CS 344: Design and Analysis of Computer Algorithms

(Spring 2022 — Sections 5,6,7,8)

Lecture 18:
Minimum Spanning Trees:
Kruskal’s and Prim’s Algorithms

The Minimum Spanning Tree Problem

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 $F = \emptyset$ 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

Proof of Correctness

A Generic Meta-Algorithm

- Let $F = \emptyset$ 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

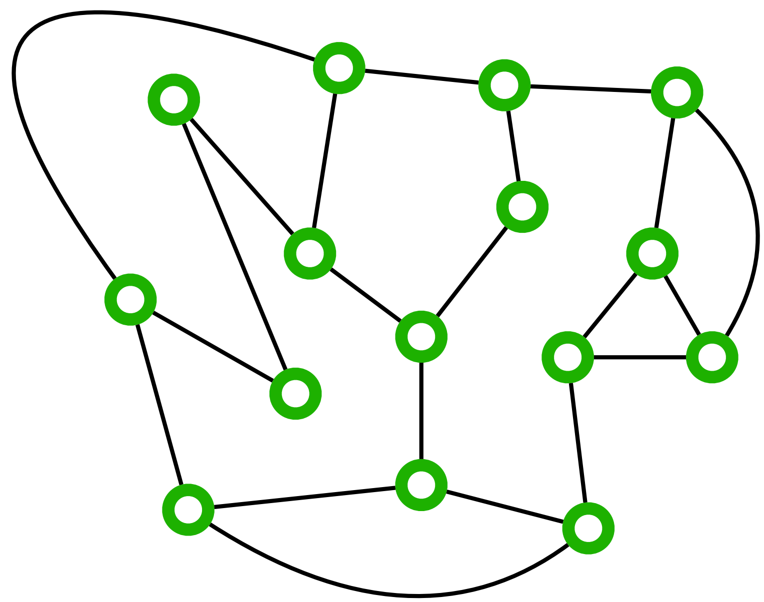
- We should now find a way of finding safe edges

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

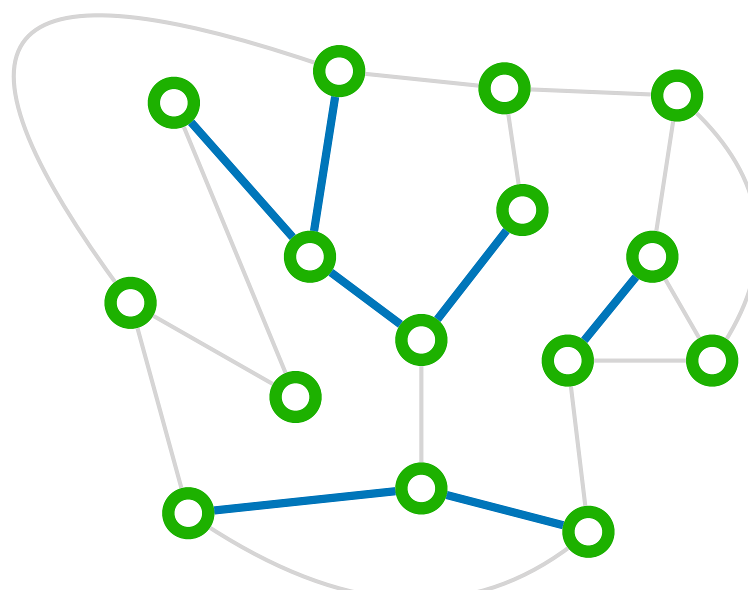
An Approach for Finding Safe Edges

- **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



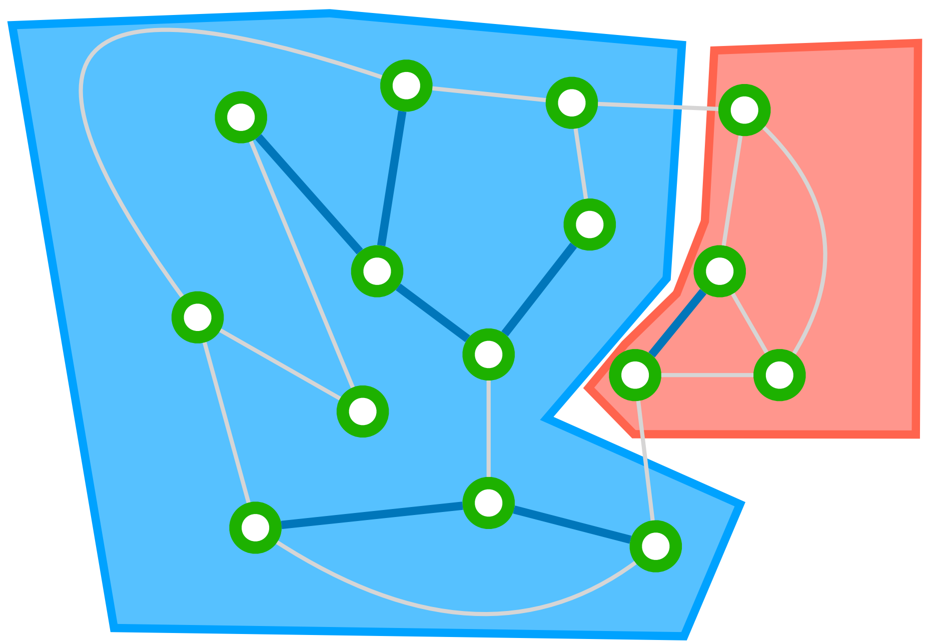
An Approach for Finding Safe Edges

- **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



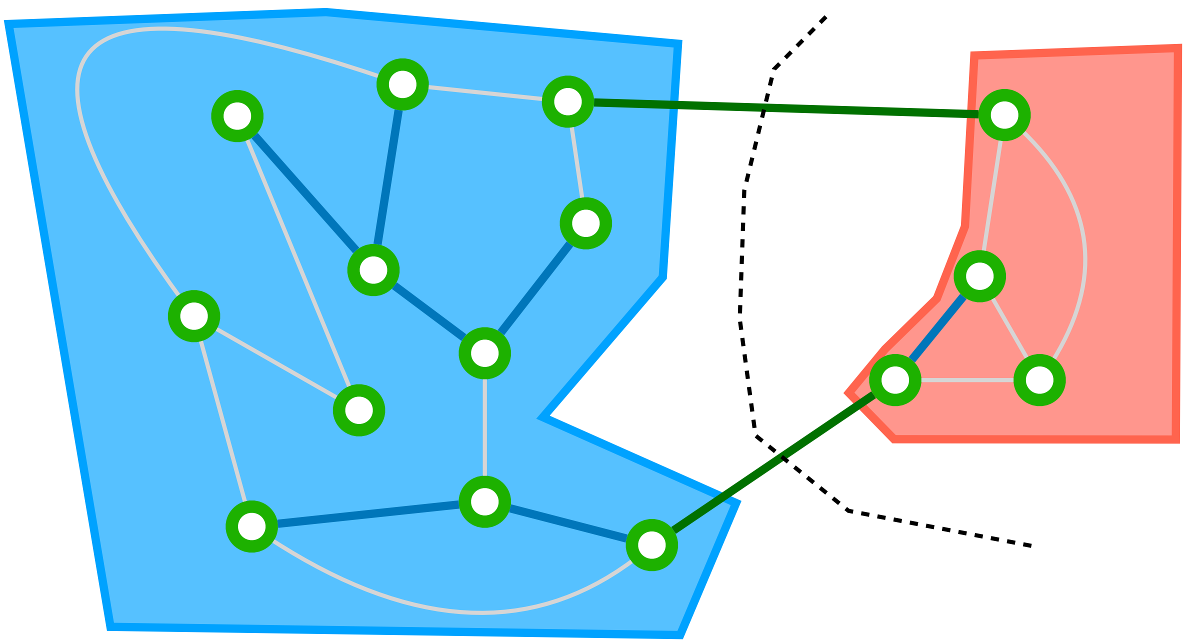
An Approach for Finding Safe Edges

- **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



Kruskal’s Algorithm

Kruskal’s Algorithm

- A canonical and efficient algorithm for MST
- Implements the strategy of on the generic meta-algorithm

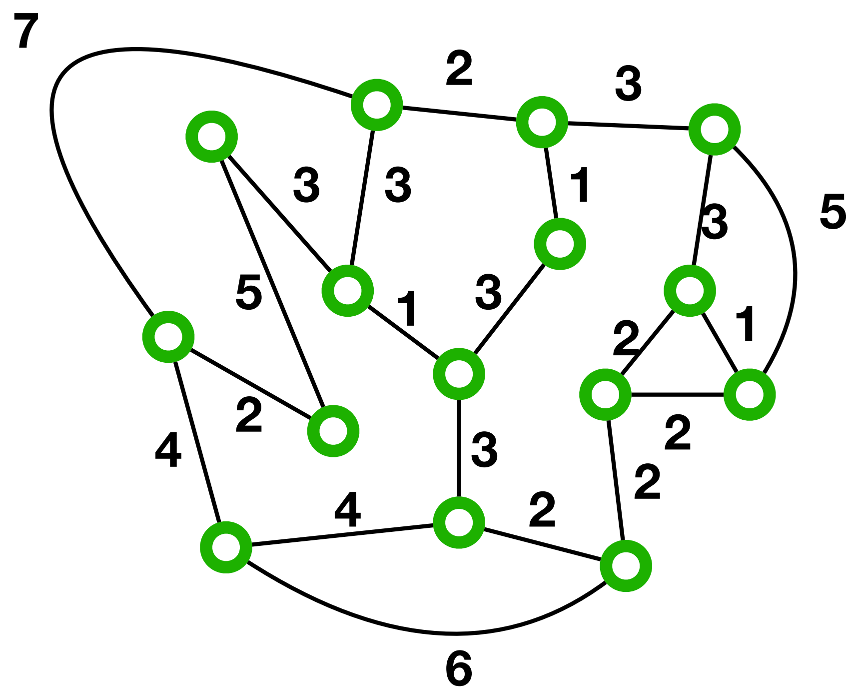
A Generic Meta-Algorithm

- Let $F = \emptyset$ 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

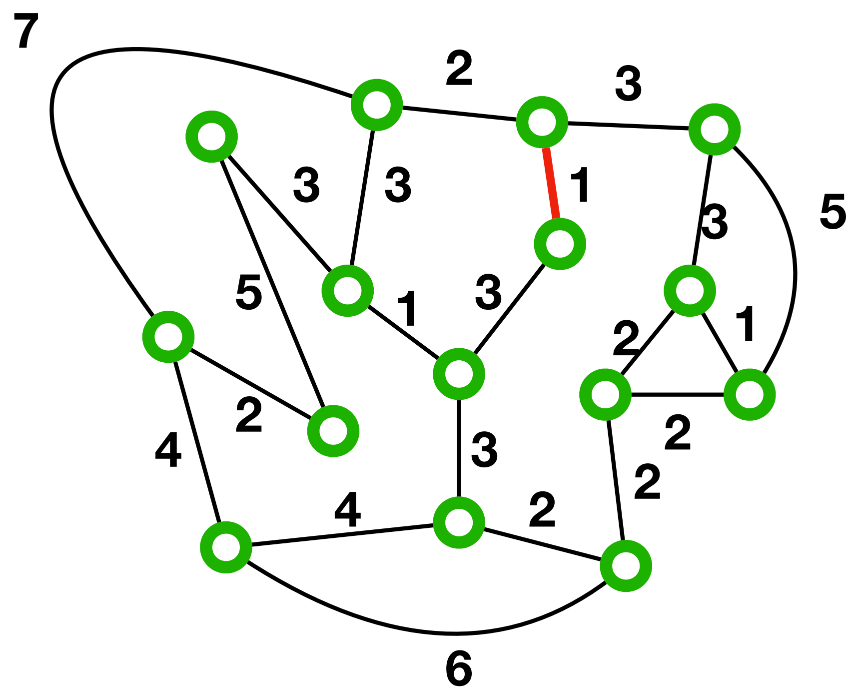
Kruskal's Algorithm

- Sort the edges in increasing (non-decreasing) order of weights
- Let $F = \emptyset$ be an empty forest initially
- For i = 1 to m:
  - if adding e_i to F does not create a cycle, let F = F + e_i
- Output F as an MST of the input

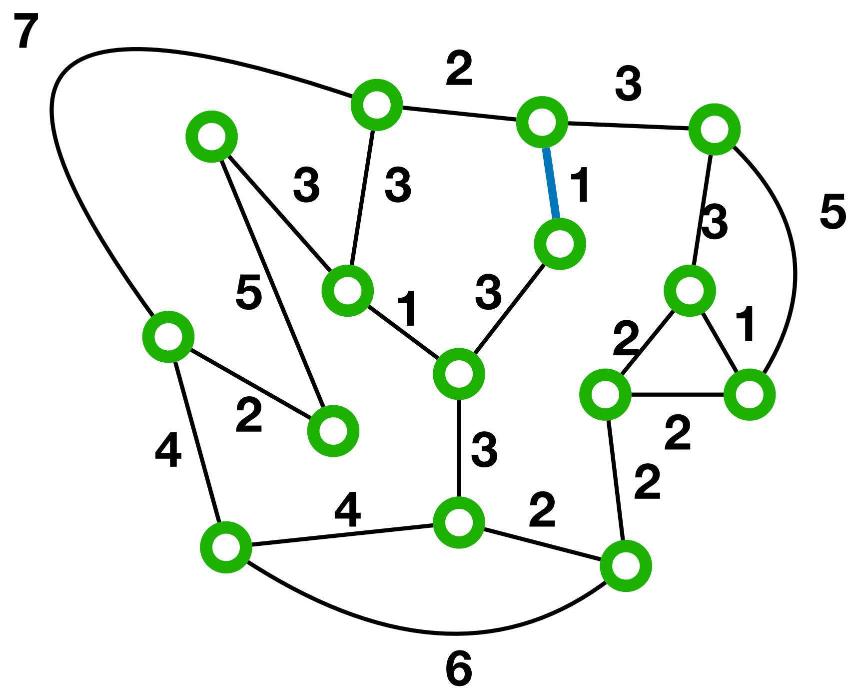
Example



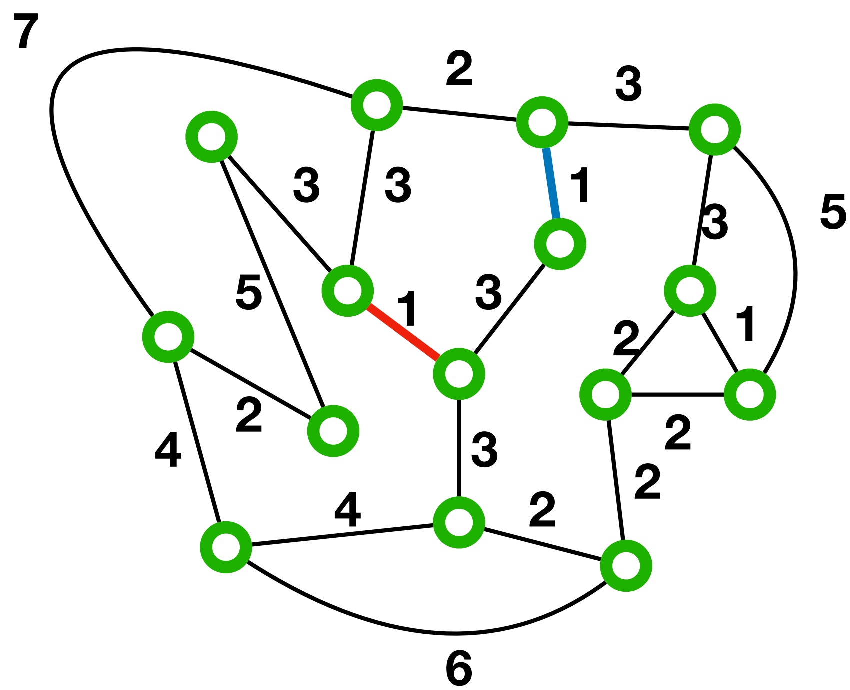
Example



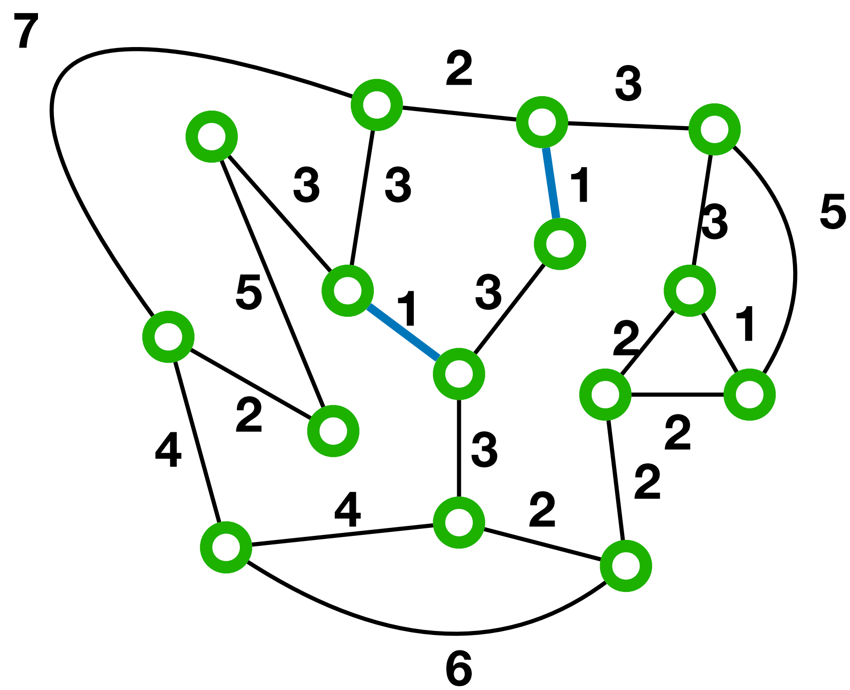
Example



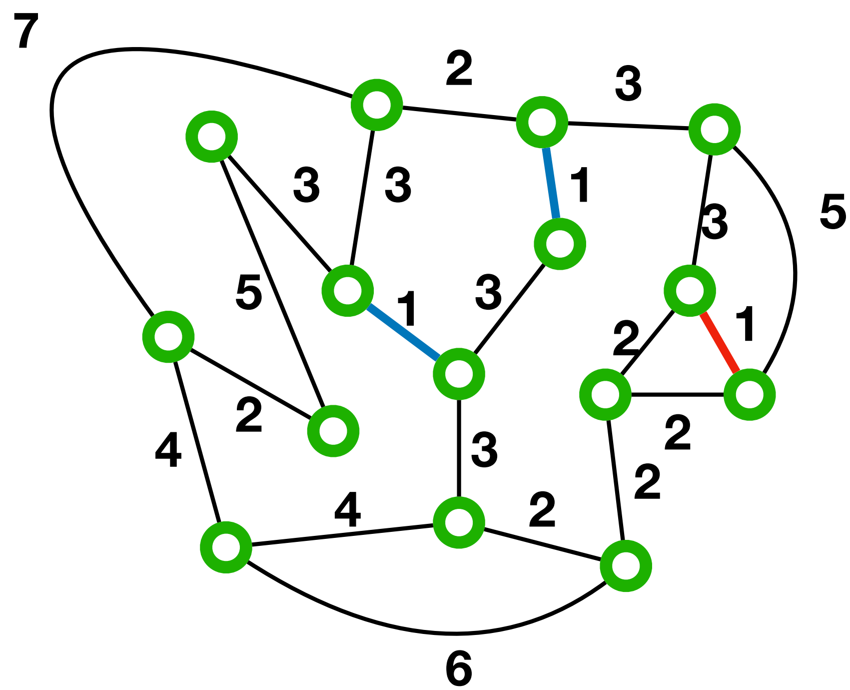
Example



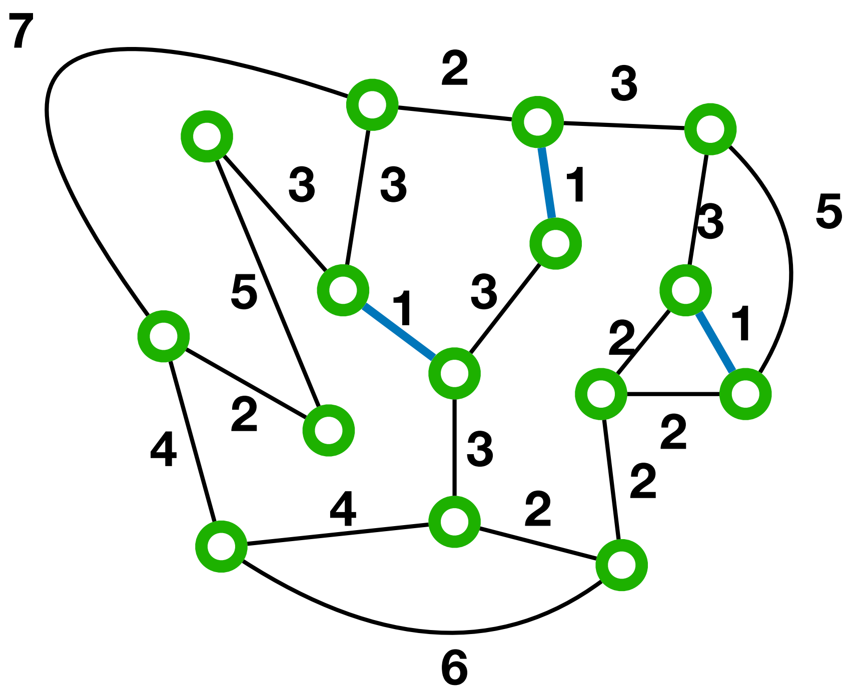
Example



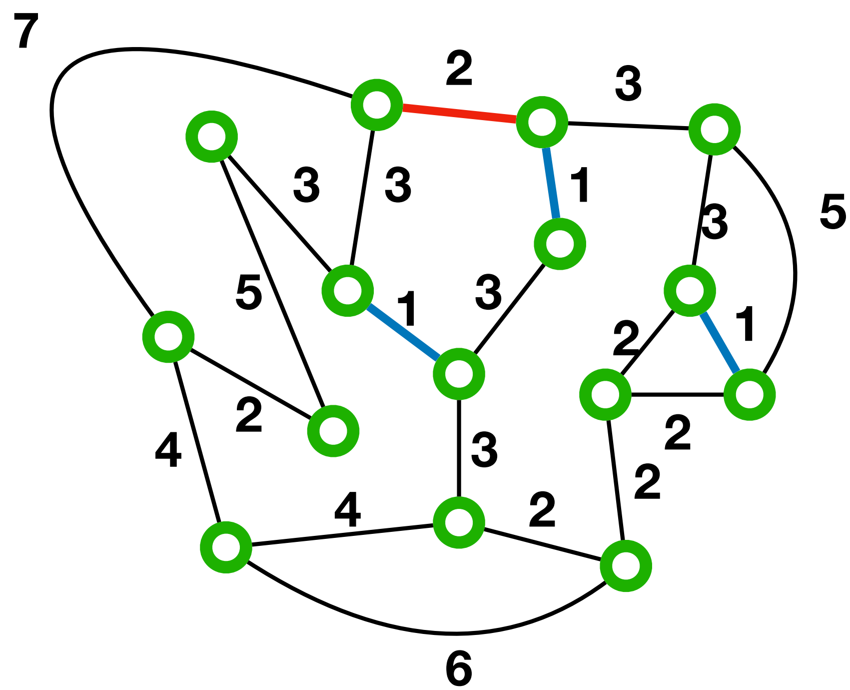
Example



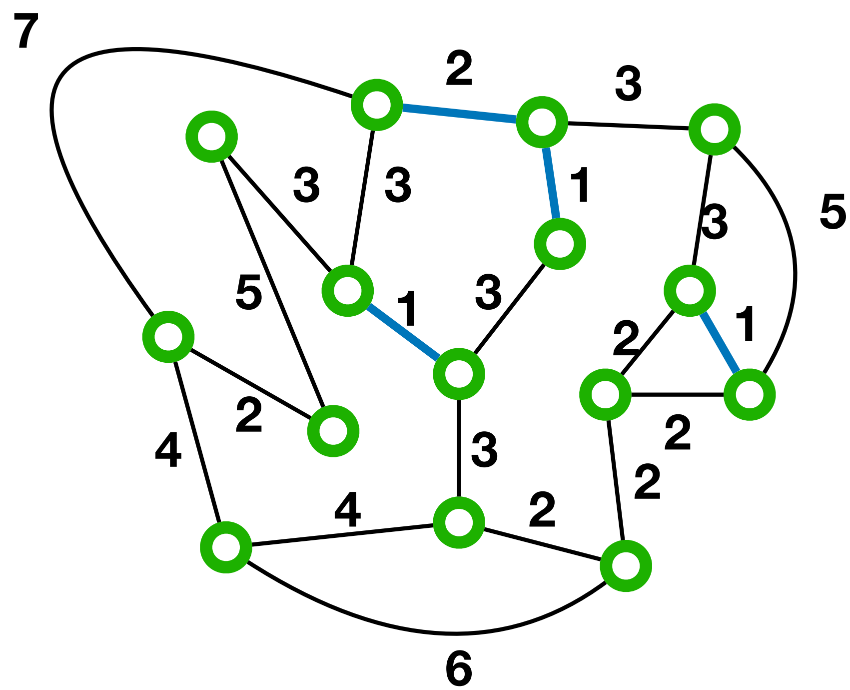
Example



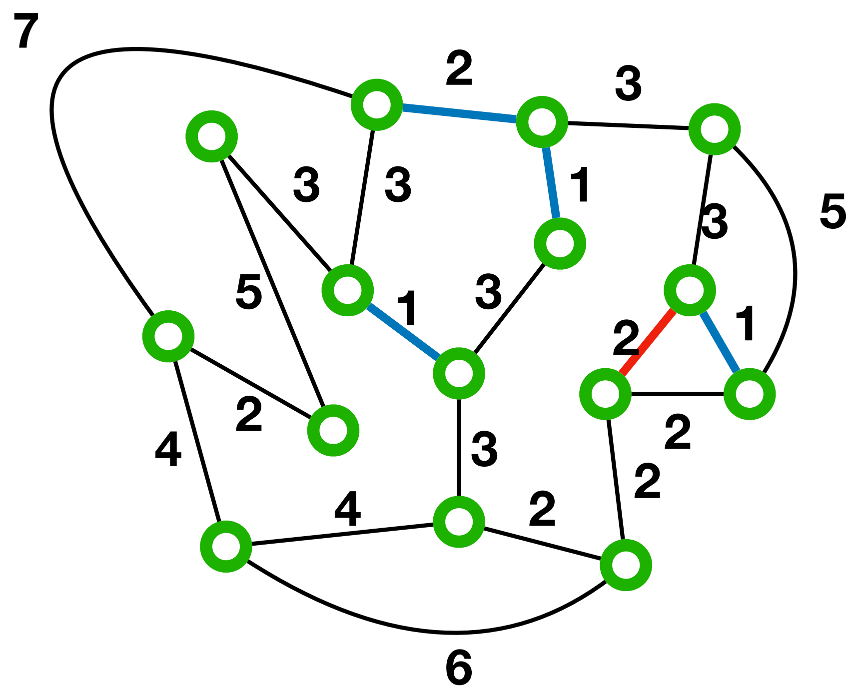
Example



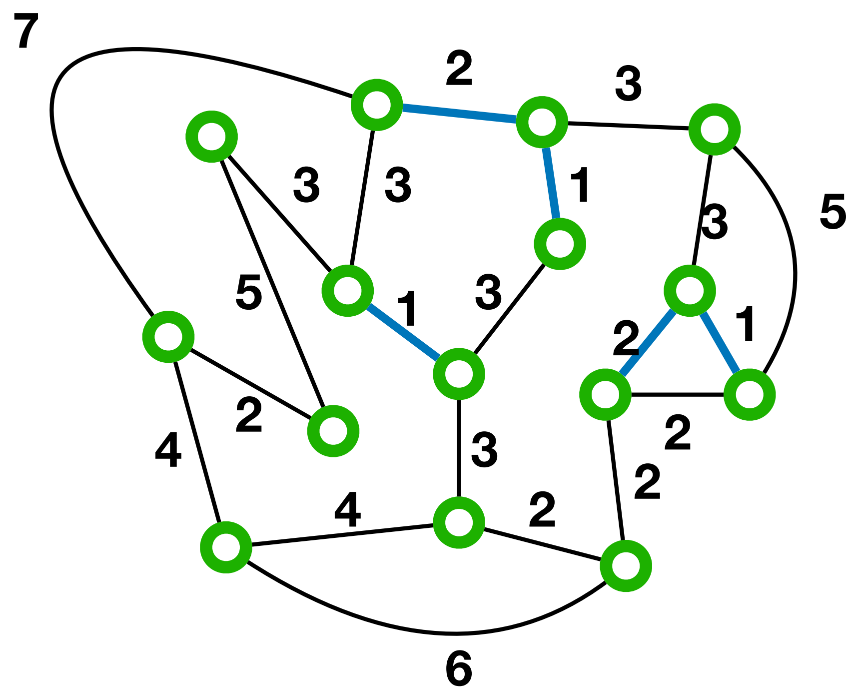
Example



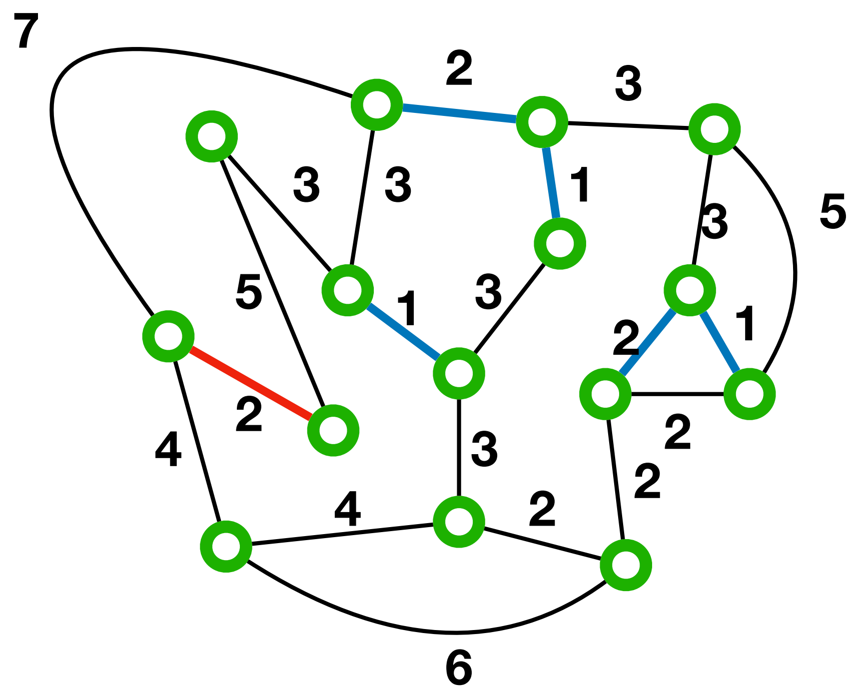
Example



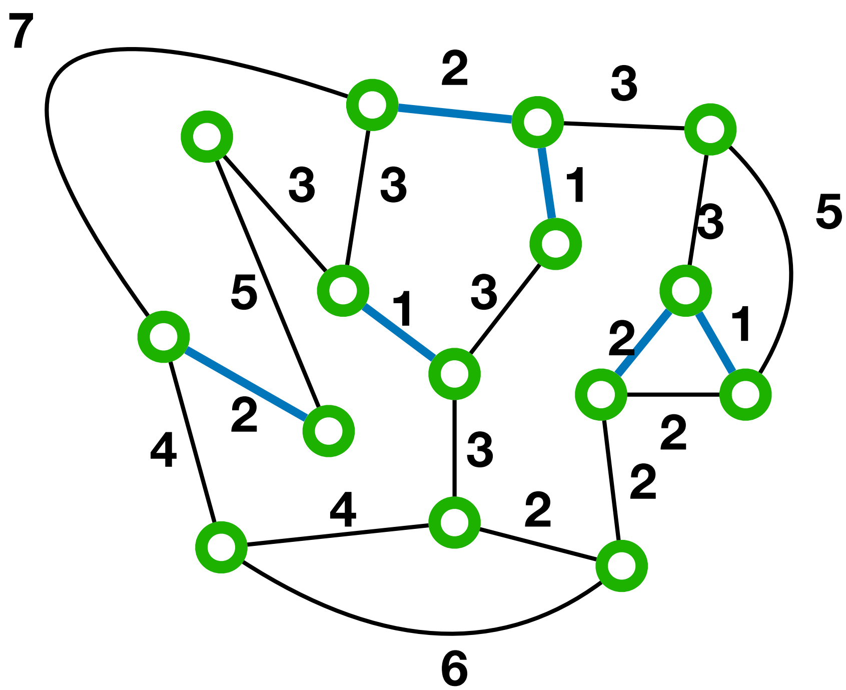
Example



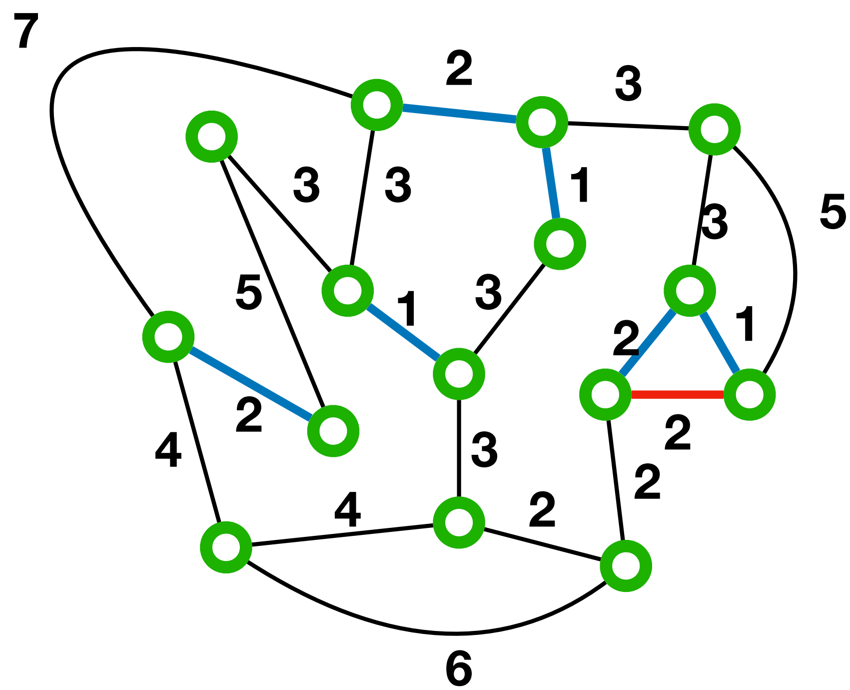
Example



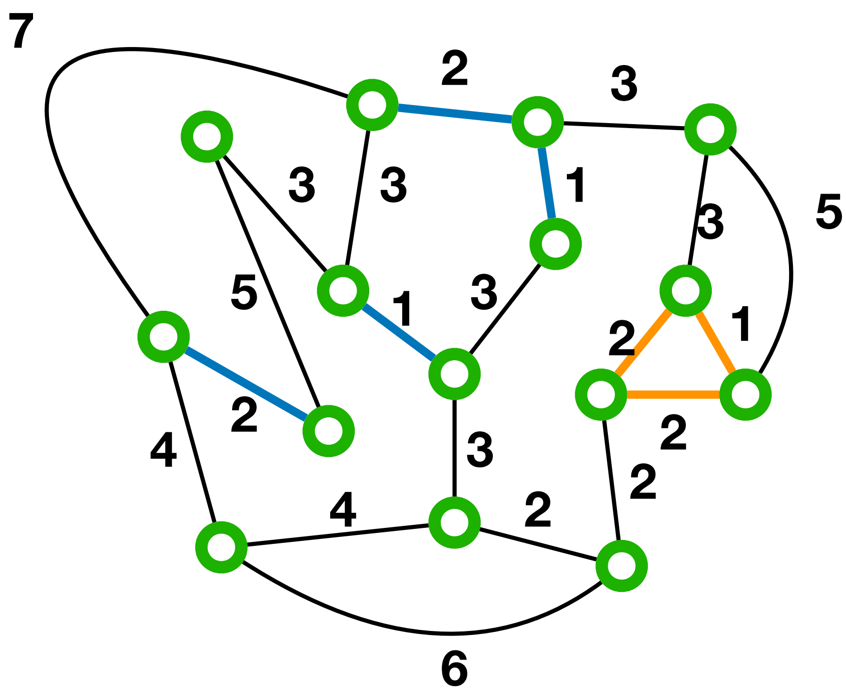
Example



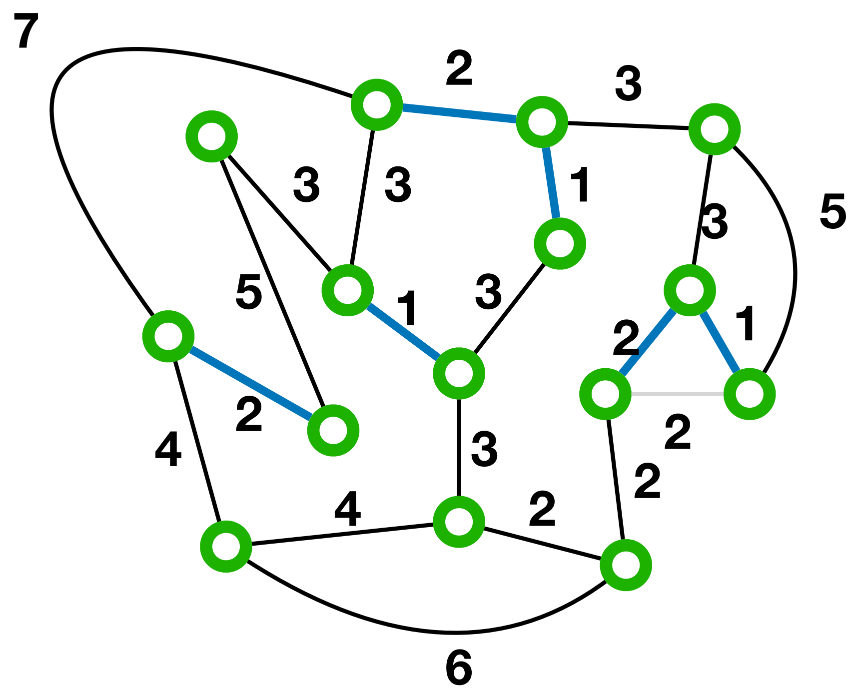
Example



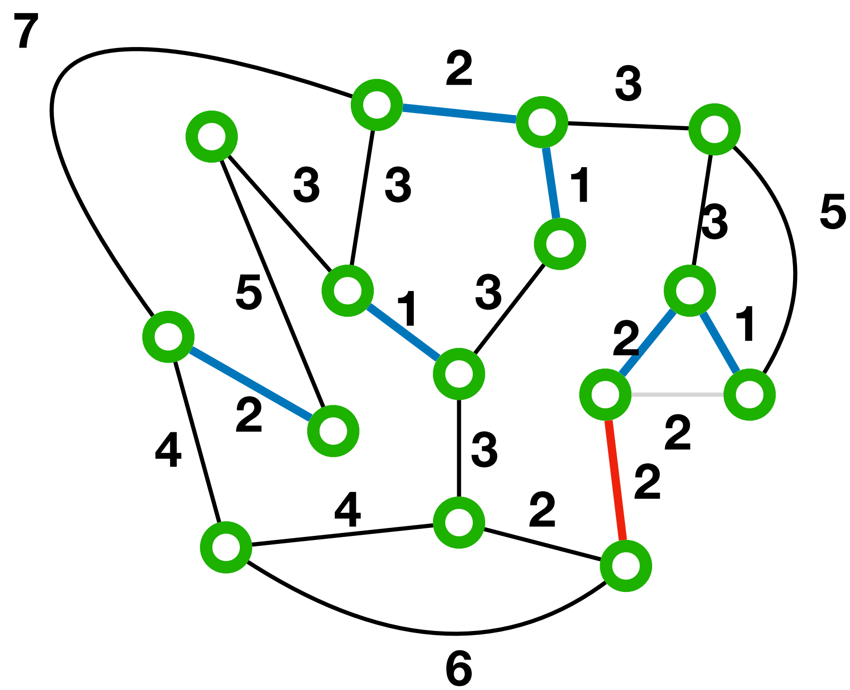
Example



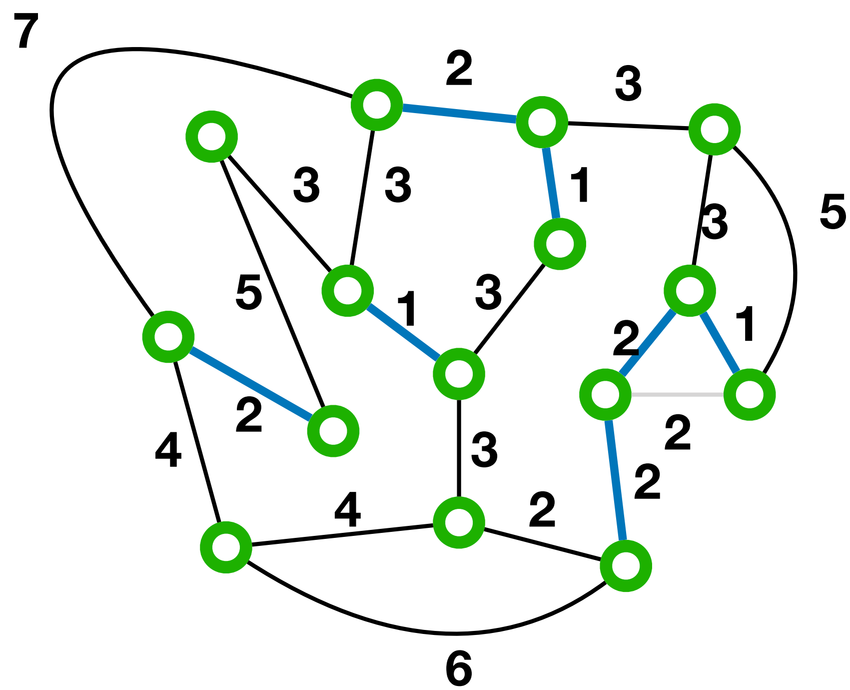
Example



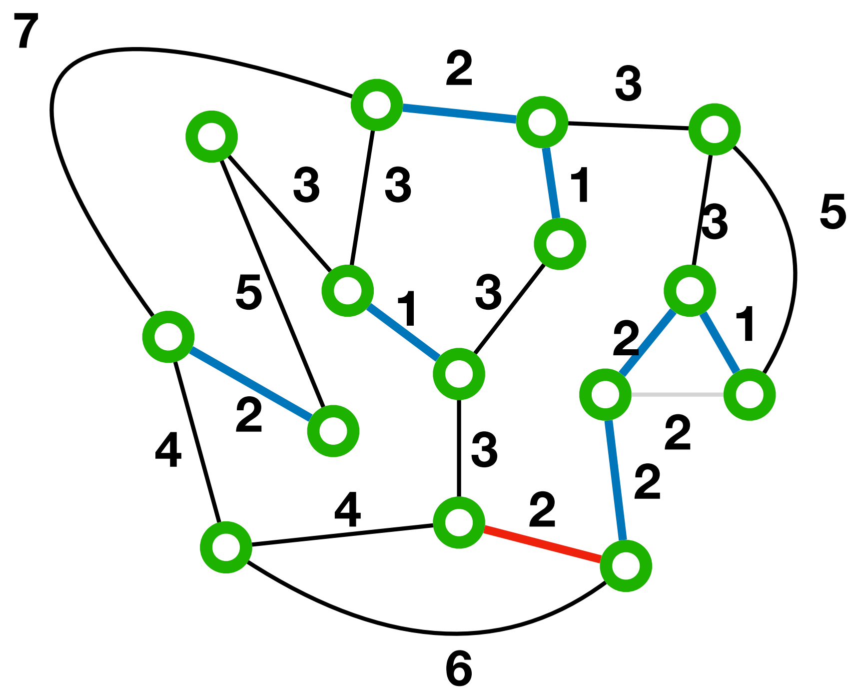
Example



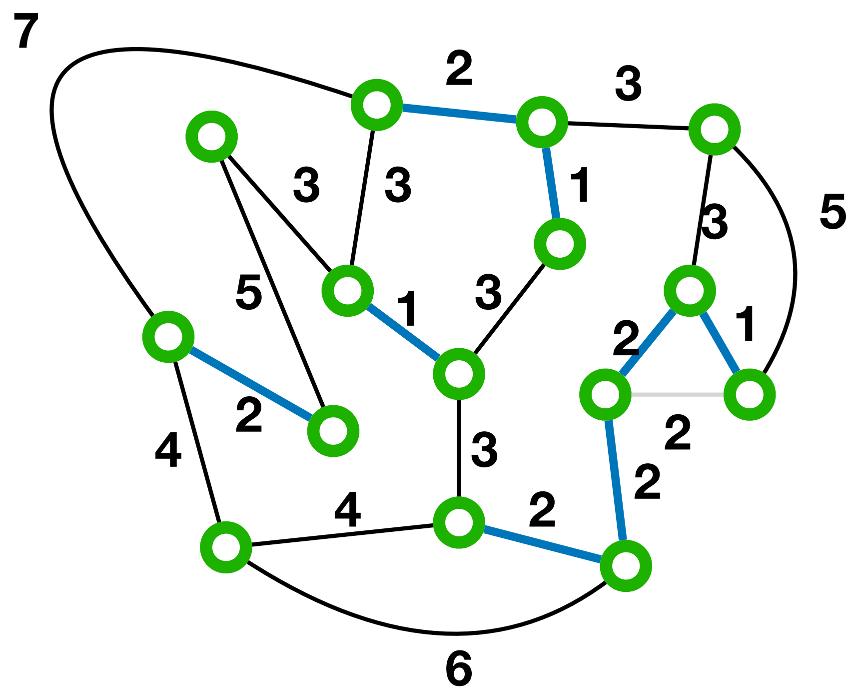
Example



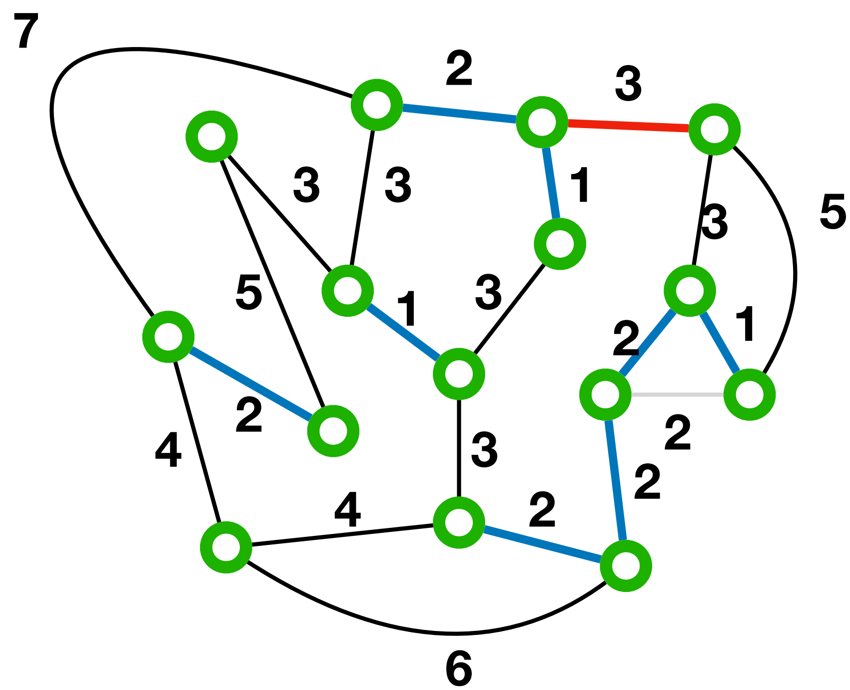
Example



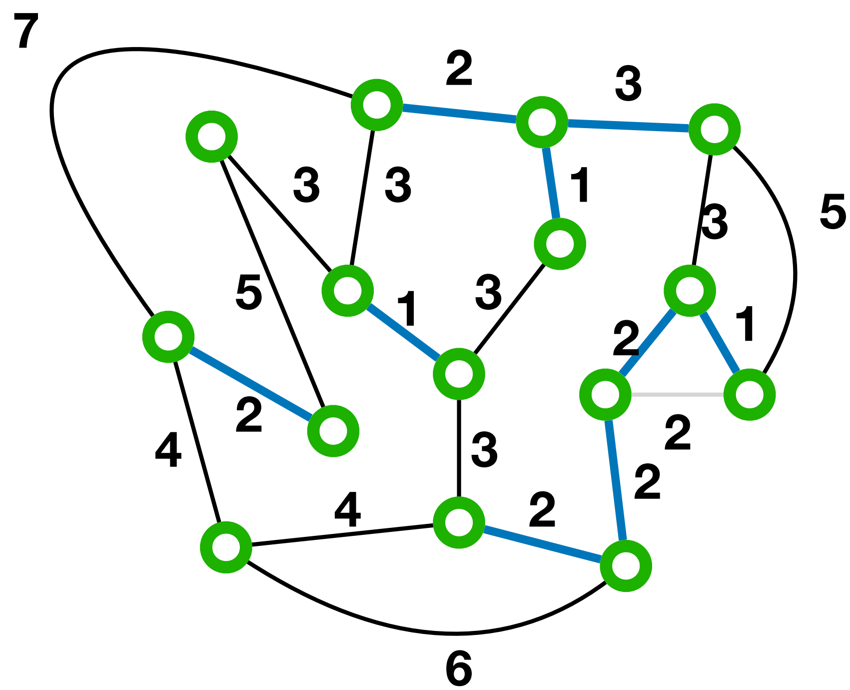
Example



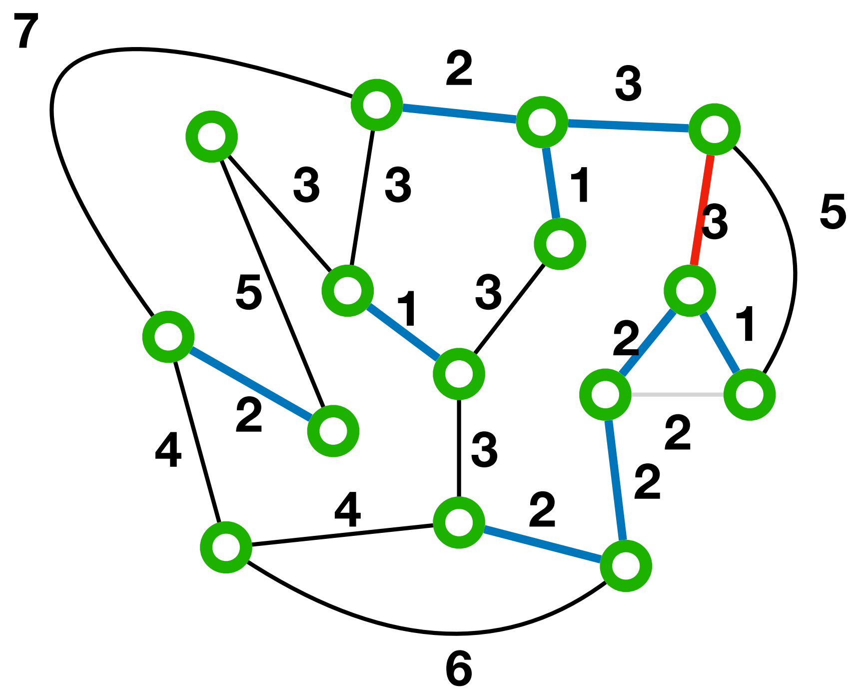
Example



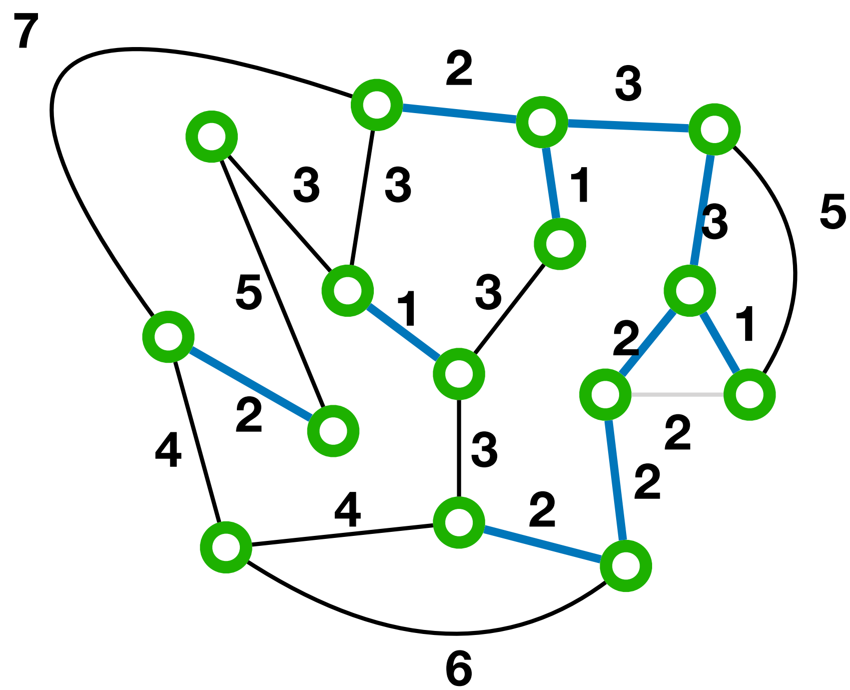
Example



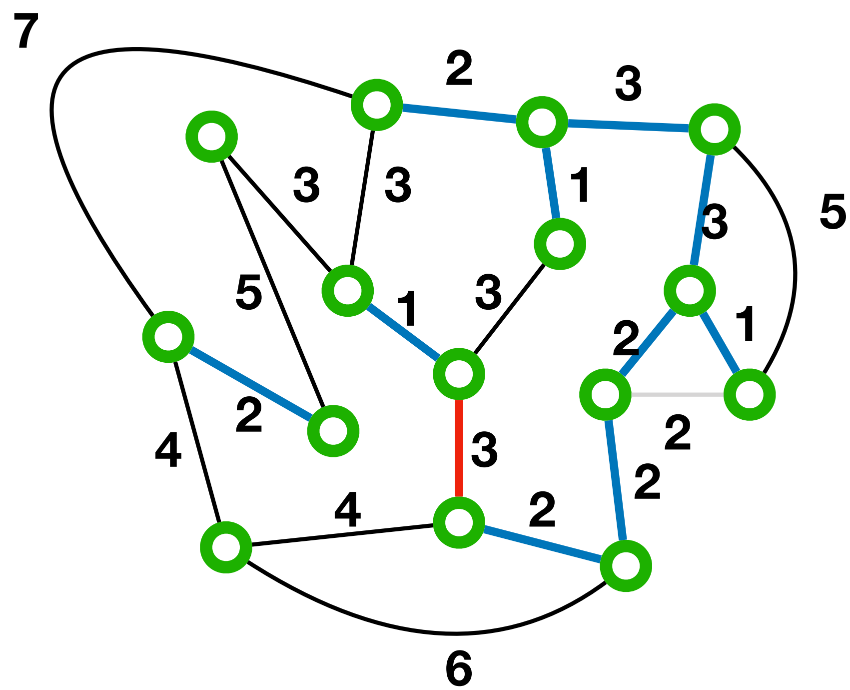
Example



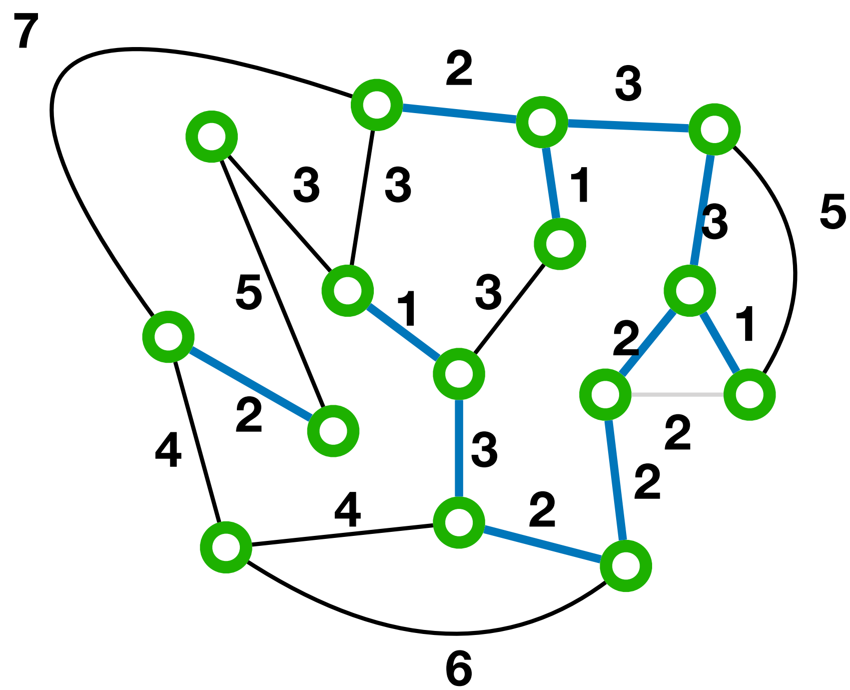
Example



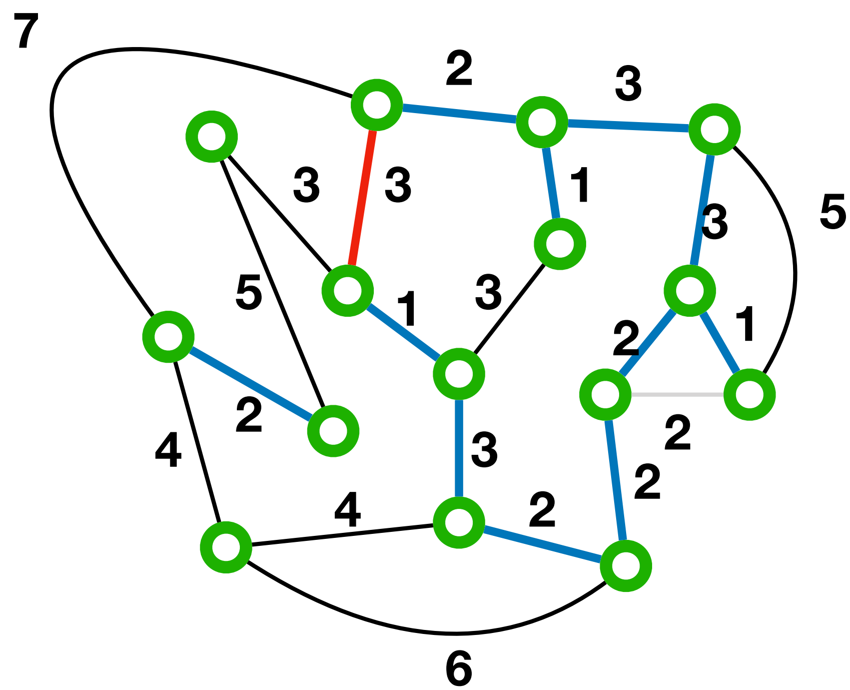
Example



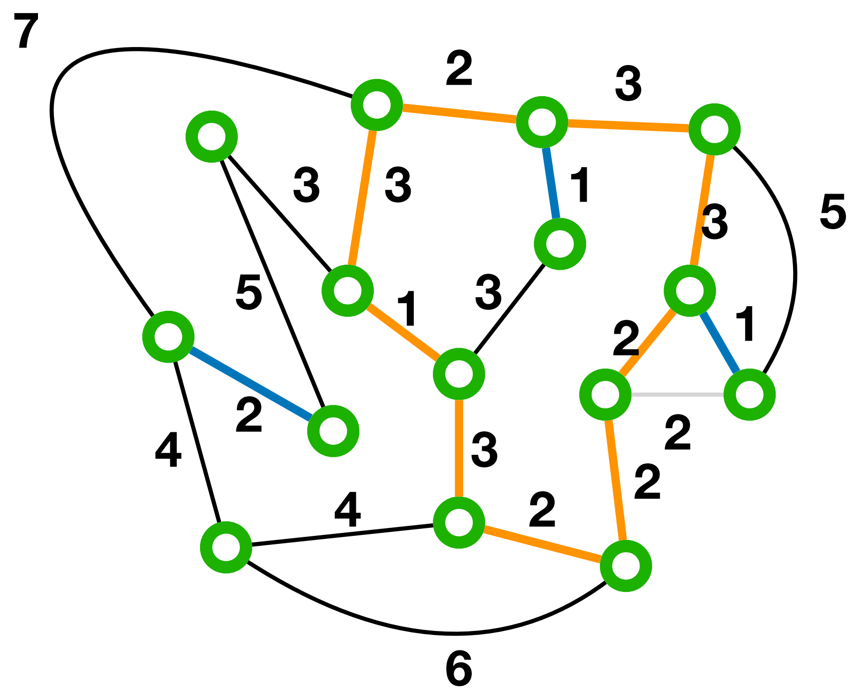
Example



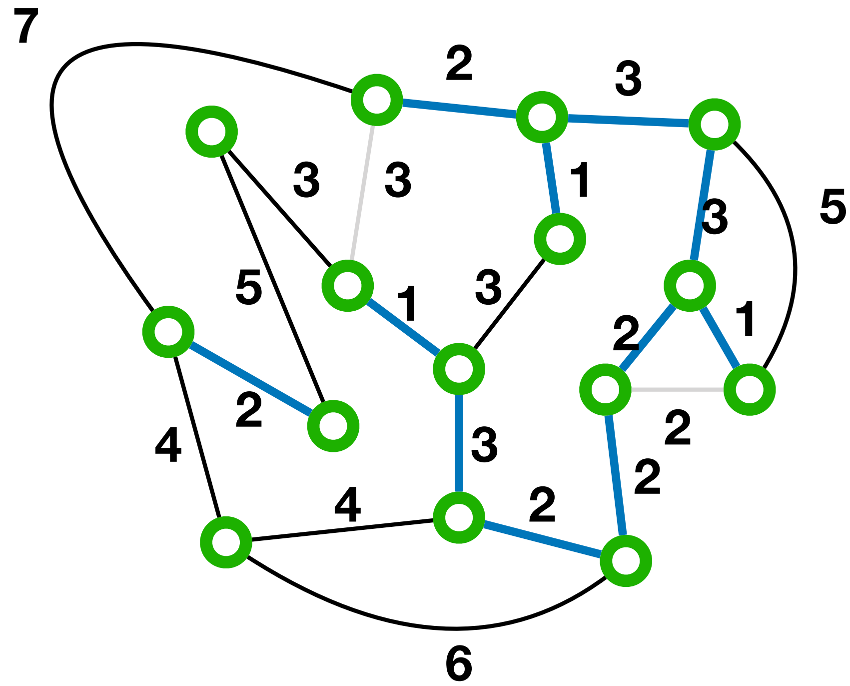
Example



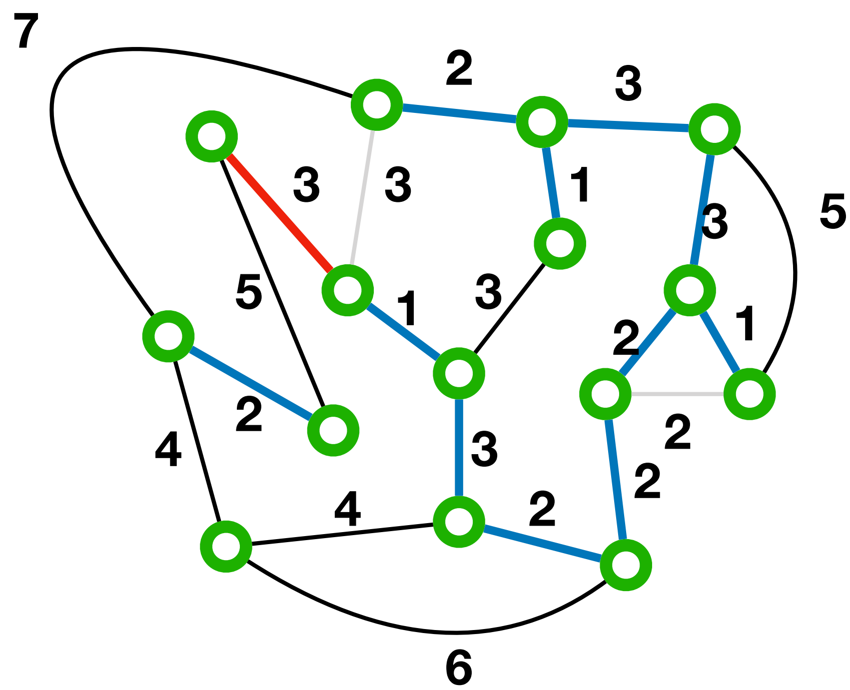
Example



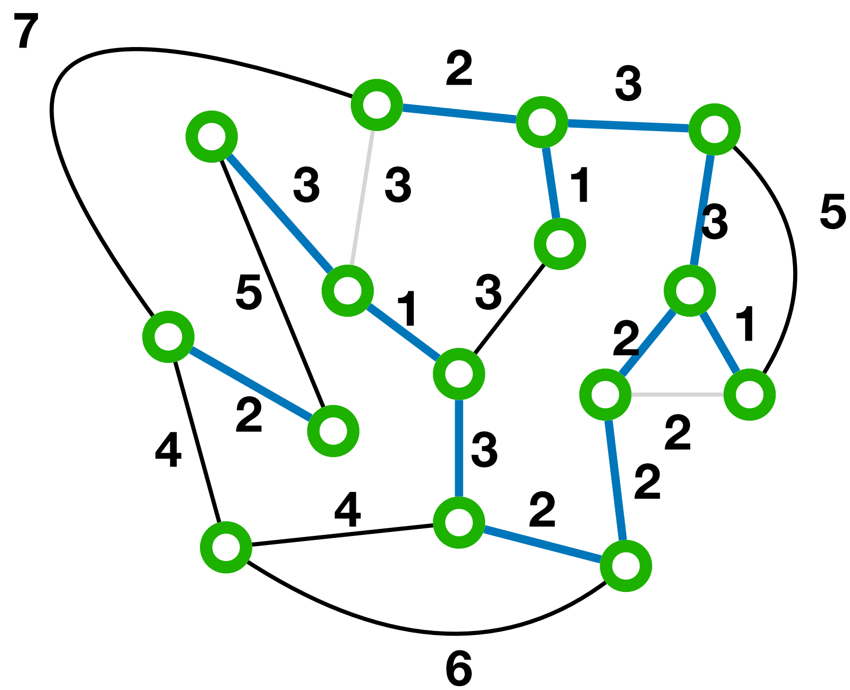
Example



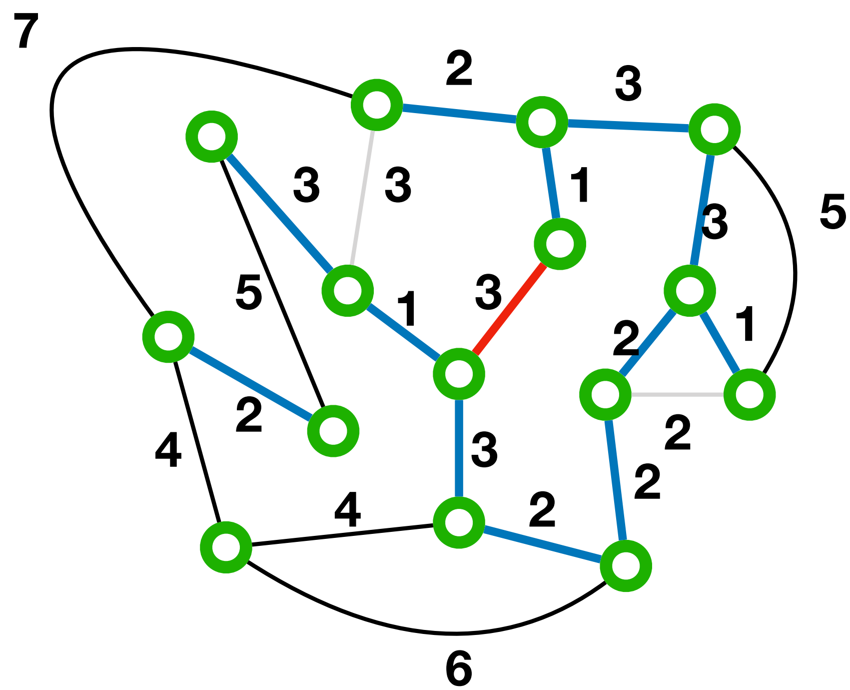
Example



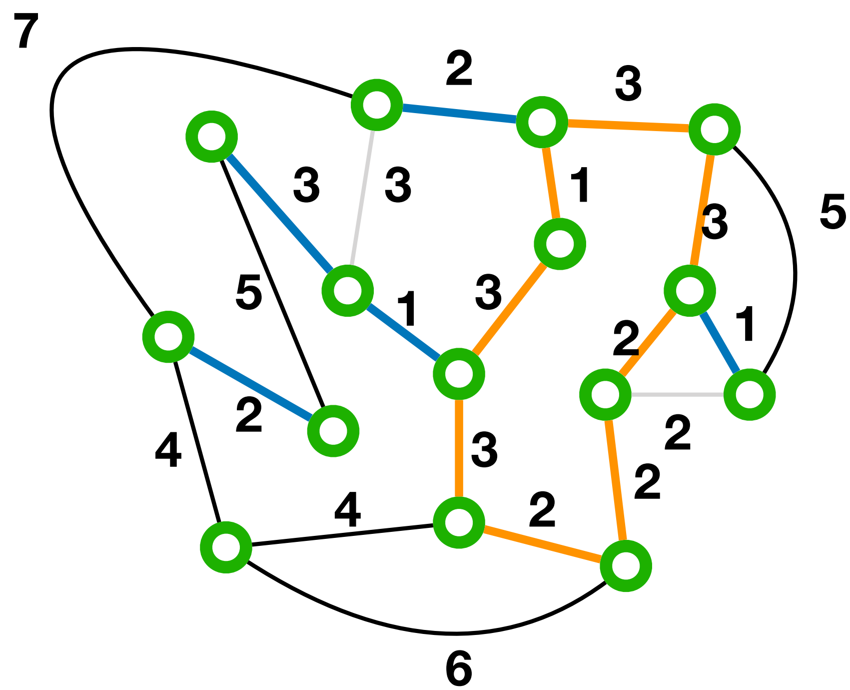
Example



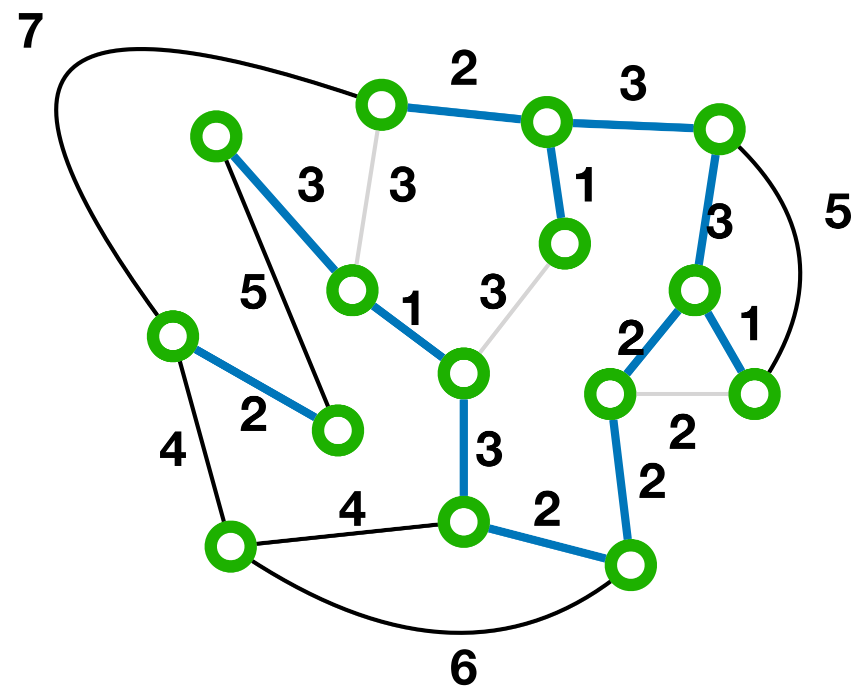
Example



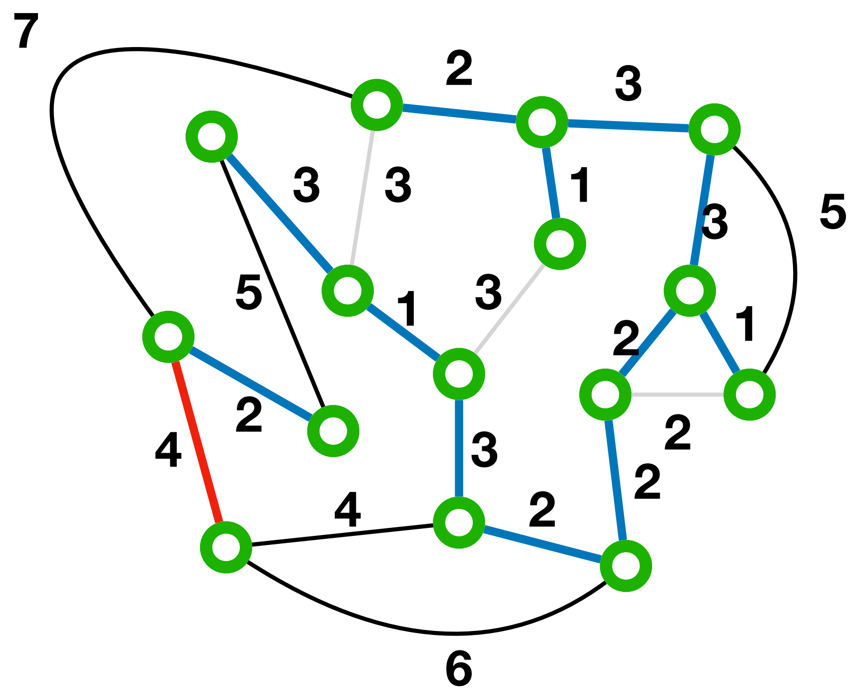
Example



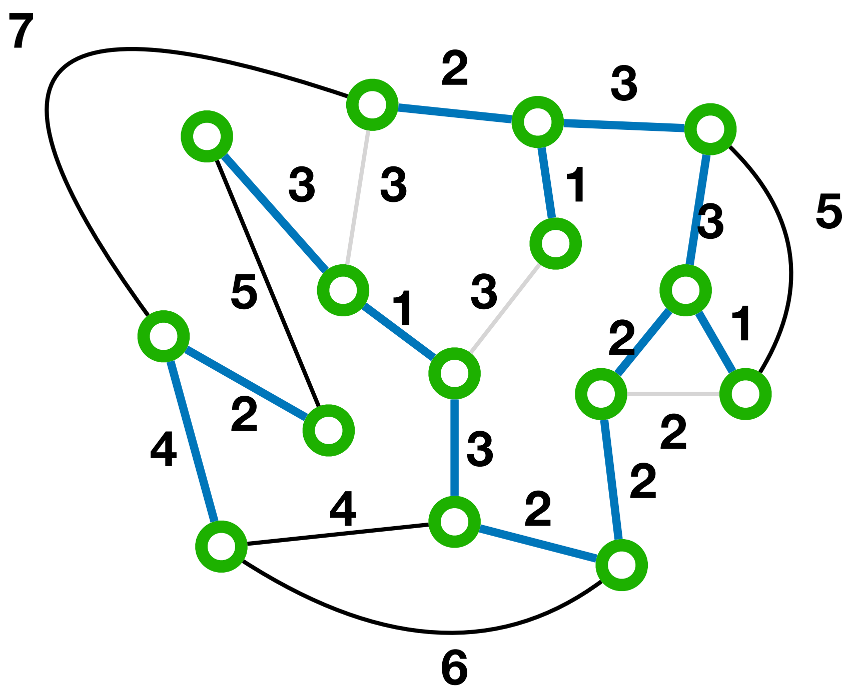
Example



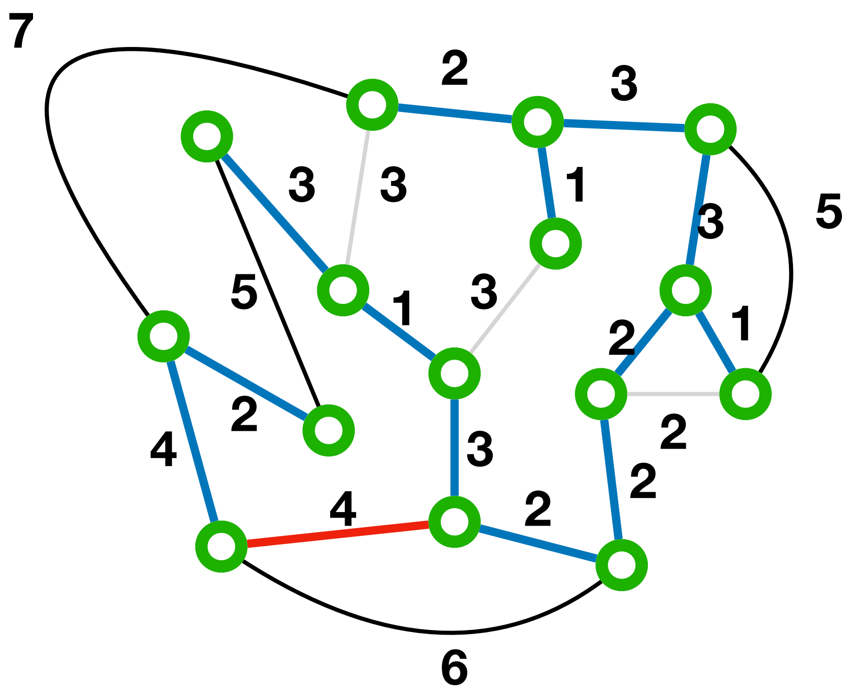
Example



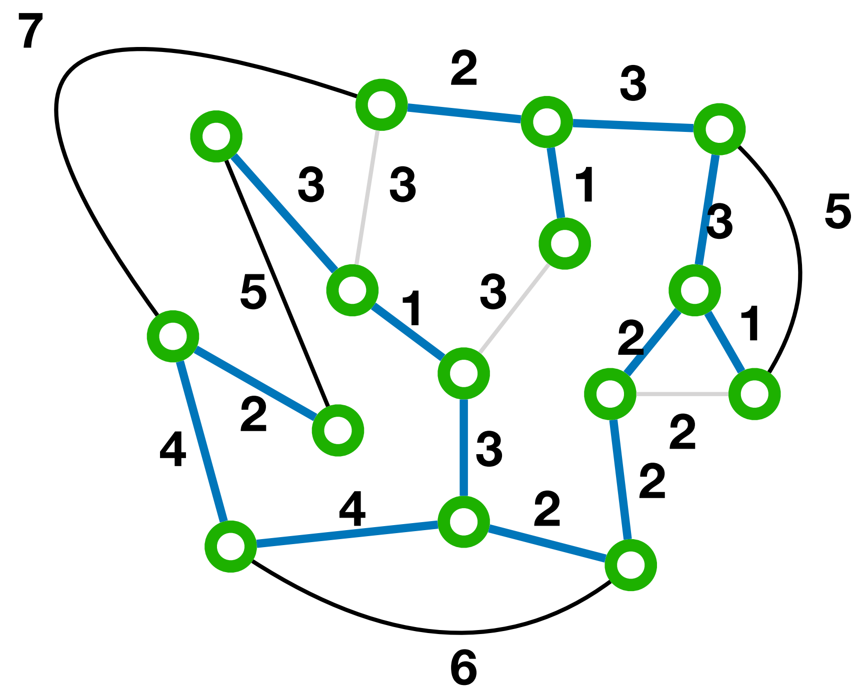
Example



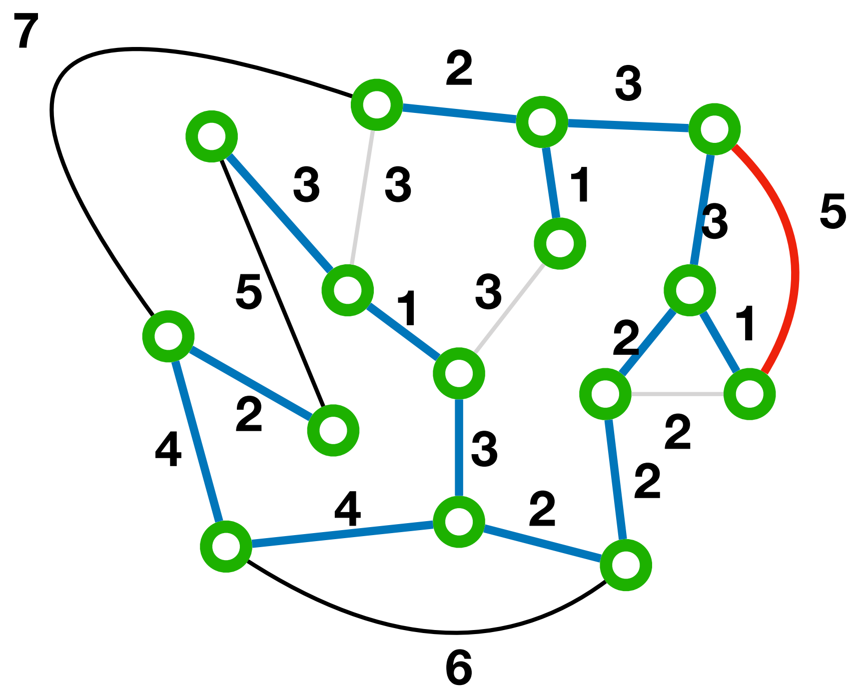
Example



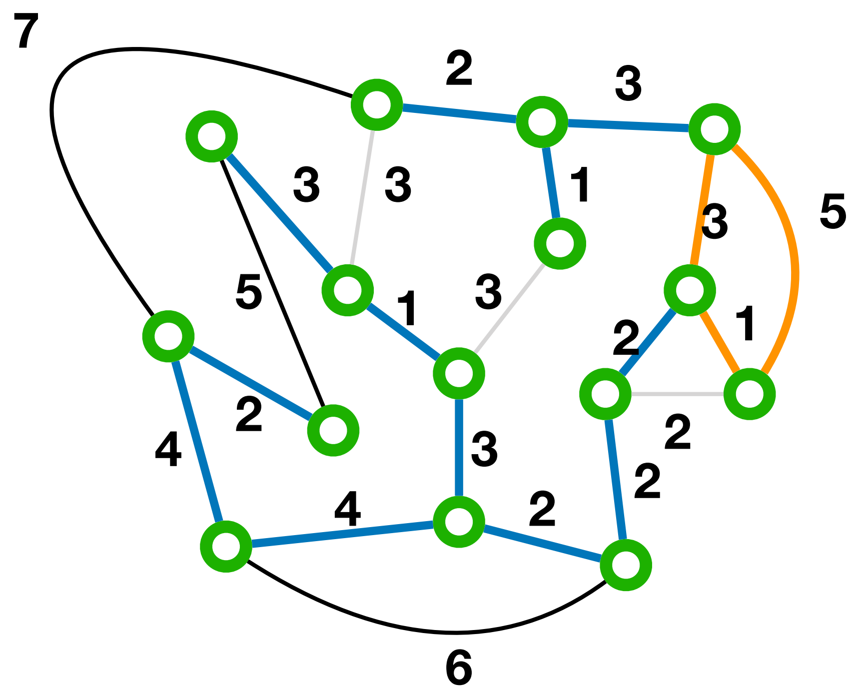
Example



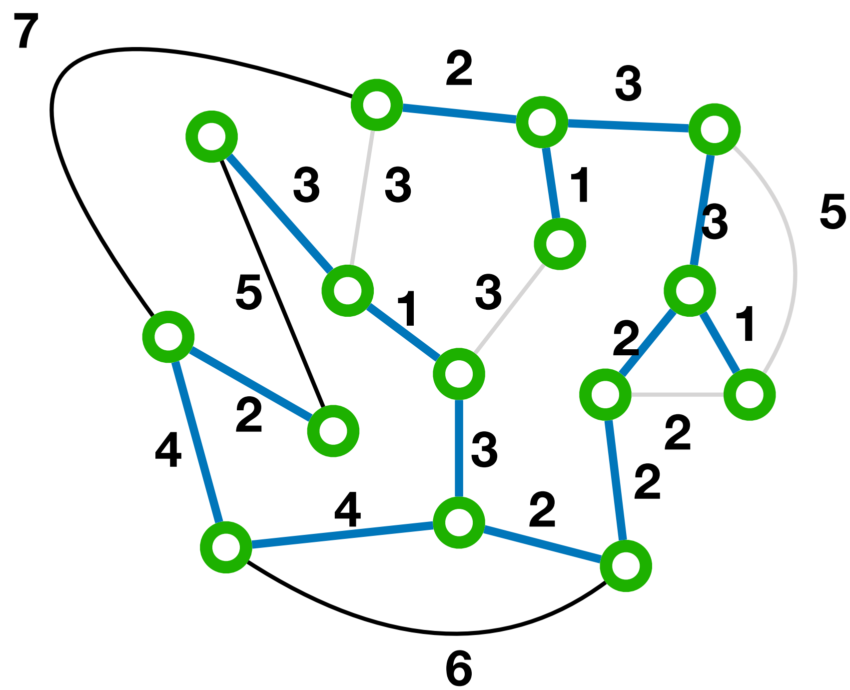
Example



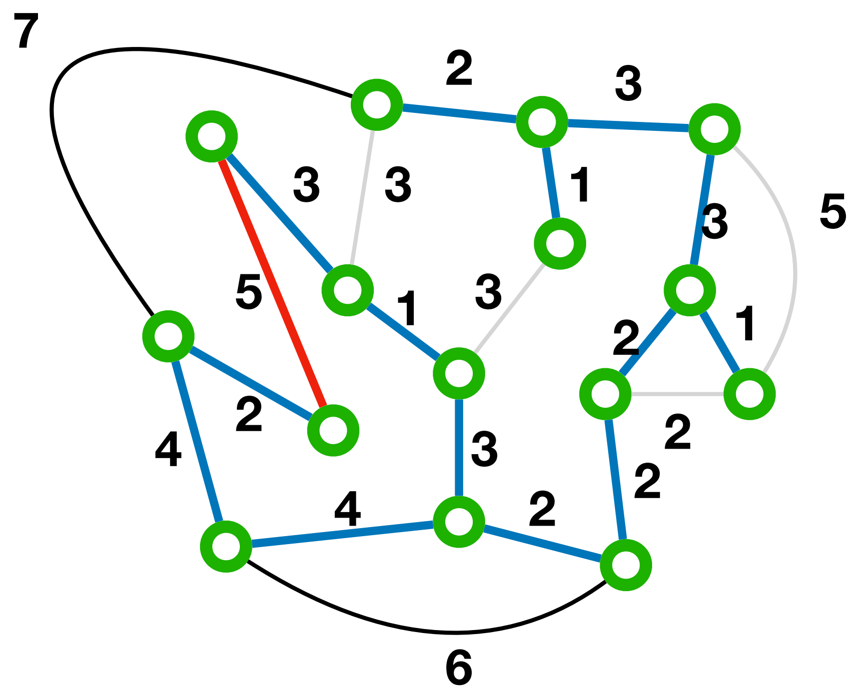
Example



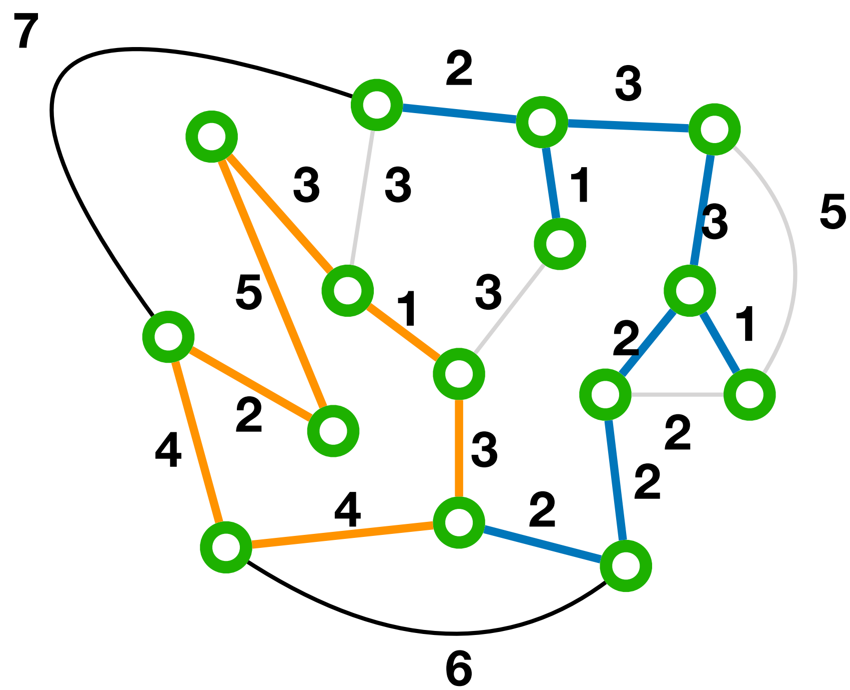
Example



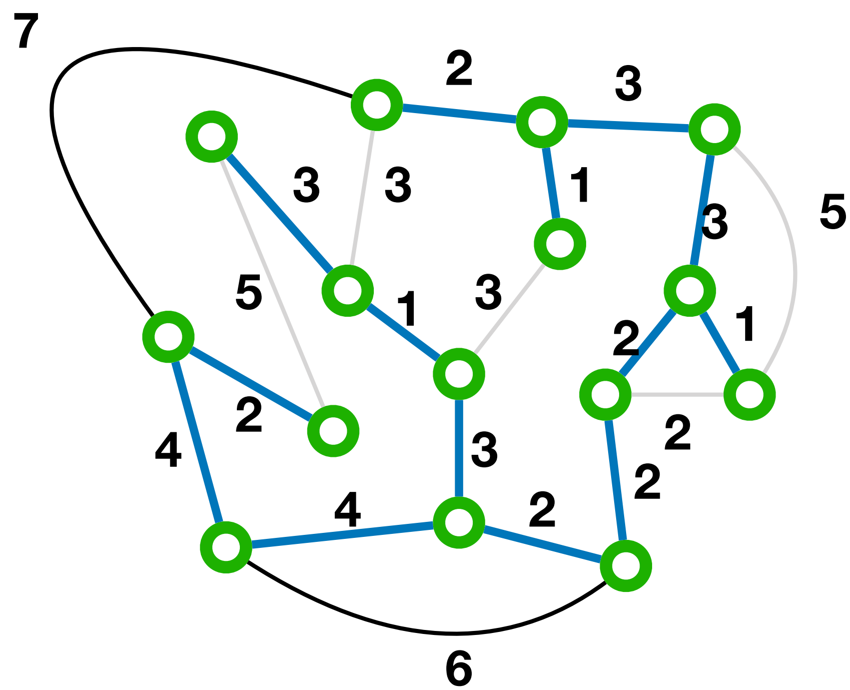
Example



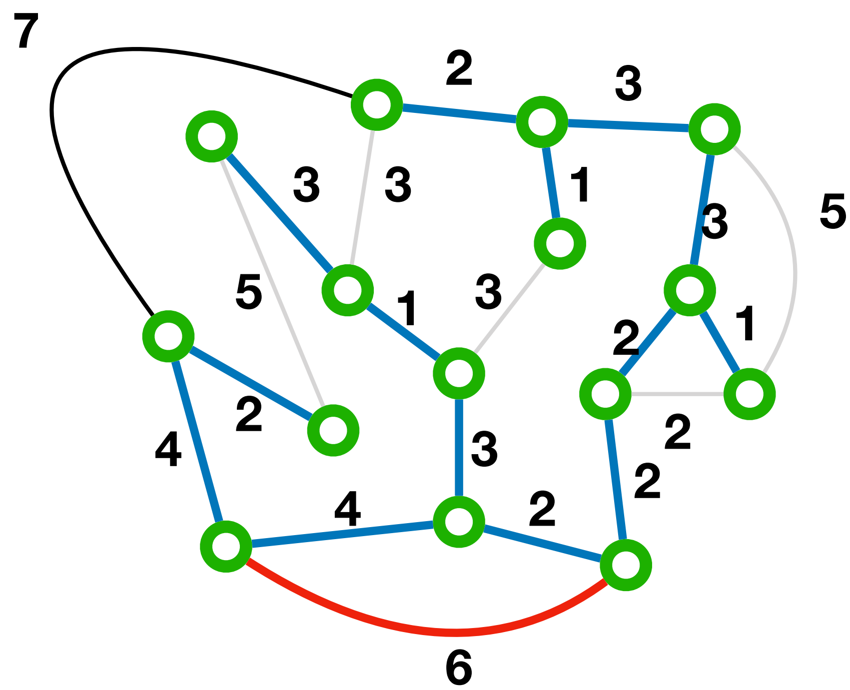
Example



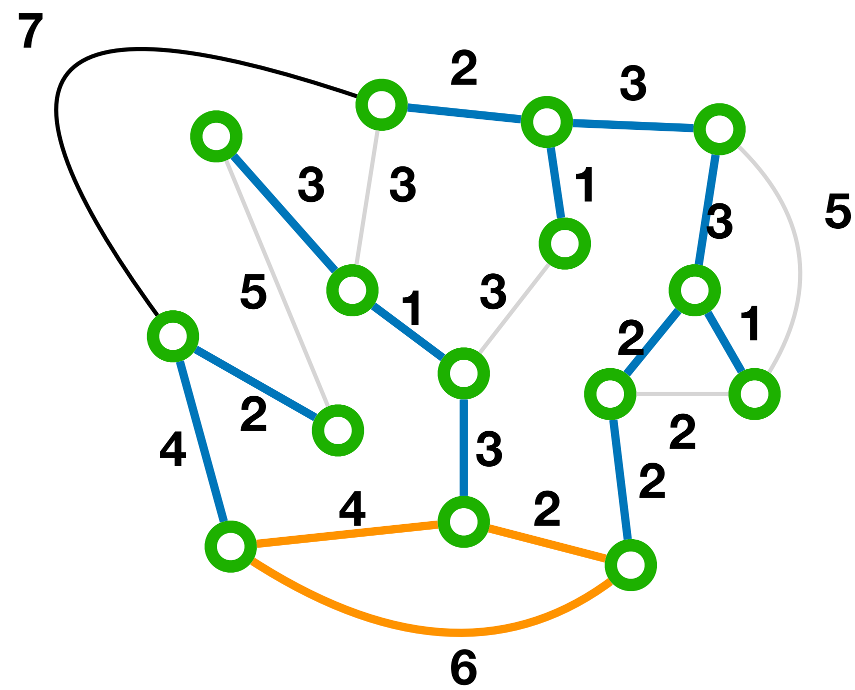
Example



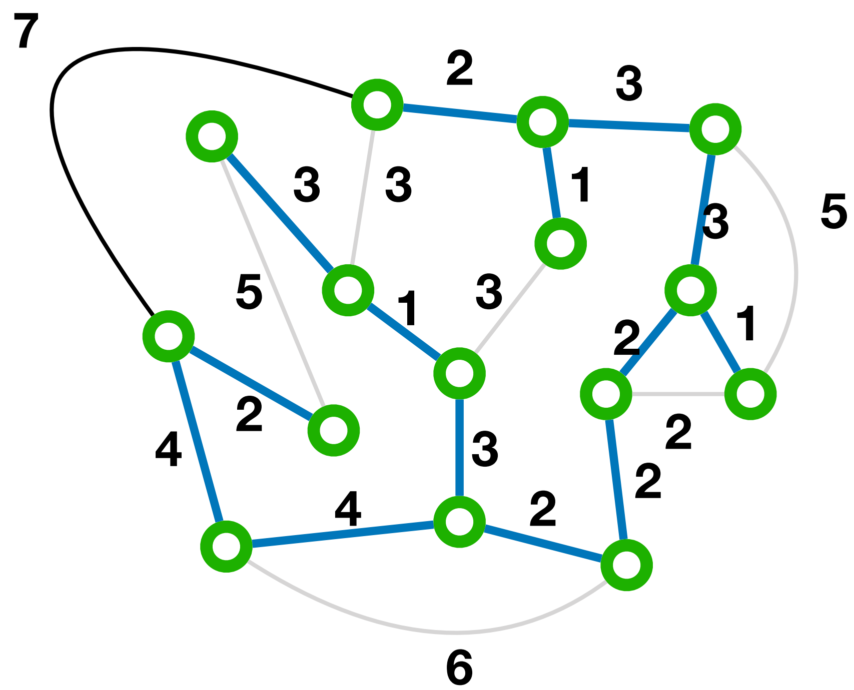
Example



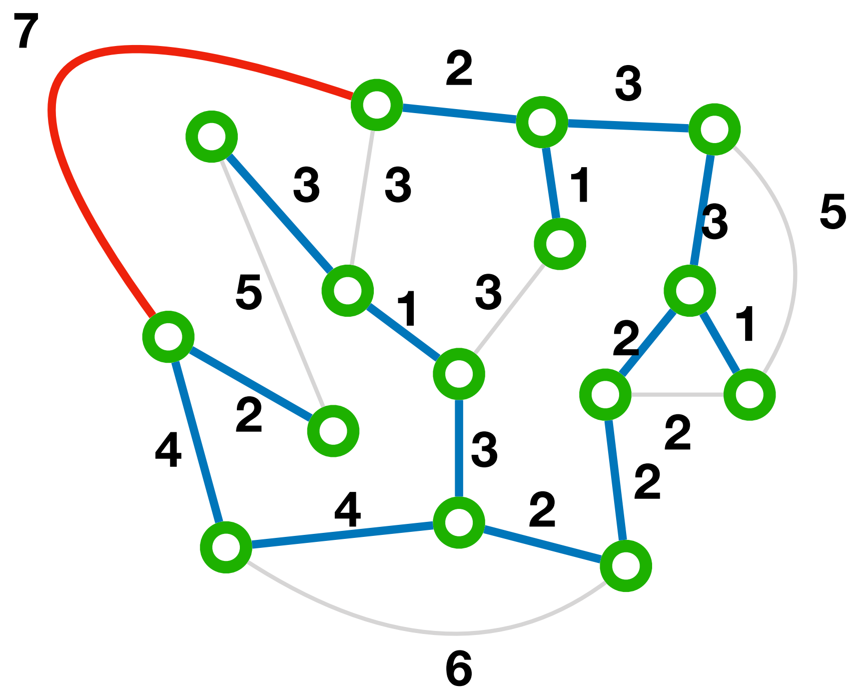
Example



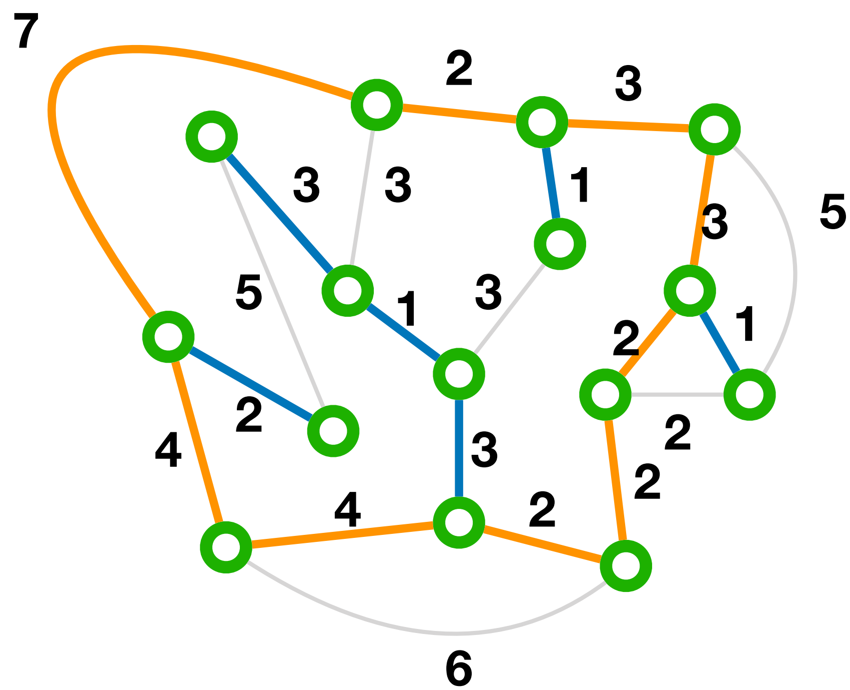
Example



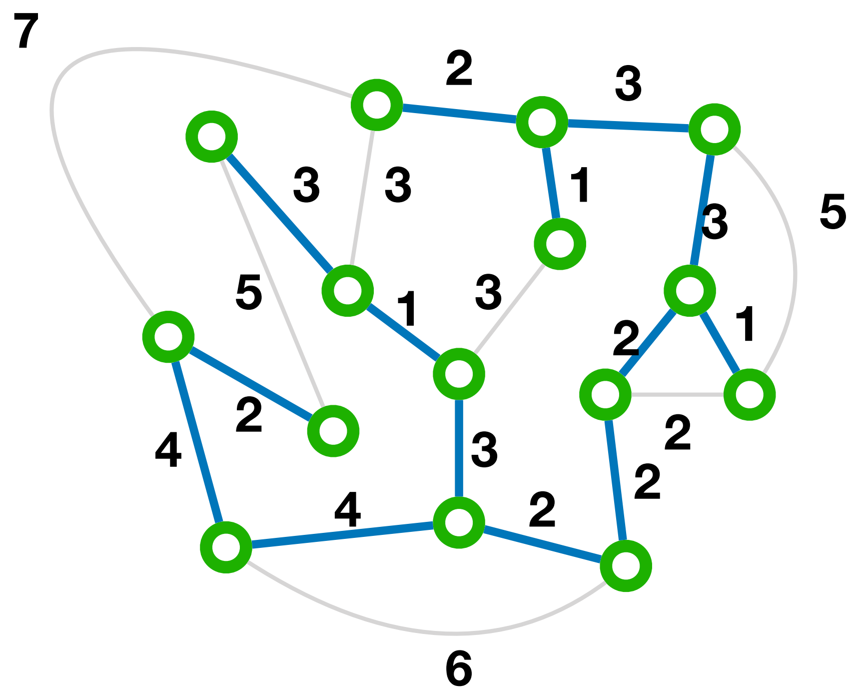
Example



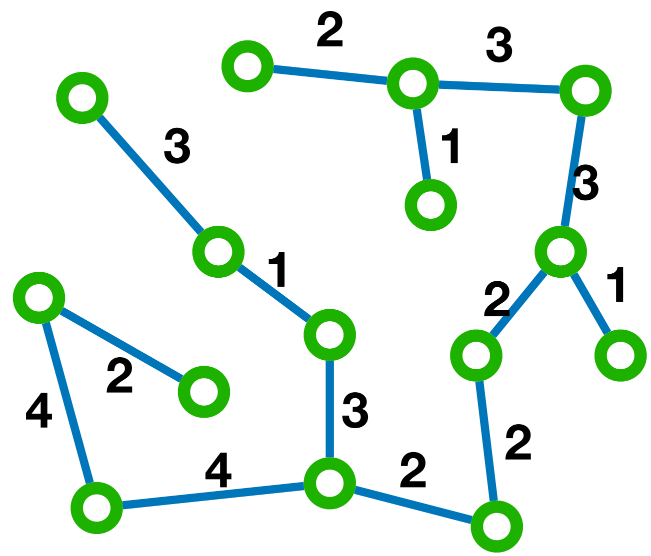
Example



Example



Example



Proof of Correctness

- Sort the edges in increasing (non-decreasing) order of weights
- Let $F = \varnothing$ be an empty forest initially
- For i = 1 to m:
  - if adding e_i to F does not create a cycle, let F = F + e_i
- Output F as an MST of the input

- **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

Runtime Analysis

- Sort the edges in increasing (non-decreasing) order of weights
- Let $F = \emptyset$ be an empty forest initially
- For i = 1 to m:
  - if adding e_i to F does not create a cycle, let F = F + e_i
- Output F as an MST of the input

- First step: O(n+m logm) time
- Checking whether e creates a cycle can be done with a DFS or BFS (check if its endpoints are connected already)
- So each iteration can be done in O(n+m) time
- This way, the algorithm takes $O(m^2 \log m)$ time in total
- This is too slow however

Runtime Analysis

- Sort the edges in increasing (non-decreasing) order of weights

- Let $F = \emptyset$ be an empty forest initially

- For i = 1 to m:

  - **if adding e_i to F does not create a cycle, let F = F + e_i**

- Output F as an MST of the input

- This step can be implemented in only $O(\log m)$ time using proper a data structure

- The data structure:

  - **Disjoint-Union-Find**

- As it will be too much of a distraction, we will not cover this data structure in the course

- They are available in your notes however

Runtime Analysis

- Sort the edges in increasing (non-decreasing) order of weights
- Let $F = \varnothing$ be an empty forest initially
- For i = 1 to m:
  - **if adding e_i to F does not create a cycle, let F = F + e_i**
- Output F as an MST of the input

- Summary:
- Kruskal can be implemented in only O(m log m) time
- This is our first efficient (actual) algorithm for MST

Prim’s Algorithm

Prim's Algorithm

- Another canonical and efficient algorithm for MST
- Another way of implementing the strategy of the meta-algorithm
- Very closely related to Dijkstra's algorithm for the shortest path problem

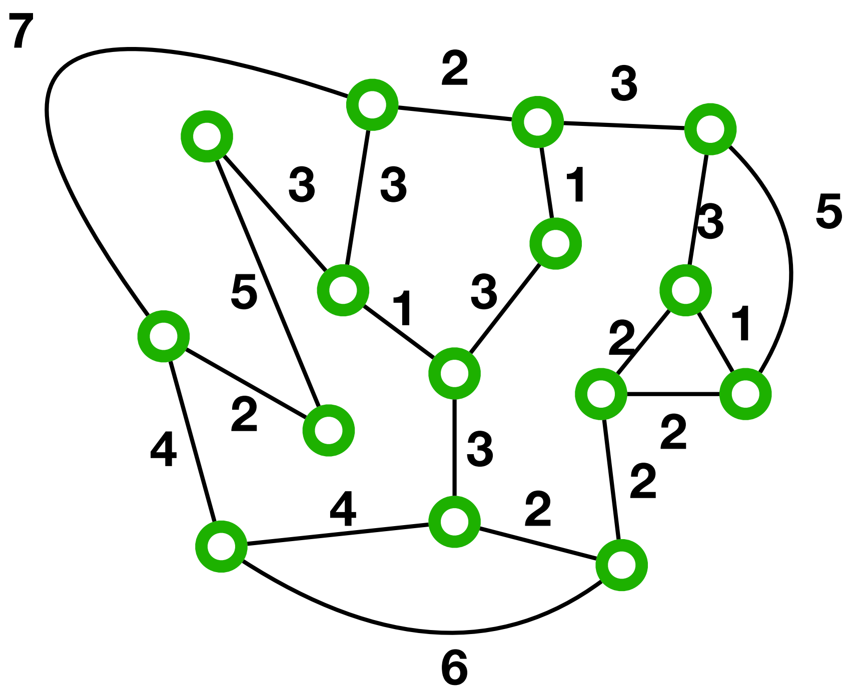
A Generic Meta-Algorithm

- Let $F = \emptyset$ 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

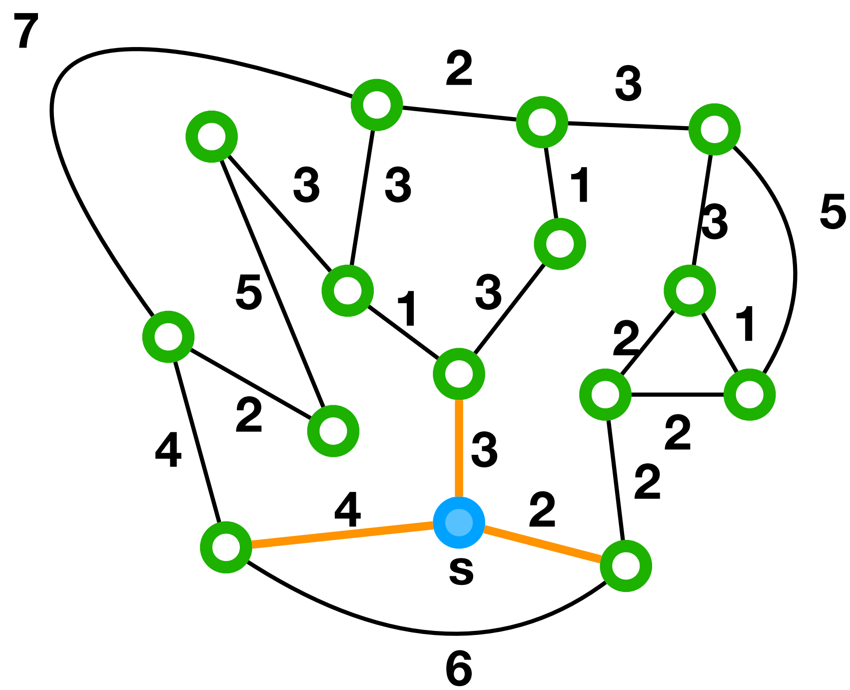
Prim's Algorithm

- 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

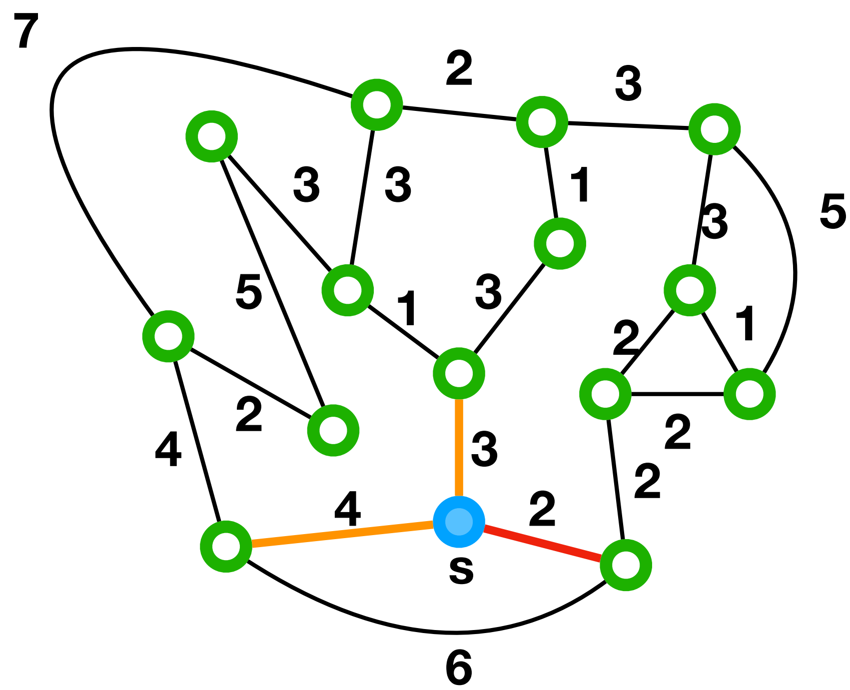
Example



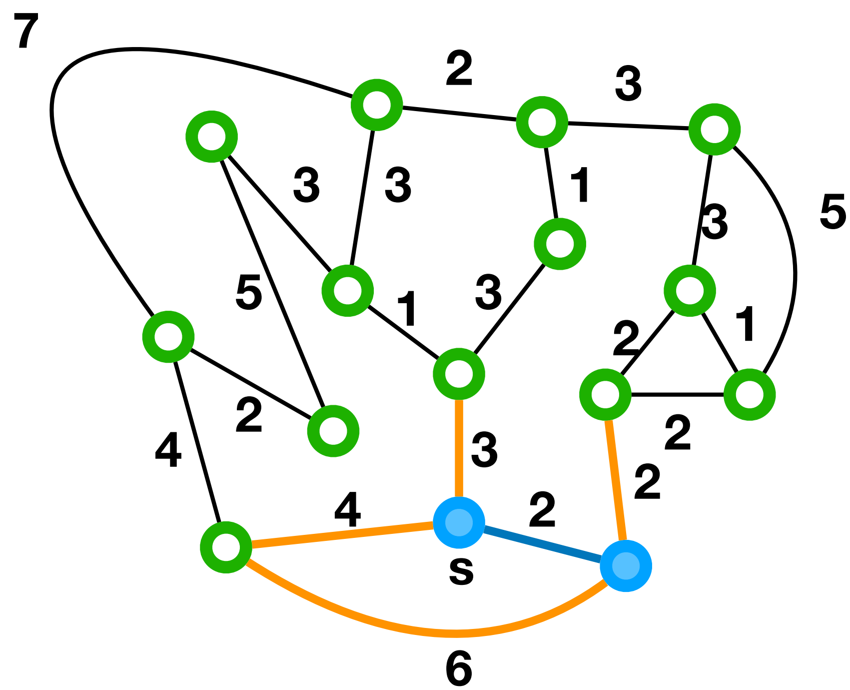
Example



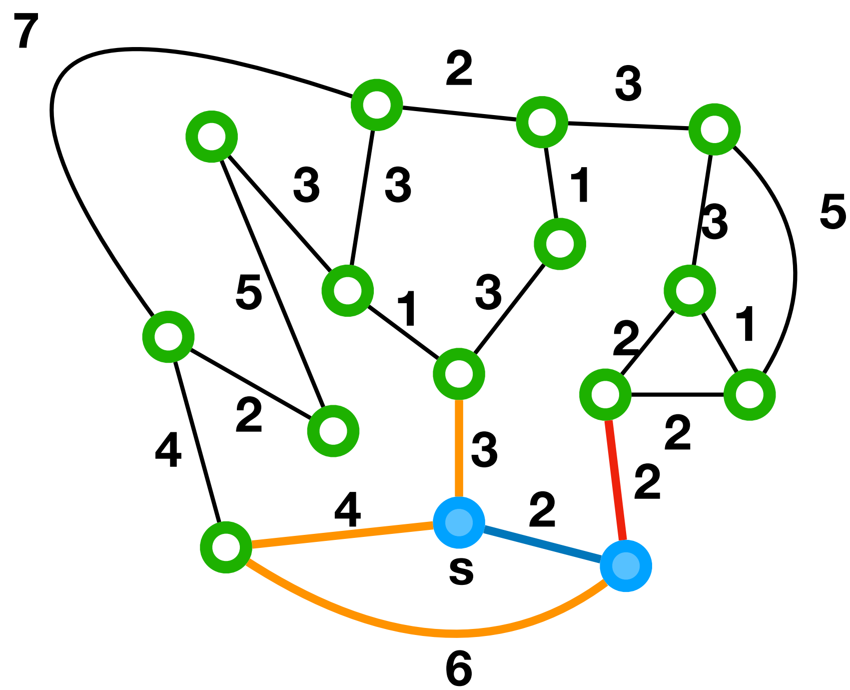
Example



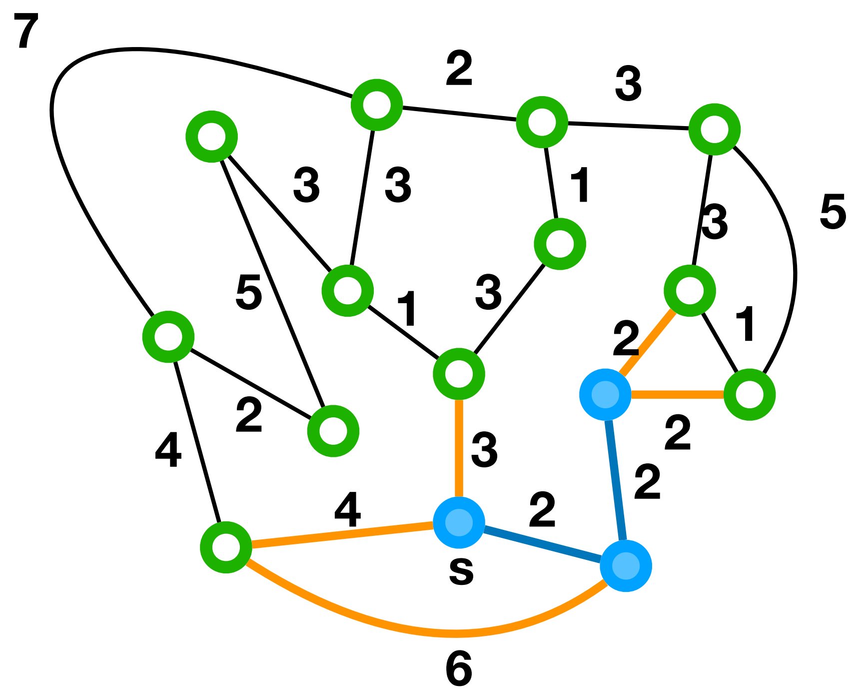
Example



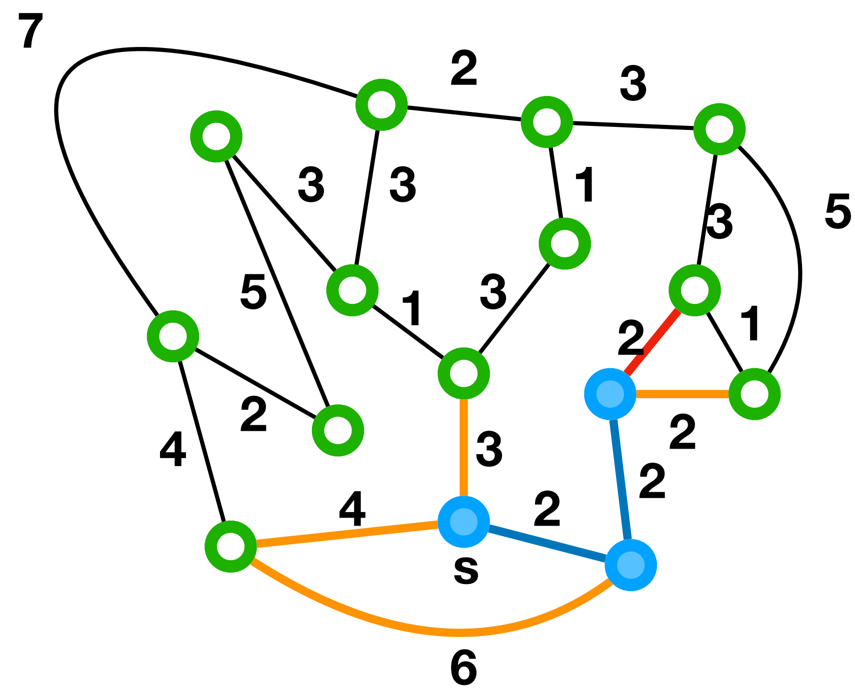
Example



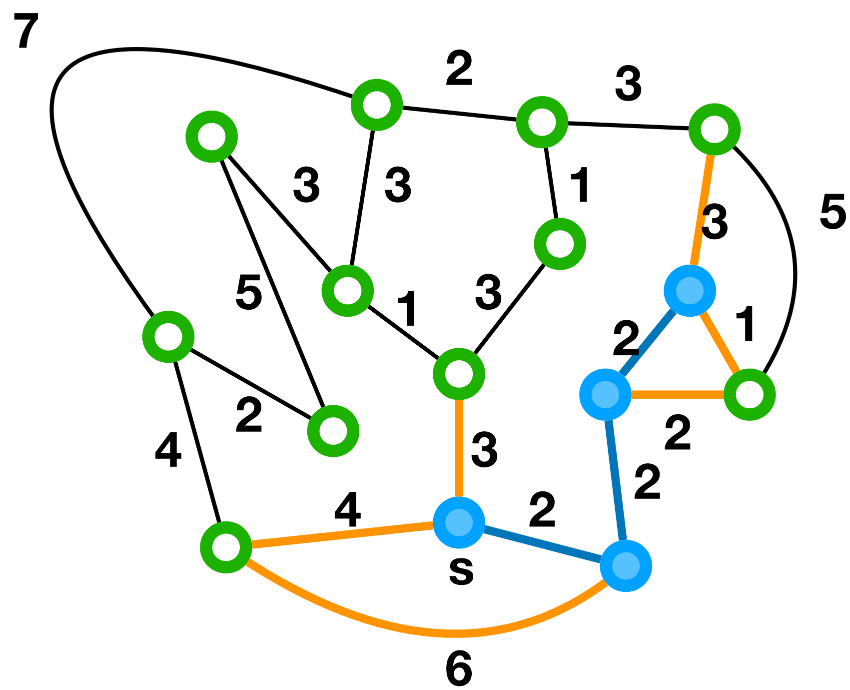
Example



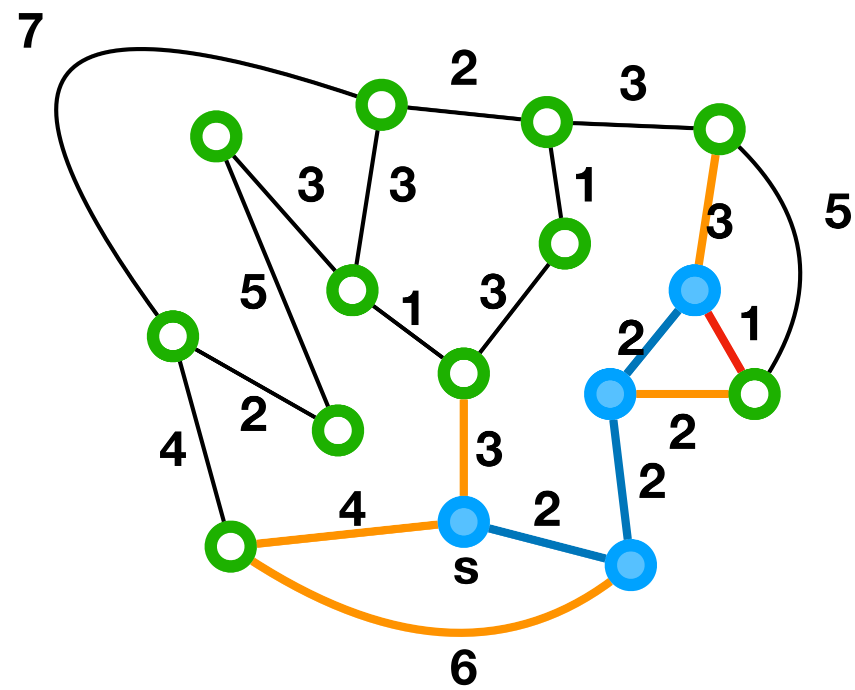
Example



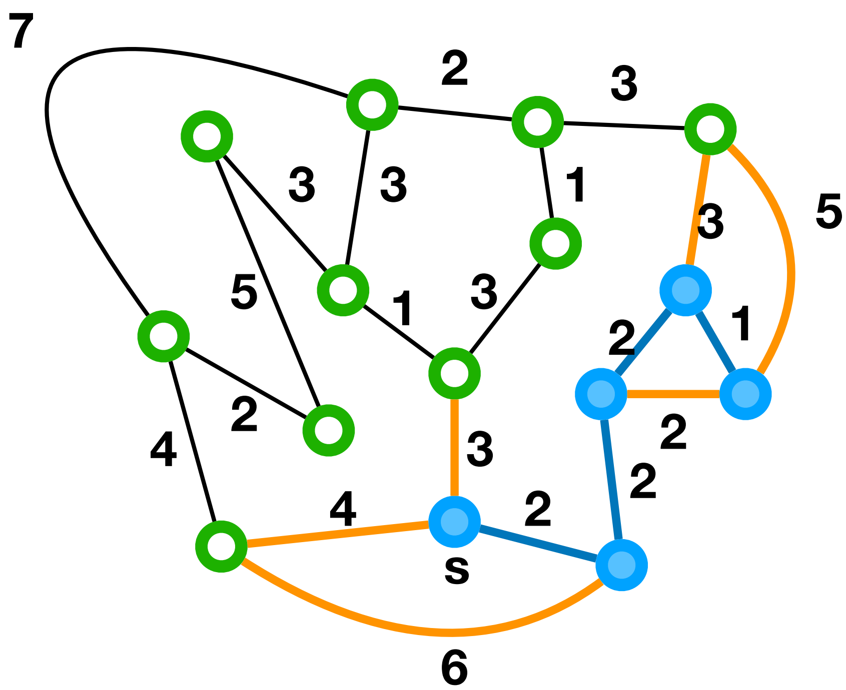
Example



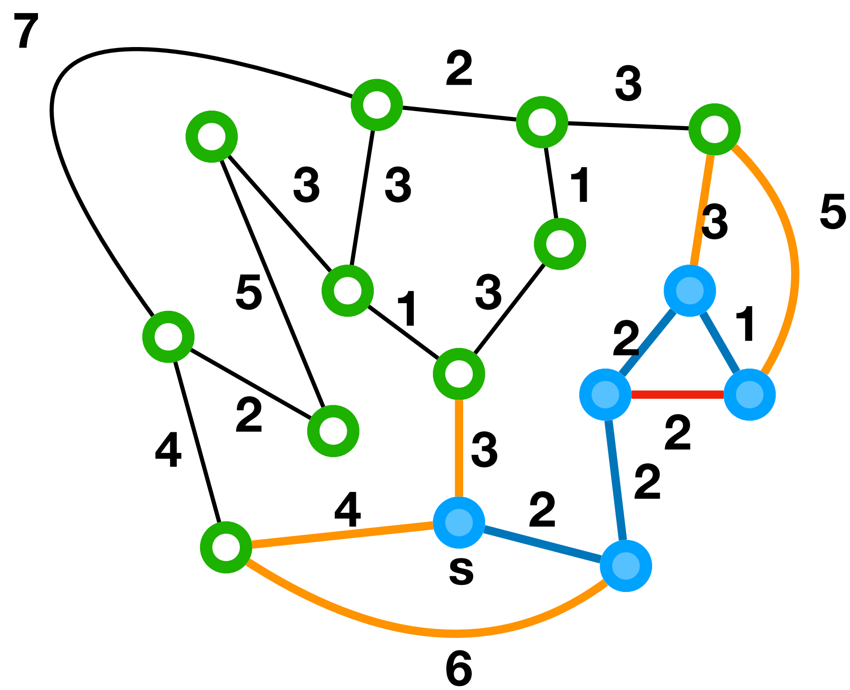
Example



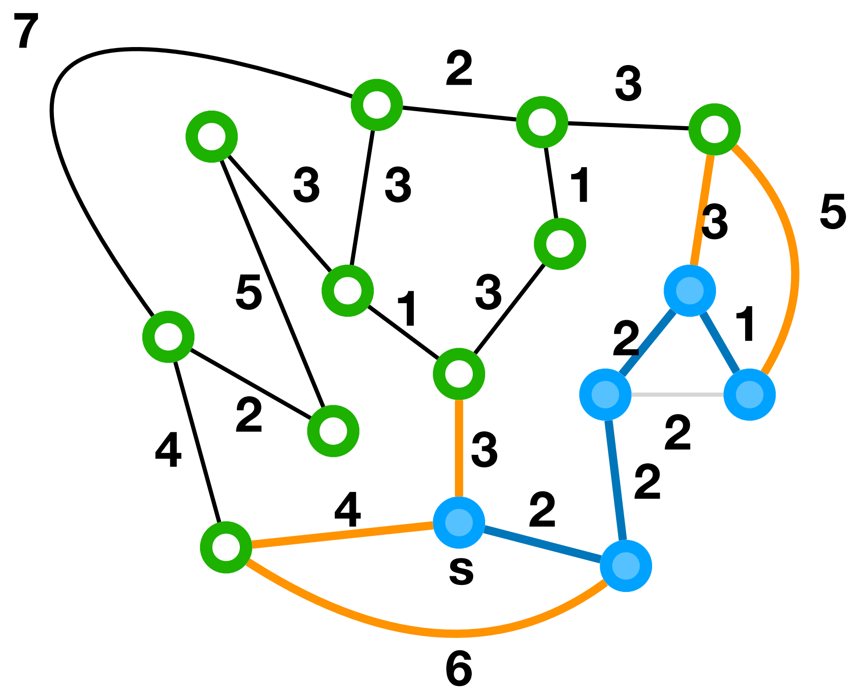
Example



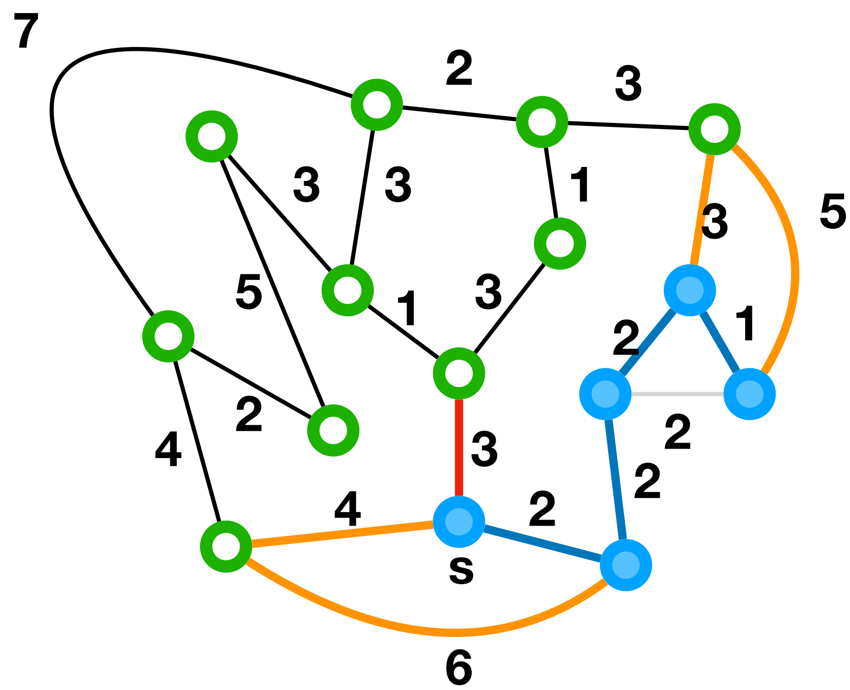
Example



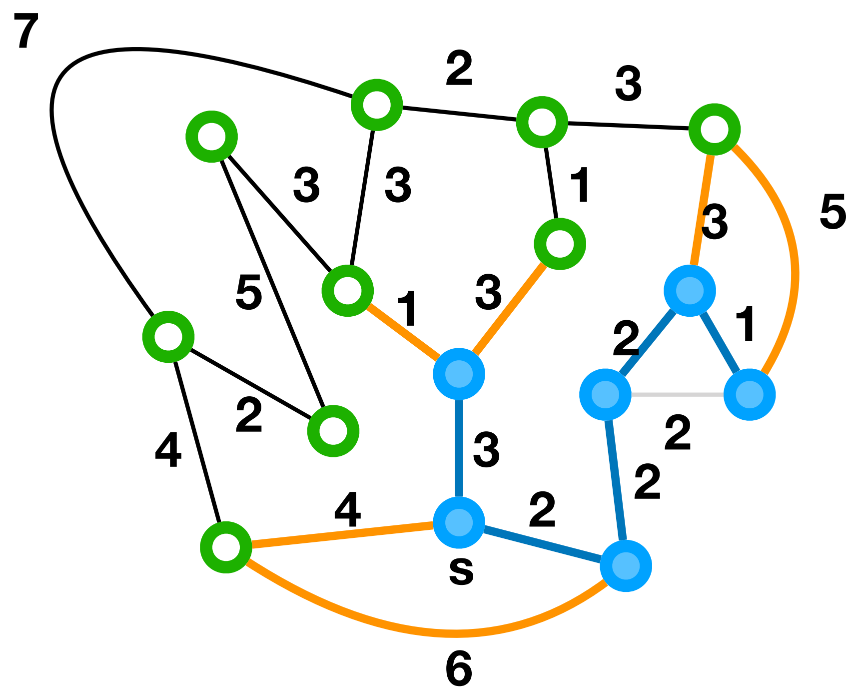
Example



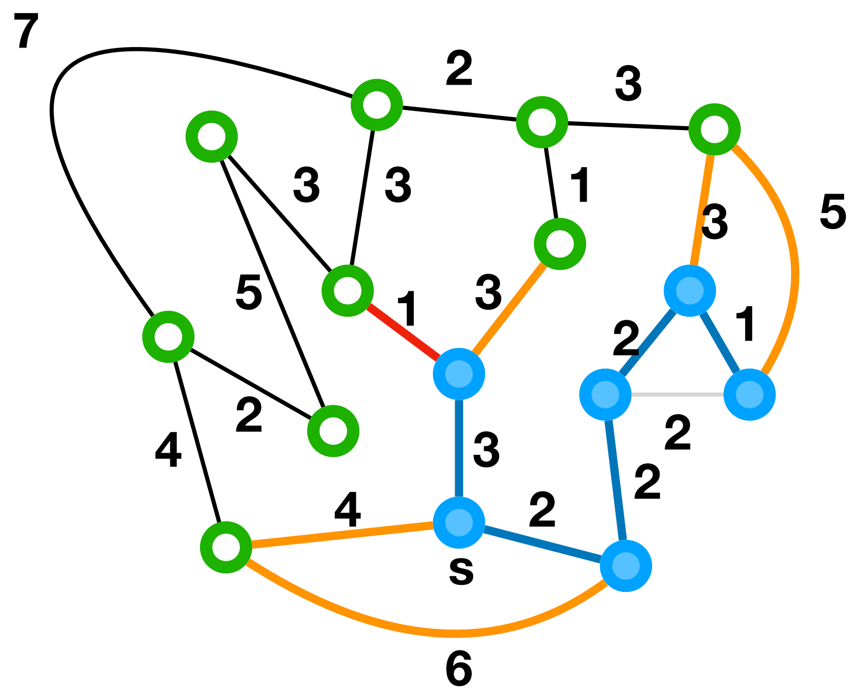
Example



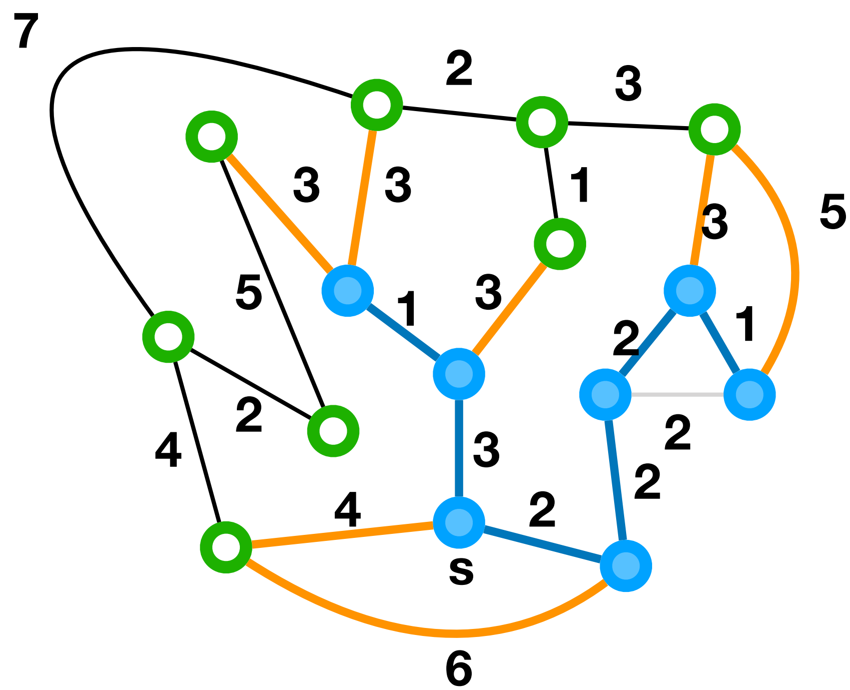
Example



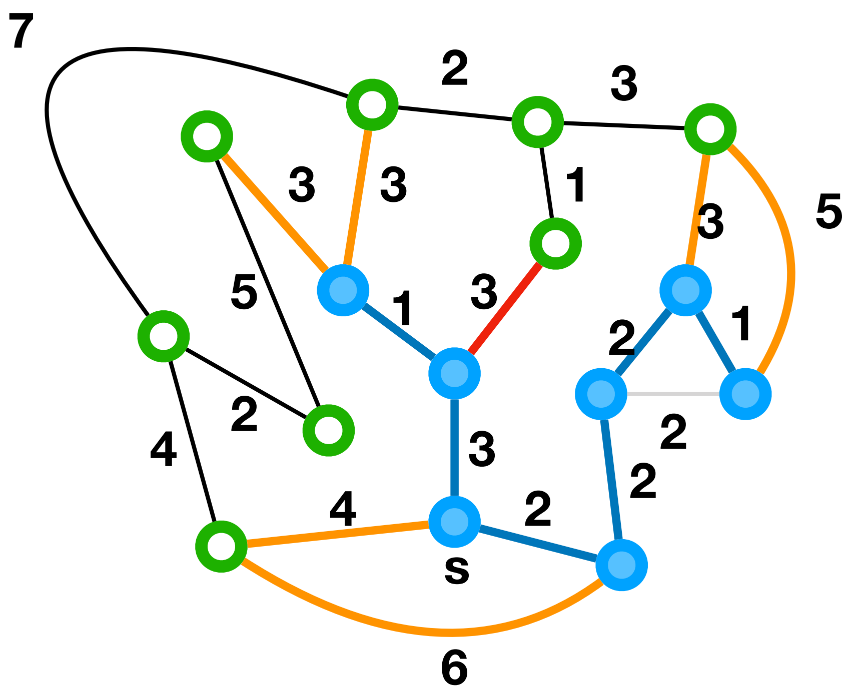
Example



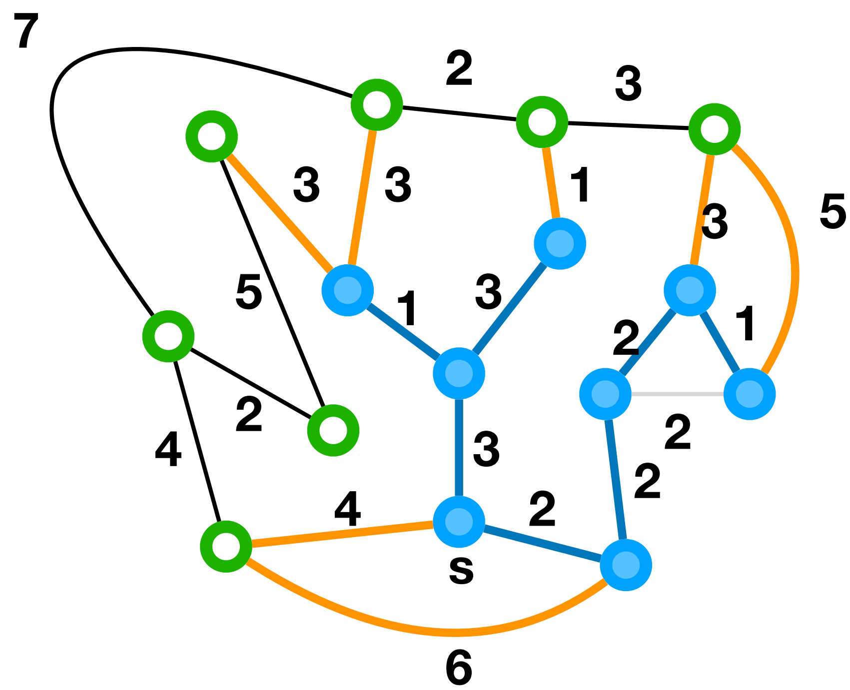
Example



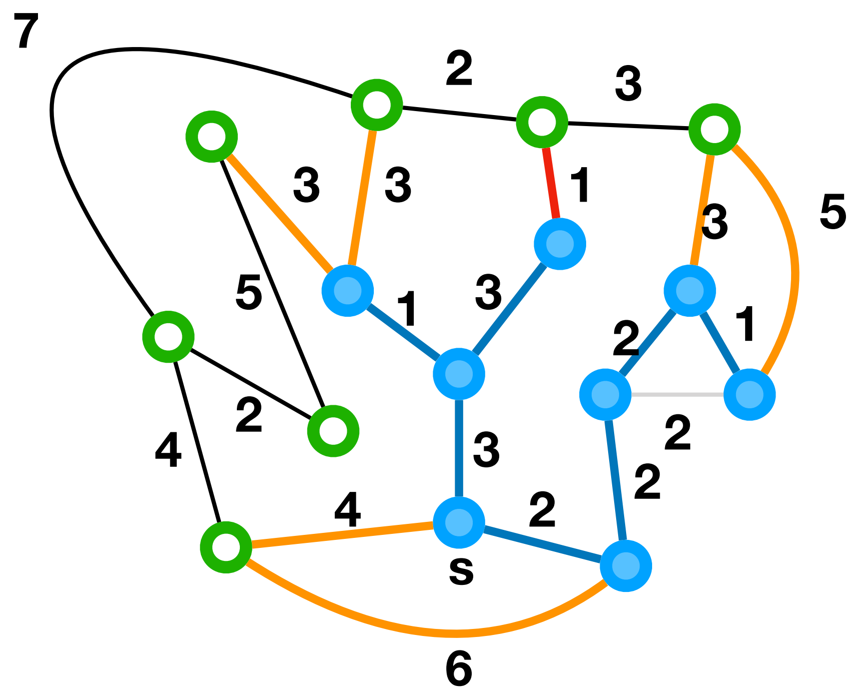
Example



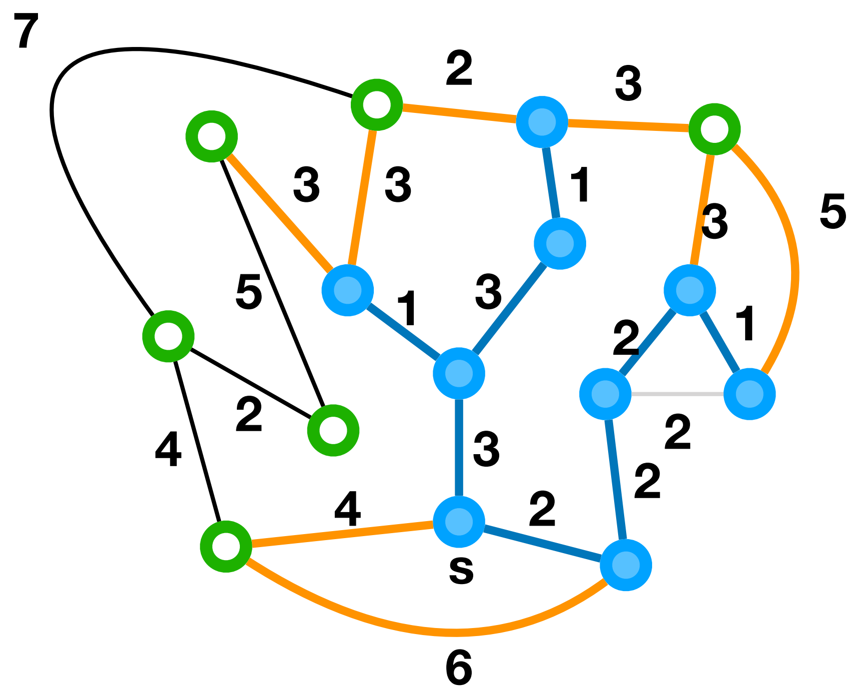
Example



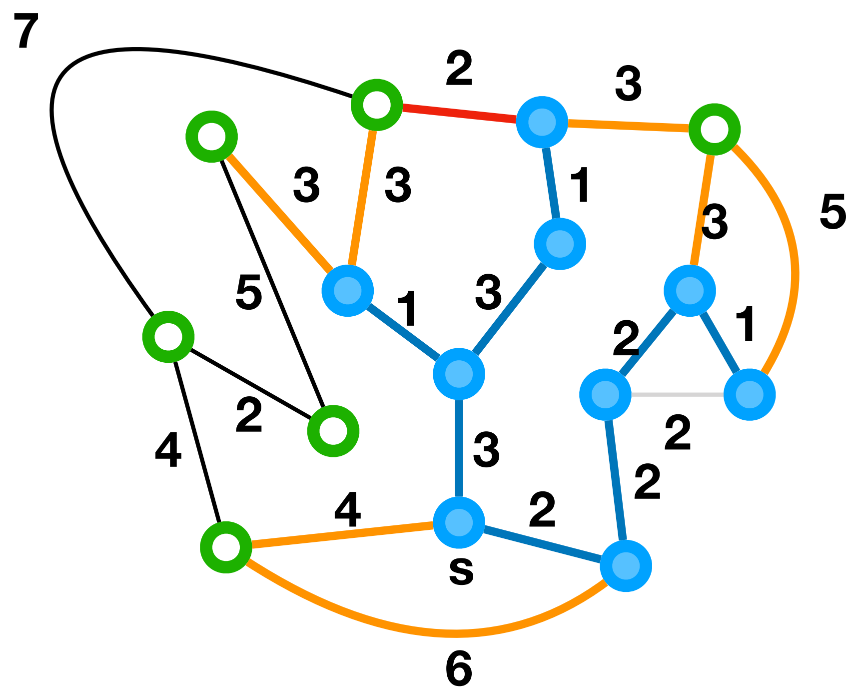
Example



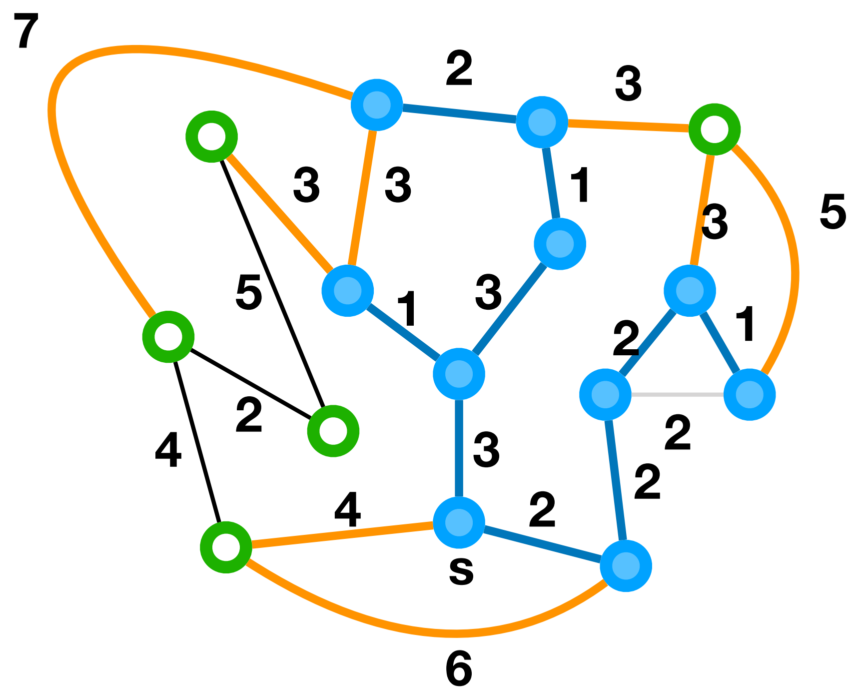
Example



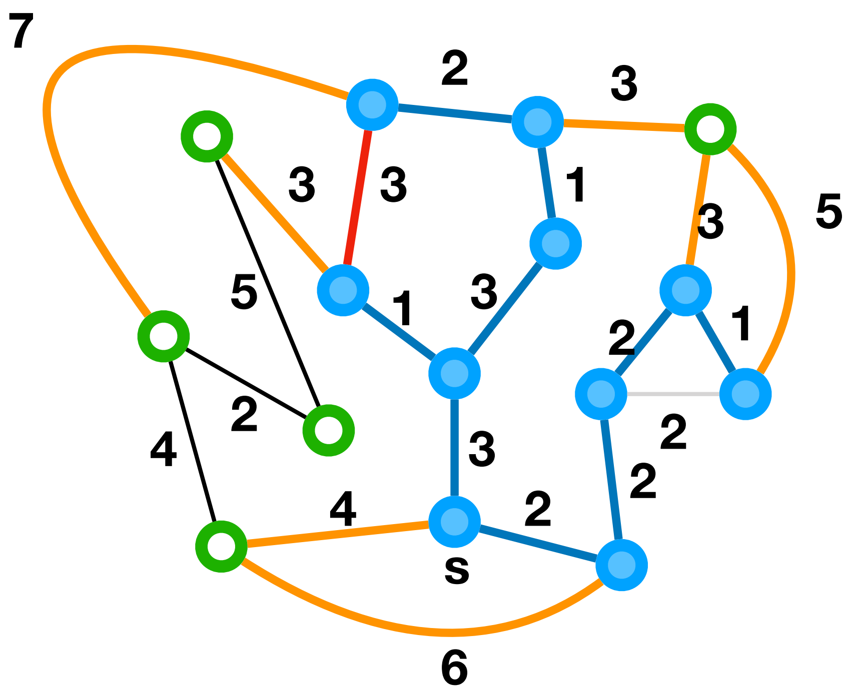
Example



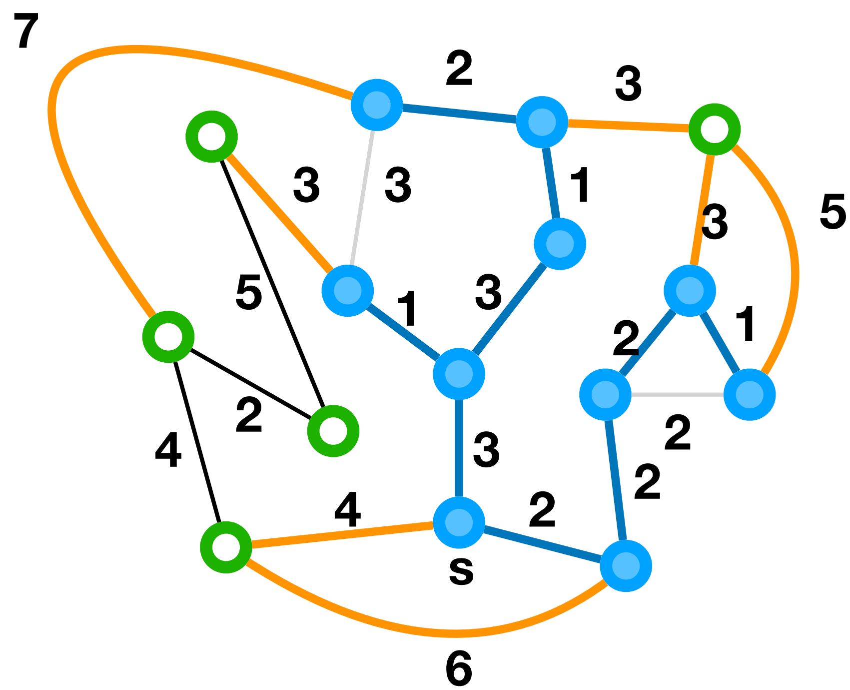
Example



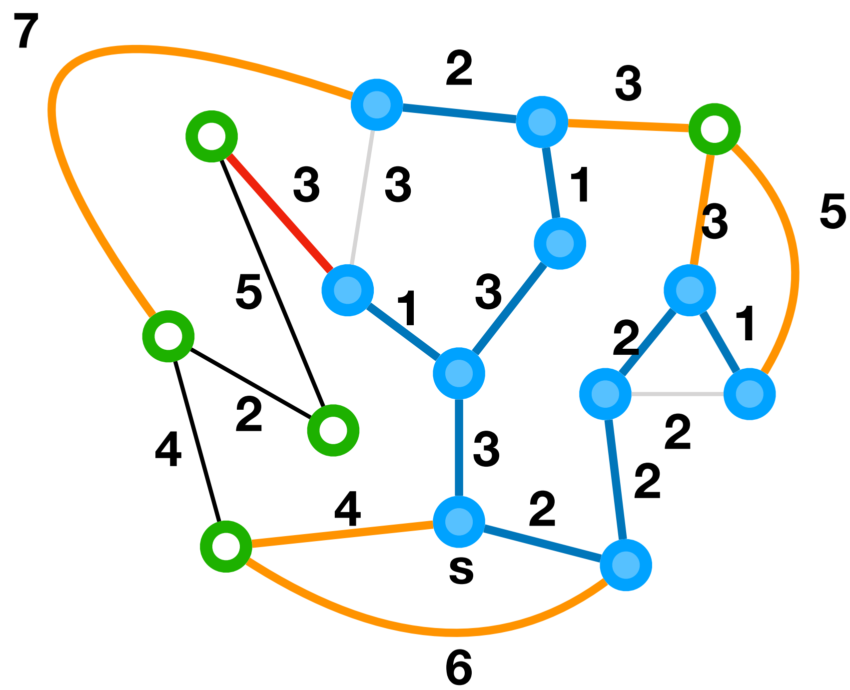
Example



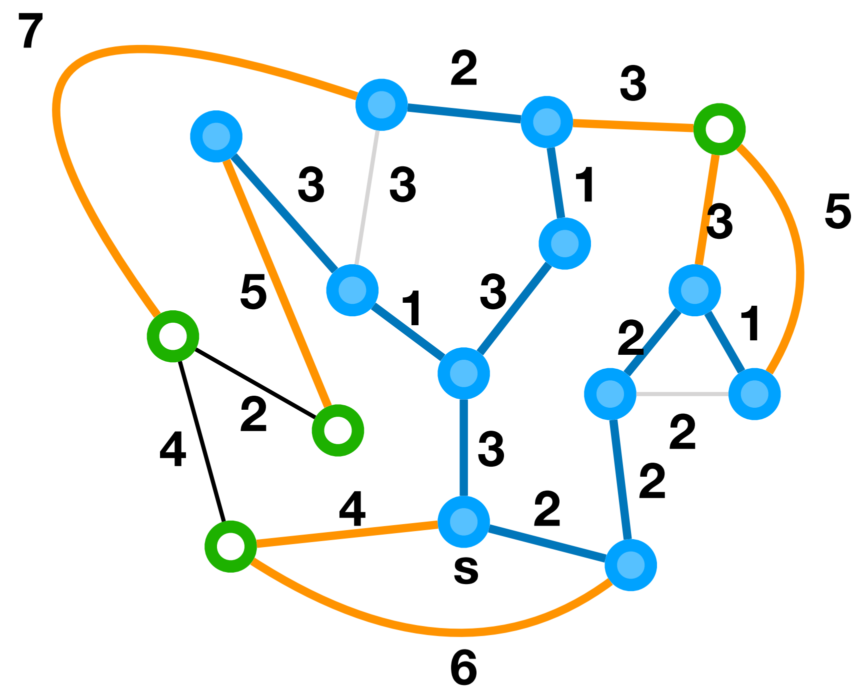
Example



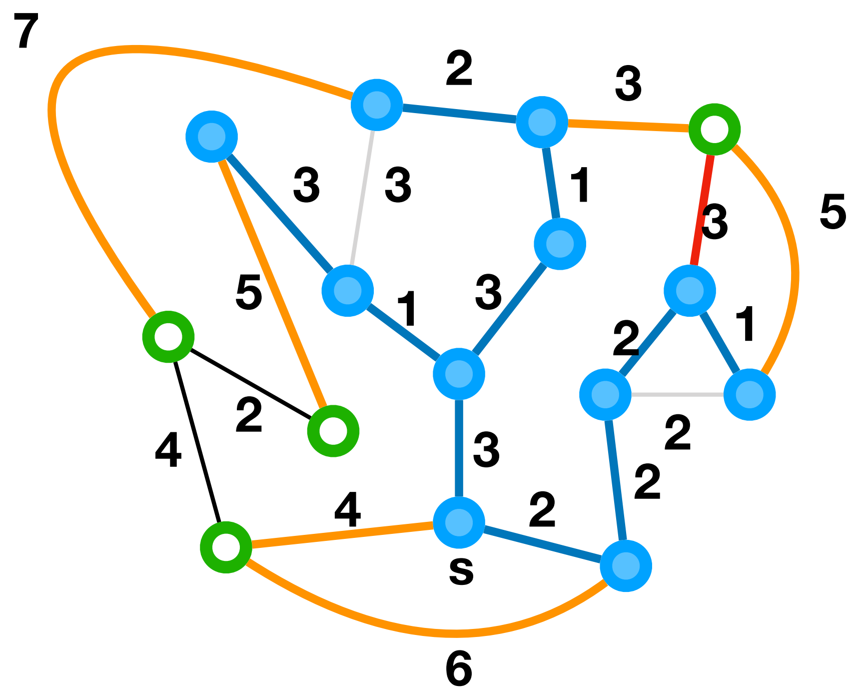
Example



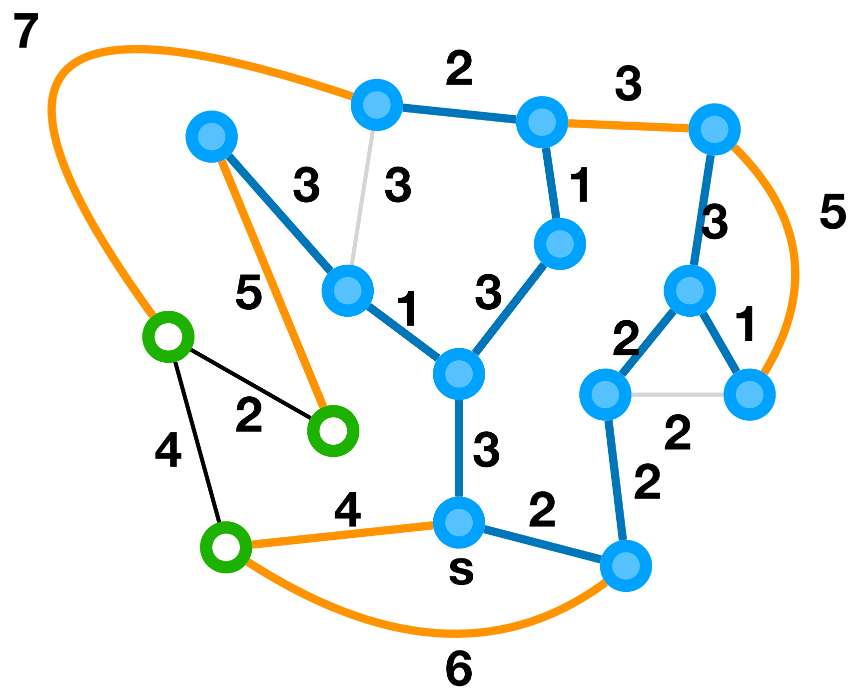
Example



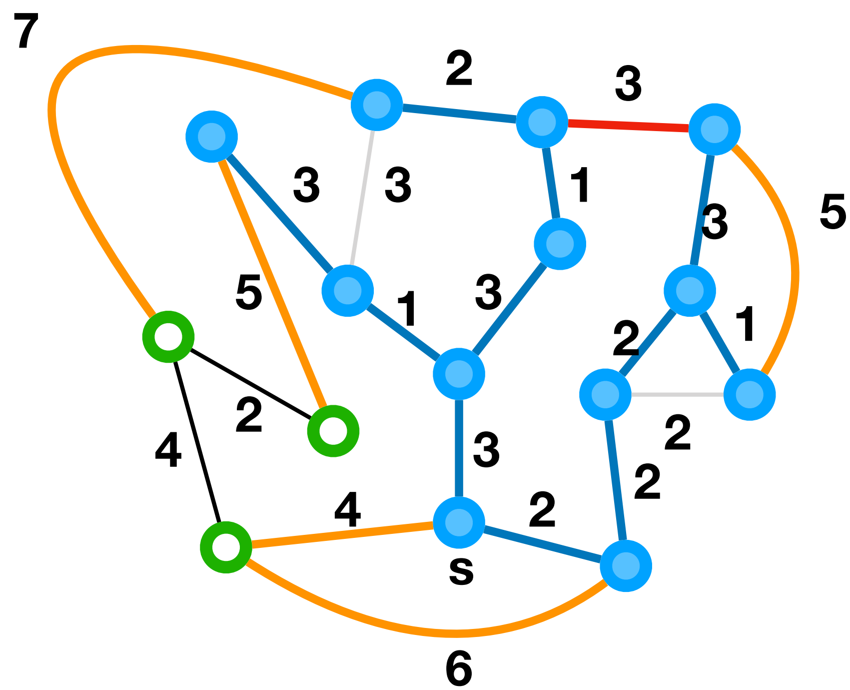
Example



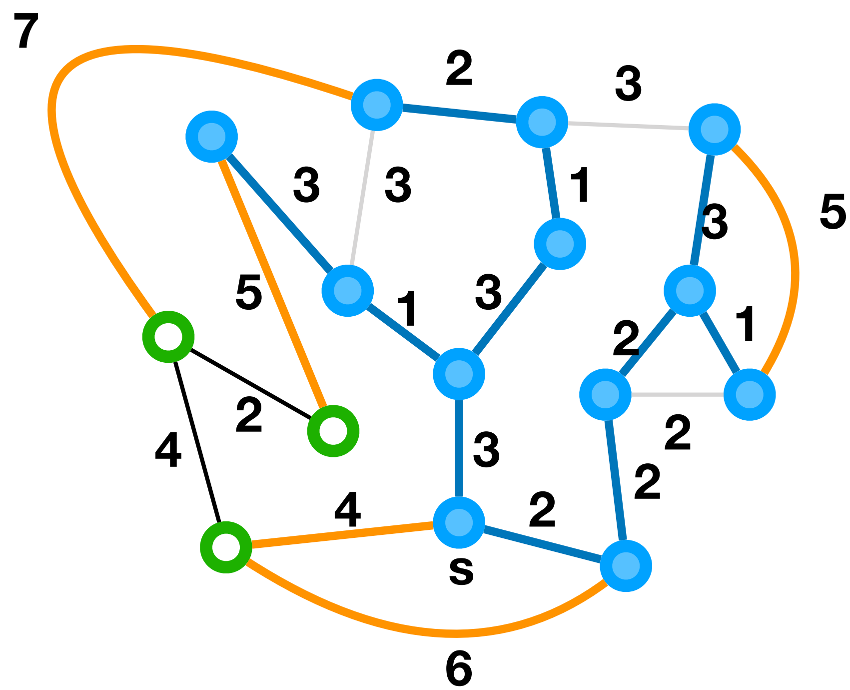
Example



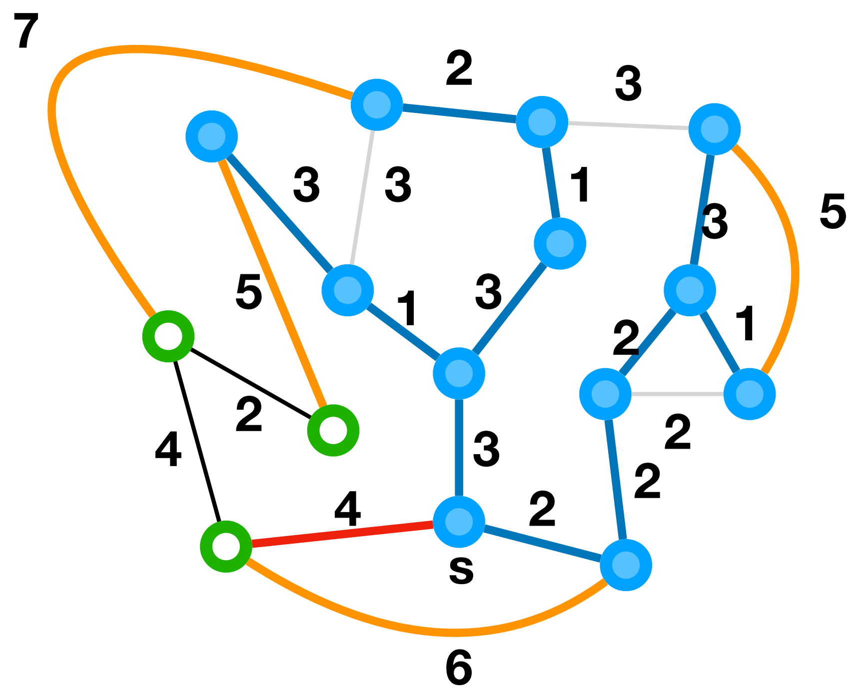
Example



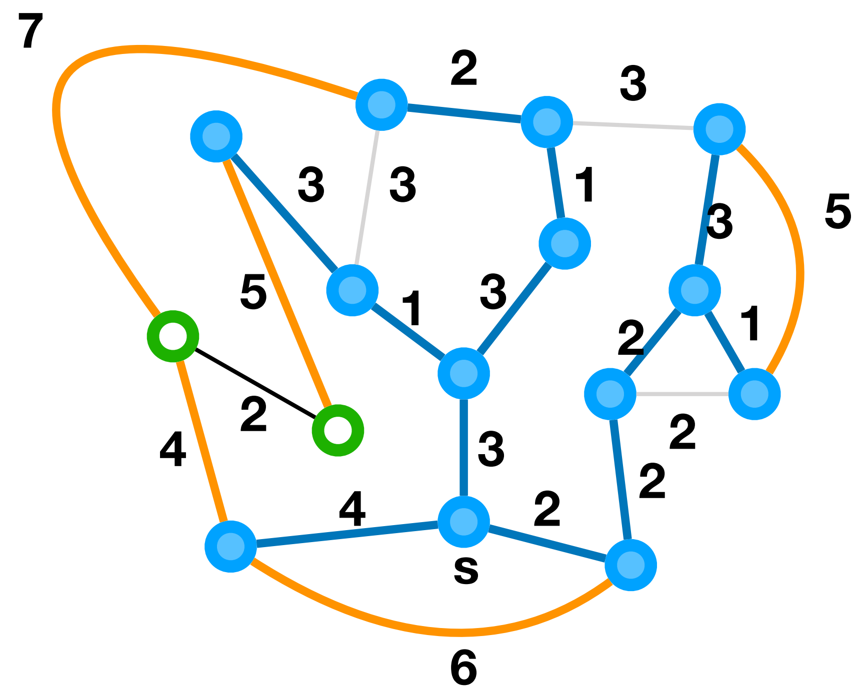
Example



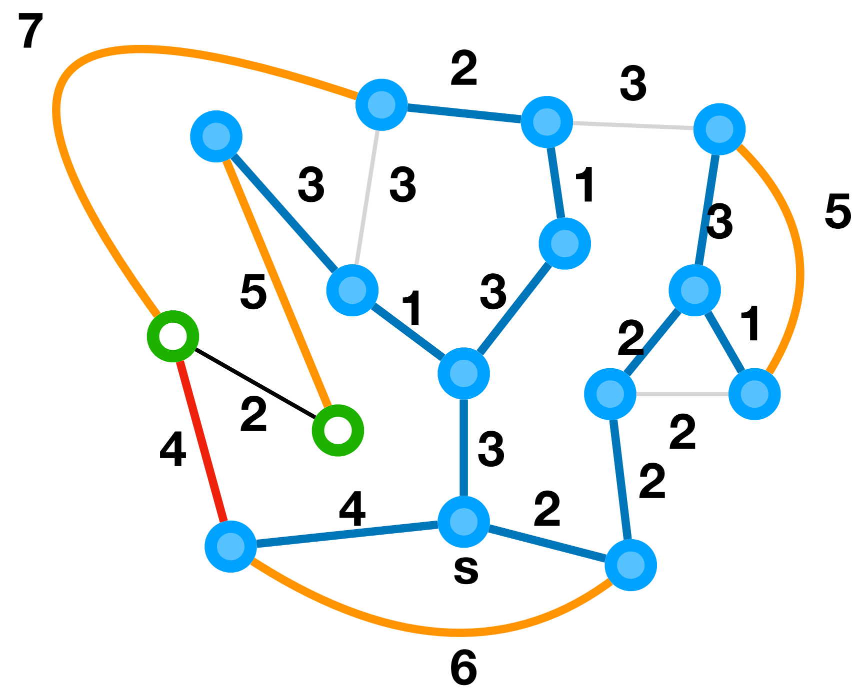
Example



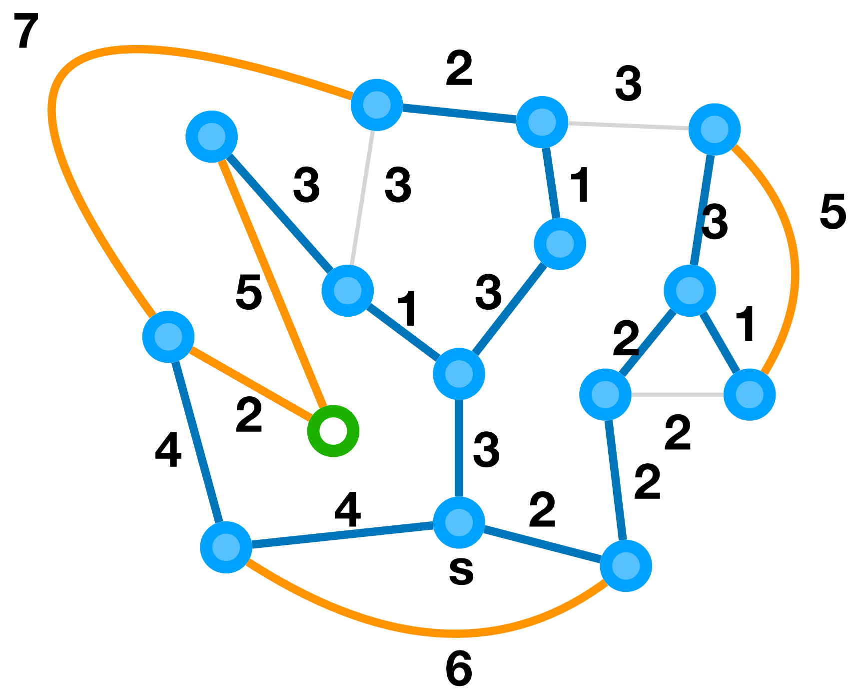
Example



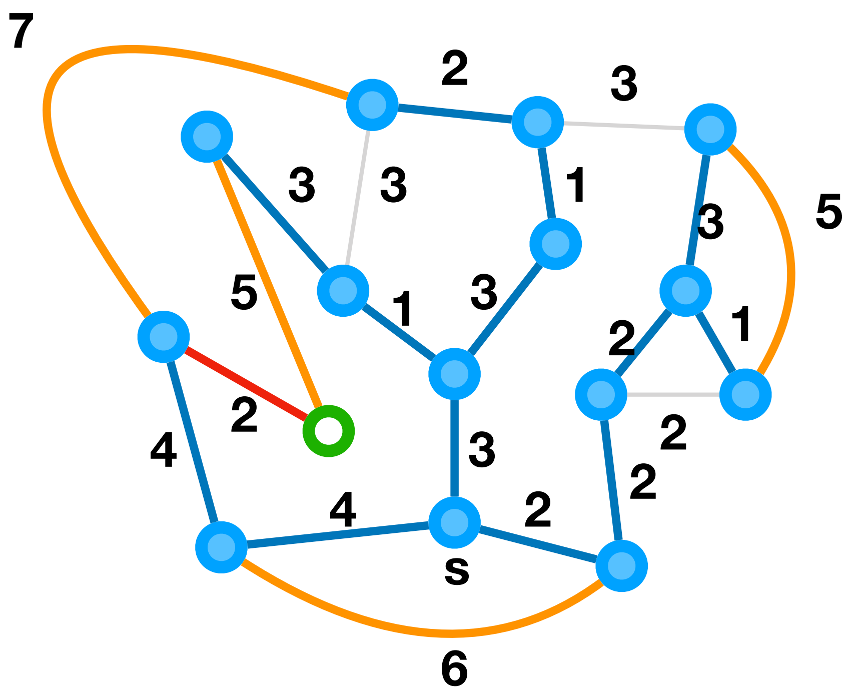
Example



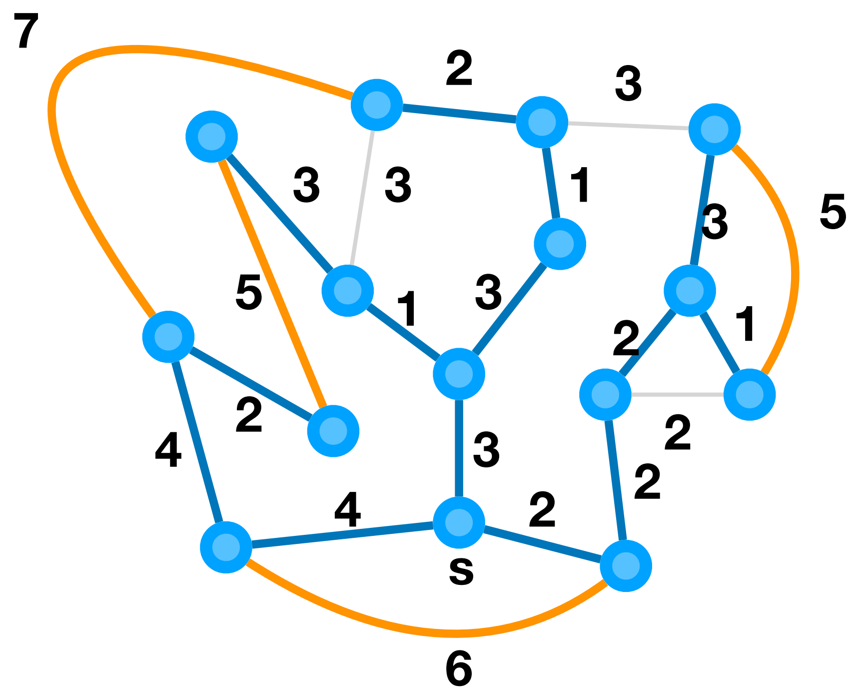
Example



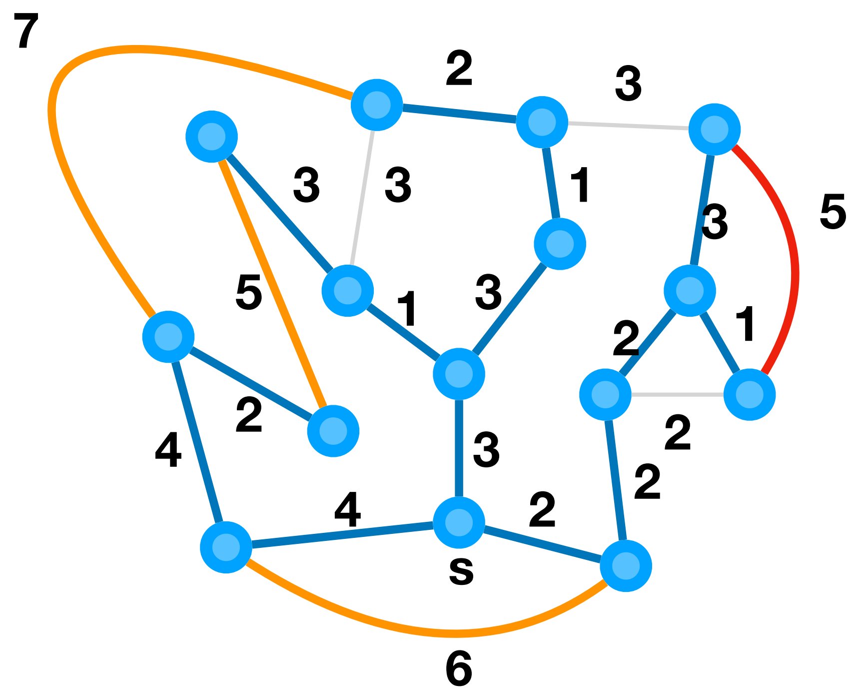
Example



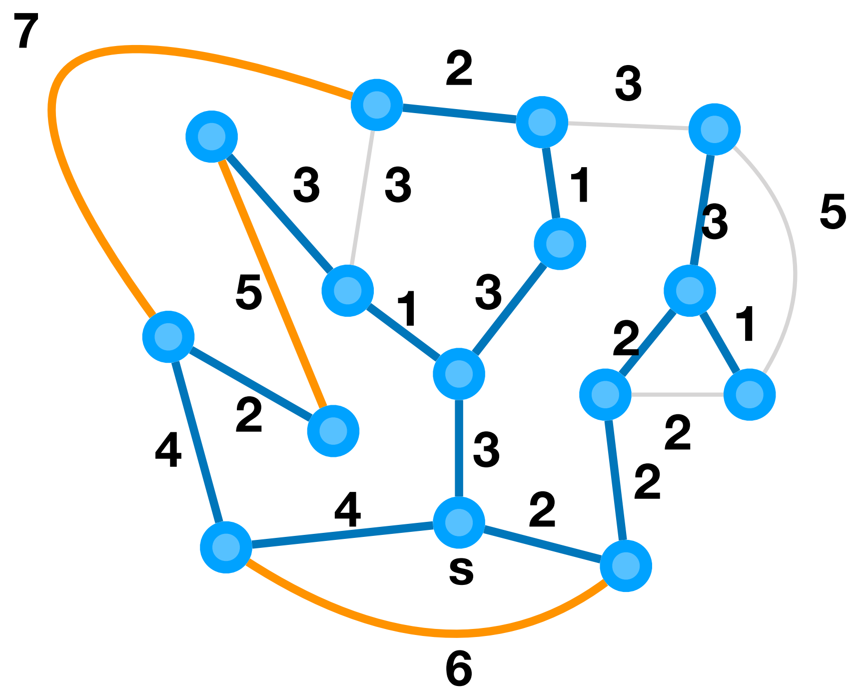
Example



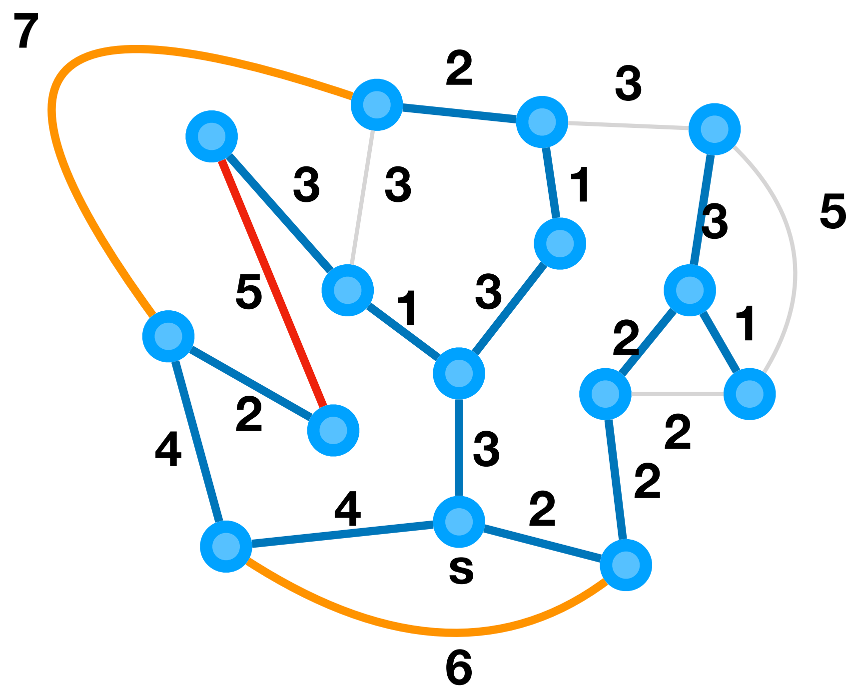
Example



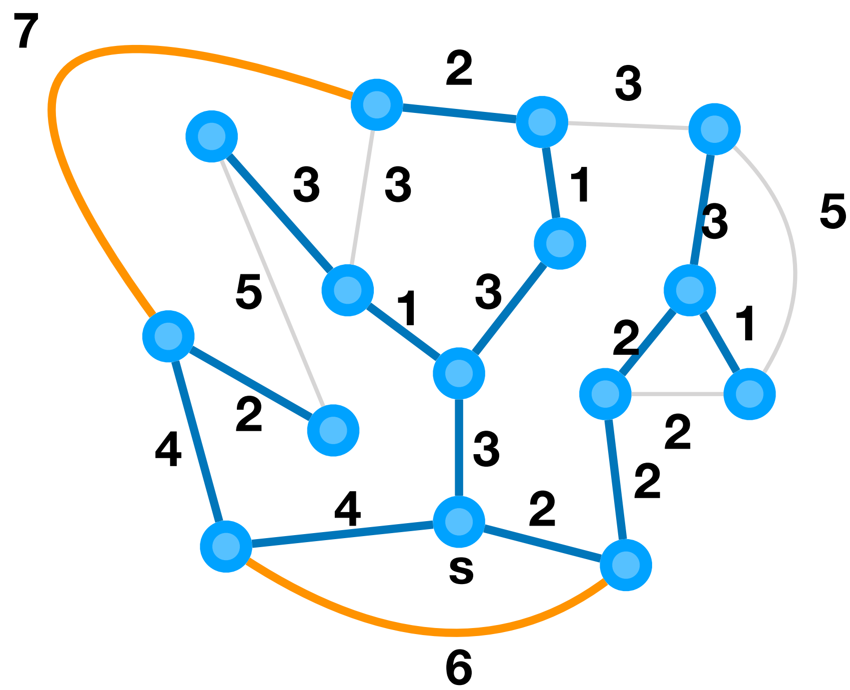
Example



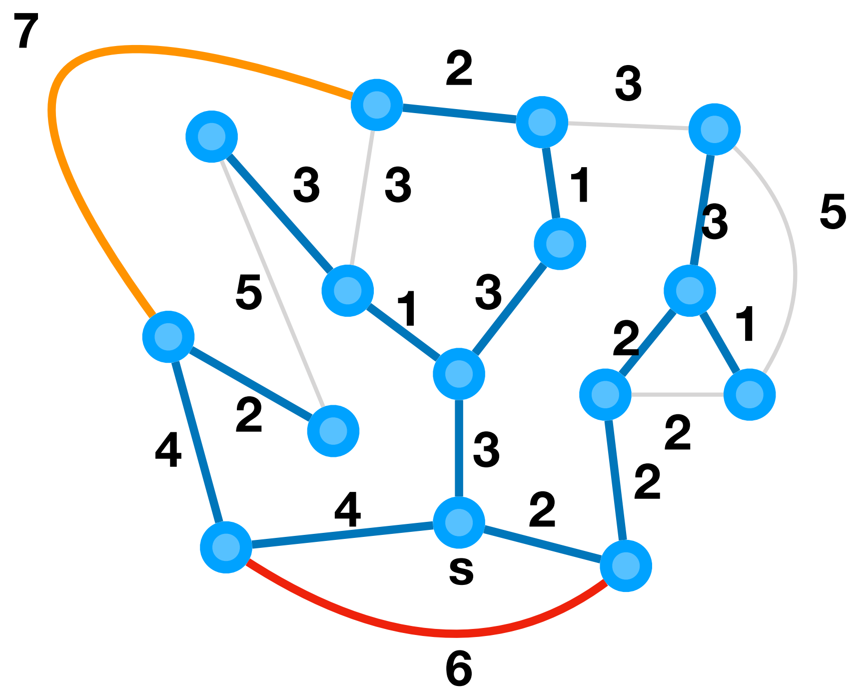
Example



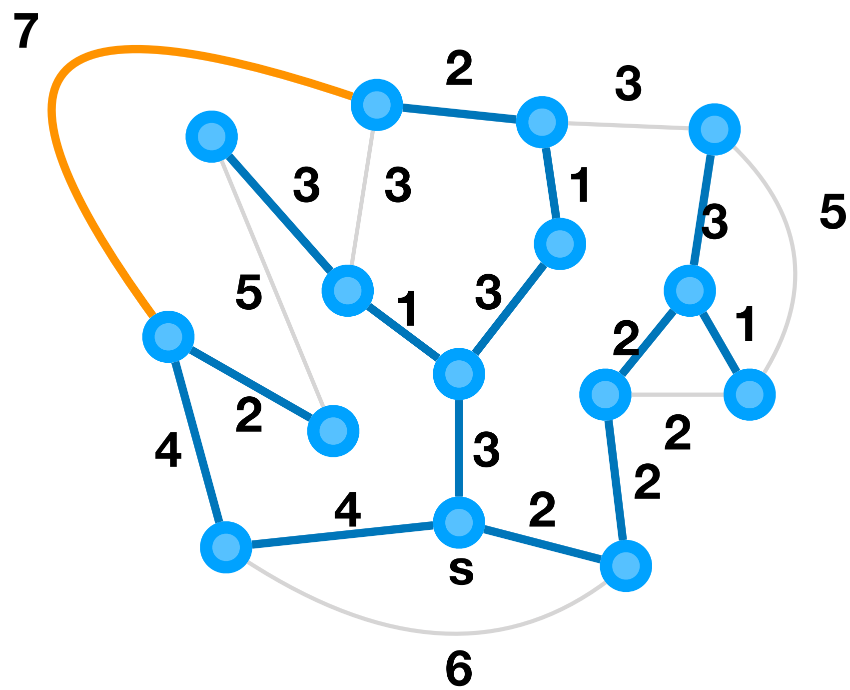
Example



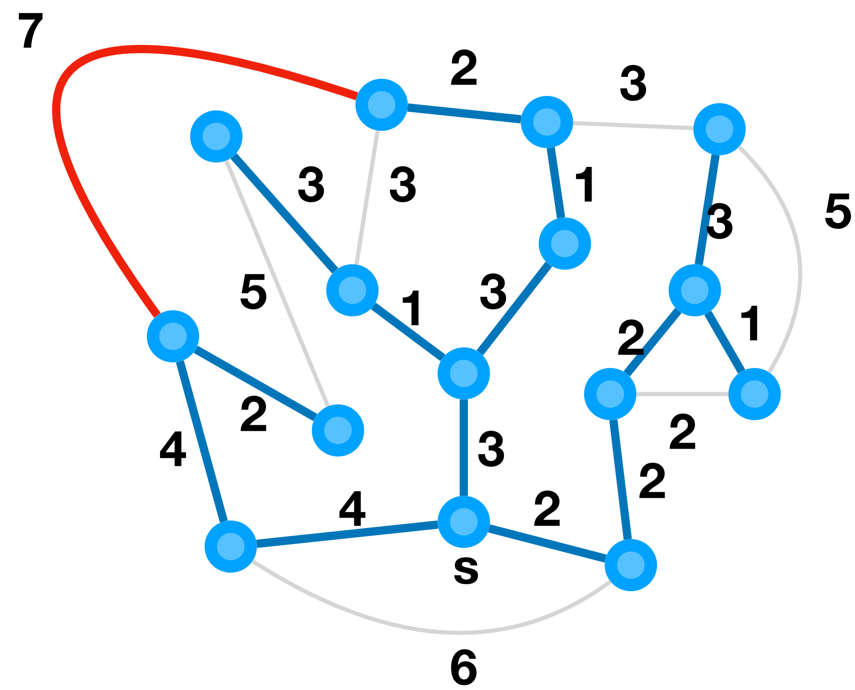
Example



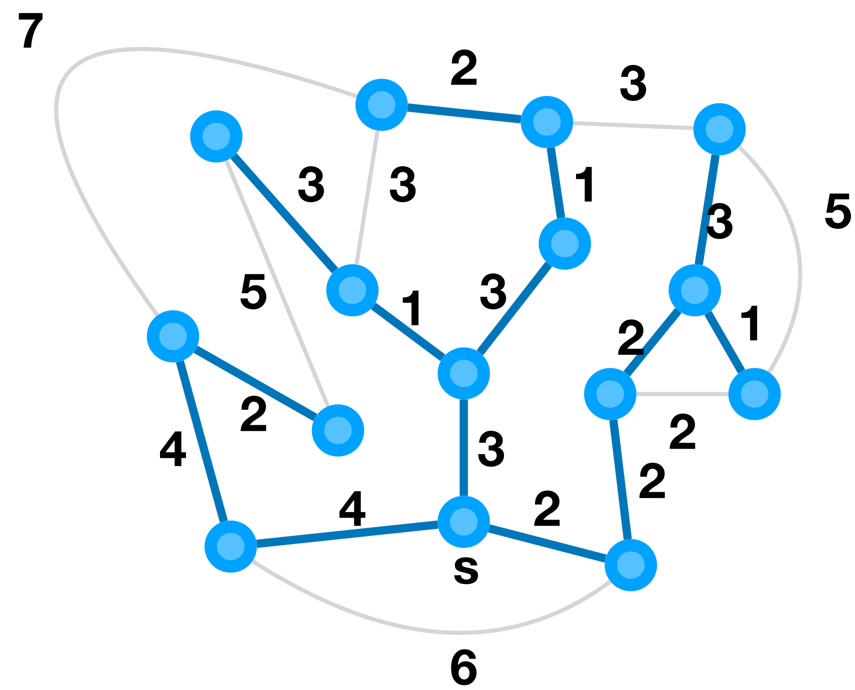
Example



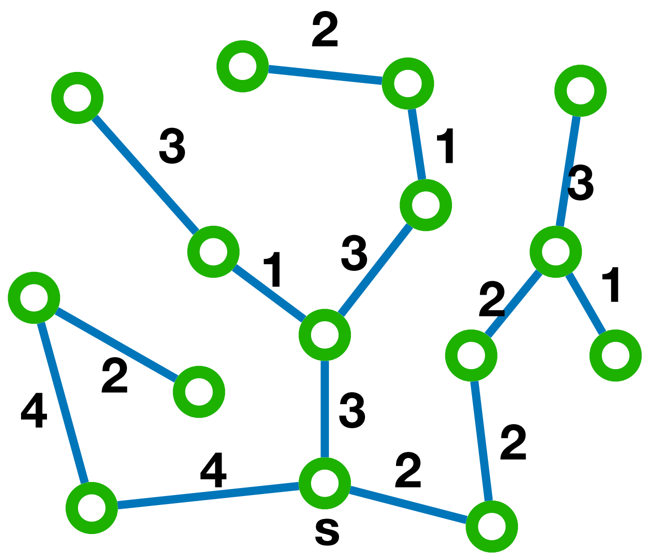
Example



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

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.

- Consider edge (u,v) inserted to F
- Let S be the connected component of s in F before inserting (u,v)
- By the same argument as DFS/BFS, S is the set of all marked vertices
- So here u belongs to S and v does not
- We claim (u,v) is the minimum weight edge of the cut (S, V-S)
- So by our theorem, (u,v) is safe

Runtime Analysis

- 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.

- If we store H in a linked list and do a linear search in every step:
- O(m) time to find minimum weight edge in H
- O(m) iterations in the while-loop
- So $O(m^2)$ time in total

Runtime Analysis

- 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.

- If we store H in a linked list and do a linear search in every step:
- O(m) time to find minimum weight edge in H
- O(m) iterations in the while-loop
- So $O(m^2)$ time in total
- Again, too slow

Runtime Analysis

- 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.

- We should store H as a min-heap
- Insertion takes O(log m) time
- Deletion takes O(log m) time
- Each vertex is marked once and takes $O(\deg(v) \cdot \log m)$ time to insert its edges to H
- We do at most m deletions from the min-heap
- So O(m log m) time in total

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