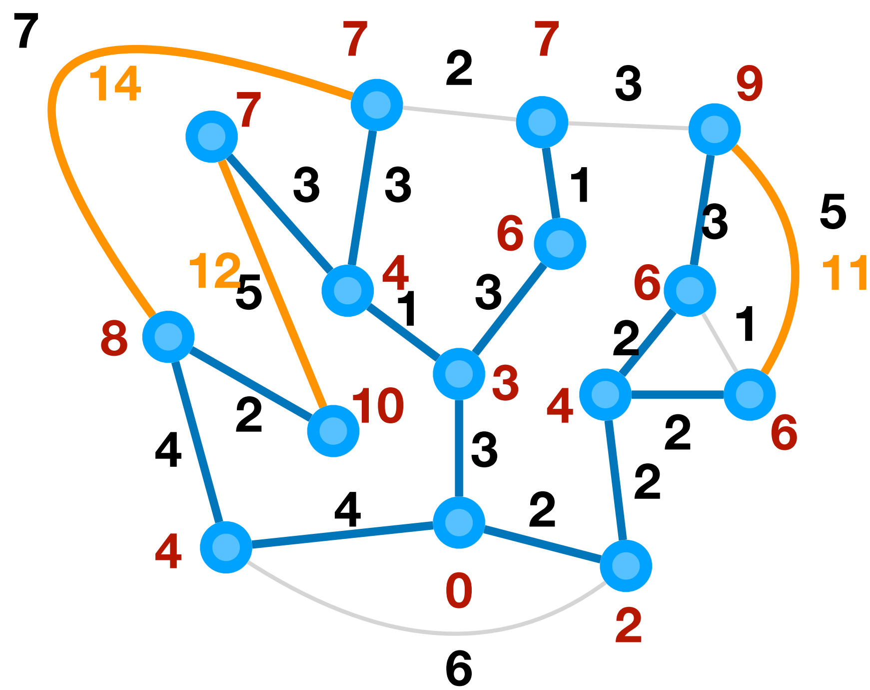
# Example



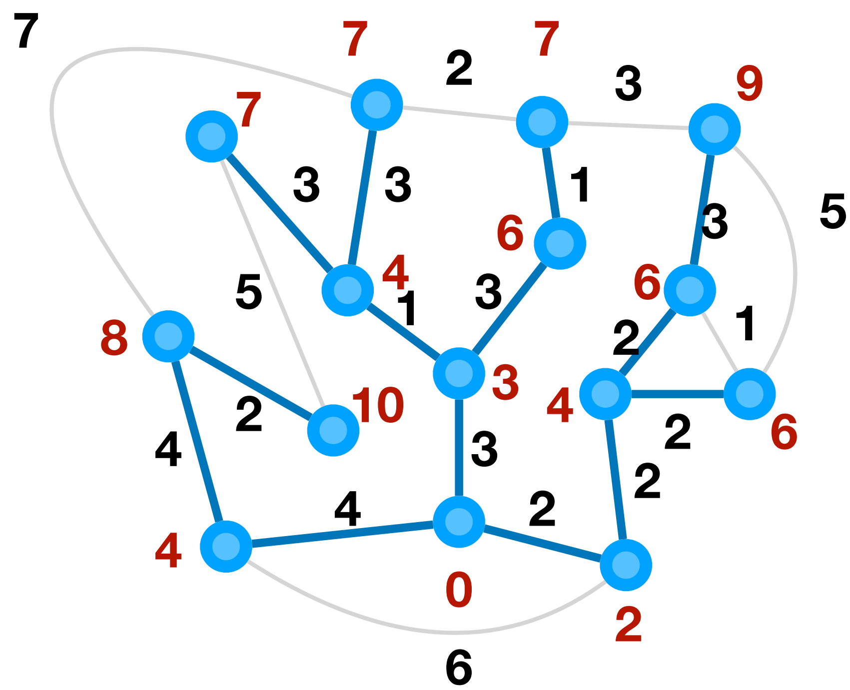
# Example



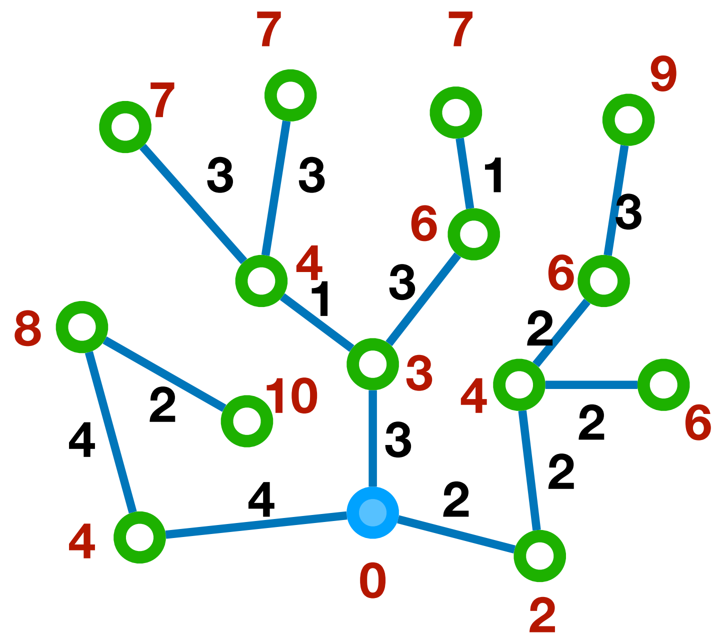
# Example



# Example



# Example



# Proof of Correctness

- Let mark[1:n] = false
- Let mark[s] = true, d[s]=0 and H be edges incident on s
- While H is not empty:
  - Remove e=(u,v) with minimum value of $d[u] + w_{uv}$ from H
  - If mark[v] = true, go to next iteration of the while-loop
  - Otherwise, set $d[v] = d[u] + w_{uv}$ and mark[v] = true, and insert all edges e incident on v with value $d[v] + w_e$ to H

- Define C = marked vertices
- **Inductive statement:** In each iteration of while-loop
- For $v \in C$, $dist(s, v) = d[v]$
- For $u \in C$ and $v \in V - C$, $dist(s, u) < dist(s, v)$

# Runtime

- Let mark[1:n] = false
- Let mark[s] = true, d[s]=0 and H be edges incident on s
- While H is not empty:
  - Remove e=(u,v) with minimum value of $d[u] + w_{uv}$ from H
  - If mark[v] = true, go to next iteration of the while-loop
  - Otherwise, set $d[v] = d[u] + w_{uv}$ and mark[v] = true, and insert all edges e incident on v with value $d[v] + w_e$ to H

- Exactly as in Prim's algorithm using a min-heap for H
- So runtime is $O(n + m \log m)$

# Summary

# Summary

- SSSP: Single-Source Shortest Path
  - Also finds the path using a simple O(n+m) time algorithm
- Bellman-Ford Algorithm:
  - Extremely simple algorithm
  - But slow: O(nm) time only
- Dijkstra's Algorithm:
  - Faster: O(n+m logm) time