CS 344: Design and Analysis of Computer Algorithms

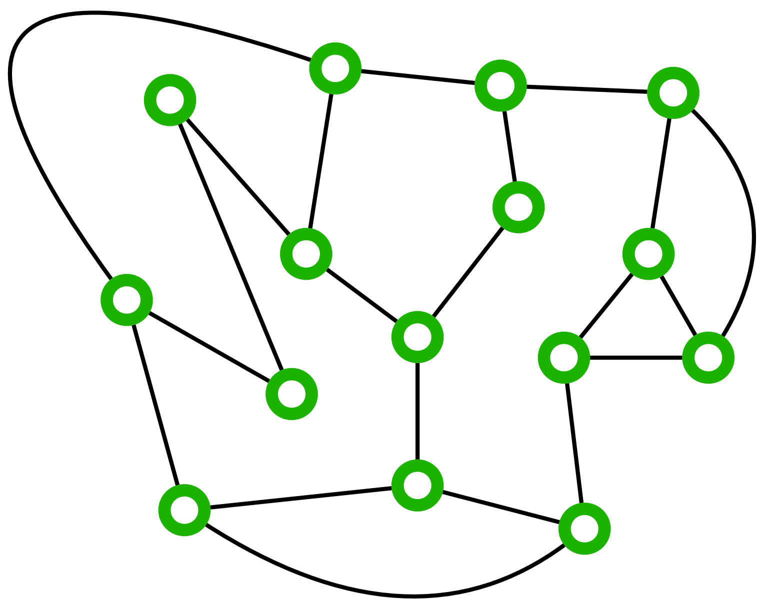
(Spring 2022 — Sections 5,6,7,8)

Lecture 17:
Minimum Spanning Trees:
The Generic “Algorithm”

The Minimum Spanning Tree Problem

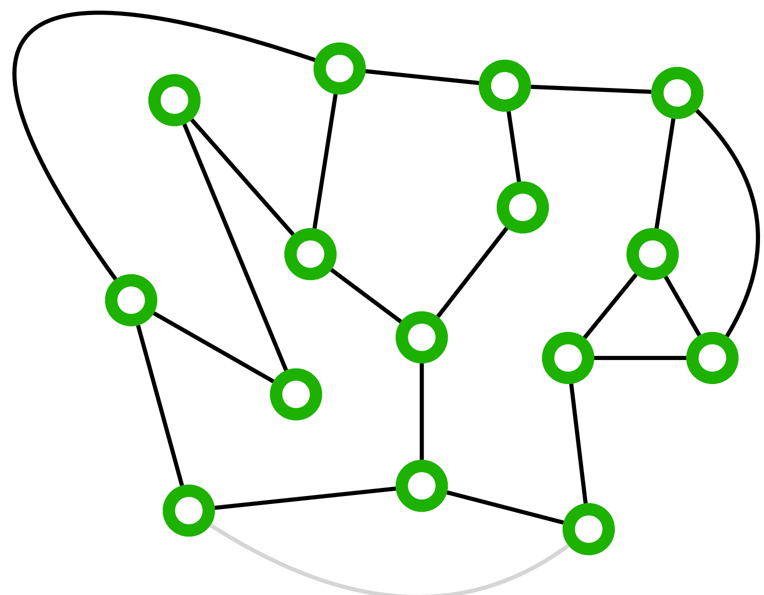
Spanning Tree

- For a connected graph $G = (V, E)$, what is a minimal subset of edges of G we can pick that it still remains connected?



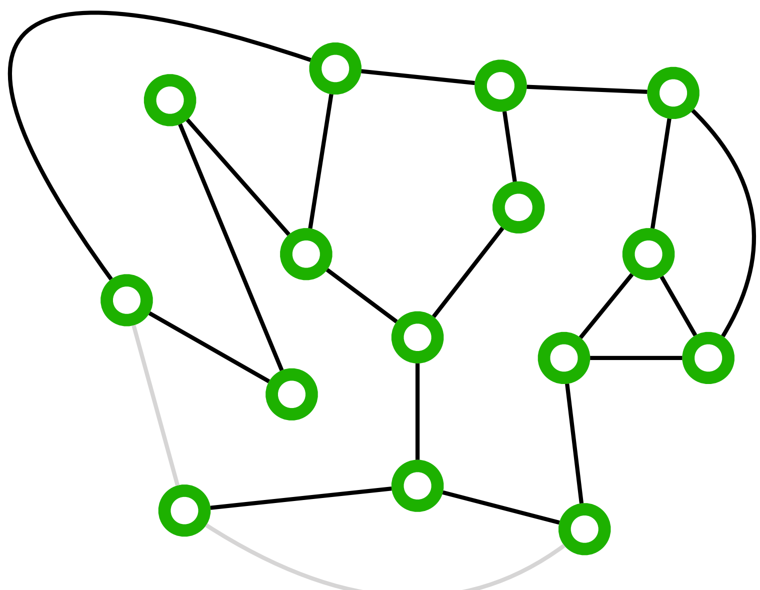
Spanning Tree

- For a connected graph $G = (V, E)$, what is a minimal subset of edges of G we can pick that it still remains connected?



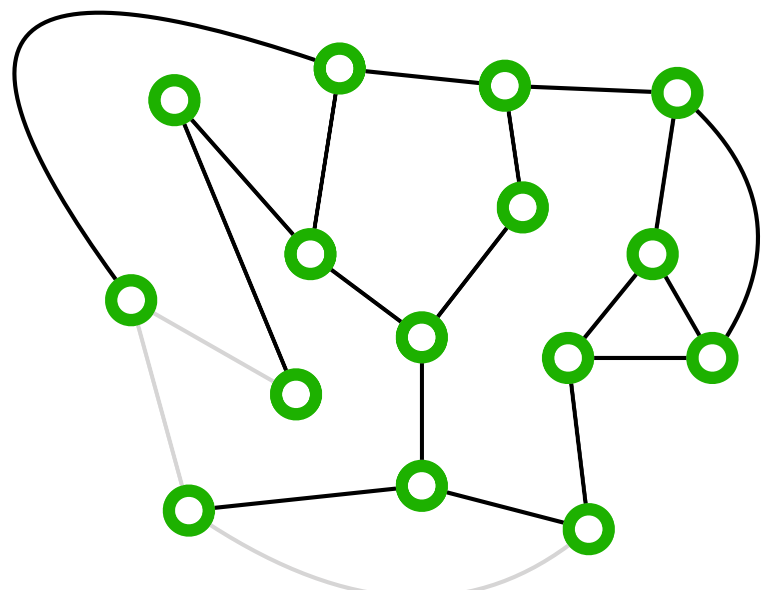
Spanning Tree

- For a connected graph $G = (V, E)$, what is a minimal subset of edges of G we can pick that it still remains connected?



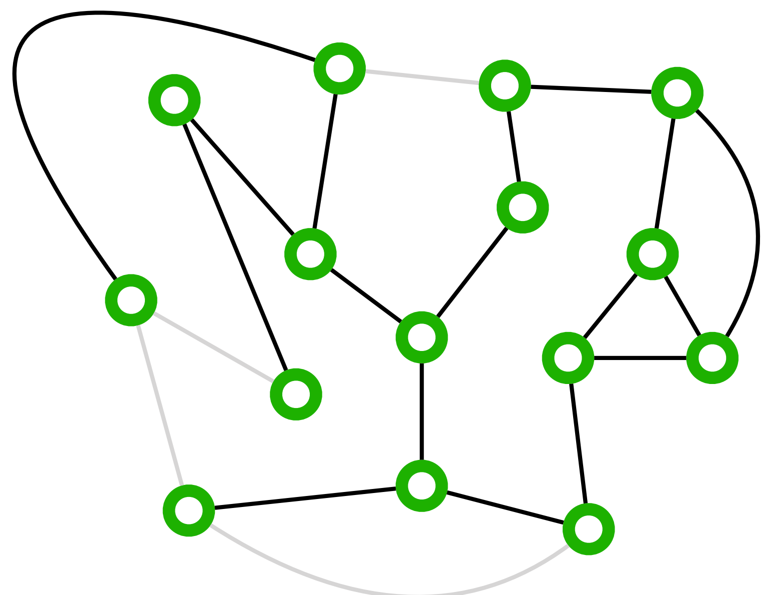
Spanning Tree

- For a connected graph $G = (V, E)$, what is a minimal subset of edges of G we can pick that it still remains connected?



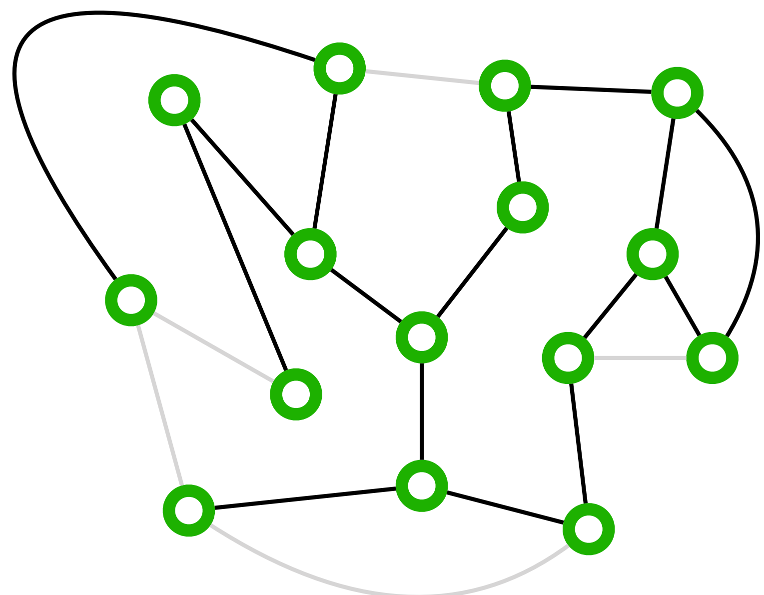
Spanning Tree

- For a connected graph $G = (V, E)$, what is a minimal subset of edges of G we can pick that it still remains connected?



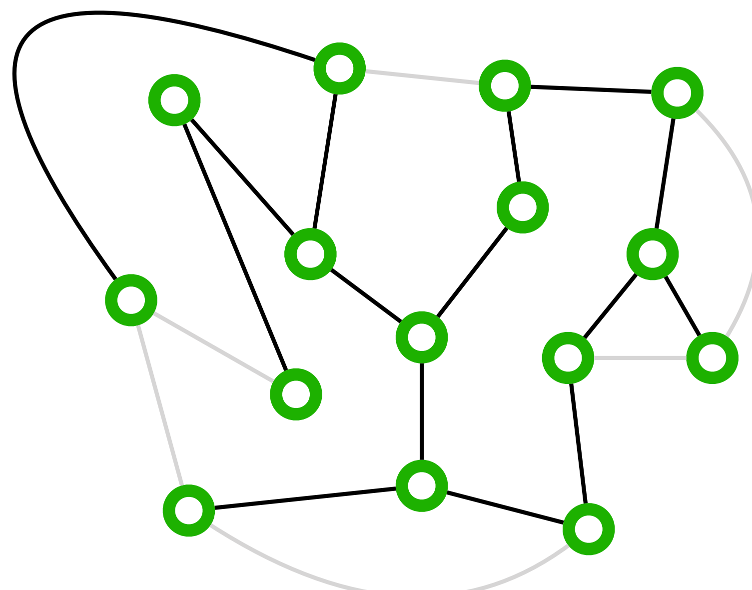
Spanning Tree

- For a connected graph $G = (V, E)$, what is a minimal subset of edges of G we can pick that it still remains connected?



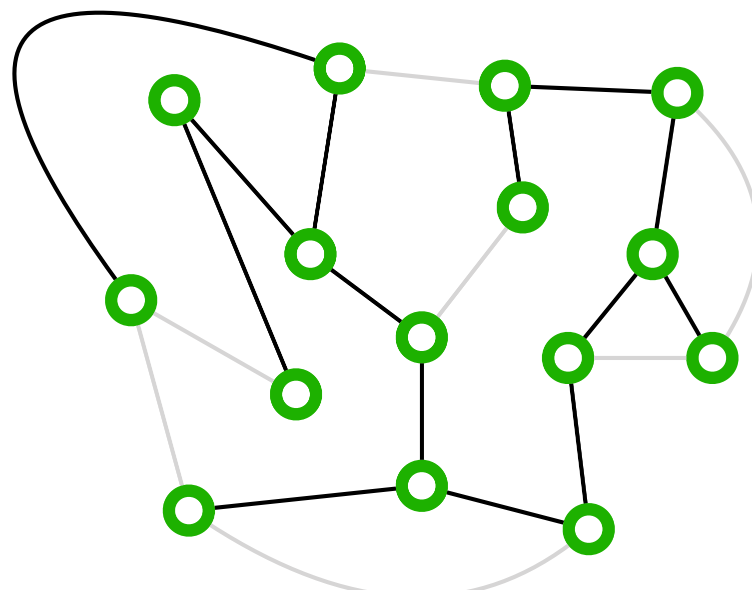
Spanning Tree

- For a connected graph $G = (V, E)$, what is a minimal subset of edges of G we can pick that it still remains connected?



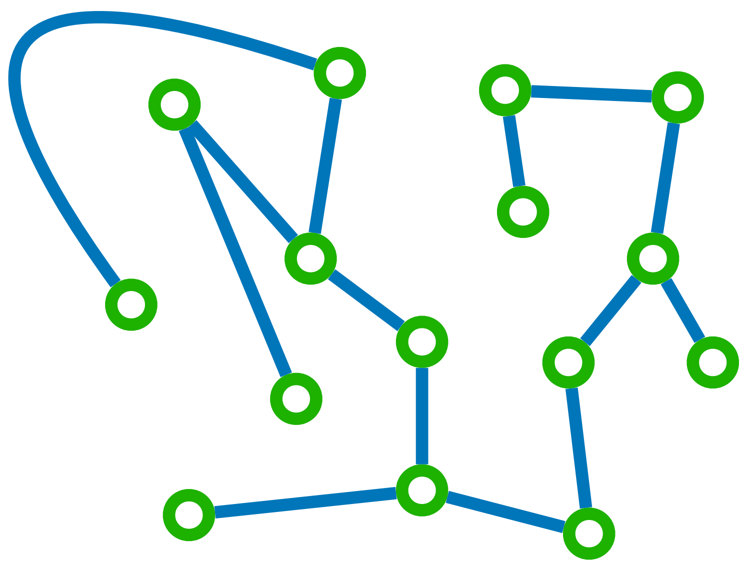
Spanning Tree

- For a connected graph $G = (V, E)$, what is a minimal subset of edges of G we can pick that it still remains connected?



Spanning Tree

- For a connected graph $G = (V, E)$, what is a minimal subset of edges of G we can pick that it still remains connected?



Spanning Tree

- For a connected graph $G = (V, E)$, what is a minimal subset of edges of G we can pick that it still remains connected?

- As long as there is a cycle in the graph, we can remove any arbitrary edge of the cycle

- We get a connected graph with no cycle: this is called a tree

  - Every tree has exactly $n - 1$ edges

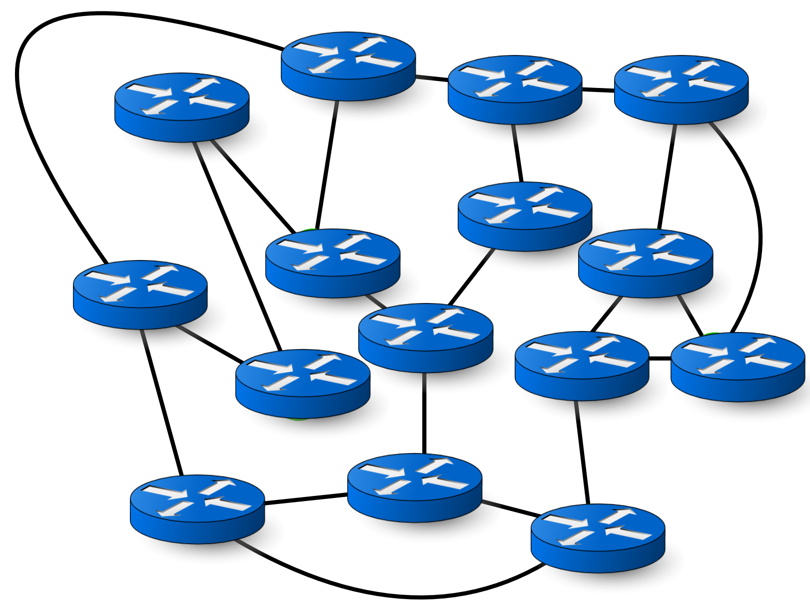
- Spanning tree: a subgraph of G on all vertices which is a tree

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 = $\sum_{e \in T} w_e$

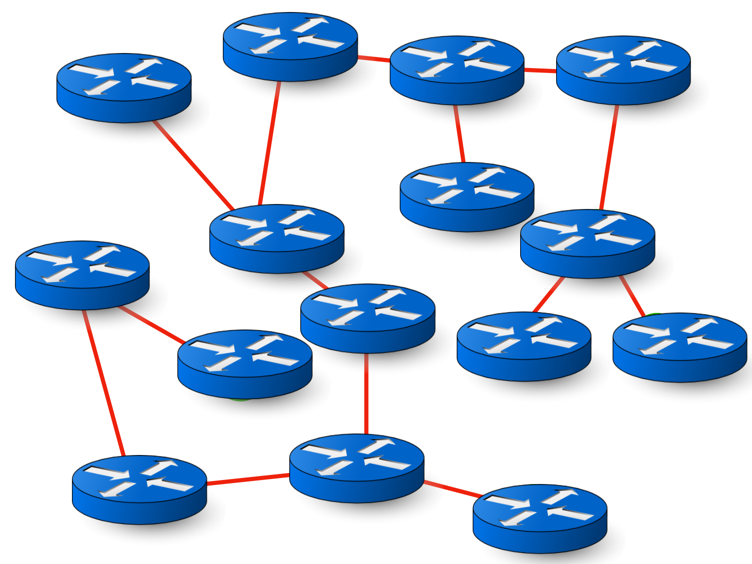
Example

- A collection of switches in a computer network
- Pick a minimum cost way of connecting all switches together



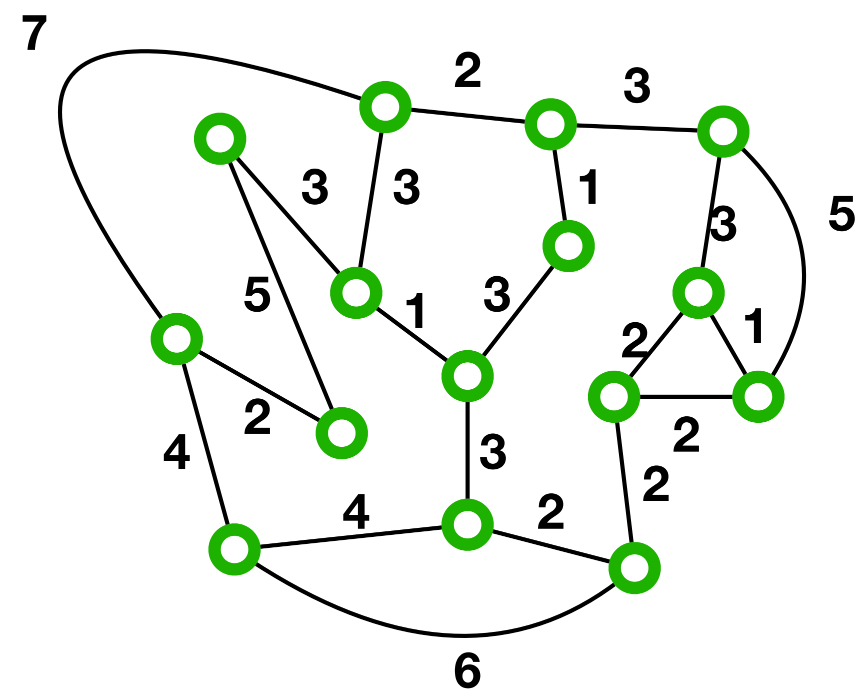
Example

- A collection of switches in a computer network
- Pick a minimum cost way of connecting all switches together



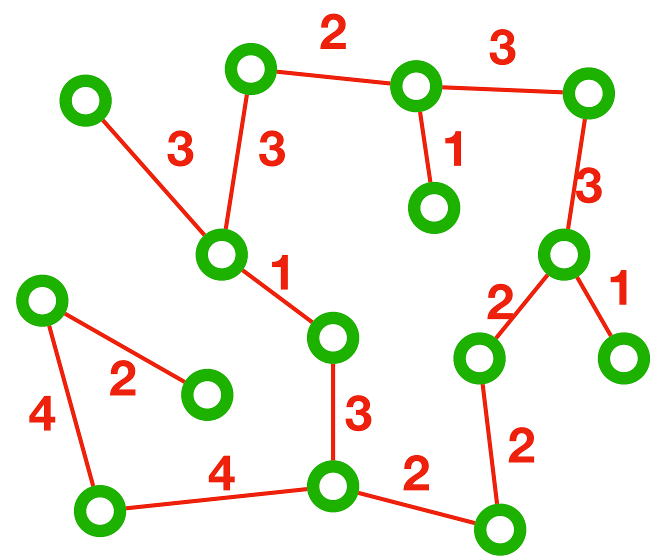
Example

- A collection of switches in a computer network
- Pick a minimum cost way of connecting all switches together



Example

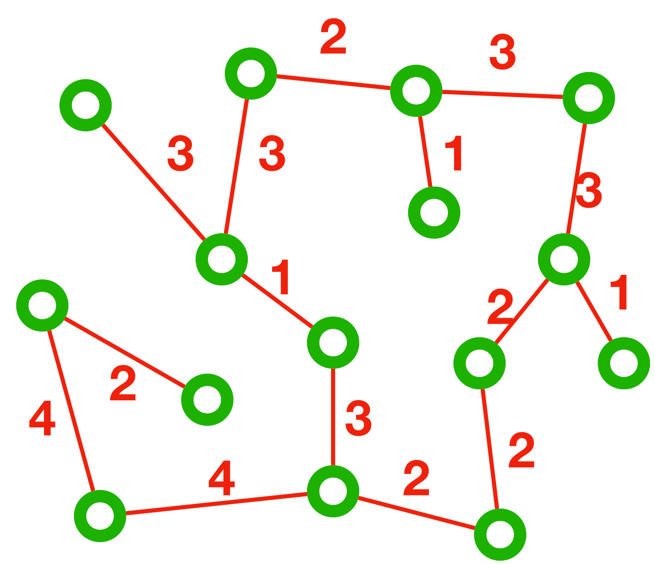
- A collection of switches in a computer network
- Pick a minimum cost way of connecting all switches together



Example

- A collection of switches in a computer network
- Pick a minimum cost way of connecting all switches together

- Total weight is:
- $3 \cdot 1 + 5 \cdot 2 + 5 \cdot 3 + 2 \cdot 4 = 36$



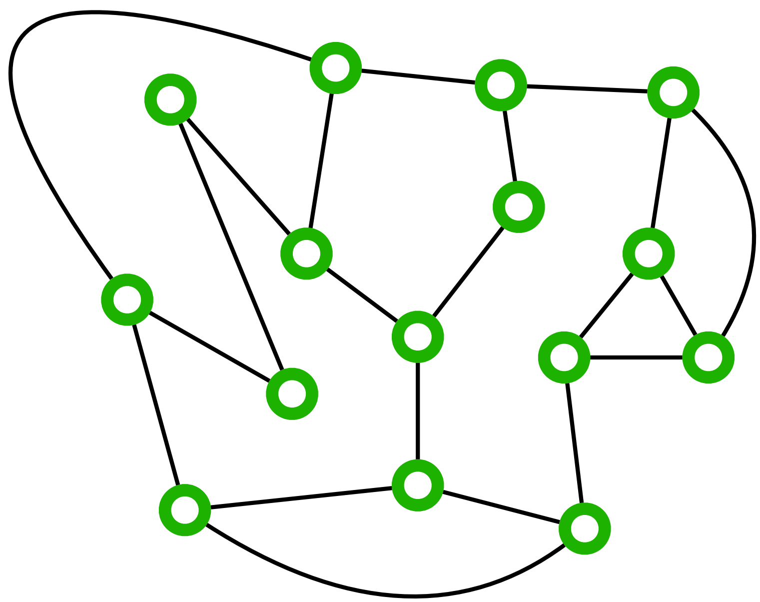
Detour: Graph Cuts

Graph Cuts

- A highly helpful notion when working with graphs
- Specially for designing algorithms related to "connectivity"

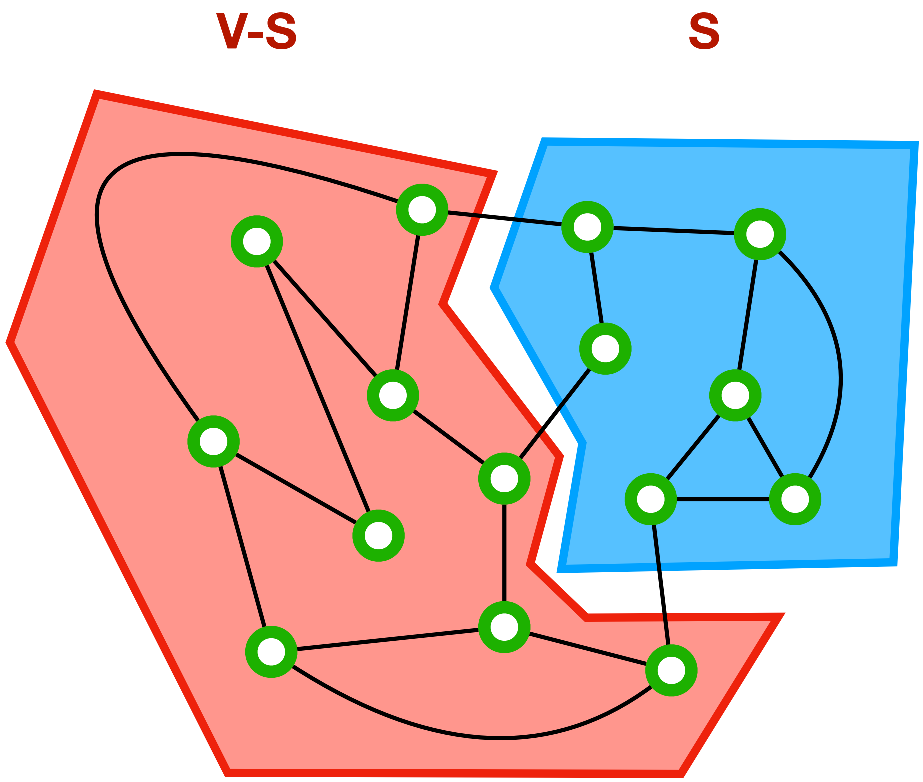
Graph Cuts

- A cut is partition of vertices into two non-empty sets S and V - S



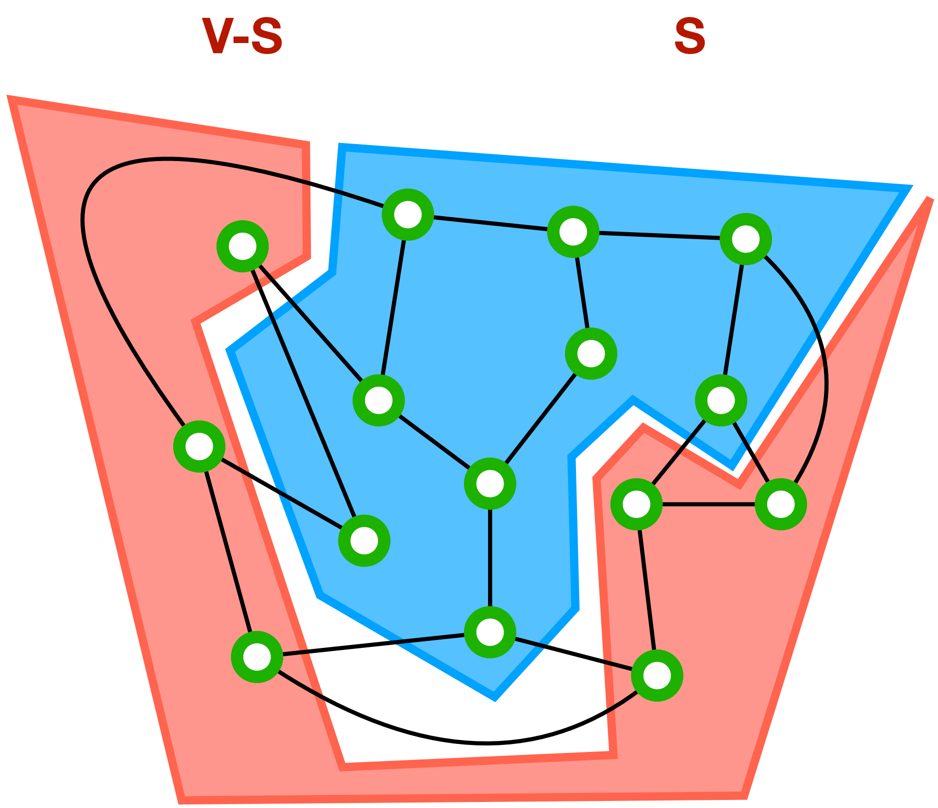
Graph Cuts

- A cut is partition of vertices into two non-empty sets S and V - S



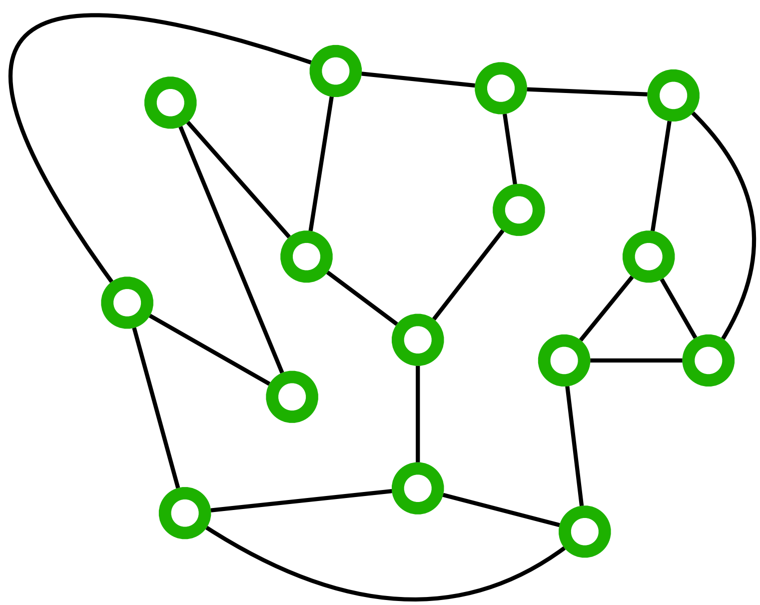
Graph Cuts

- A cut is partition of vertices into two non-empty sets S and V - S



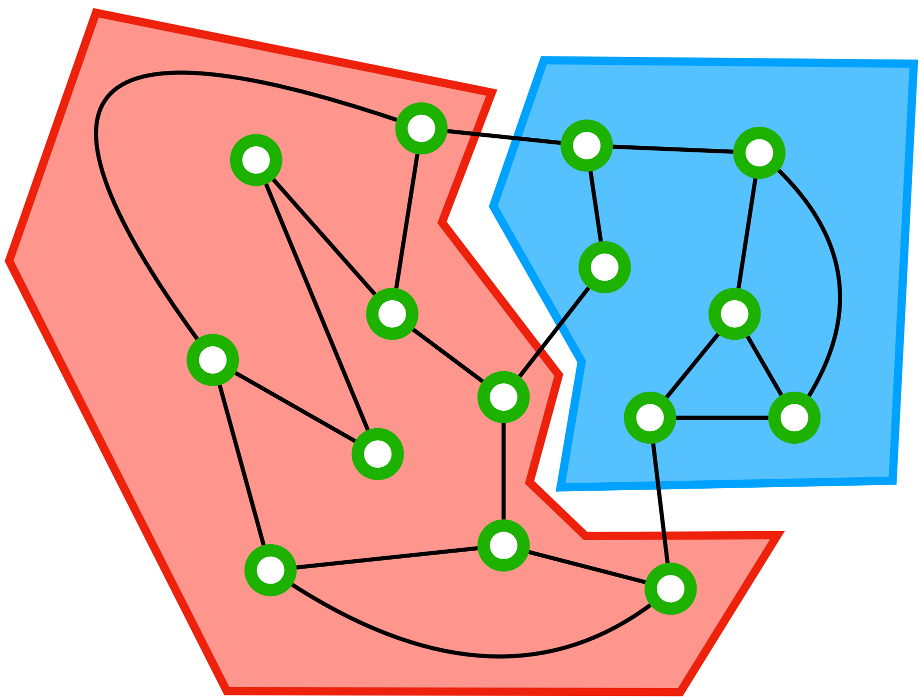
Graph Cuts

- A cut is partition of vertices into two non-empty sets S and V - S
- Cut edge: any edge between S and V-S



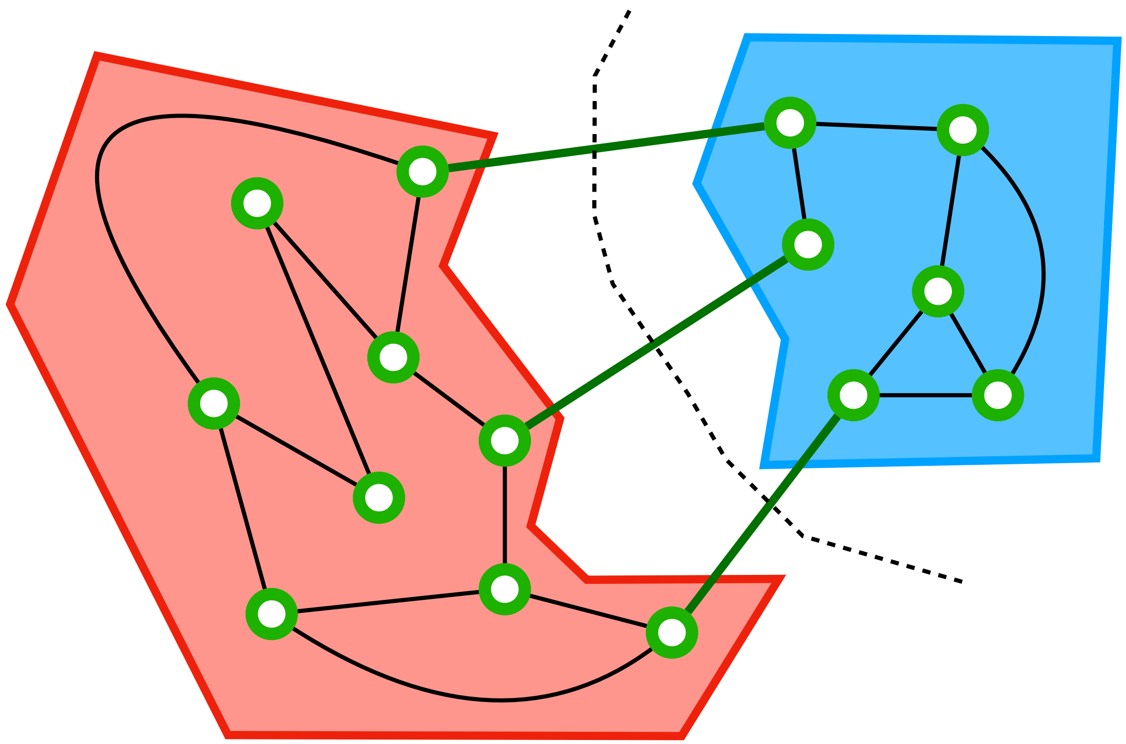
Graph Cuts

- A cut is partition of vertices into two non-empty sets S and V - S
- Cut edge: any edge between S and V-S



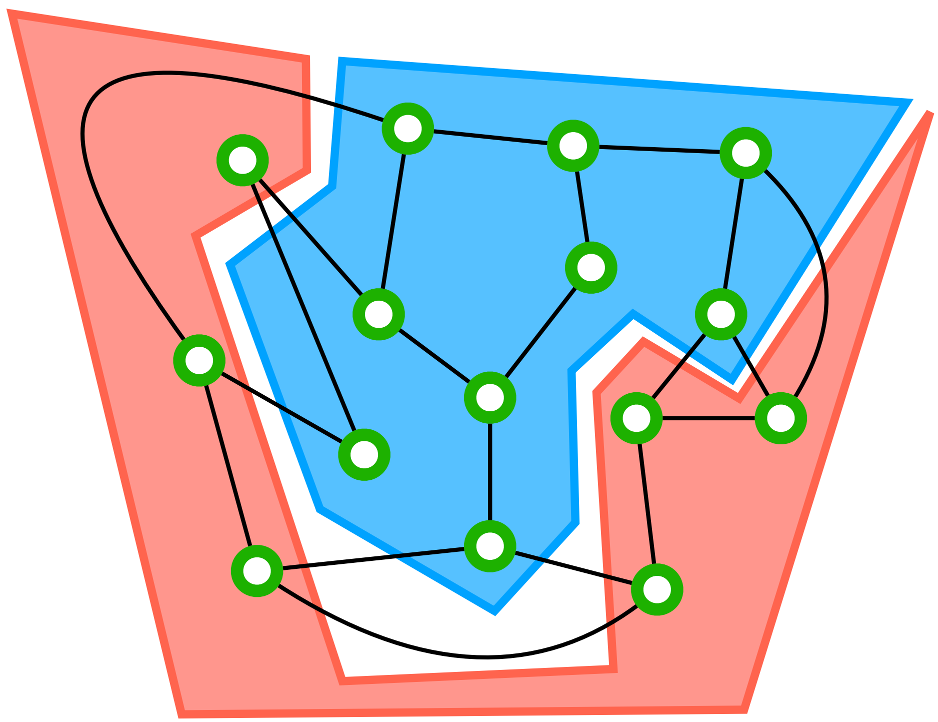
Graph Cuts

- A cut is partition of vertices into two non-empty sets S and V - S
- Cut edge: any edge between S and V-S



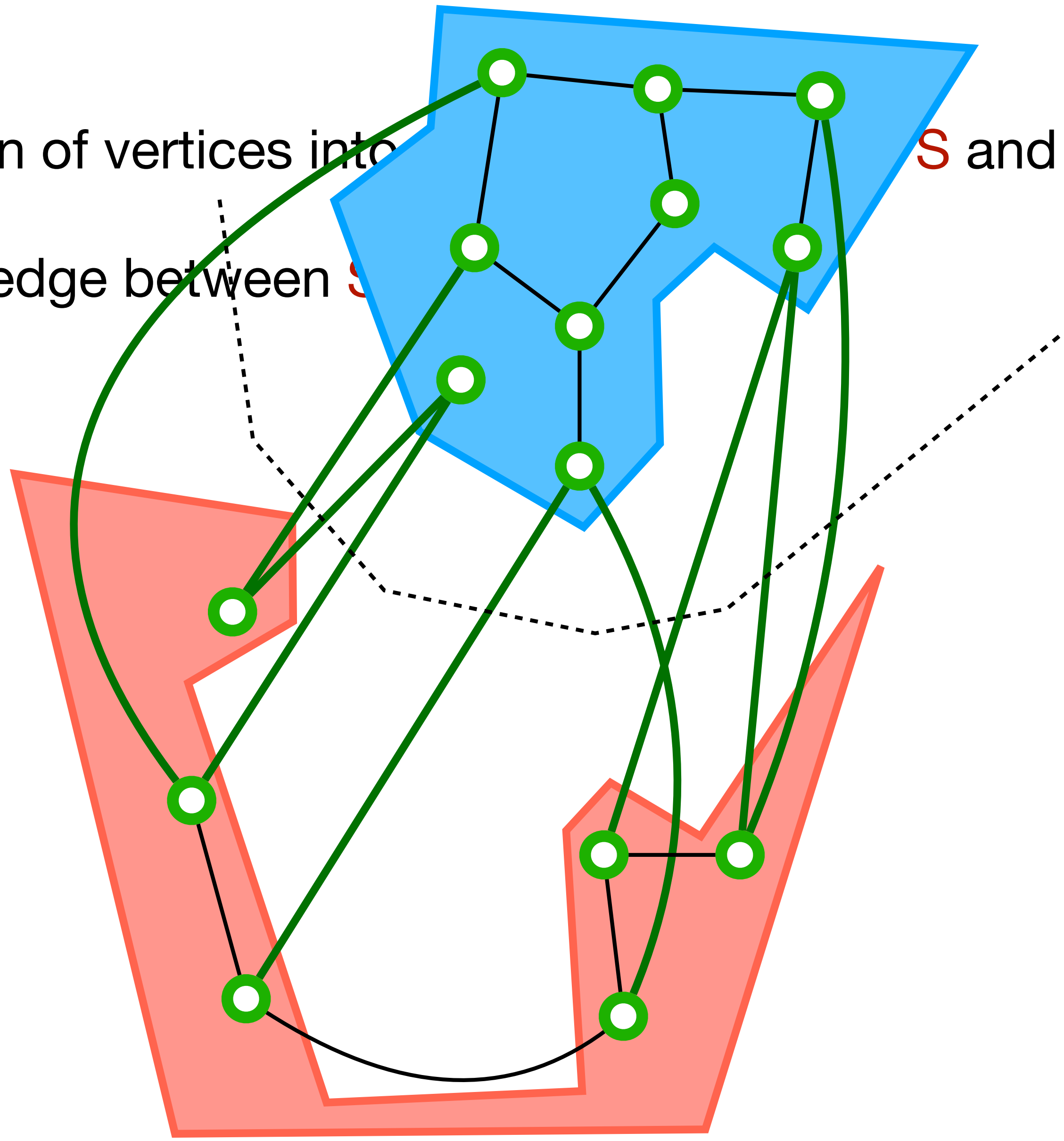
Graph Cuts

- A cut is partition of vertices into two non-empty sets S and V - S
- Cut edge: any edge between S and V-S



Graph Cuts

- A cut is partition of vertices into … S and V - S
- Cut edge: any edge between S



Graph Cuts

- A cut is partition of vertices into two non-empty sets S and V - S
- Cut edge: any edge between S and V-S
- We use $\delta(S)$ to denote the set of all cut edges

Exercise with Graph Cuts

- How many different cuts are in a graph?

Exercise with Graph Cuts

- How many different cuts are in a graph? 2^{n-1}

Exercise with Graph Cuts

- How many different cuts are in a graph? 2^{n-1}
- Number of cut edges in a singleton cut $(\{v\}, V - \{v\})$?

Exercise with Graph Cuts

- How many different cuts are in a graph? 2^{n-1}
- Number of cut edges in a singleton cut $(\{v\}, V - \{v\})$? $\deg(v)$

Exercise with Graph Cuts

- An undirected graph G=(V,E) is connected if and only if for every cut (S,V-S) in G, there is at least one cut edge, i.e., $|\delta(S)| > 0$

Exercise with Graph Cuts

- Let (S,V-S) be a cut with no cut edges in a graph G. Then, if we add any edge e to this cut, G+e will not have a cycle

A Generic “Algorithm” for MST

Algorithm Design Process

- What is the problem?
- How do we solve it?
- Why our solution is correct?
- How efficient is our solution?

- We will see a great example of this process in designing algorithms for the MST problem

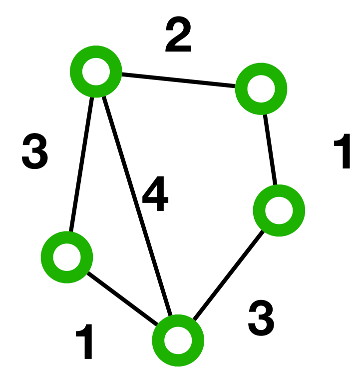
A Meta-Algorithm (“Algorithm”)

- A high level strategy to solve the MST problem
- Less detailed and precise than an actual algorithm
- So we call it a meta-algorithm
  - A rather made-up term with no precise definition

Some Definitions

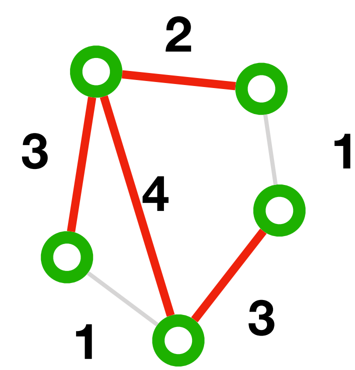
- Forest:
  - Any subgraph of a tree
- MST-good forest:
  - A forest that is a subgraph of some MST of the input graph
- Safe edge for an MST-good forest F
  - An edge e is safe for F if $F \cup \{e\}$ is another MST-good forest

Illustration



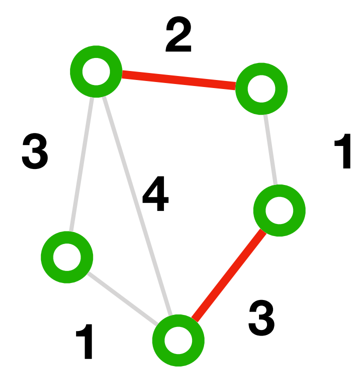
Illustration

- Forest



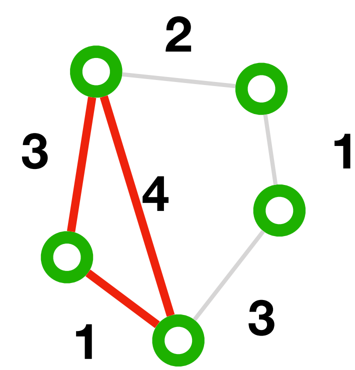
Illustration

- Forest



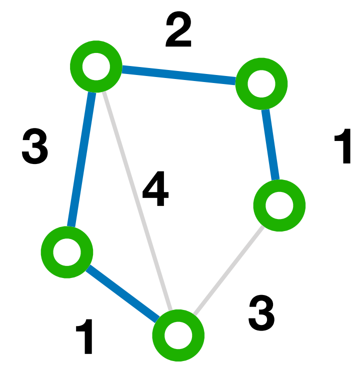
Illustration

- NOT a forest



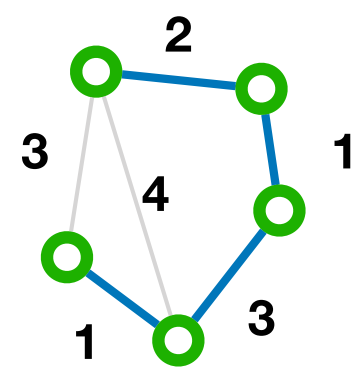
Illustration

- One MST



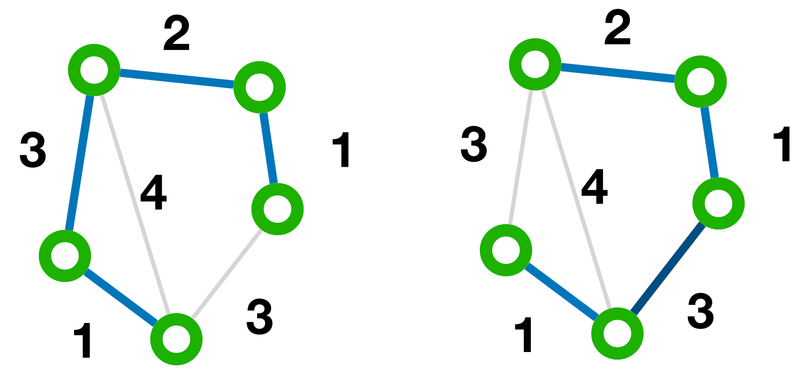
Illustration

- Another MST



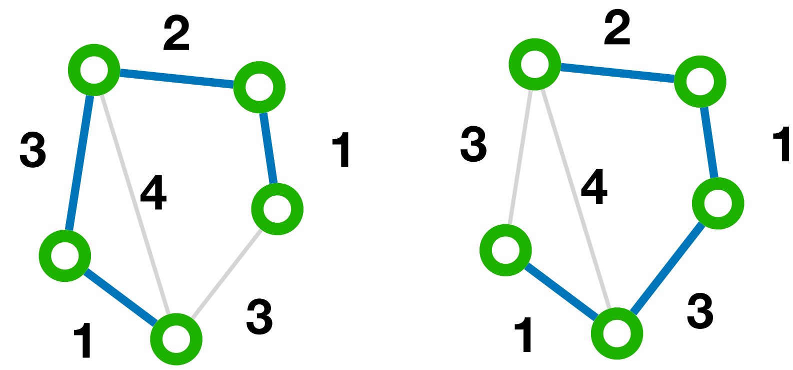
Illustration

- All MSTs of the input graph

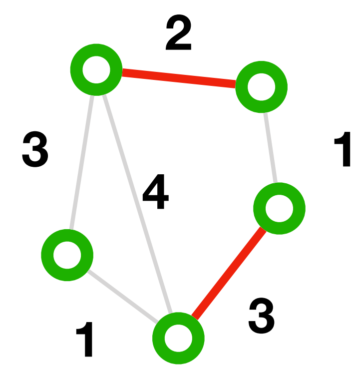


Illustration

- All MSTs of the input graph

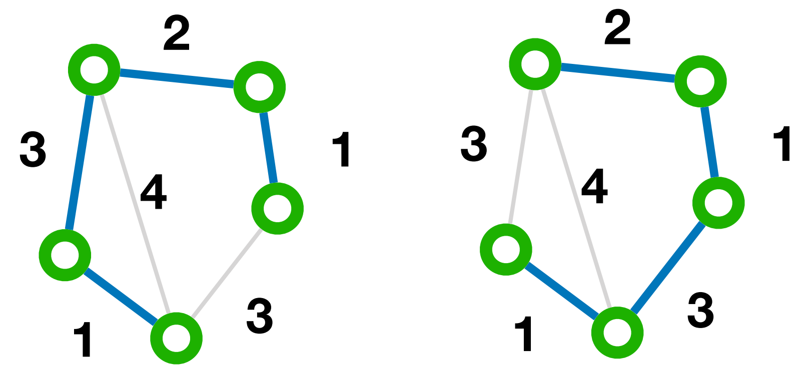


- An MST-good forest

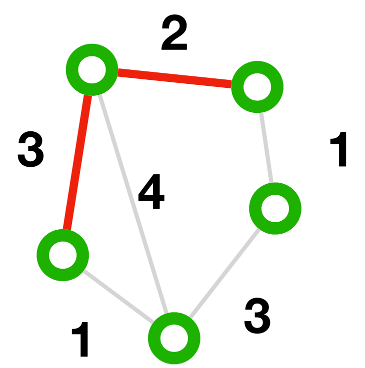


Illustration

- All MSTs of the input graph

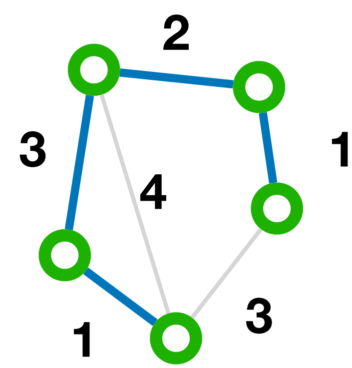


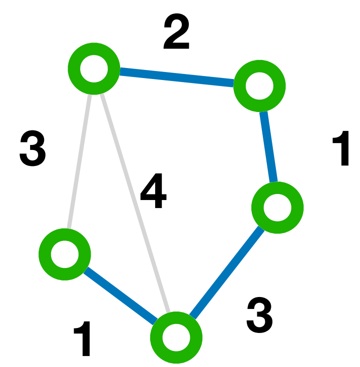
- Another MST-good forest



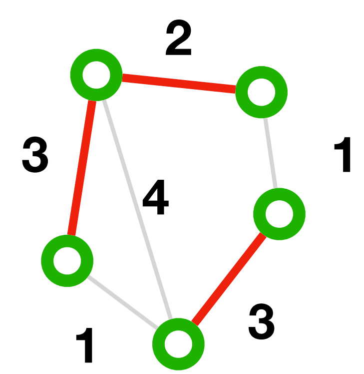
Illustration

- All MSTs of the input graph



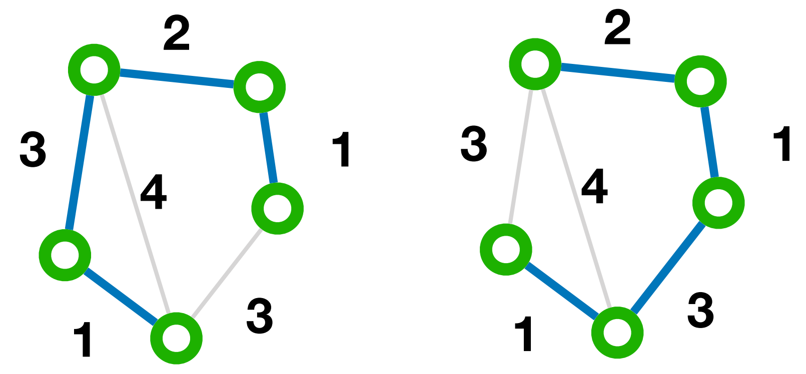


- NOT an MST-good forest

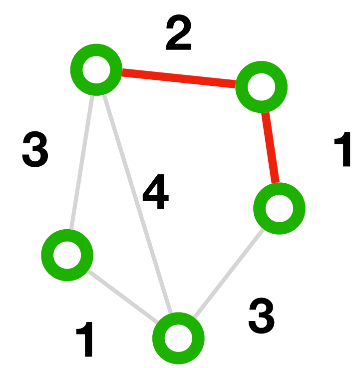


Illustration

- All MSTs of the input graph

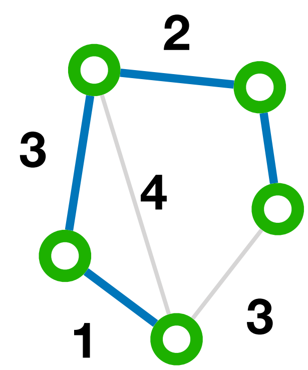


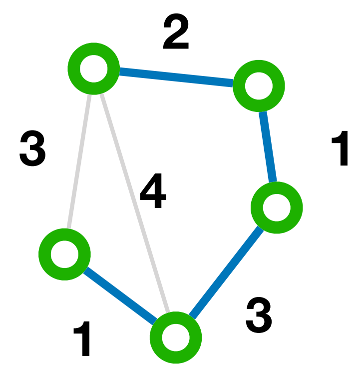
- An MST-good forest



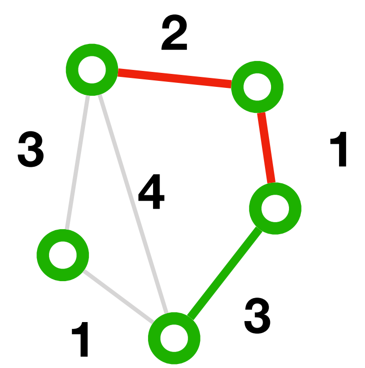
Illustration

- All MSTs of the input graph



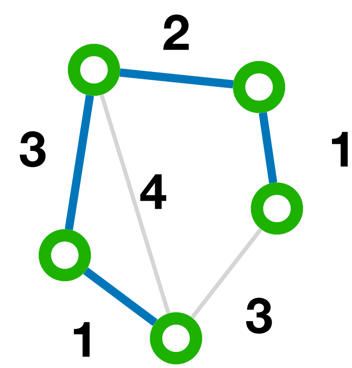


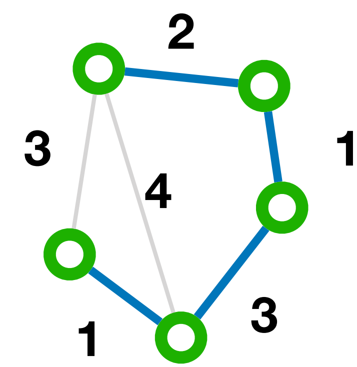
- An MST-good forest
  - A safe edge for this forest



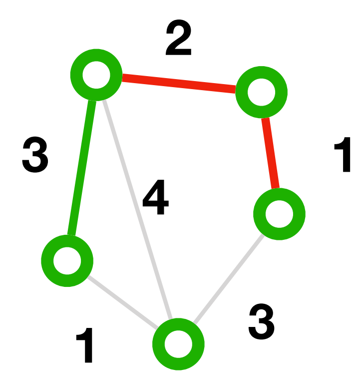
Illustration

- All MSTs of the input graph



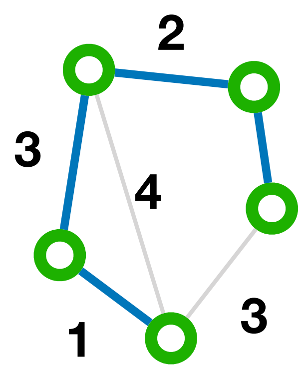


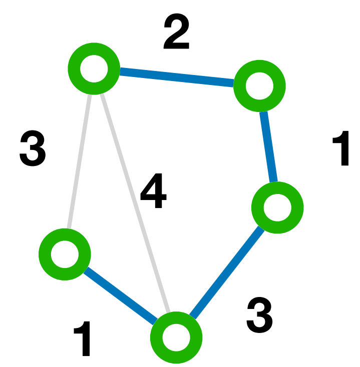
- An MST-good forest
  - Another safe edge for this forest



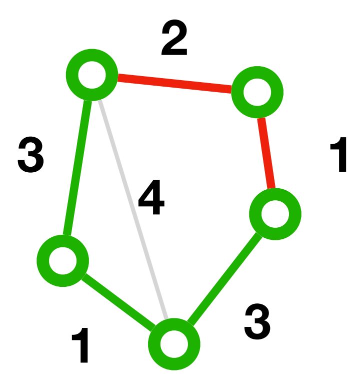
Illustration

- All MSTs of the input graph



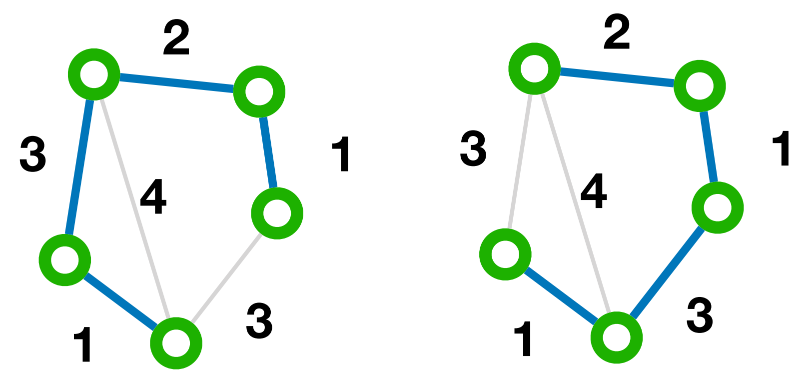


- An MST-good forest
  - All safe edge for this forest

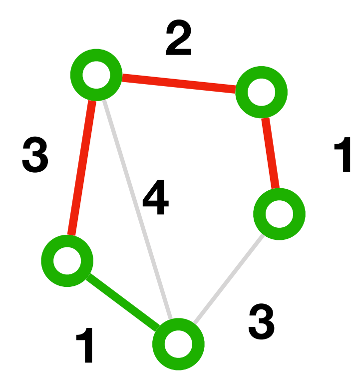


Illustration

- All MSTs of the input graph



- Another MST-good forest
  - All safe edge for this forest



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

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

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

Proof of Correctness

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

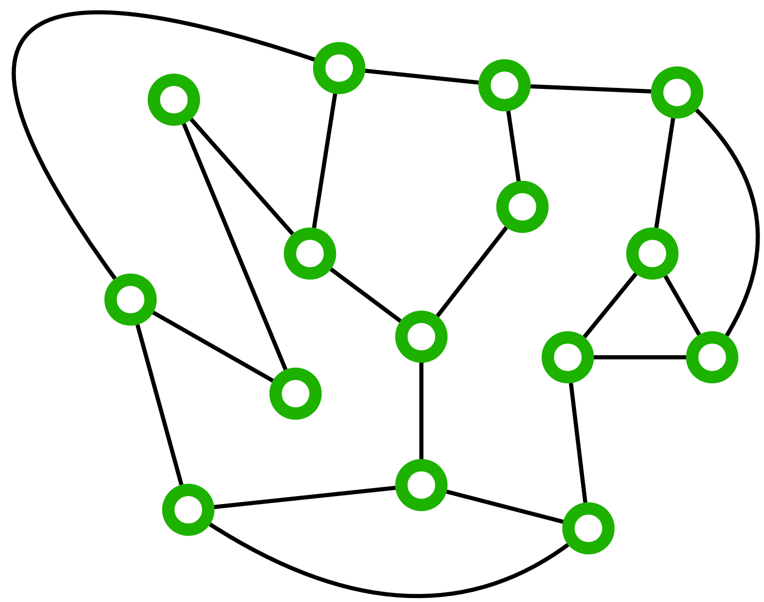
- We should now find a way of finding safe edges

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

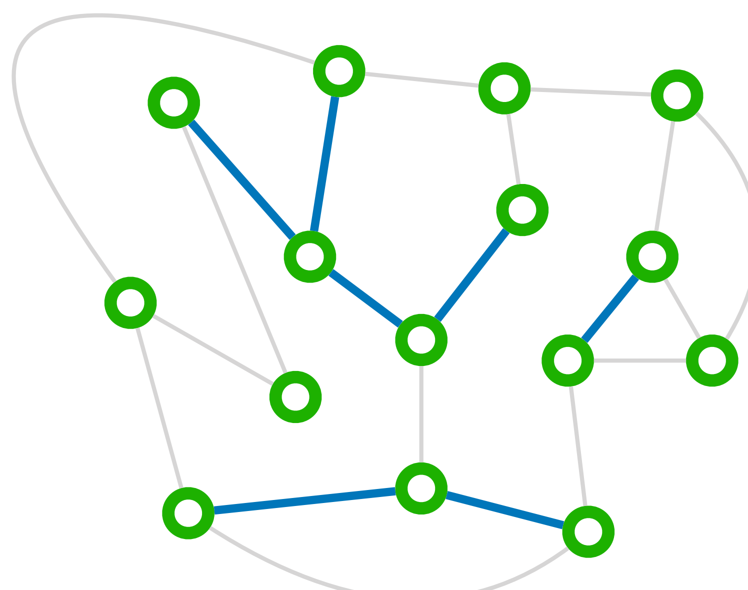
An Approach for Finding Safe Edges

- **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



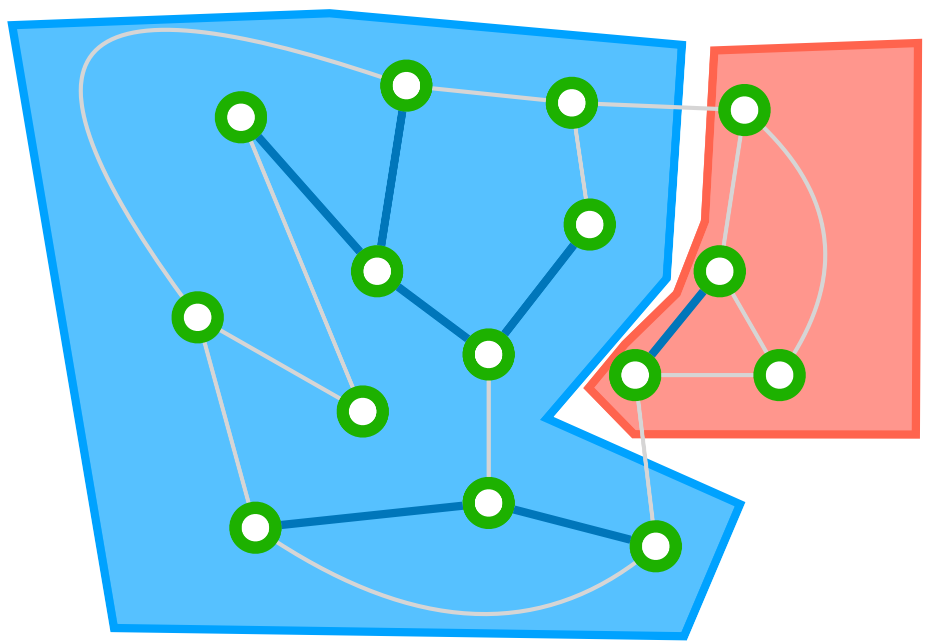
An Approach for Finding Safe Edges

- **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



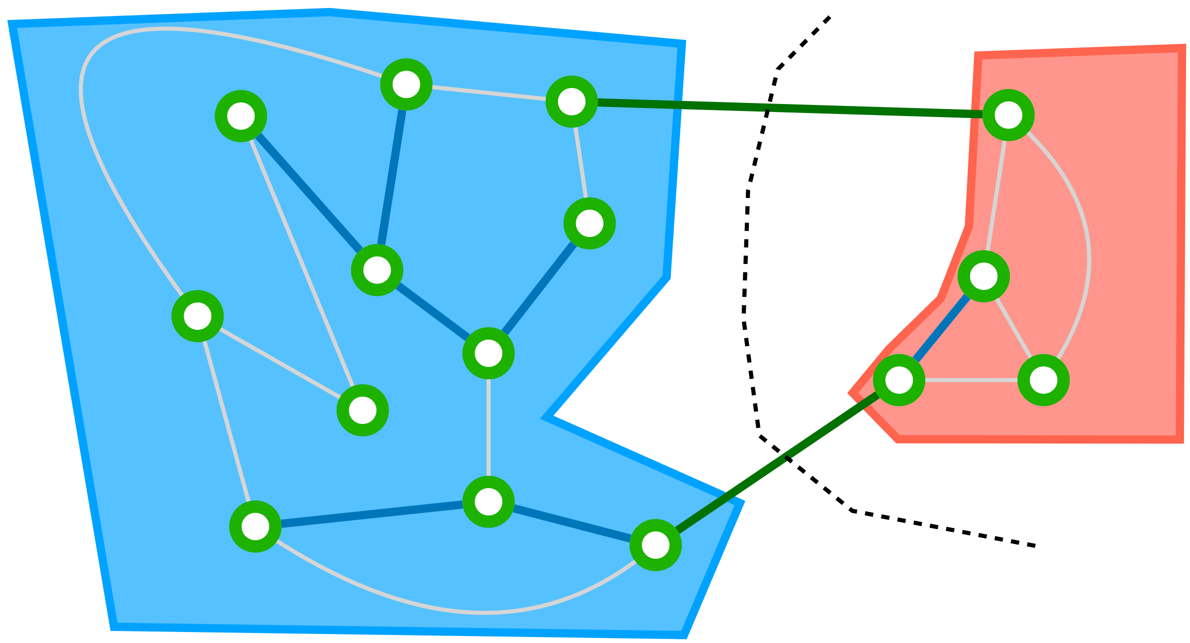
An Approach for Finding Safe Edges

- **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



Proof