CS 344: Design and Analysis of Computer Algorithms

(Spring 2022 — Sections 5,6,7,8)

Lecture 19:
Prim's Algorithm for MST, Shortest Path Algorithms

The Minimum Spanning Tree Problem

- **Input:**
  - An undirected connected graph $G = (V, E)$
  - Positive weights on edges of G: edge e has weight $w_e > 0$
- **Output:**
  - A spanning tree T in G with minimum weight
    - Weight of T = $\sum_{e \in T} w_e$

A Generic "Algorithm" for MST

A Generic Meta-Algorithm

- Let $F = \varnothing$ be an empty forest initially
- For i = 1 to n-1 steps:
  - **Find a safe edge e for the current forest F**
  - Update F = F + e
- Output the final F as an MST

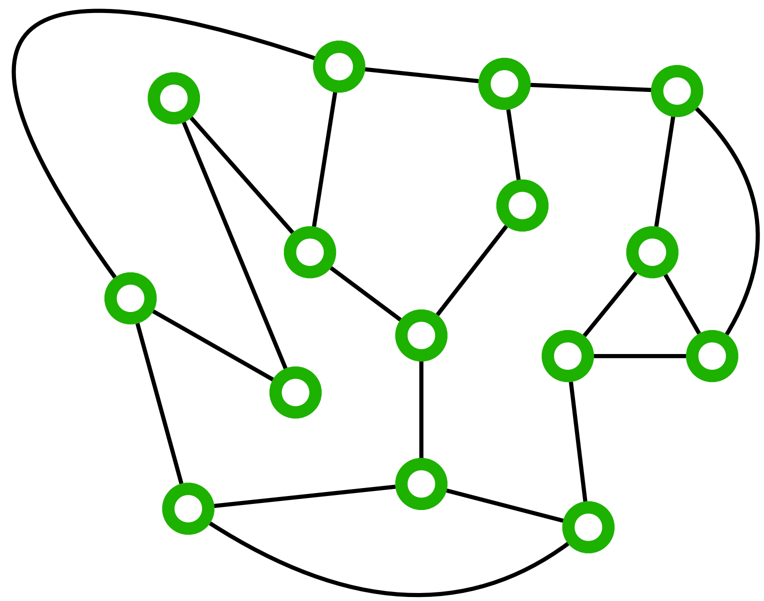
- This is NOT really an algorithm

An Approach for Finding Safe Edges

- **Theorem:**
  - Suppose F is MST-good but not a tree yet
  - Let (S,V-S) be any cut with no cut edge in F
  - Then edge e in G-F with minimum weight among cut edges of (S,V-S) is safe for F

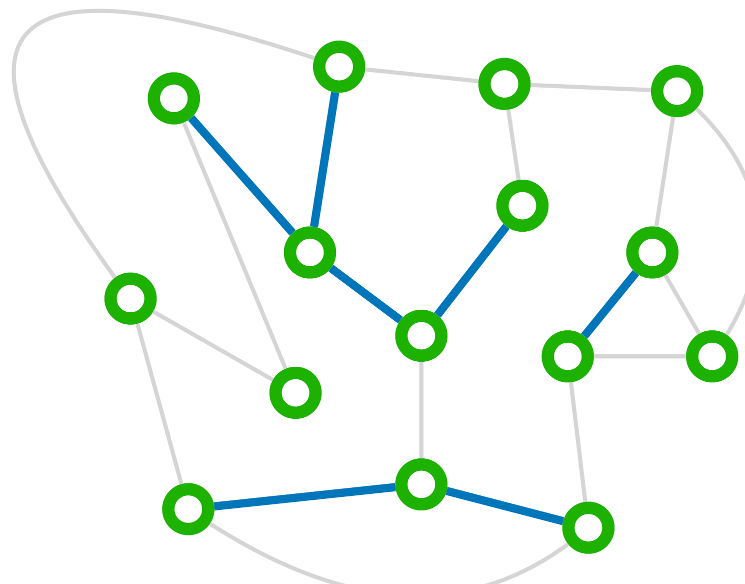
An Approach for Finding Safe Edges

- **Theorem:**
  - Suppose F is MST-good but not a tree yet
  - Let (S,V-S) be any cut with no cut edge in F
  - Then edge e in G-F with minimum weight among cut edges of (S,V-S) is safe for F



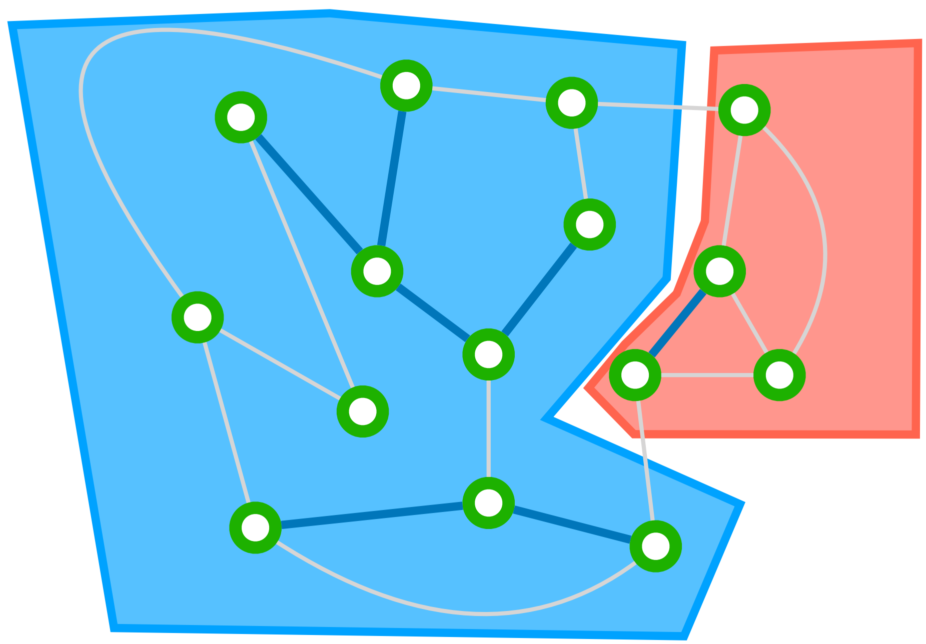
An Approach for Finding Safe Edges

- **Theorem:**
  - Suppose F is MST-good but not a tree yet
  - Let (S,V-S) be any cut with no cut edge in F
  - Then edge e in G-F with minimum weight among cut edges of (S,V-S) is safe for F



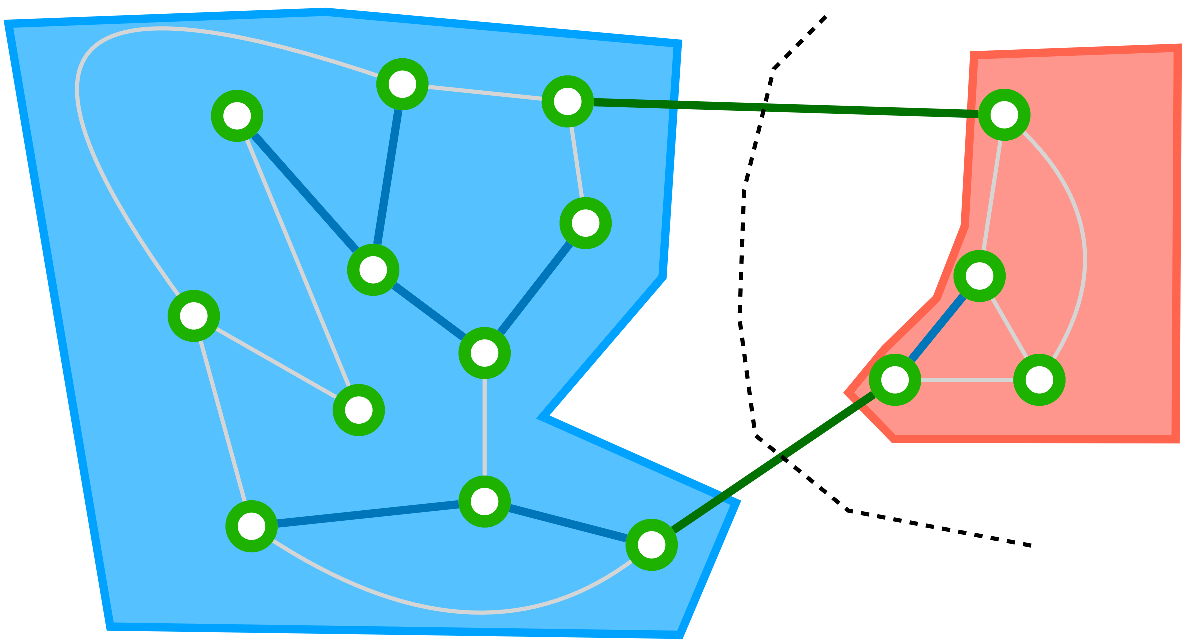
An Approach for Finding Safe Edges

- **Theorem:**
  - Suppose F is MST-good but not a tree yet
  - Let (S,V-S) be any cut with no cut edge in F
  - Then edge e in G-F with minimum weight among cut edges of (S,V-S) is safe for F



An Approach for Finding Safe Edges

- **Theorem:**
  - Suppose F is MST-good but not a tree yet
  - Let (S,V-S) be any cut with no cut edge in F
  - Then edge e in G-F with minimum weight among cut edges of (S,V-S) is safe for F

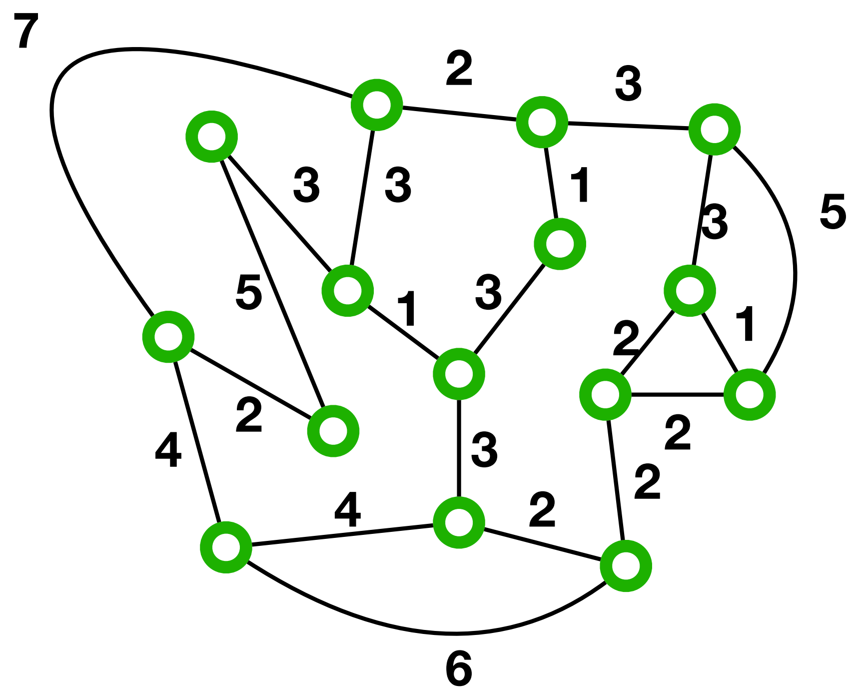


Prim’s Algorithm

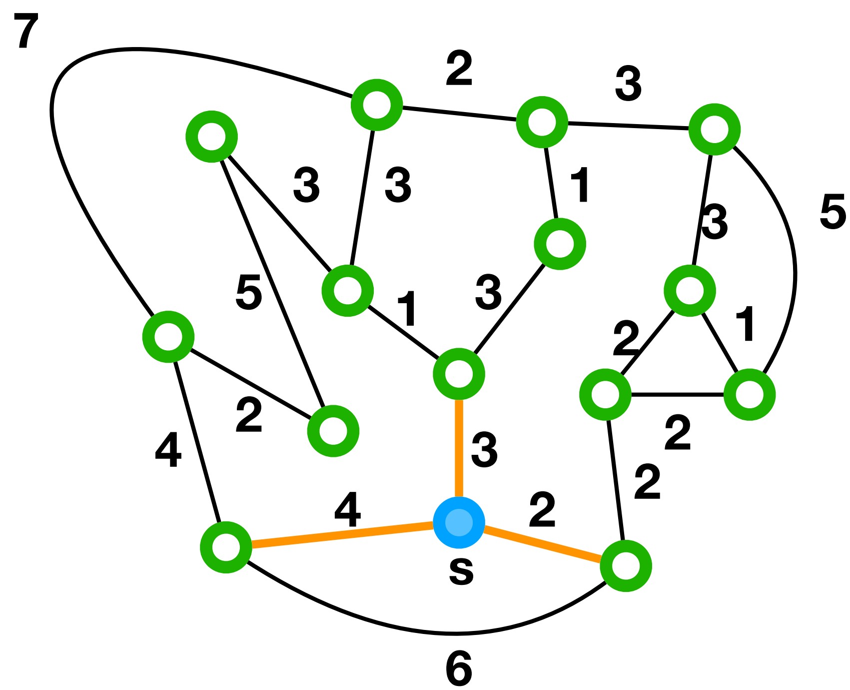
Prim's Algorithm

- Let mark[1:n] = false for all vertices and s be any arbitrary vertex
- Let F = ∅, mark[s] = true and H be the set of edges incident on s
- While H is not empty:
  - Remove the minimum weight edge e=(u,v) from H
  - If mark[u]=mark[v] = true, ignore this edge and go to the next iteration of the while-loop
  - Otherwise, let us assume by symmetry mark[u] = true only
  - Add the edge (u,v) to F and all edges incident on v to H; set mark[v] = true.
- Return F as an MST of the input graph

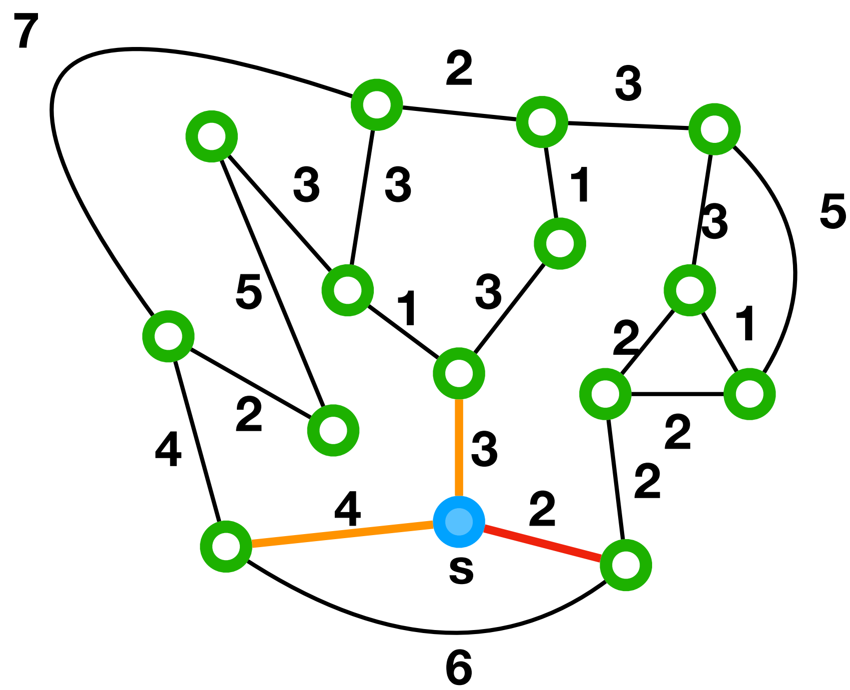
Example



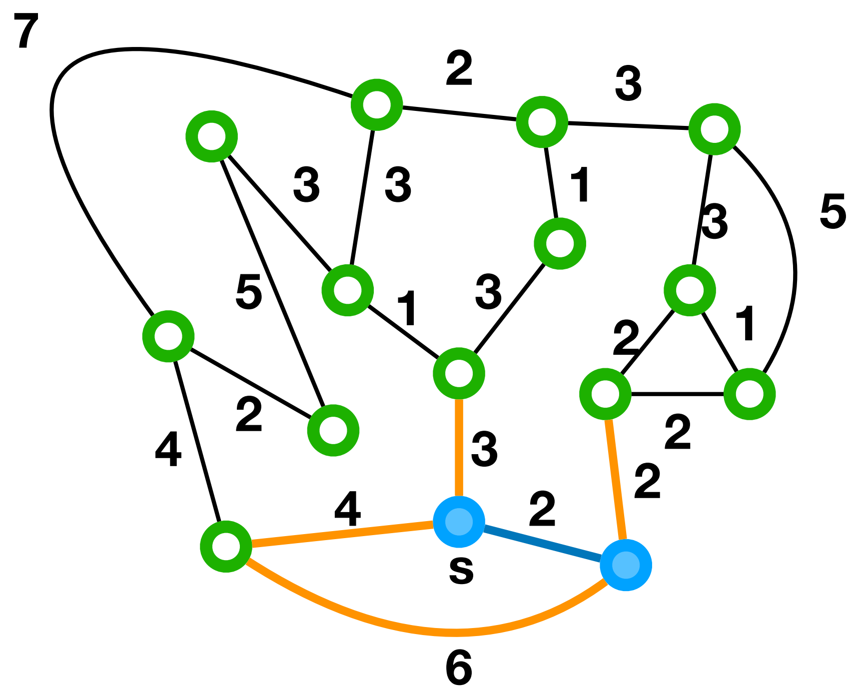
Example



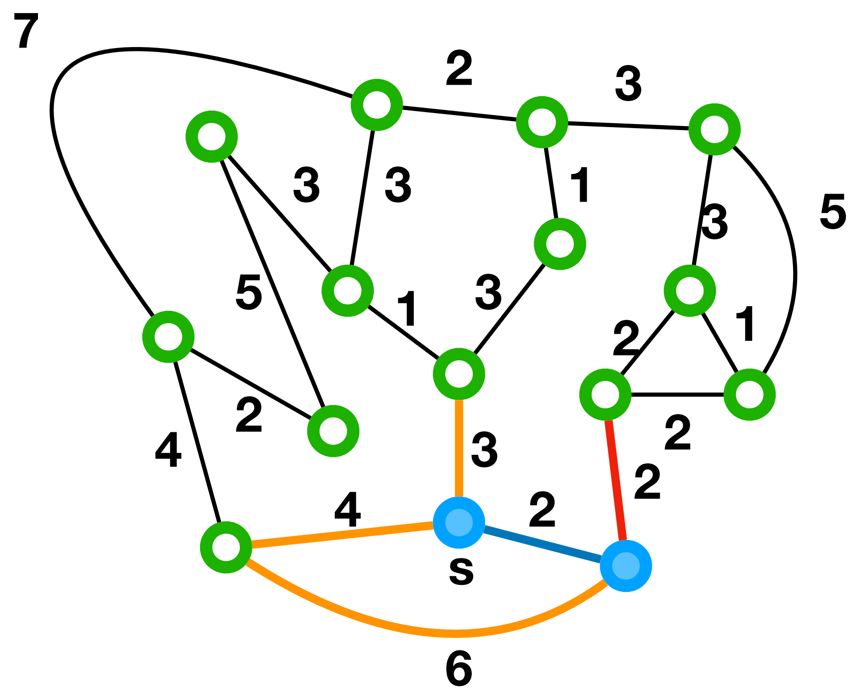
Example



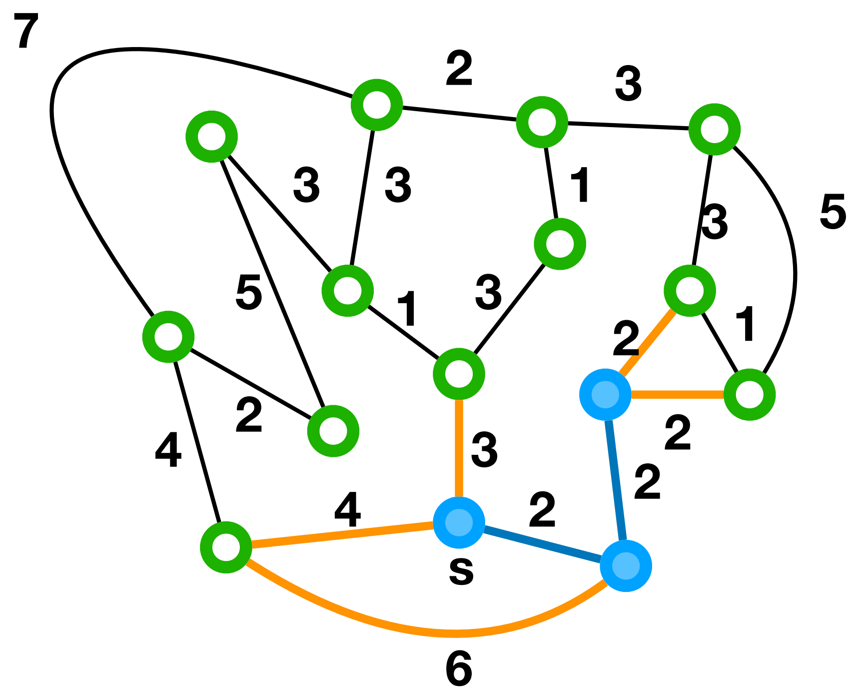
Example



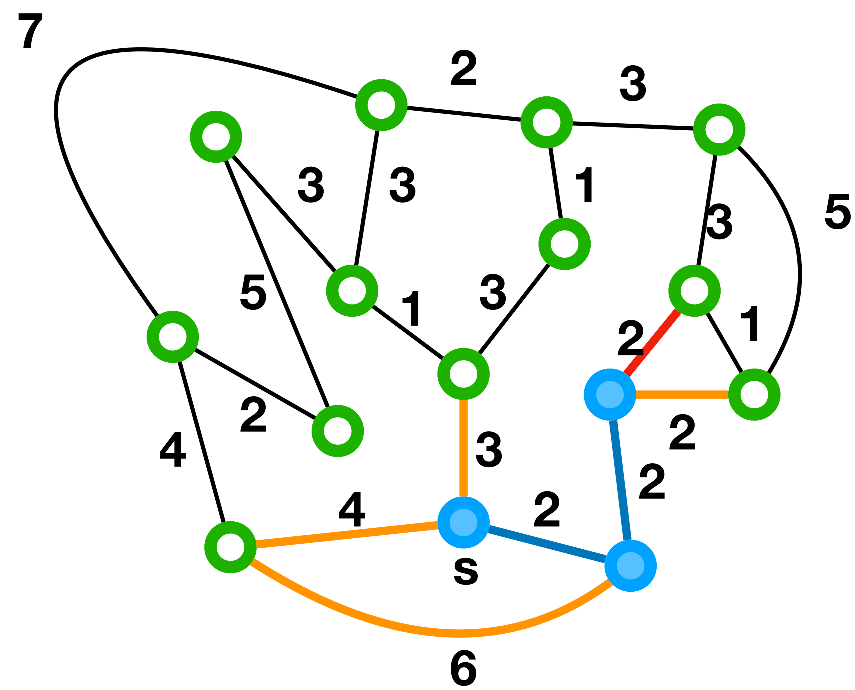
Example



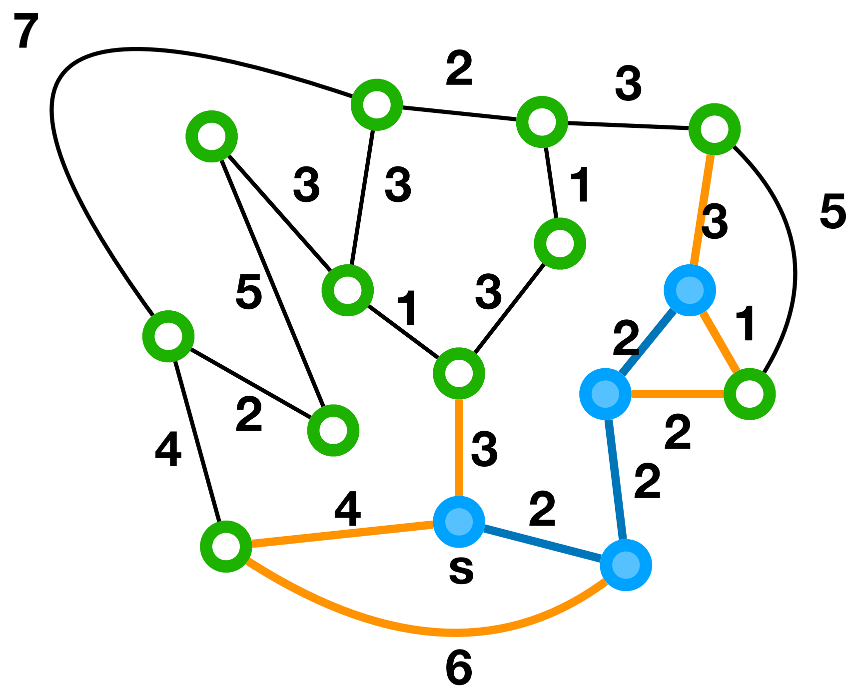
Example



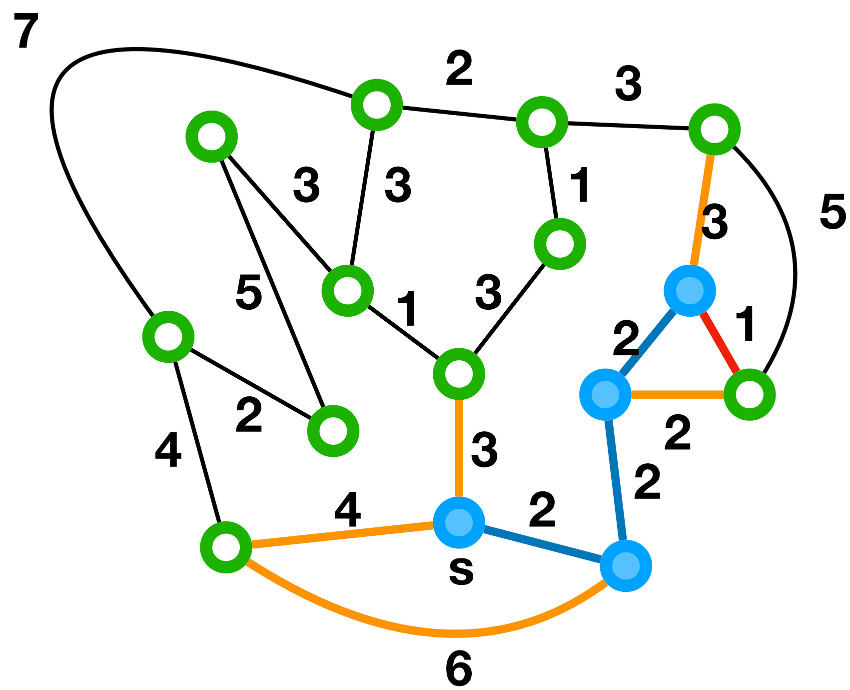
Example



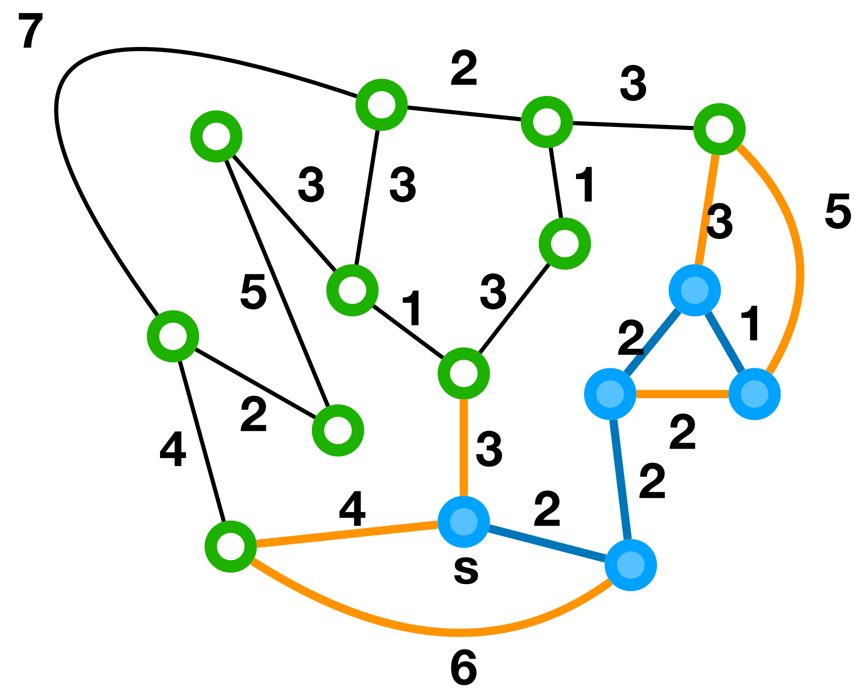
Example



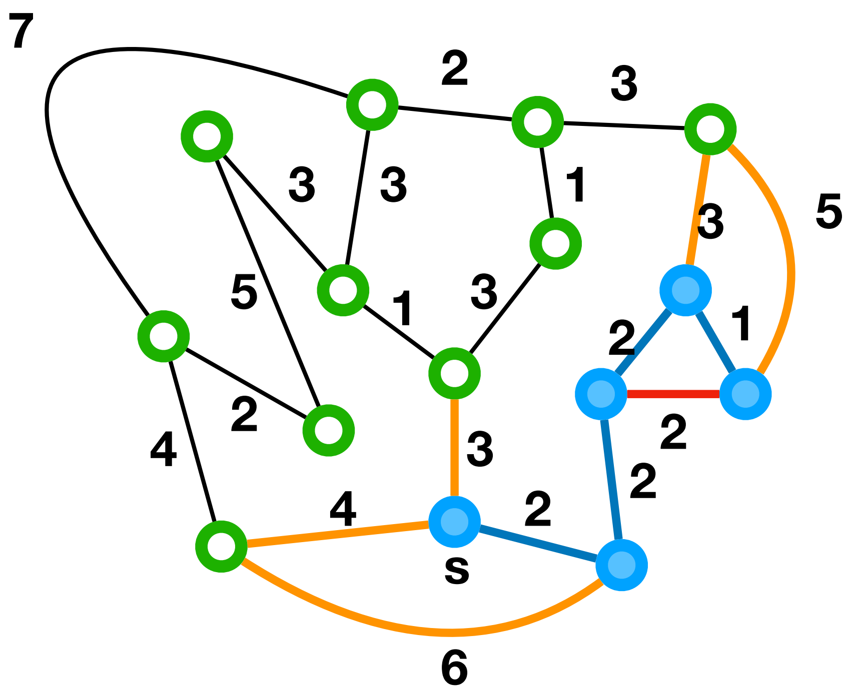
Example



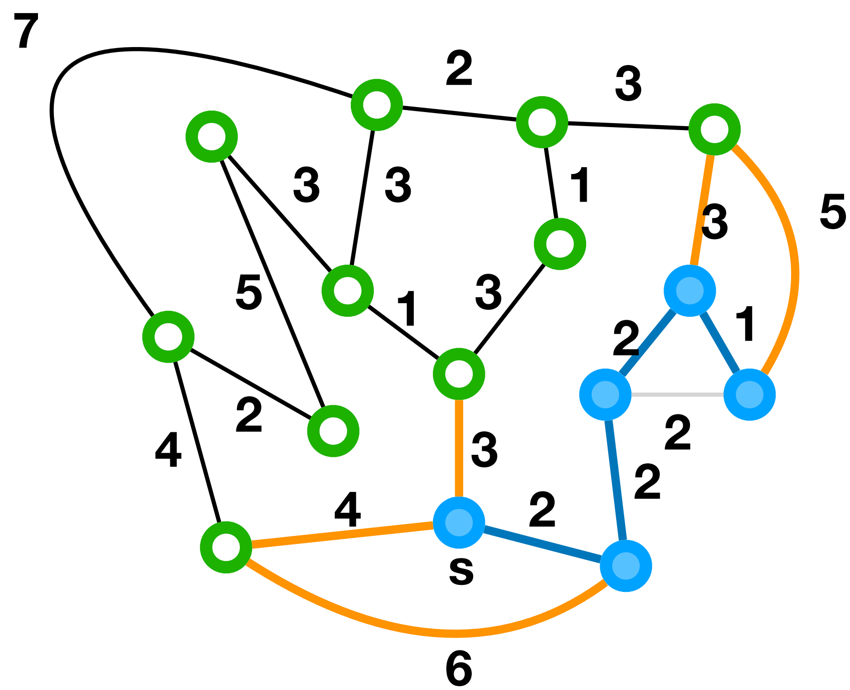
Example



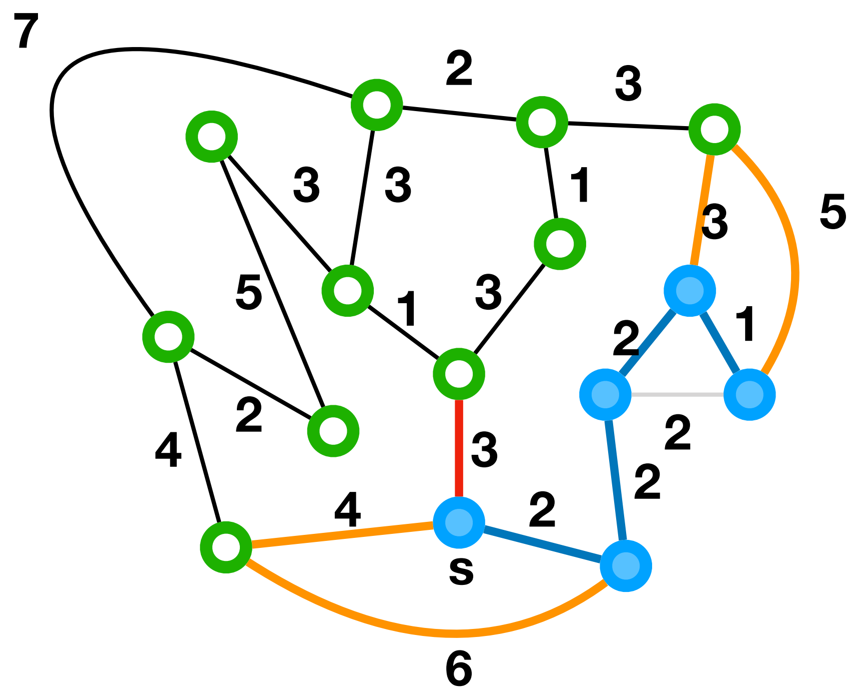
Example



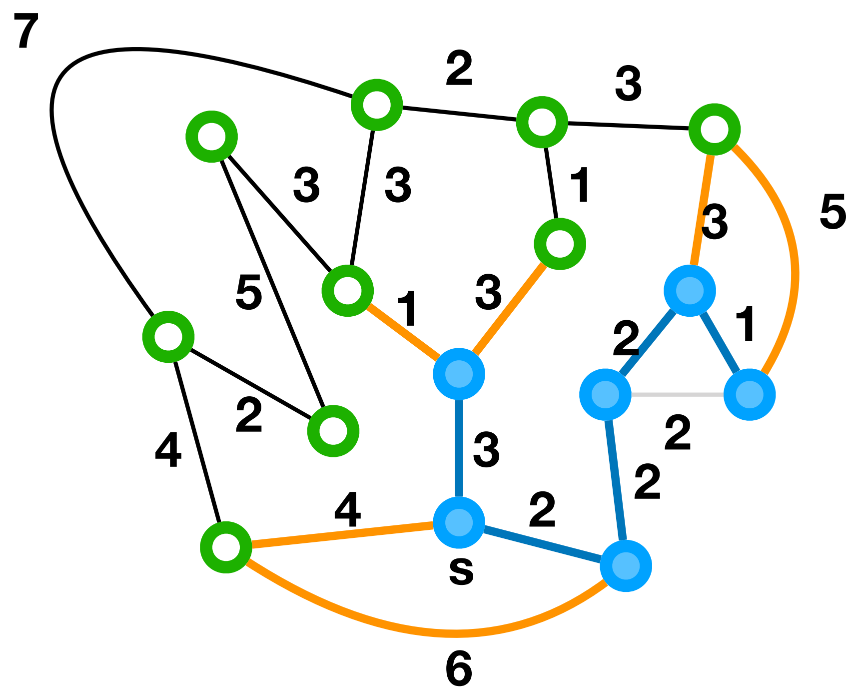
Example



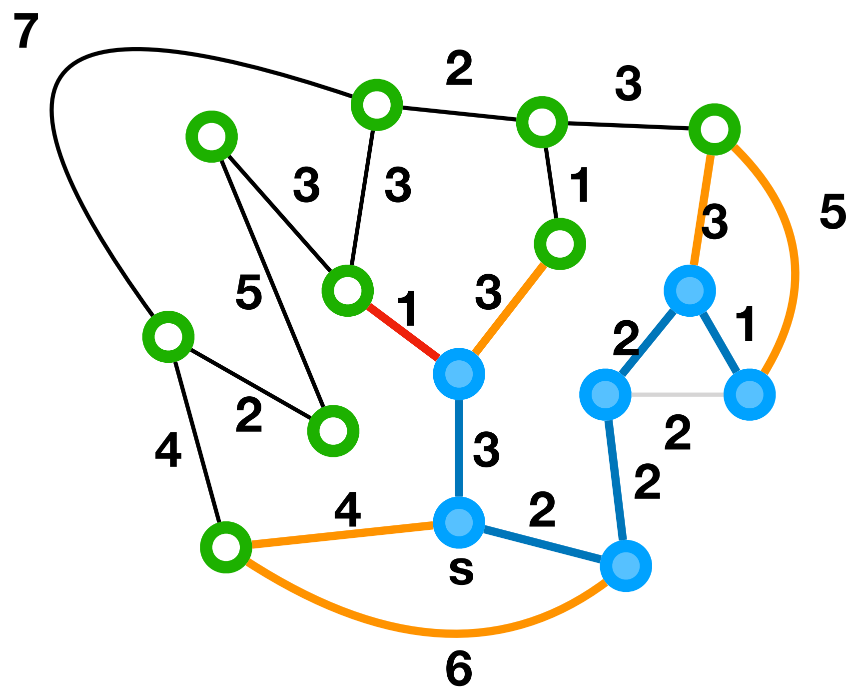
Example



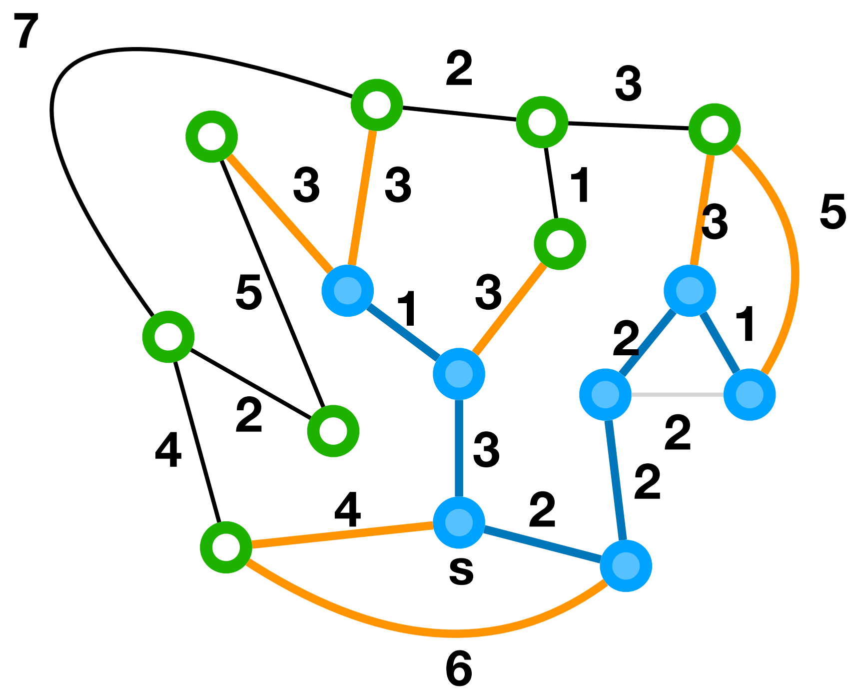
Example



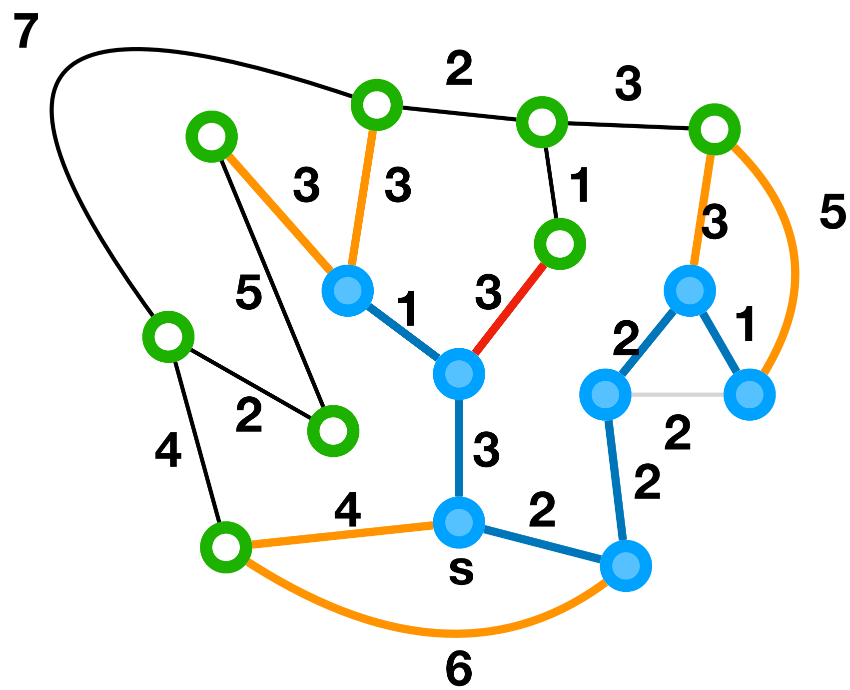
Example



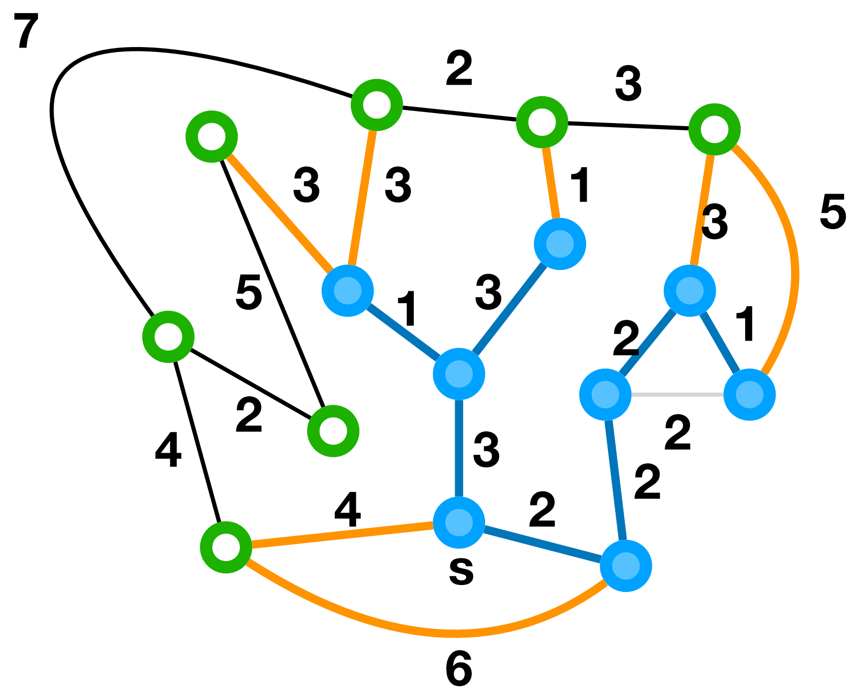
Example



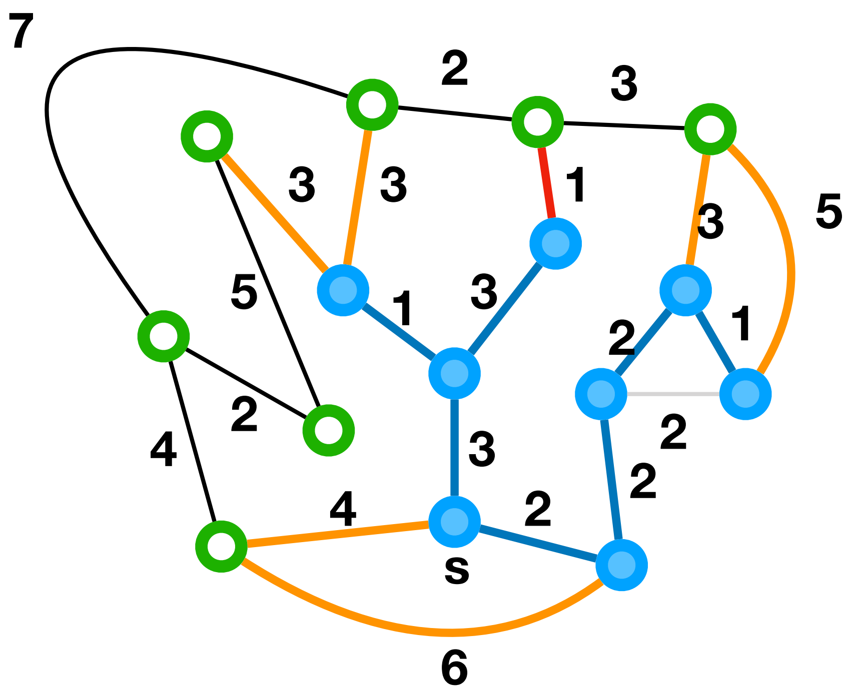
Example



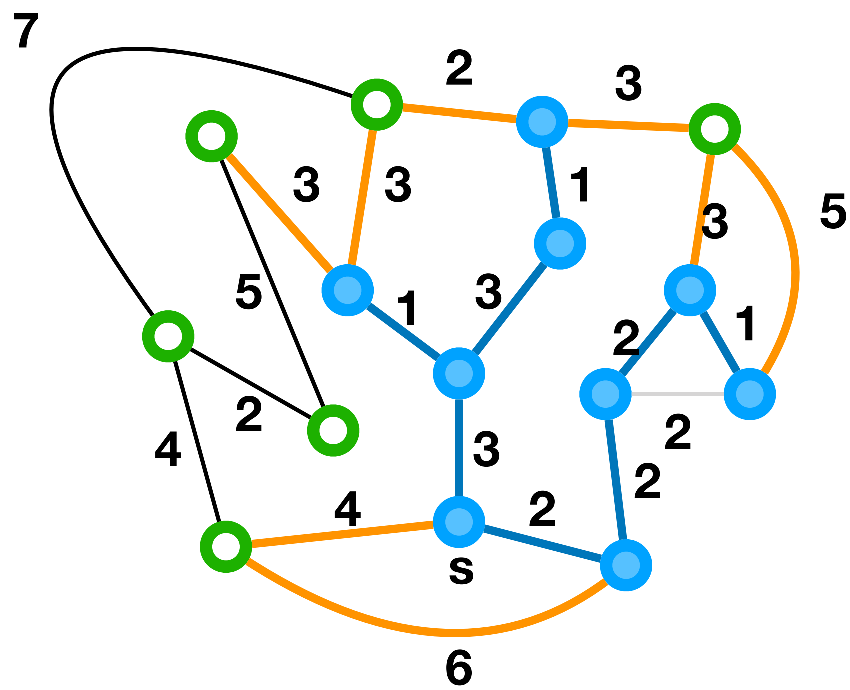
Example



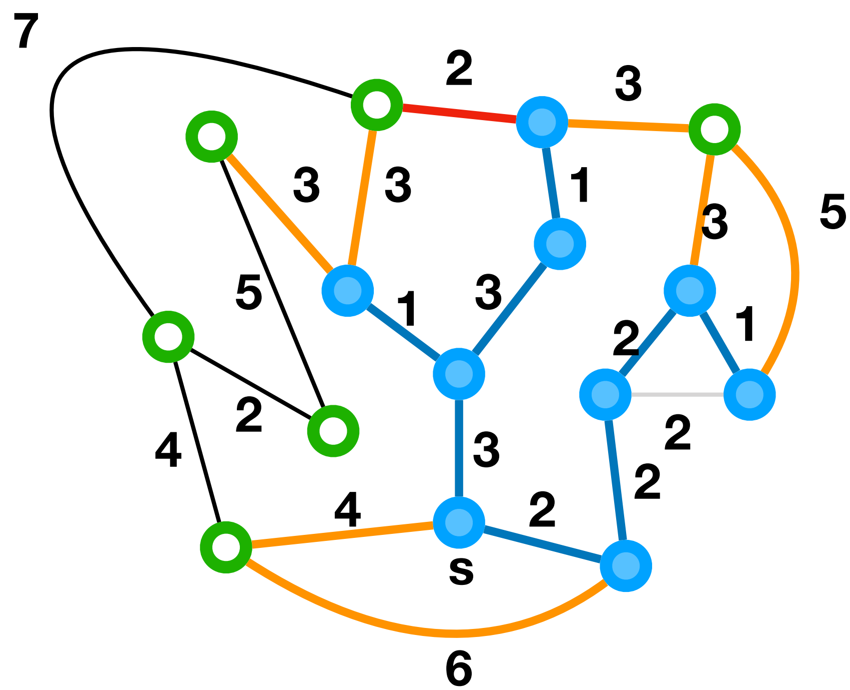
Example



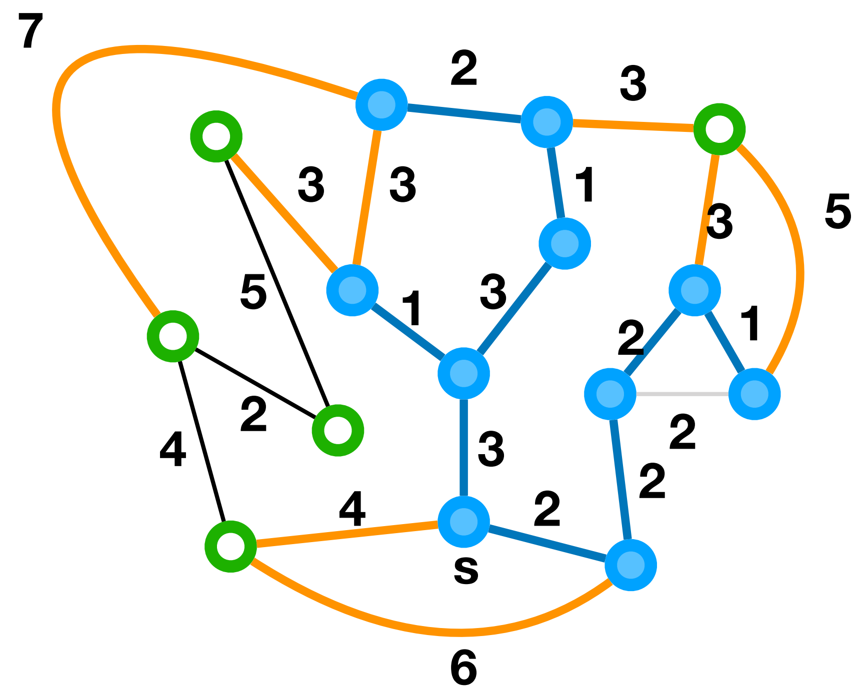
Example



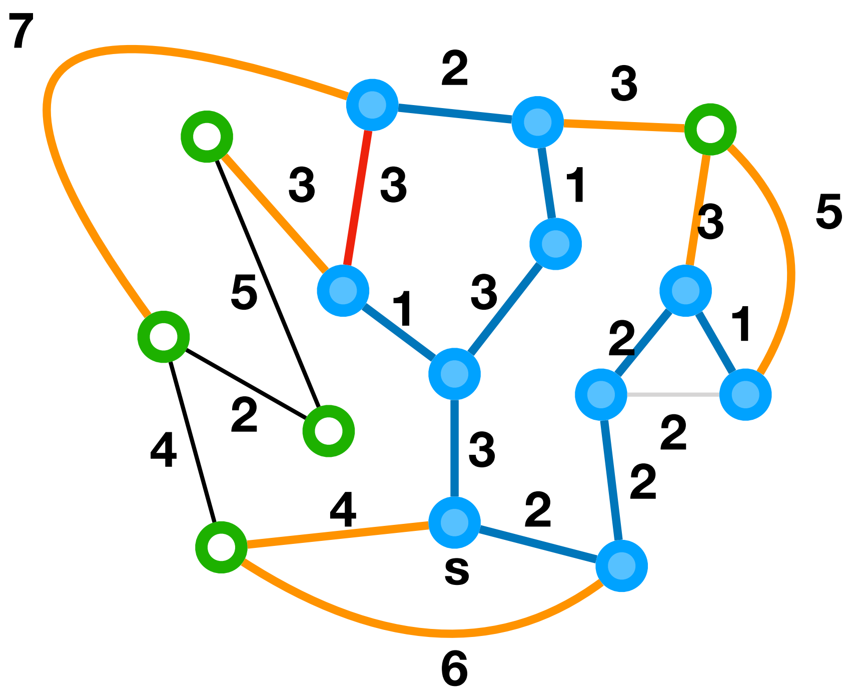
Example



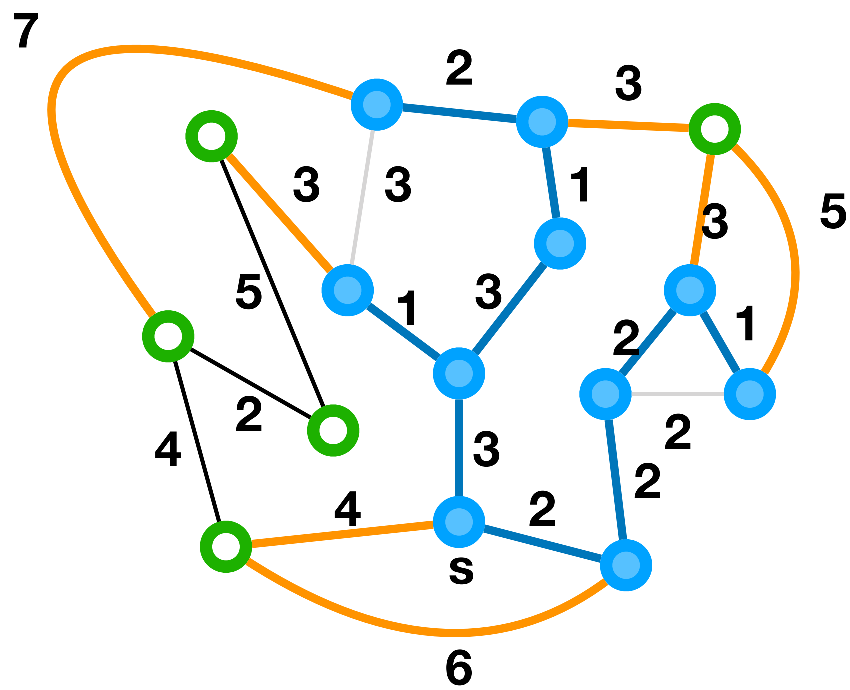
Example



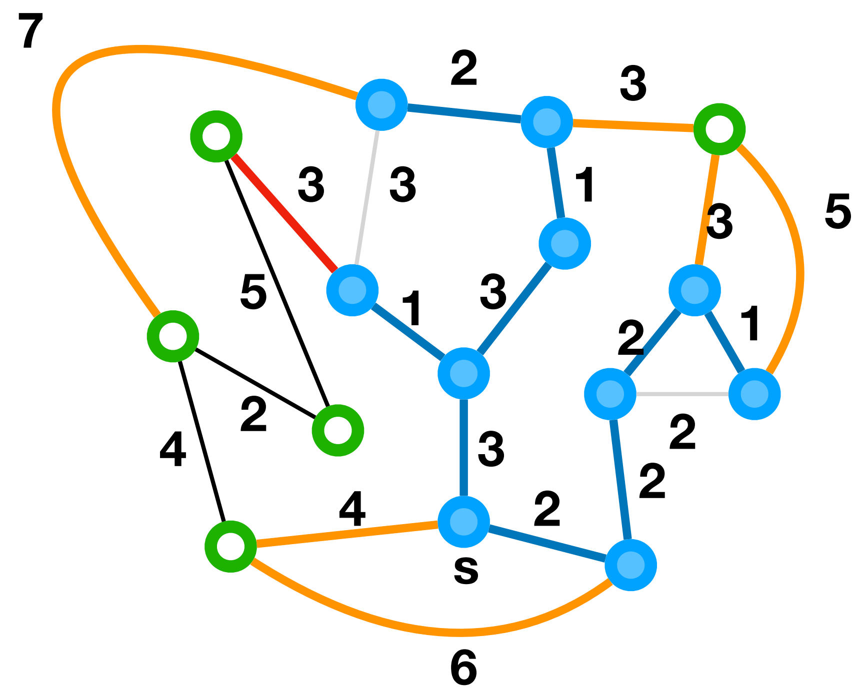
Example



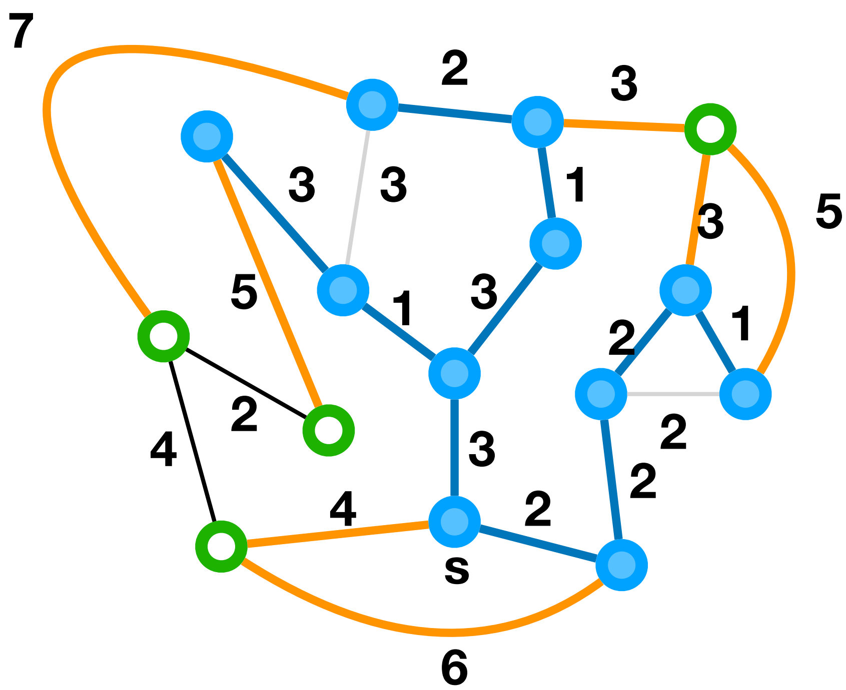
Example



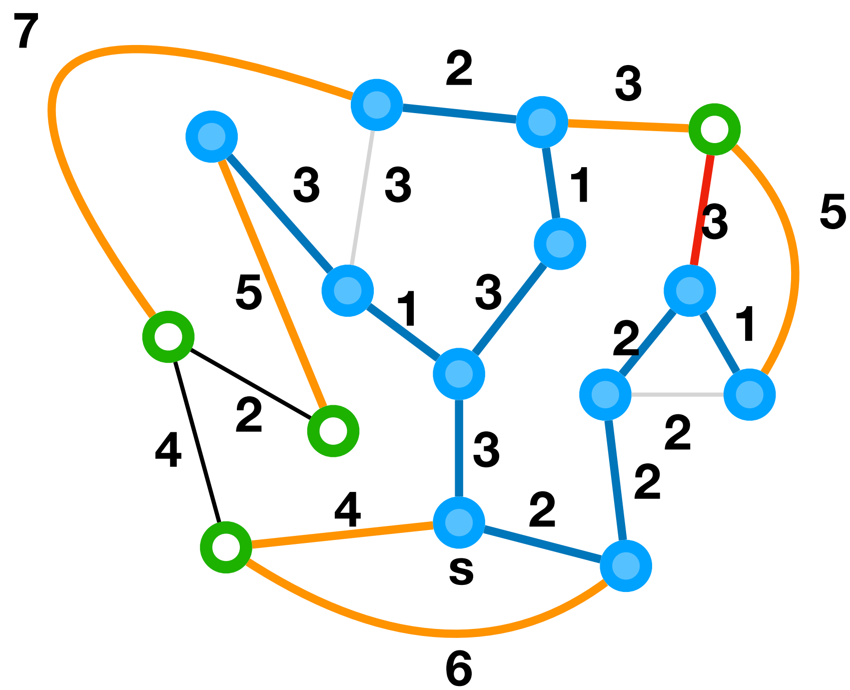
Example



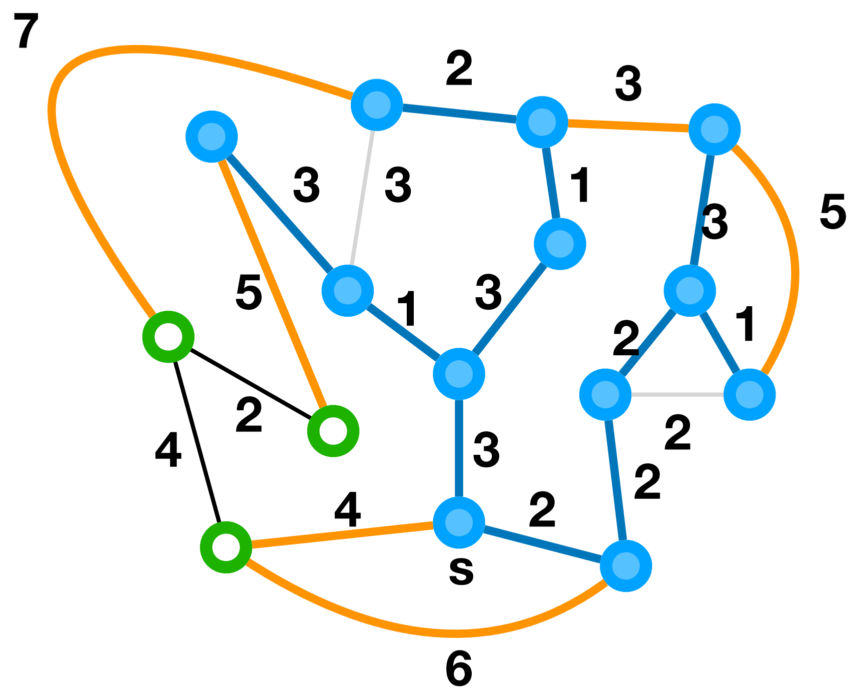
Example



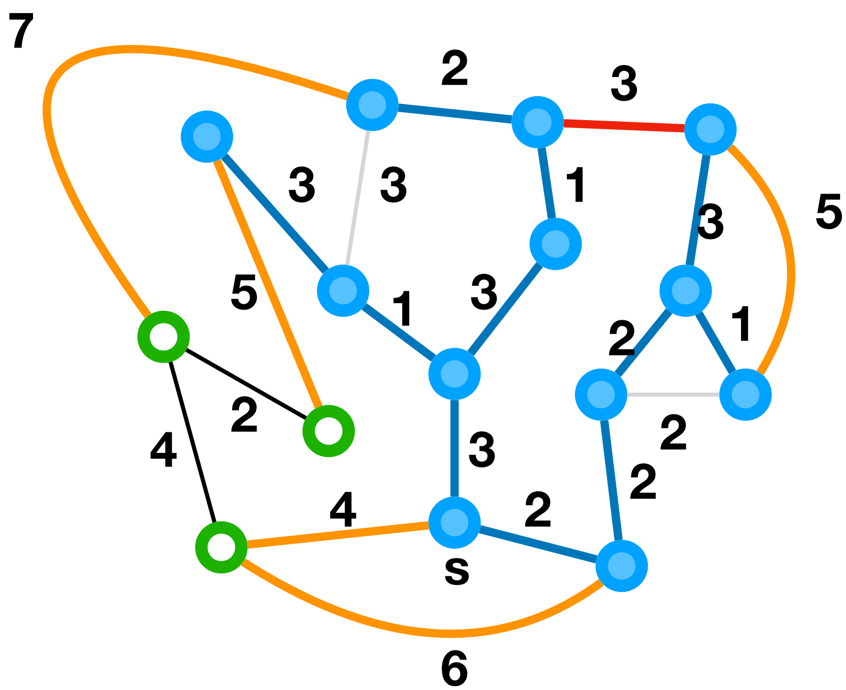
Example



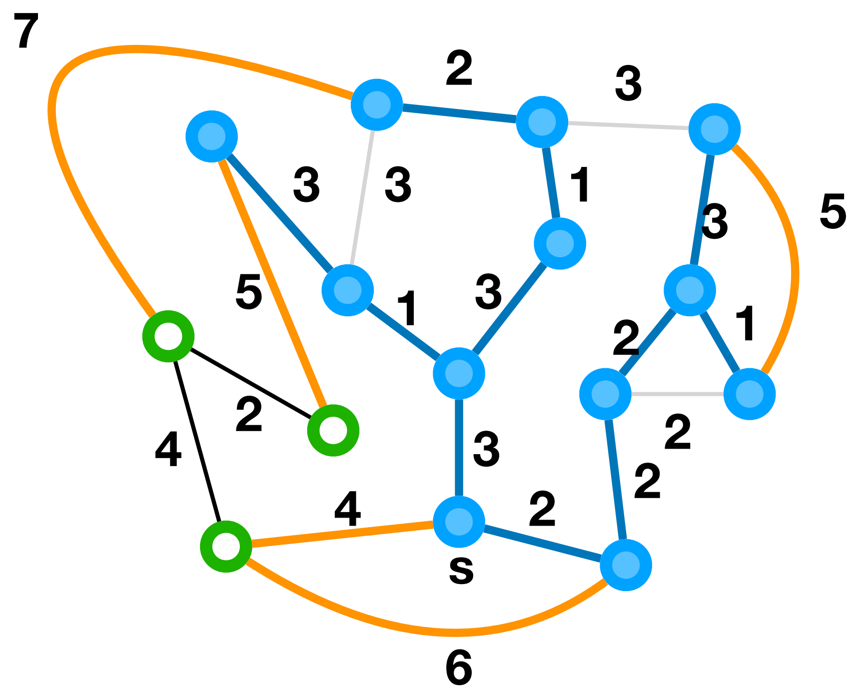
Example



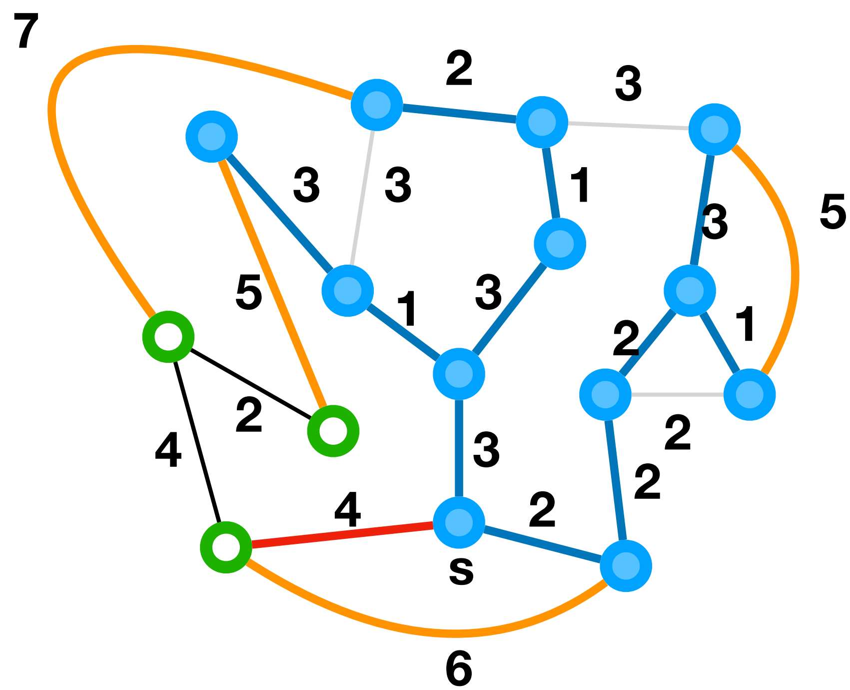
Example



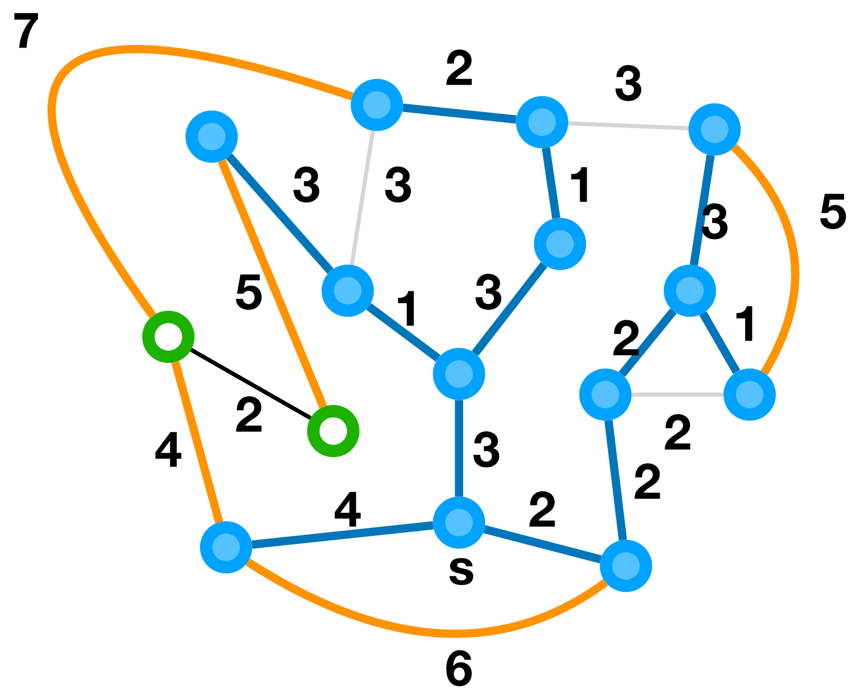
Example



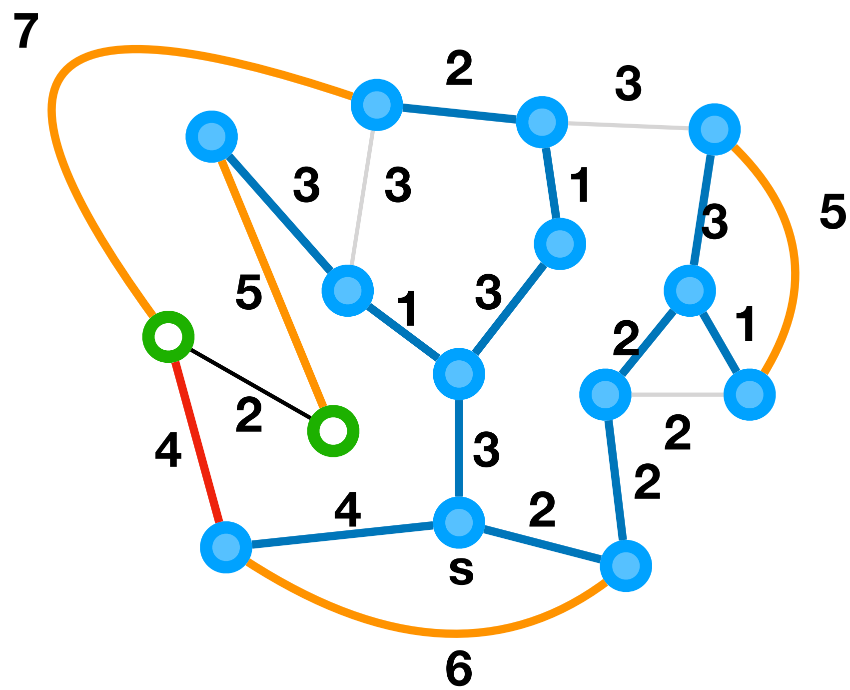
Example



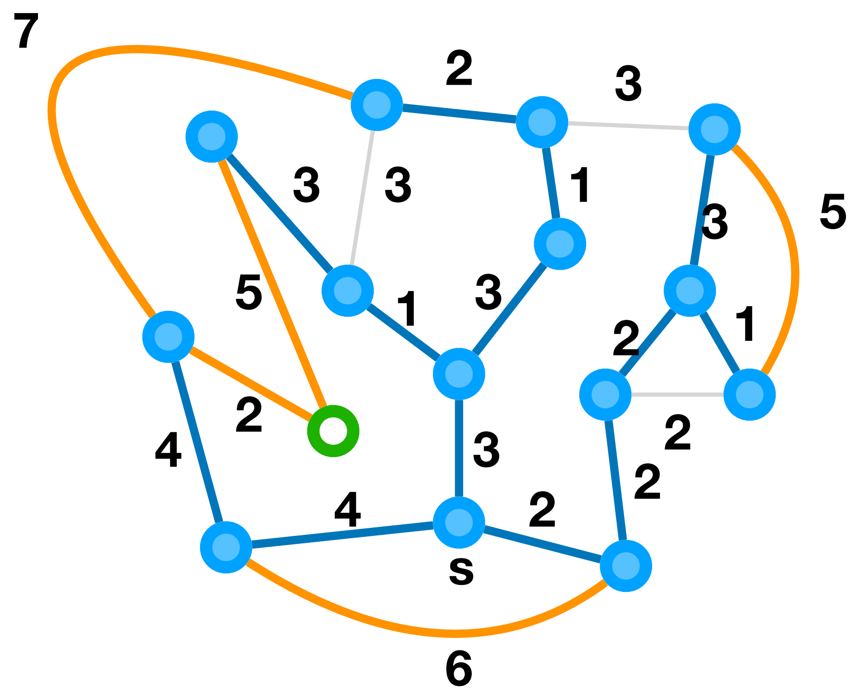
Example



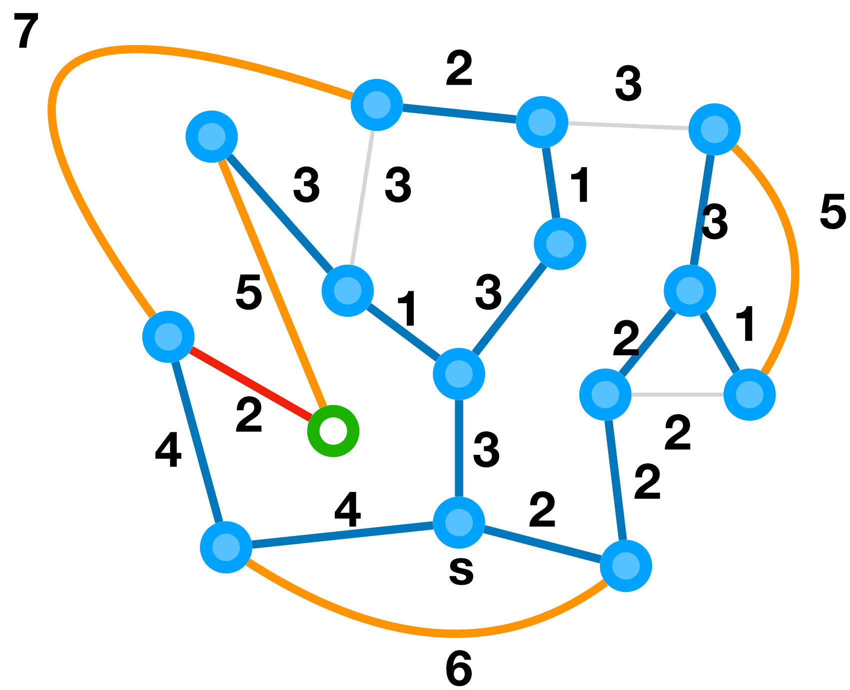
Example



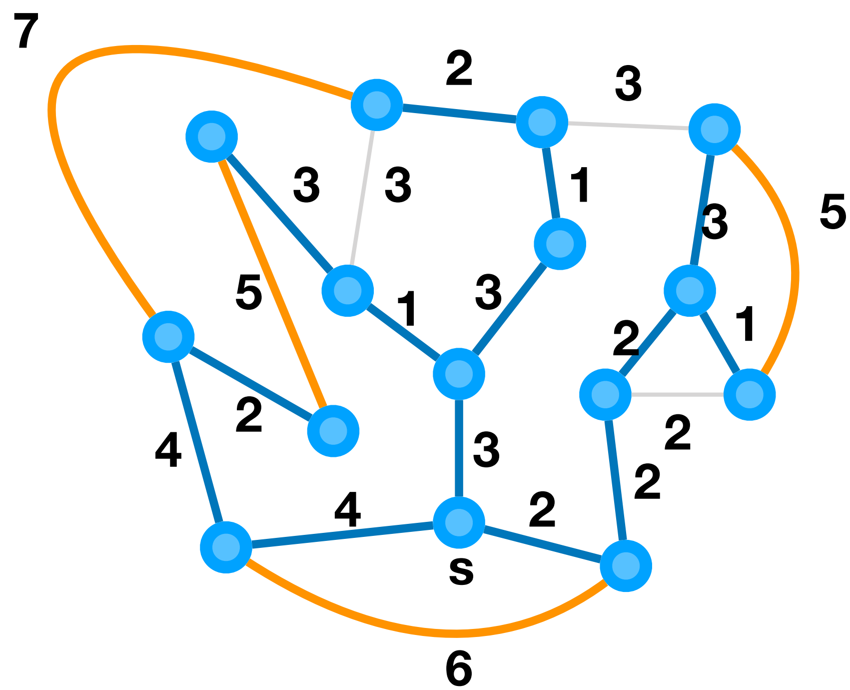
Example



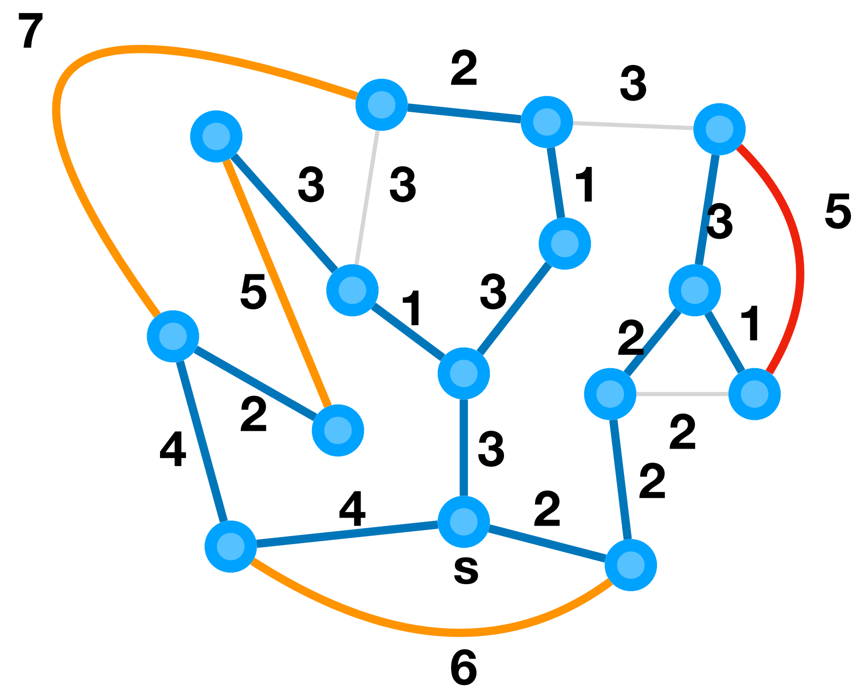
Example



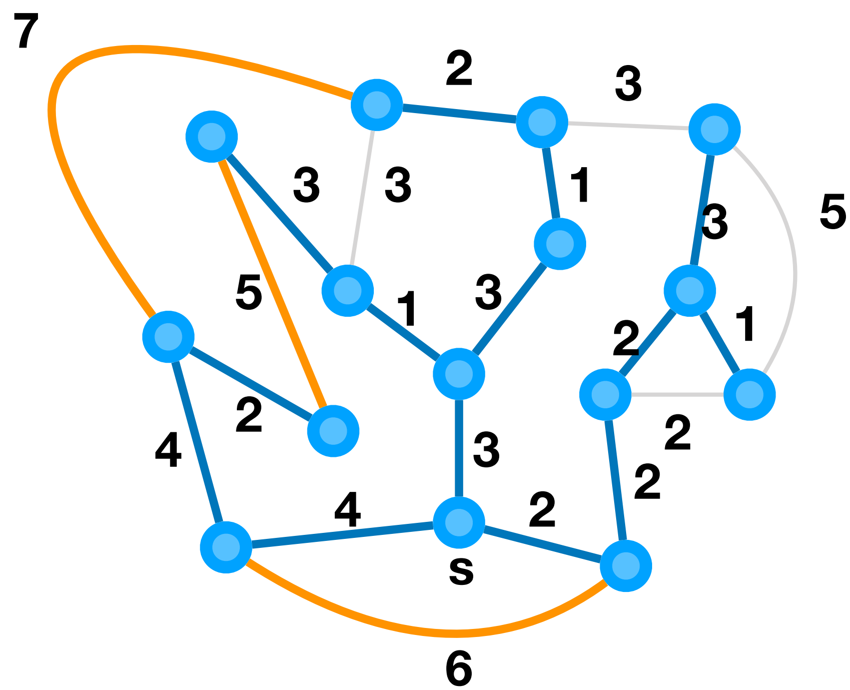
Example



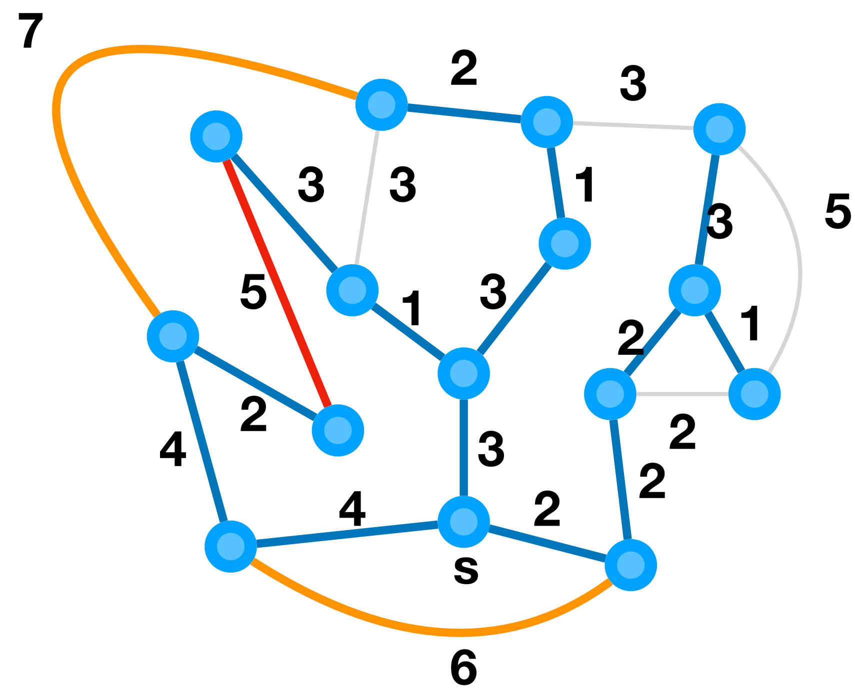
Example



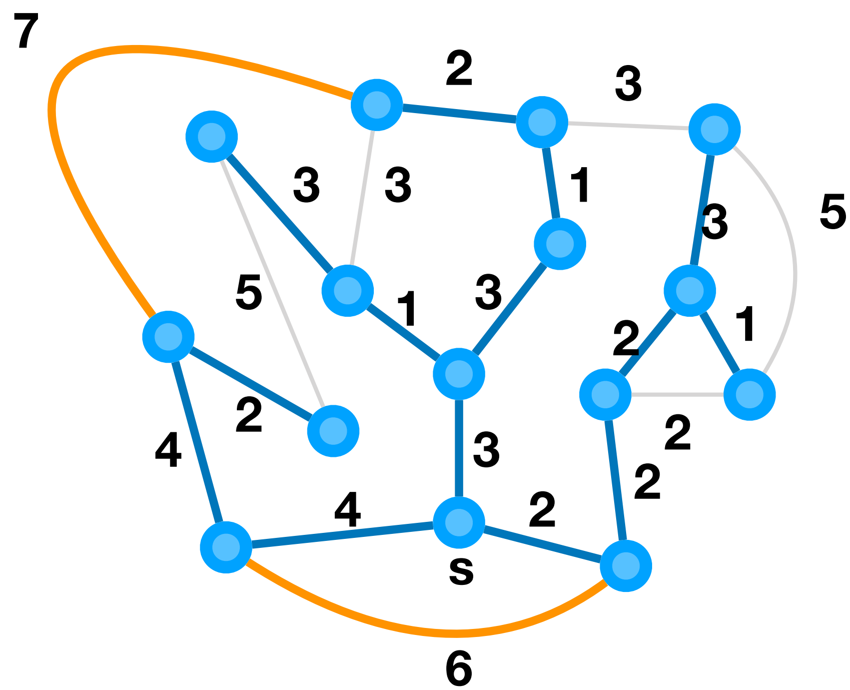
Example



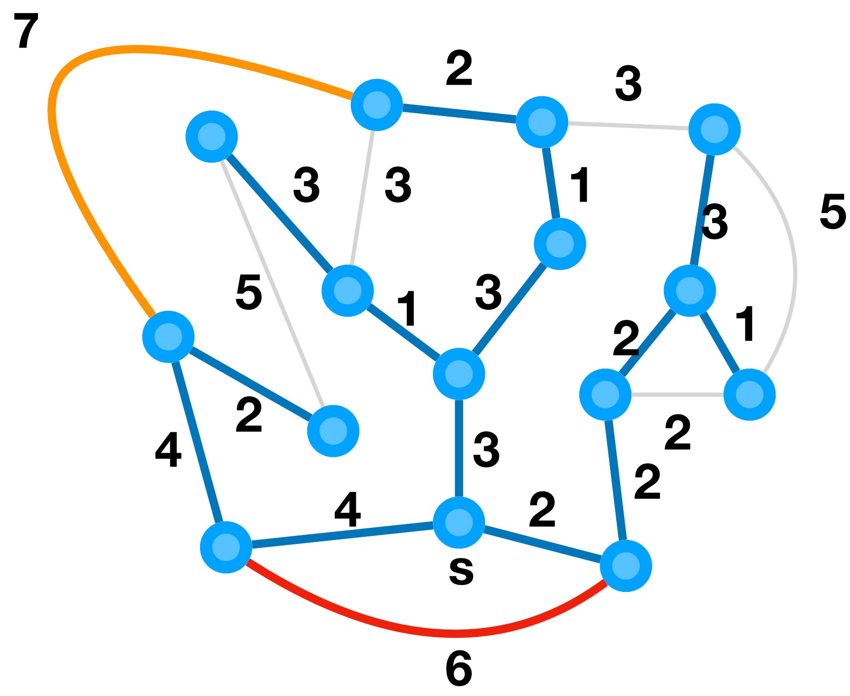
Example



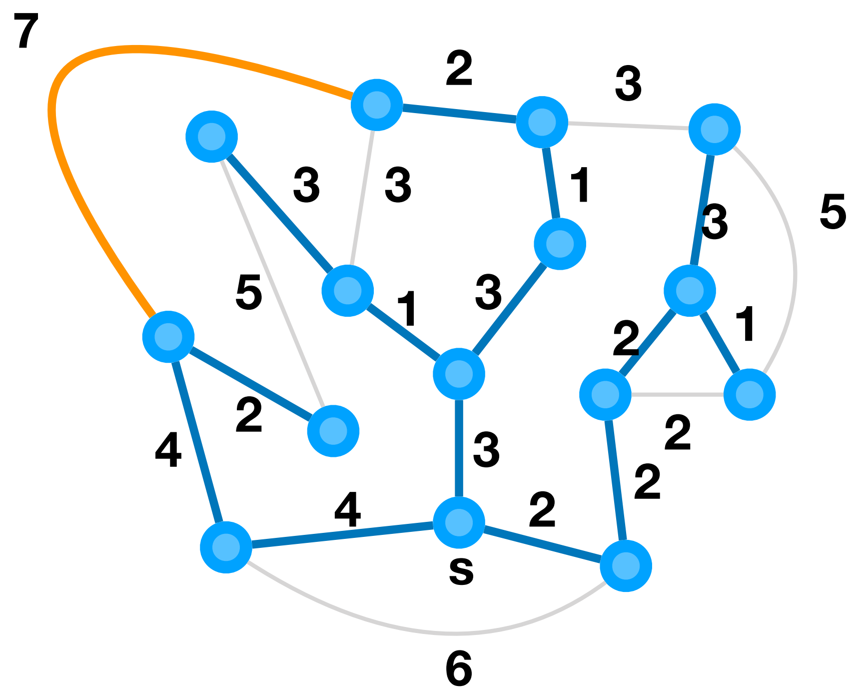
Example



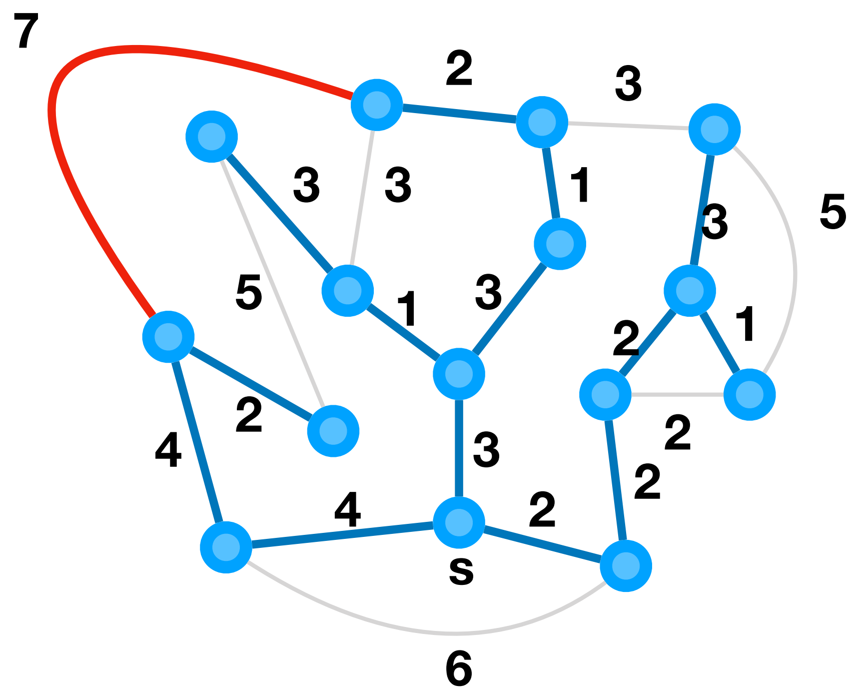
Example



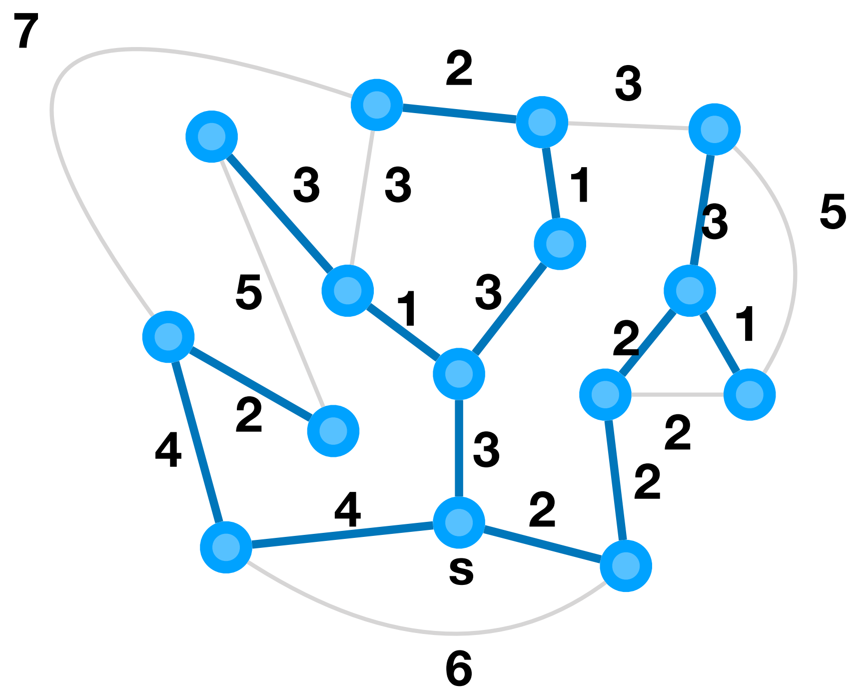
Example



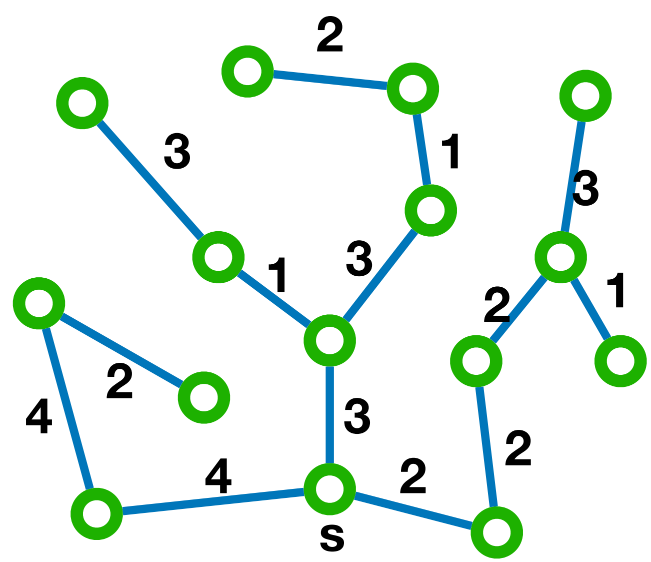
Example



Example



Example



Proof of Correctness

- Let mark[1:n] = false for all vertices and s be any arbitrary vertex
- Let F = ∅, mark[s] = true and H be the set of edges incident on s
- While H is not empty:
  - Remove minimum weight edge (u,v) from H
  - If mark[u]=mark[v] = true, ignore the edge and go to the next iteration of while-loop
  - Otherwise, let us assume by symmetry mark[u] = true only
  - Add (u,v) to F and all edges incident on v to H; set mark[v] = true.

- **Theorem:**
  - Suppose F is MST-good but not a tree yet
  - Let (S,V-S) be any cut with no cut edge in F
  - Then edge e in G-F with minimum weight among cut edges of (S,V-S) is safe for F

Summary

Summary

- MST problem: finding a spanning tree with minimum weight
- We saw two different algorithms for finding MST
  - Kruskal: based on sorting edges first
  - Prim: based on a graph search + min-heap
- They are both different implementation of a generic meta-algorithm based on safe edges and minimum weight edge of cuts
- Both algorithms take O(m log m) time

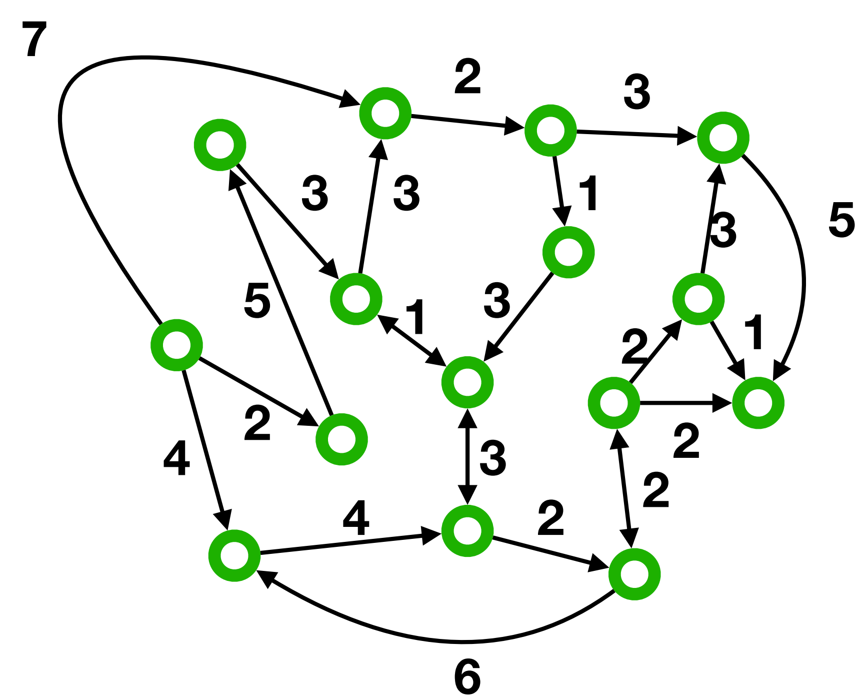
Summary

- MST problem: finding a spanning tree with minimum weight
- We saw two different algorithms for finding MST
  - Kruskal: based on sorting edges first
  - Prim: based on a graph search + min-heap
- They are both different implementation of a generic meta-algorithm based on safe edges and minimum weight edge of cuts
- Both algorithms take $O(m \log m)$ time
- There are even faster algorithms for this problem but they are way beyond the scope of our course

The Single-Source Shortest Path Problem

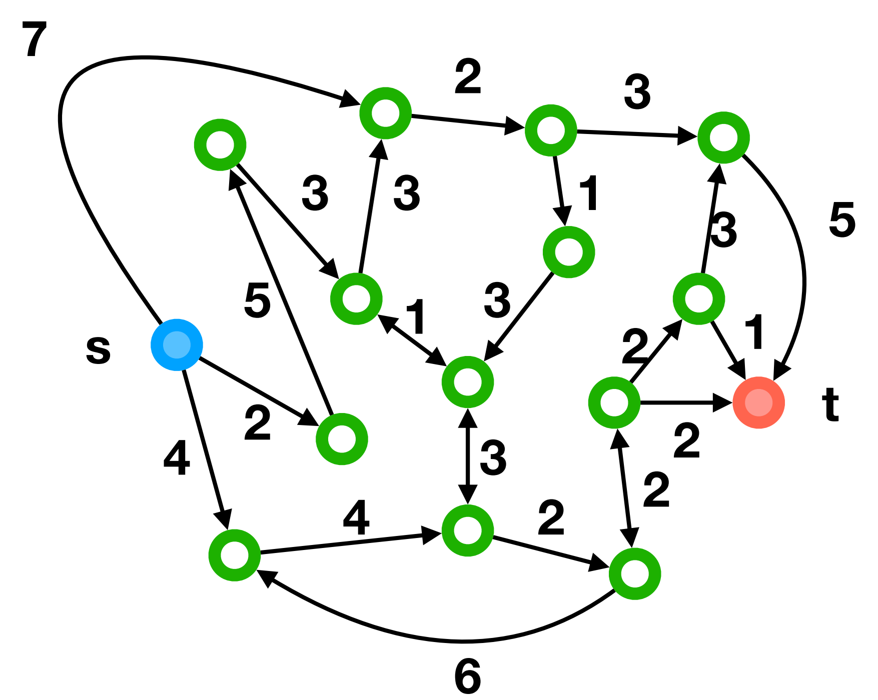
Shortest Paths

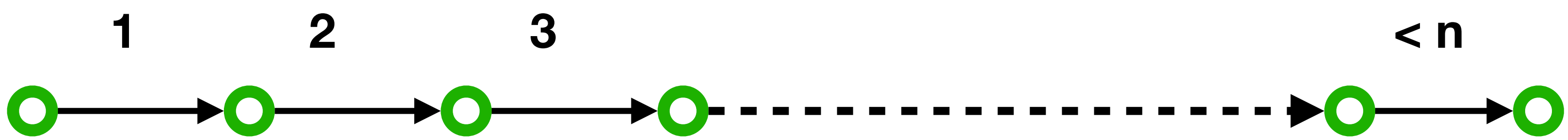
- For a graph G=(V,E) (directed or undirected) with weights w_e over each edge e



Shortest Paths

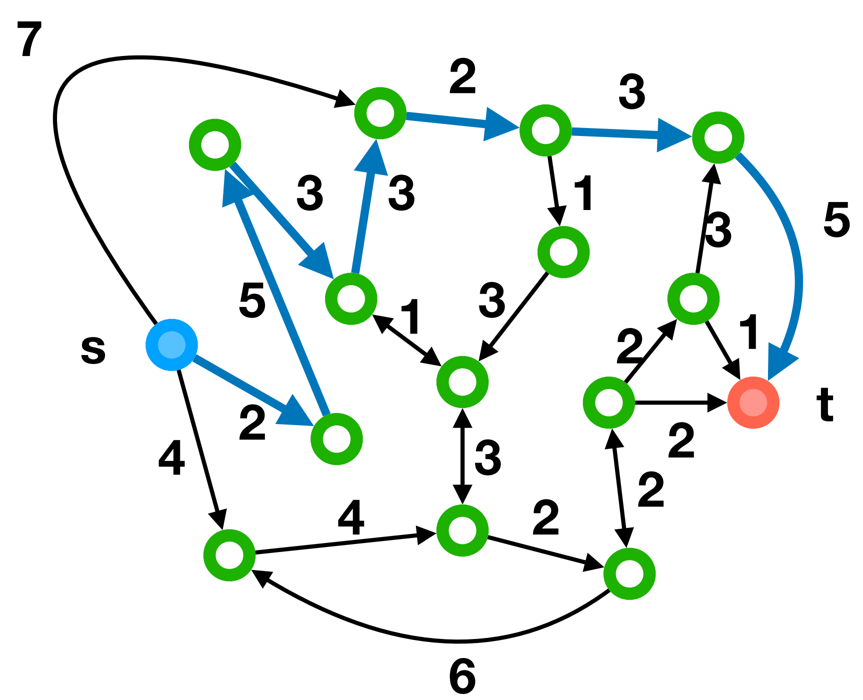
- For a graph G=(V,E) (directed or undirected) with weights w_e over each edge e
- Let P_{st} be any path between vertices s and t




1
2
3
< n

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



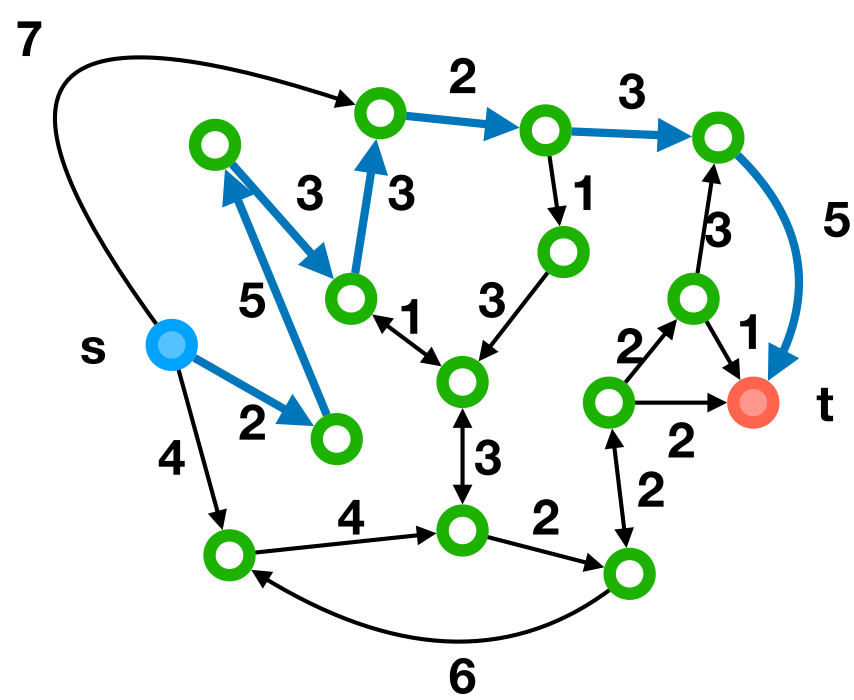
Shortest Paths

- 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:

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



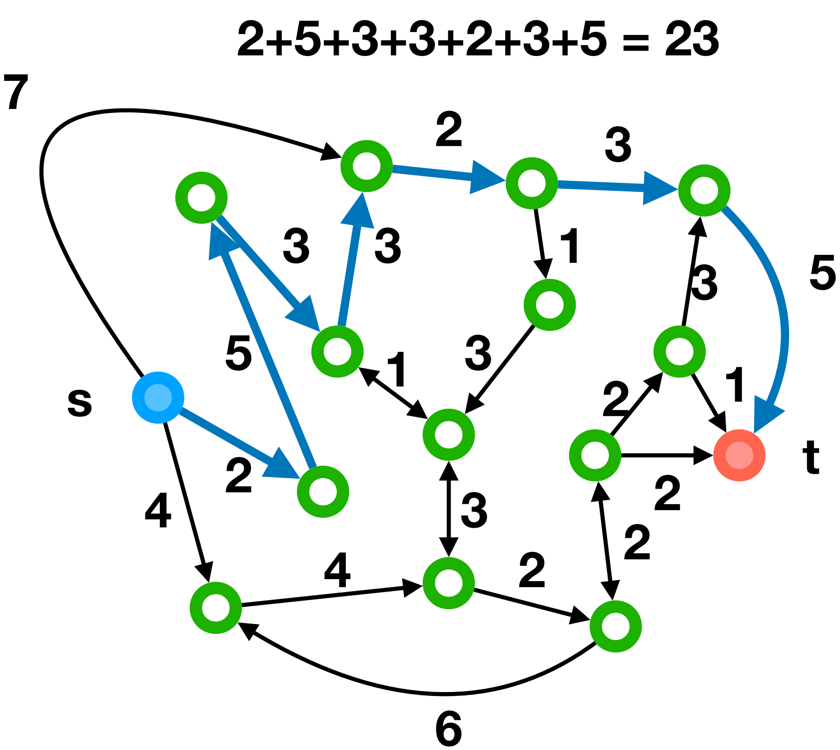
Shortest Paths

- 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:

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



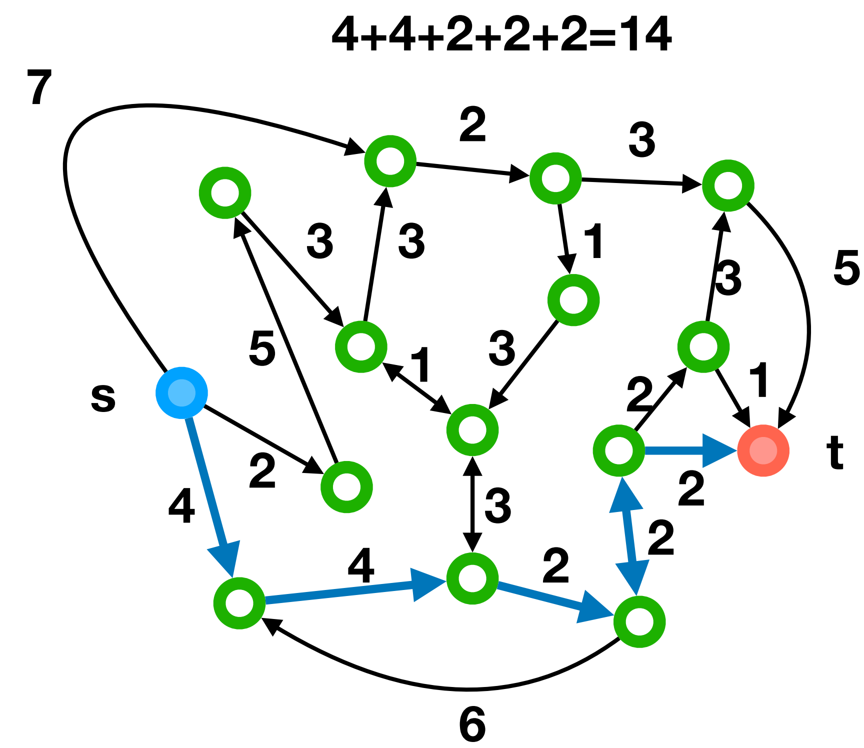
Shortest Paths

- 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

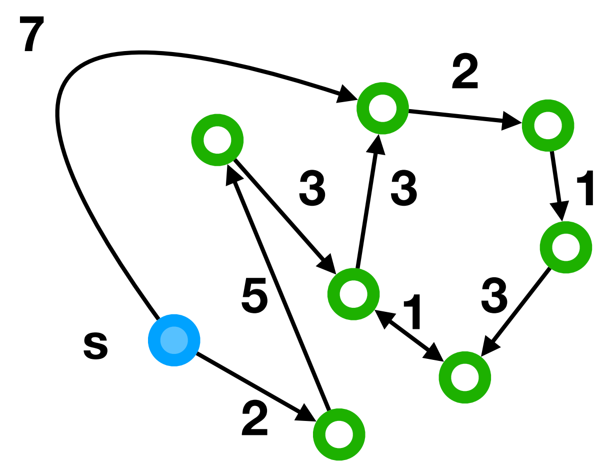
- Shortest s-t Path:
  - The path with minimum weight
  - P_{st} that minimizes $w(P_{st})$
- Distance of s to t, $dist(s, t)$:
  - Weight of shortest path from s to t

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
- Application? Any navigation app/method you ever use

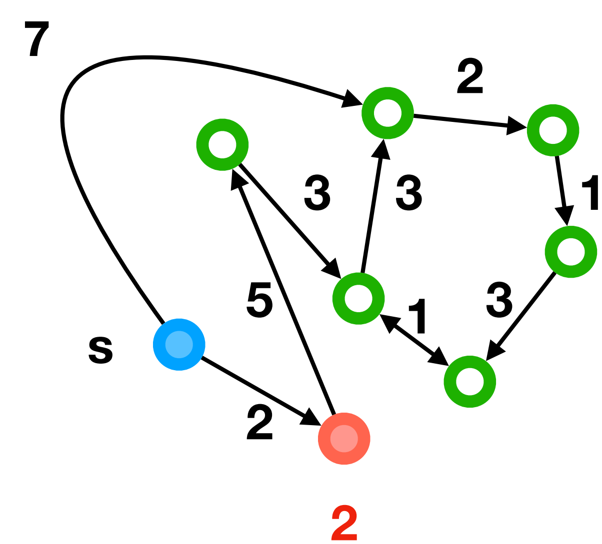
Shortest Paths

- 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



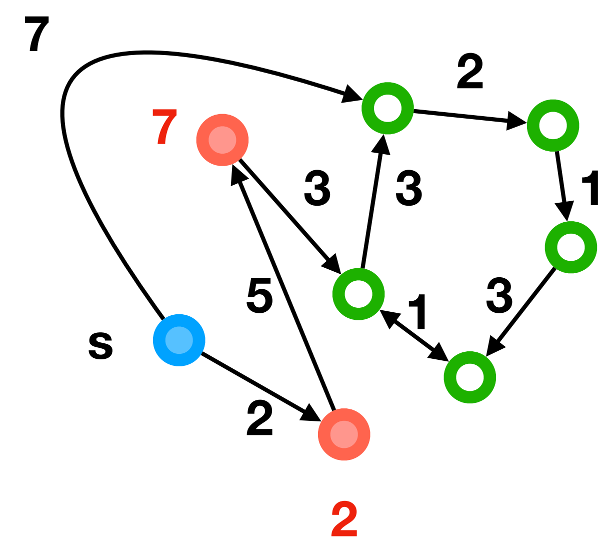
Shortest Paths

- 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



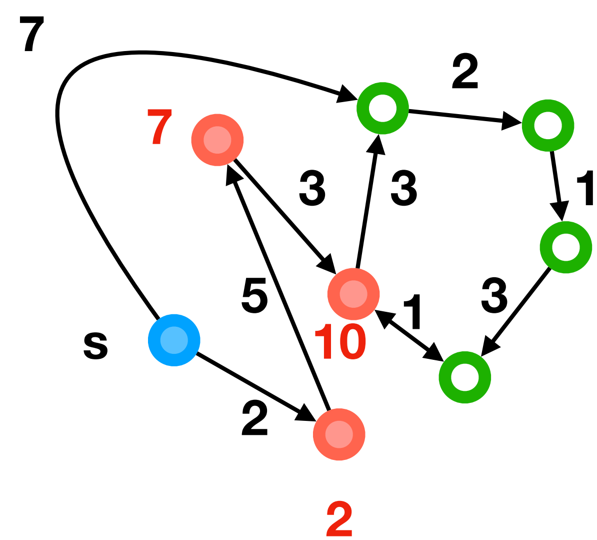
Shortest Paths

- 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



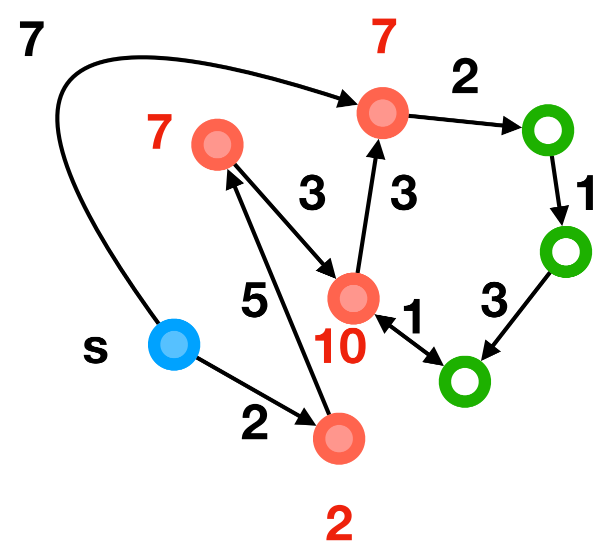
Shortest Paths

- 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



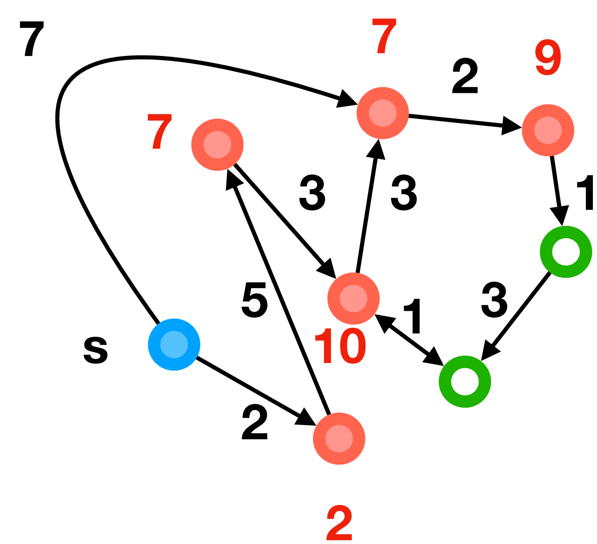
Shortest Paths

- 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



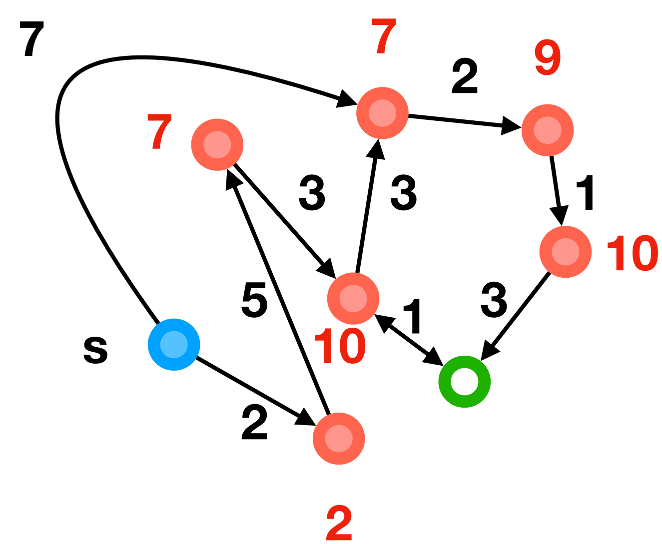
Shortest Paths

- 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



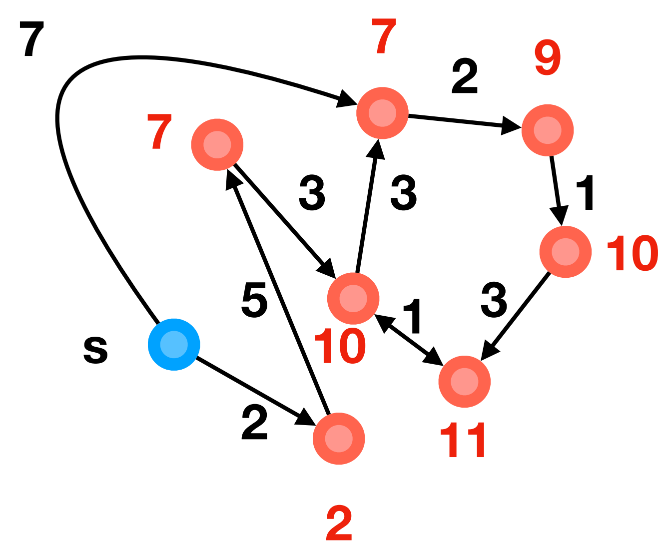
Shortest Paths

- 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

SSSP and Distances

- The SSSP problem we defined only outputs the distances
- What if we want to find a shortest path from s to some vertex t?

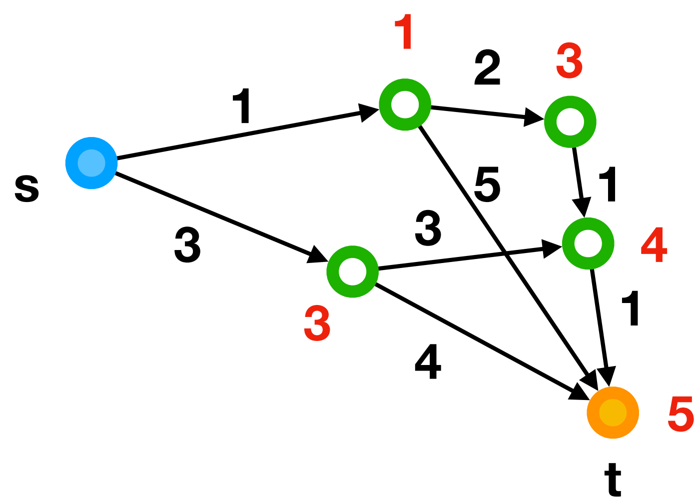
SSSP and Distances

- The SSSP problem we defined only outputs the distances
- What if we want to find a shortest path from s to some vertex t?
- We can build the path given the distances easily

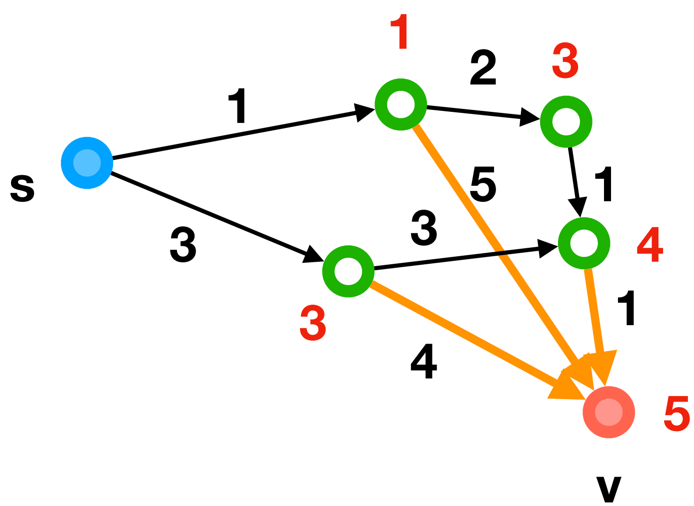
Finding the Path

- Let v=t and $P_{st} = \varnothing$
- While $v \neq s$
  - Find the in-neighbor u of v such that $dist(s, v) = dist(s, u) + w_{uv}$
  - Add the edge (u,v) to the beginning of P_{st}
  - Let $v \leftarrow u$
- Output P_{st}

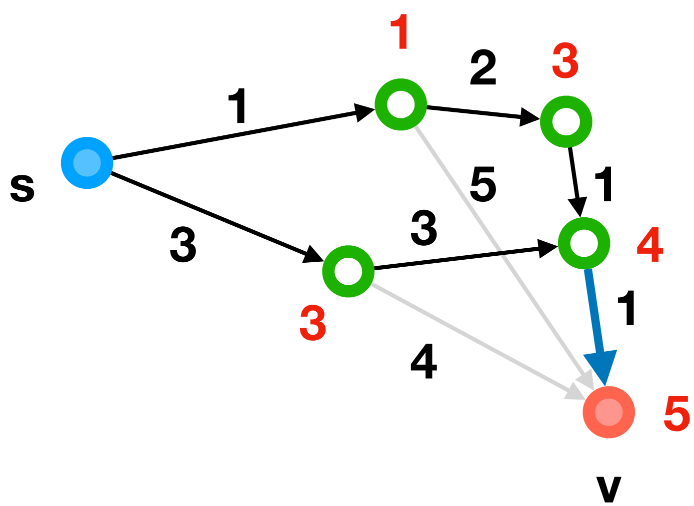
Example



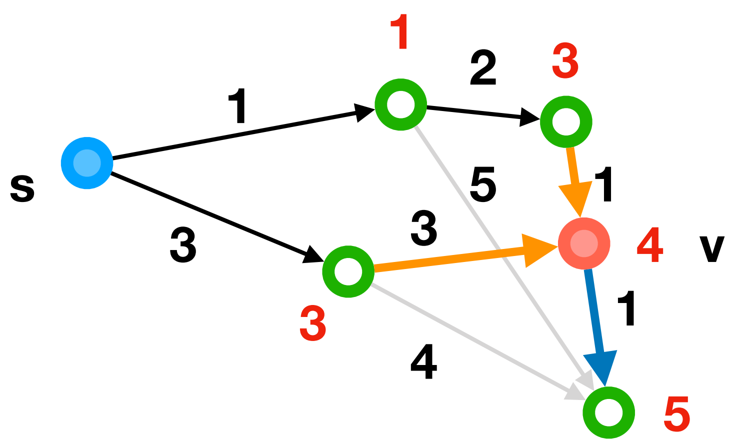
Example



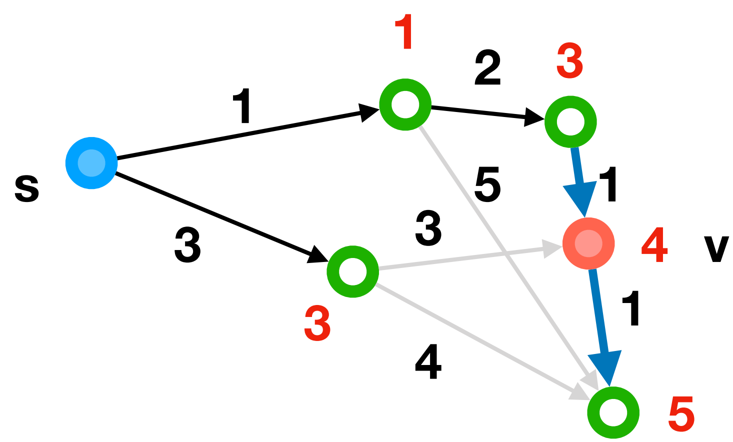
Example



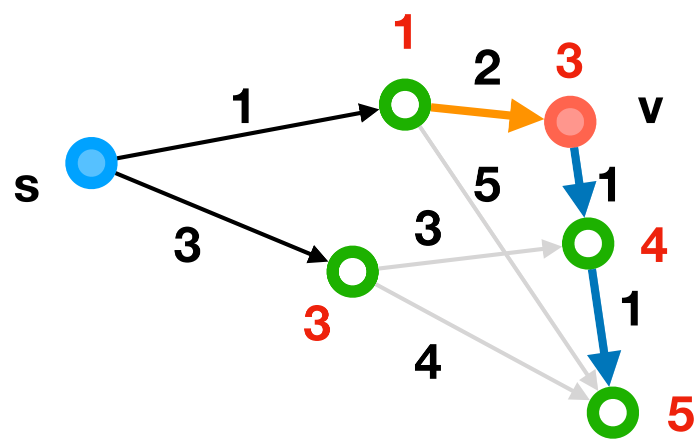
Example



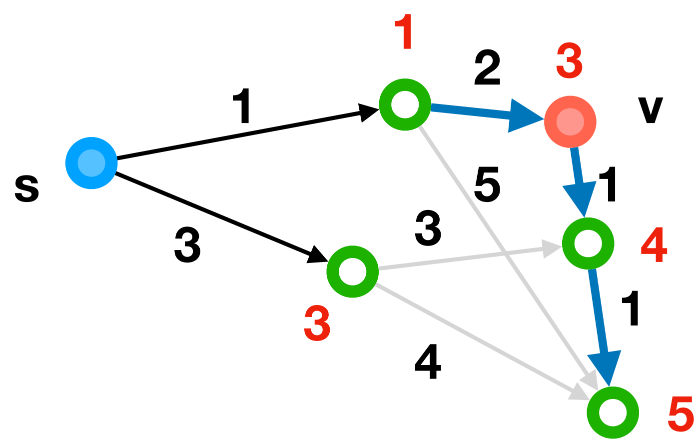
Example



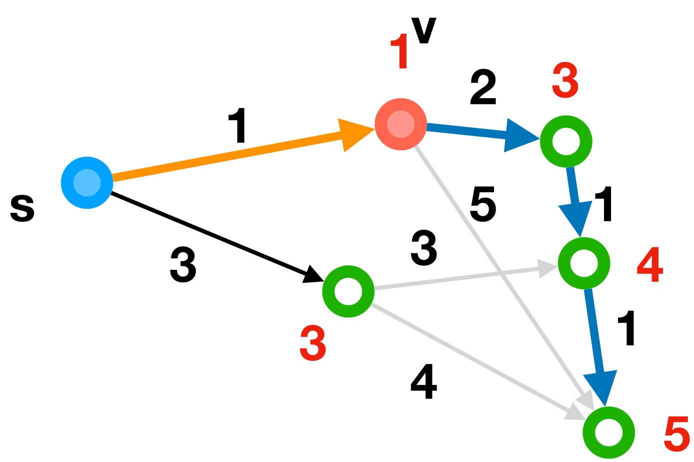
Example



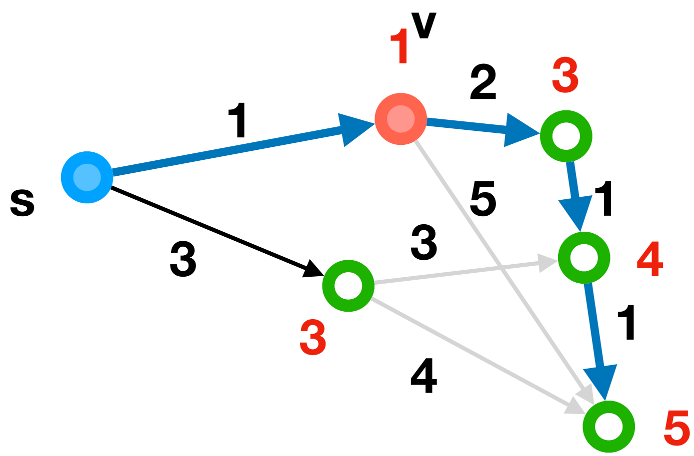
Example



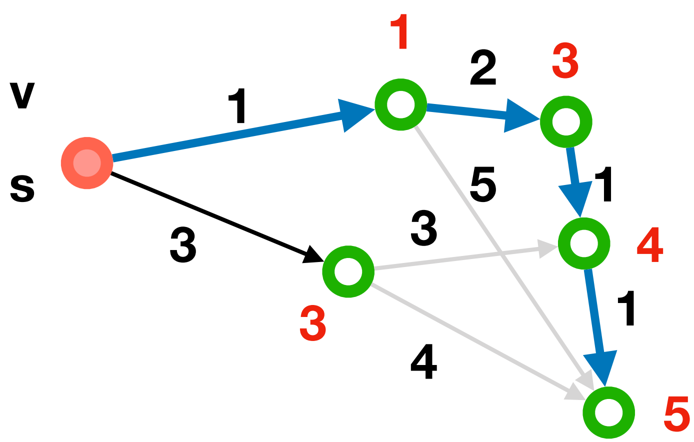
Example



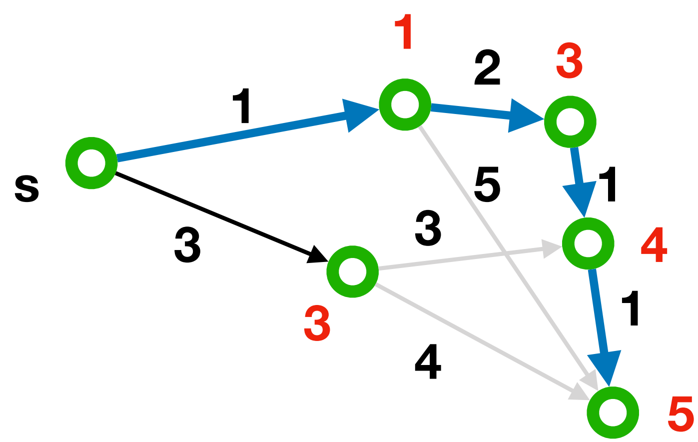
Example



Example



Example



Proof of Correctness

- Let v=t and $P_{st} = \emptyset$
- While $v \neq s$
  - Find the in-neighbor u of v such that $dist(s, v) = dist(s, u) + w_{uv}$
  - Add the edge (u,v) to the beginning of P_{st}
  - Let $v \leftarrow u$
- Output P_{st}

- The weight of the path is equal to $dist(s, t)$
- So it is a shortest path from s to t
- We will see more on this later in the lecture

Runtime

- Let v=t and $P_{st} = \emptyset$
- While $v \neq s$
  - Find the in-neighbor u of v such that $dist(s,u) = dist(s,v) + w_{uv}$
  - Add the edge (u,v) to the beginning of P_{st}
  - Let $v \leftarrow u$
- Output P_{st}

- No edge or vertex is visited more than once
- So O(n+m) time at most

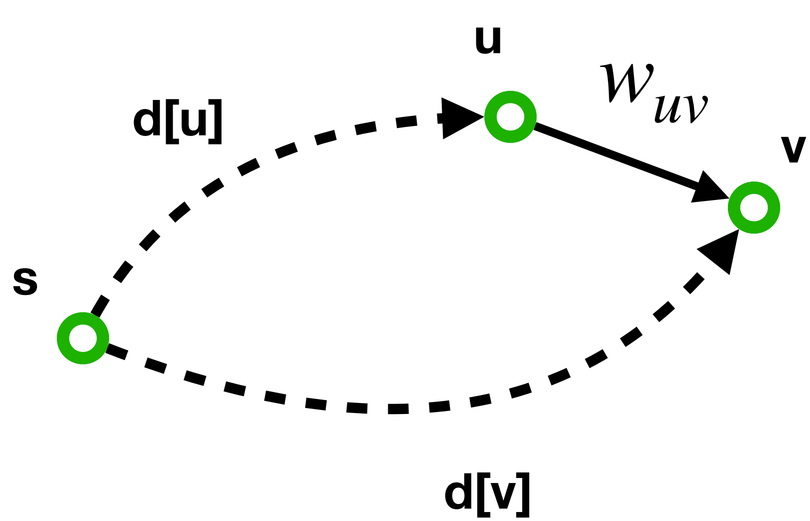
Bellman-Ford Algorithm

Bellman-Ford Algorithm

- An extremely simple algorithm for SSSP
- General idea:
- We start with some value d[v] for every vertex v
- We make sure that d[v] is always equal to weight of some s-v path
- We would like to eventually have d[v] = dist(s,v) but originally d[v] can be much larger
- We update values like this:
  - for any edge (u,v), if $d[v] > d[u] + w_{uv}$ set $d[v] \leftarrow d[u] + w_{uv}$

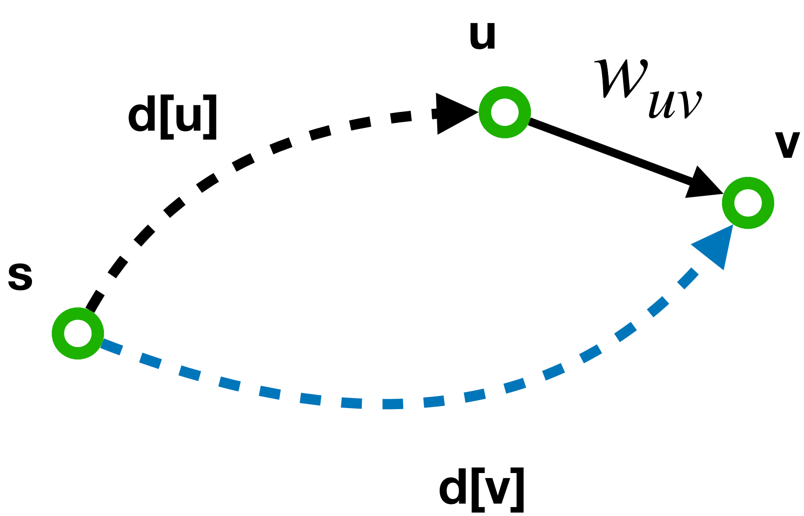
Bellman-Ford Algorithm

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



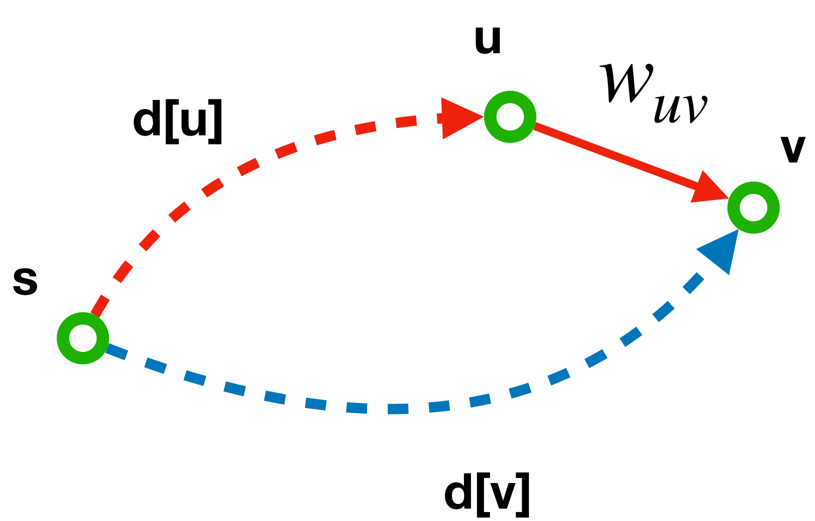
Bellman-Ford Algorithm

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



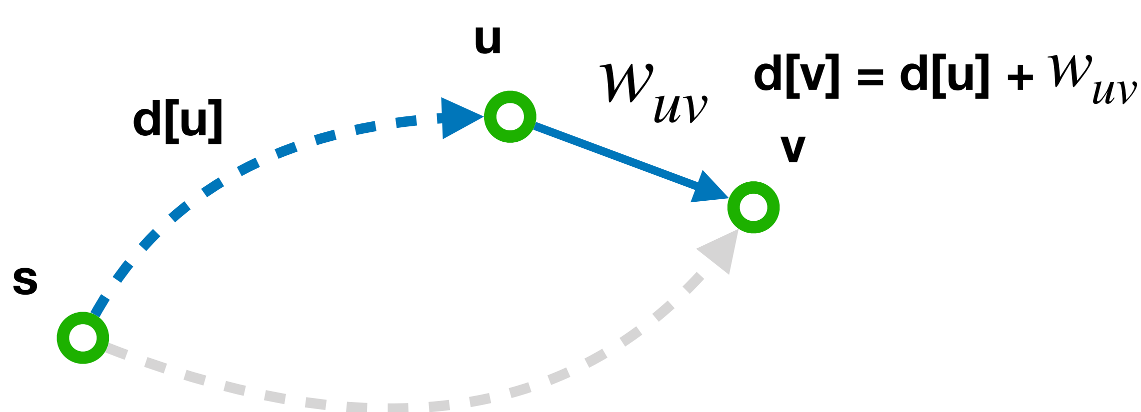
Bellman-Ford Algorithm

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



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

- 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.

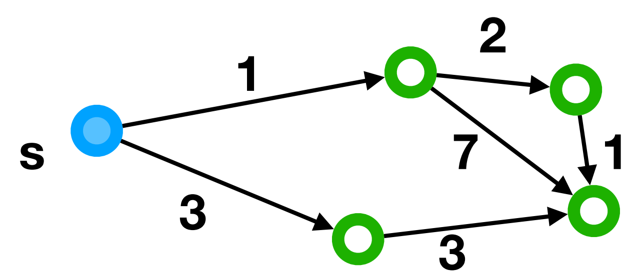
Bellman-Ford Algorithm: Example

- 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.



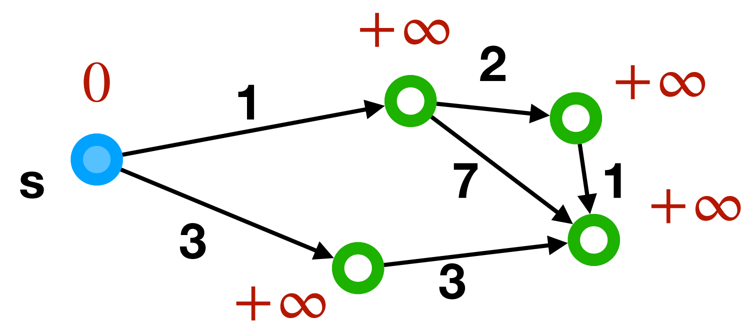
Bellman-Ford Algorithm: Example

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

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

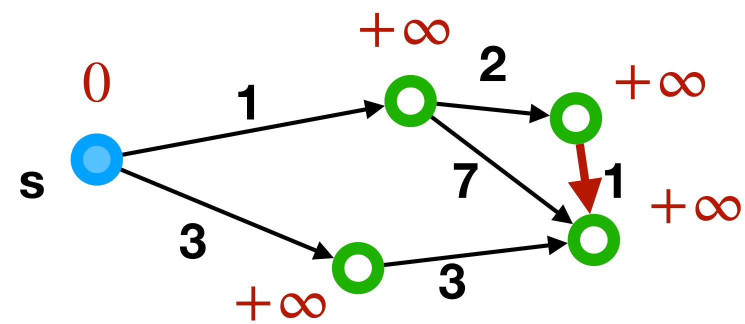
  - If $d[v] > d[u] + w_{uv}$
    update $d[v] \leftarrow d[u] + w_{uv}$

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



Bellman-Ford Algorithm: Example

- 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.



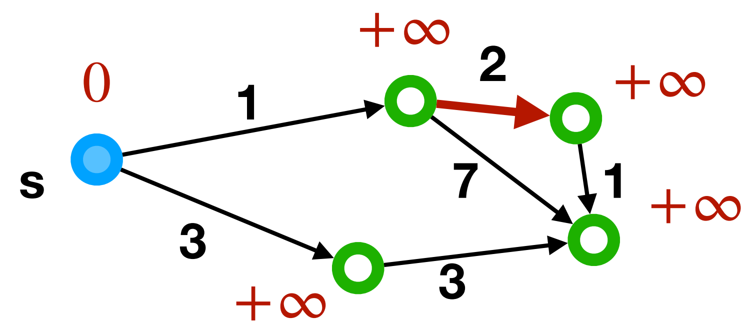
Bellman-Ford Algorithm: Example

- 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.



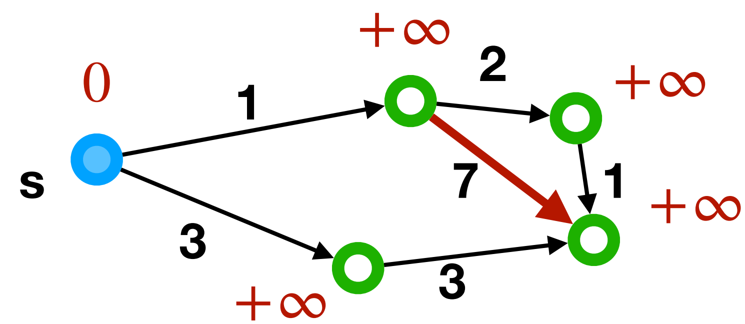
Bellman-Ford Algorithm: Example

- 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.



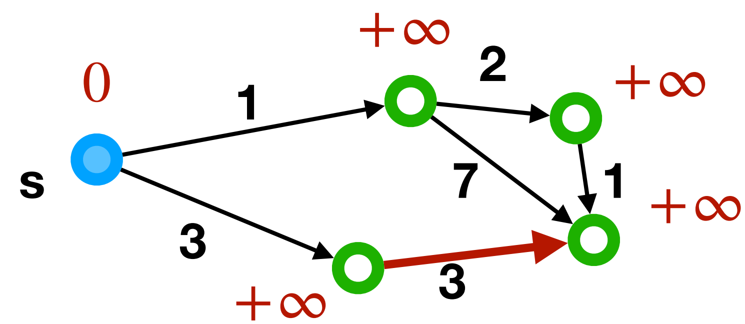
Bellman-Ford Algorithm: Example

- 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.



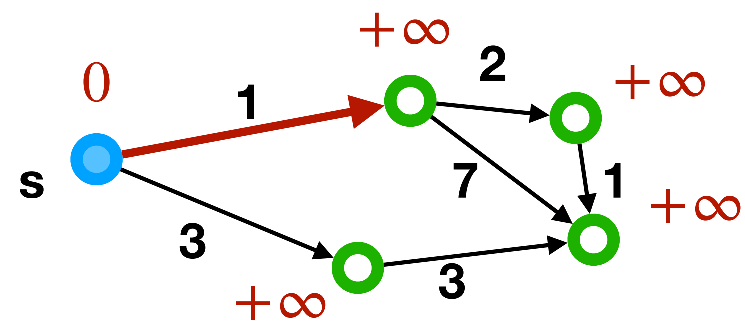
Bellman-Ford Algorithm: Example

- 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.



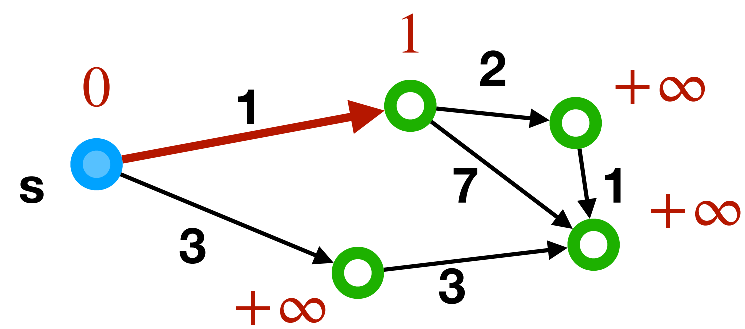
Bellman-Ford Algorithm: Example

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

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

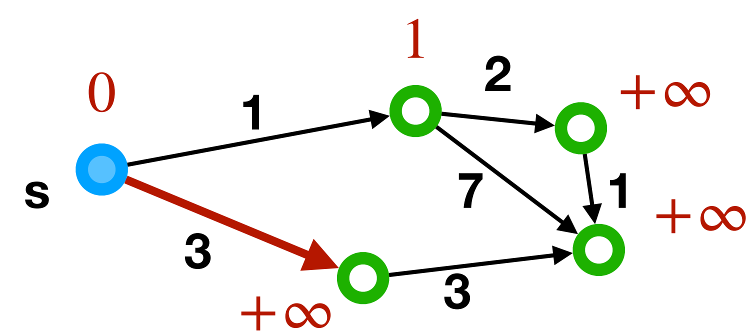
  - If $d[v] > d[u] + w_{uv}$ **update** $d[v] \leftarrow d[u] + w_{uv}$

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



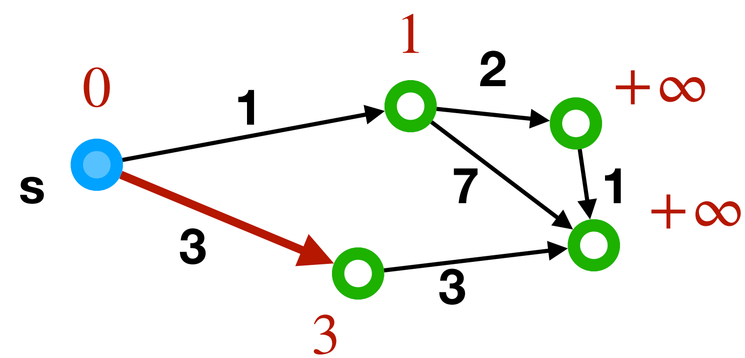
Bellman-Ford Algorithm: Example

- 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: Example

- 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.



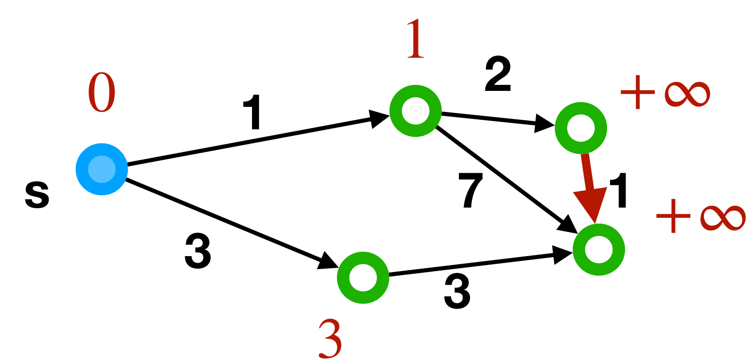
Bellman-Ford Algorithm: Example

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

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

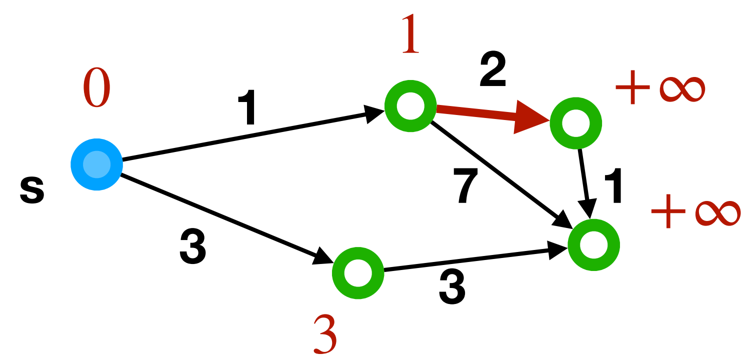
  - If $d[v] > d[u] + w_{uv}$ **update** $d[v] \leftarrow d[u] + w_{uv}$

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



Bellman-Ford Algorithm: Example

- 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.



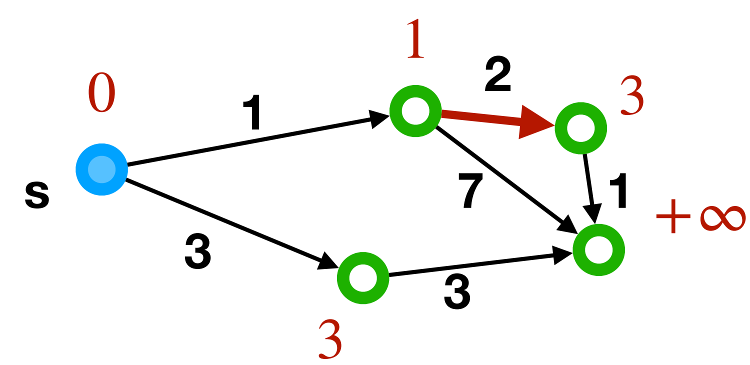
Bellman-Ford Algorithm: Example

- 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.



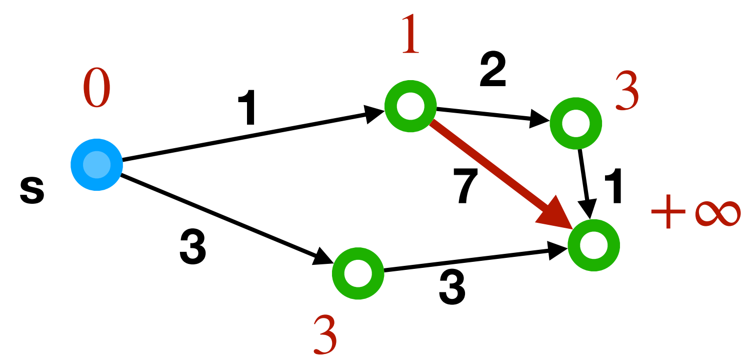
Bellman-Ford Algorithm: Example

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

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

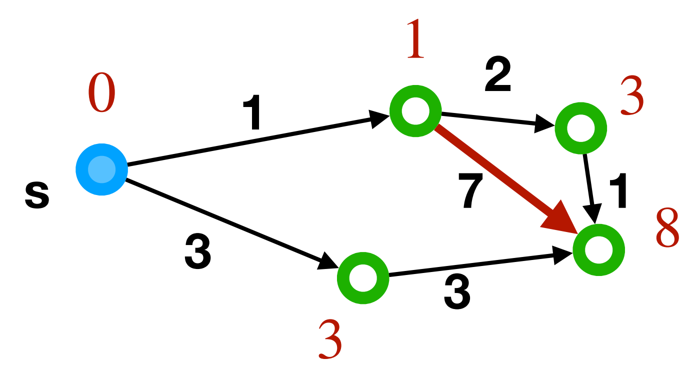
  - If $d[v] > d[u] + w_{uv}$
    update $d[v] \leftarrow d[u] + w_{uv}$

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



Bellman-Ford Algorithm: Example

- 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.



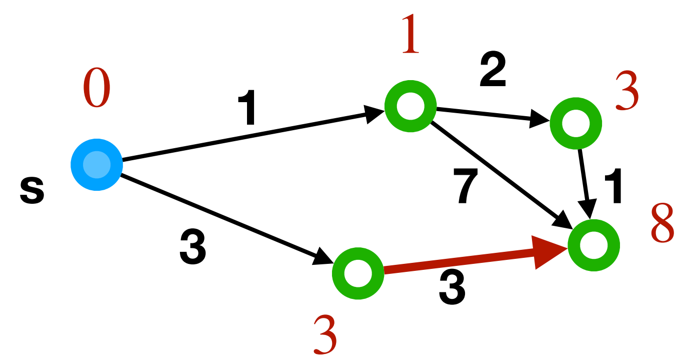
Bellman-Ford Algorithm: Example

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

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

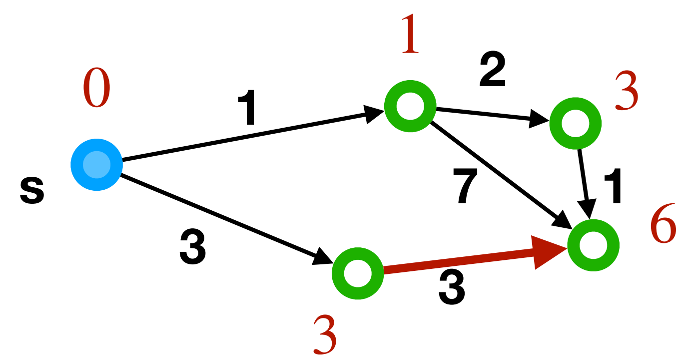
  - If $d[v] > d[u] + w_{uv}$ **update** $d[v] \leftarrow d[u] + w_{uv}$

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



Bellman-Ford Algorithm: Example

- 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.



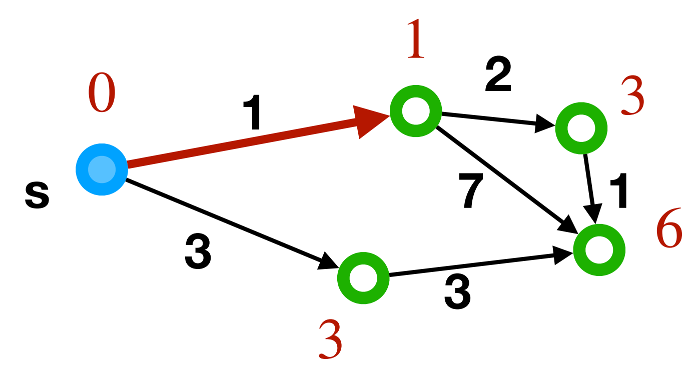
Bellman-Ford Algorithm: Example

- 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.



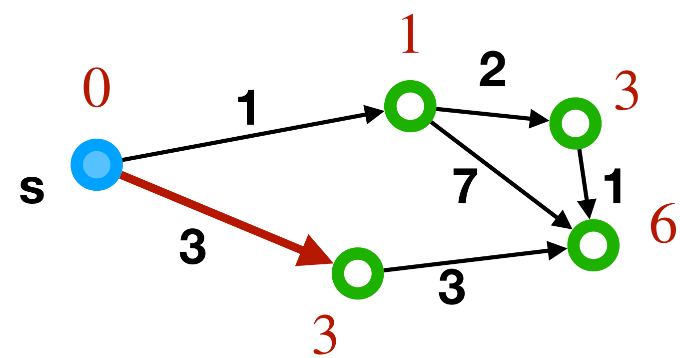
Bellman-Ford Algorithm: Example

- 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.



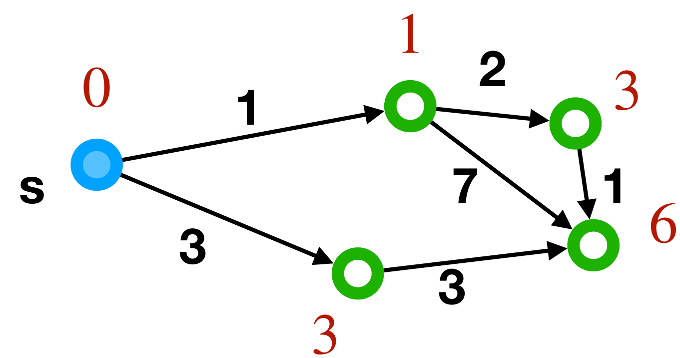
Bellman-Ford Algorithm: Example

- 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.



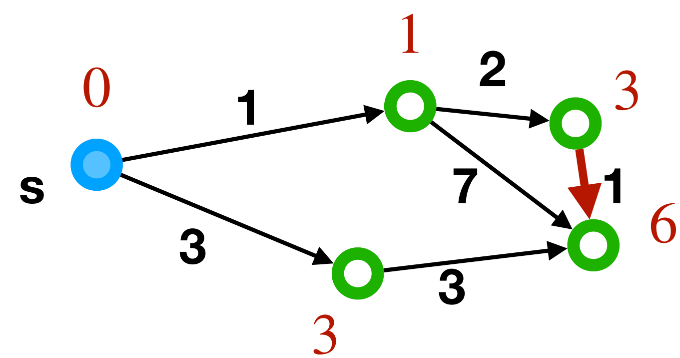
Bellman-Ford Algorithm: Example

- 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.



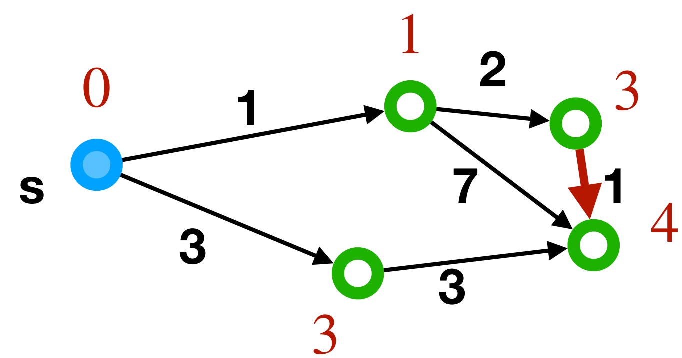
Bellman-Ford Algorithm: Example

- 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.

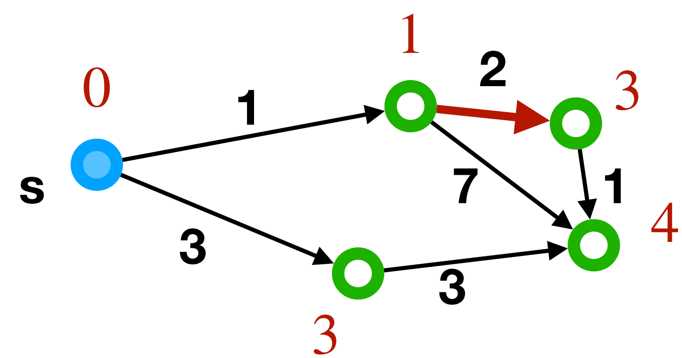


Bellman-Ford Algorithm: Example

- 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.



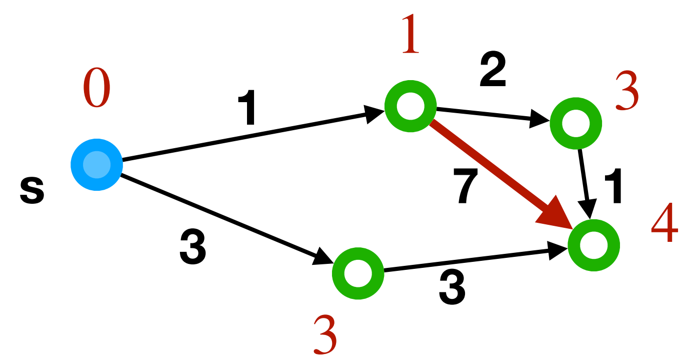
Bellman-Ford Algorithm: Example

- 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.



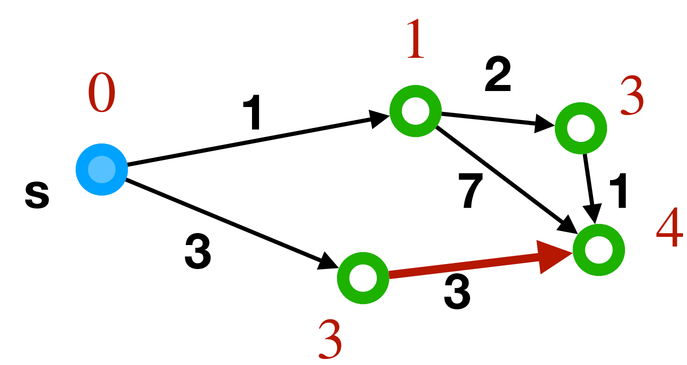
Bellman-Ford Algorithm: Example

- 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.



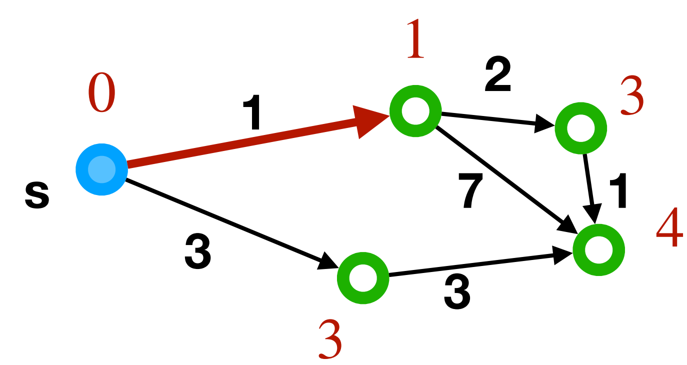
Bellman-Ford Algorithm: Example

- 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.



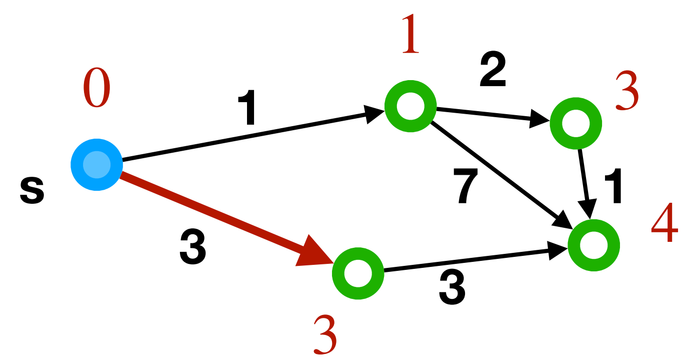
Bellman-Ford Algorithm: Example

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

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

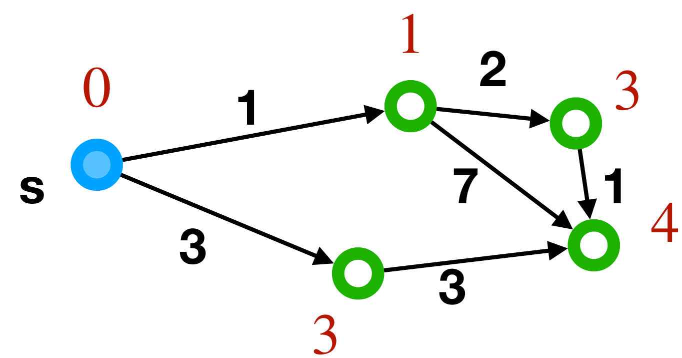
  - If $d[v] > d[u] + w_{uv}$ **update** $d[v] \leftarrow d[u] + w_{uv}$

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



Bellman-Ford Algorithm: Example

- 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.



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!