Spanning Trees: Efficient Algorithms

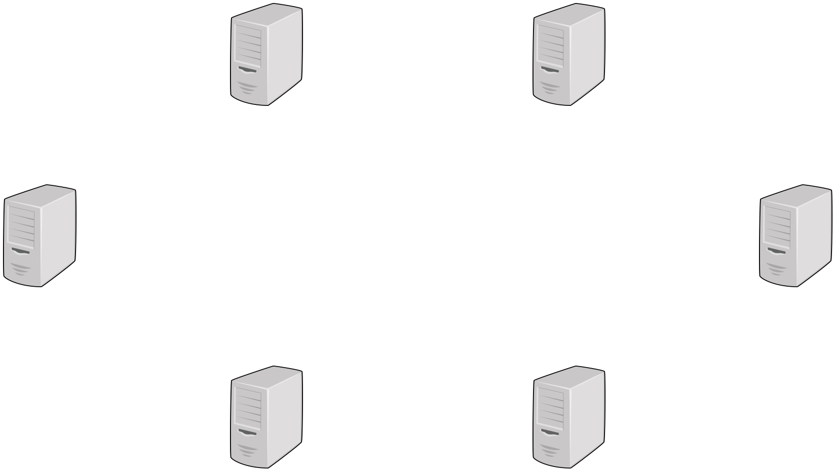
Alexander S. Kulikov

Steklov Institute of Mathematics at St. Petersburg
Russian Academy of Sciences

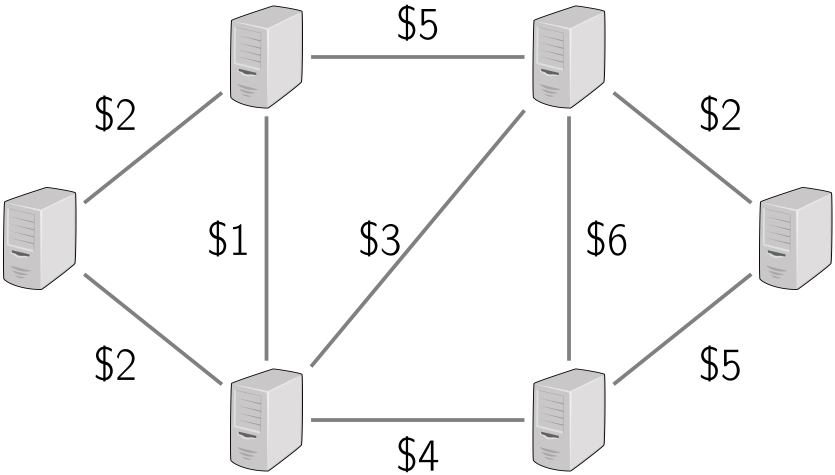
Graph Algorithms
Data Structures and Algorithms

Outline

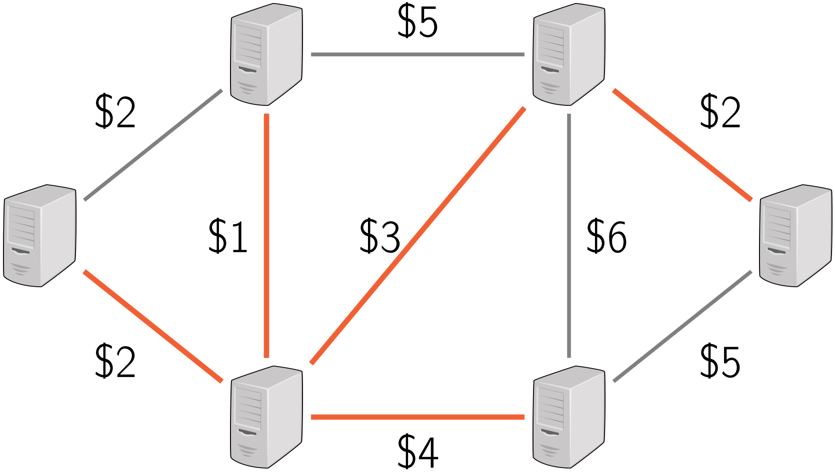
Connecting Computers by Wires



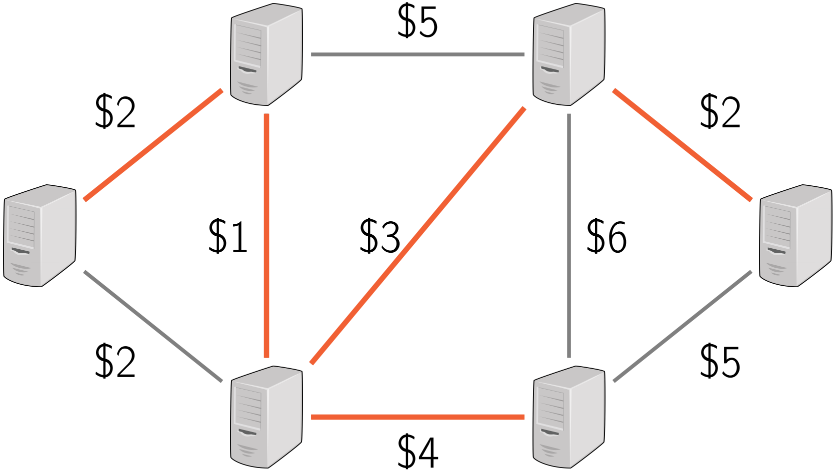
Connecting Computers by Wires



Connecting Computers by Wires

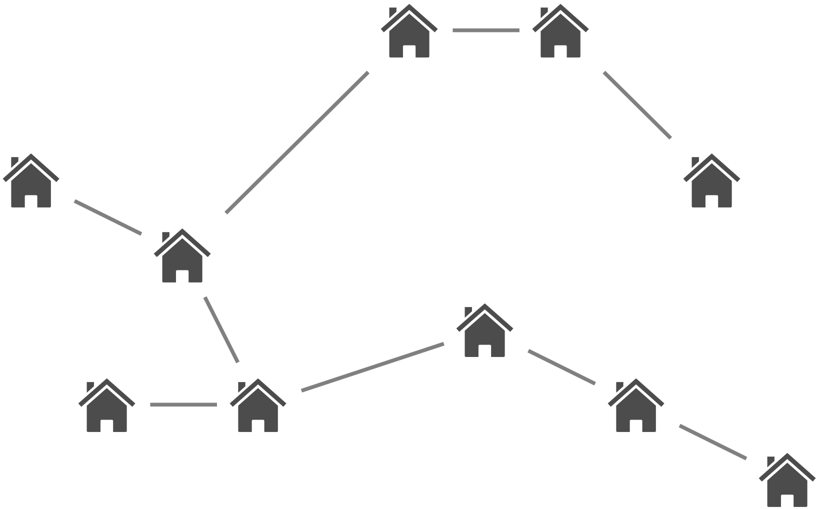


Connecting Computers by Wires



Building Roads

Building Roads



Minimum spanning tree (MST)

Input: A connected, undirected graph $G = (V, E)$ with positive edge weights.

Output: A subset of edges $E' \subseteq E$ of minimum total weight such that the graph (V, E') is connected.

Minimum spanning tree (MST)

Input: A connected, undirected graph $G = (V, E)$ with positive edge weights.

Output: A subset of edges $E' \subseteq E$ of minimum total weight such that the graph (V, E') is connected.

Remark

The set E' always forms a tree.

Properties of Trees

- A tree is an undirected graph that is connected and acyclic.

Properties of Trees

- A tree is an undirected graph that is connected and acyclic.
- A tree on n vertices has $n - 1$ edges.

Properties of Trees

- A tree is an undirected graph that is connected and acyclic.
- A tree on n vertices has $n - 1$ edges.
- Any connected undirected graph $G(V, E)$ with $|E| = |V| - 1$ is a tree.

Properties of Trees

- A tree is an undirected graph that is connected and acyclic.
- A tree on n vertices has $n - 1$ edges.
- Any connected undirected graph $G(V, E)$ with $|E| = |V| - 1$ is a tree.
- An undirected graph is a tree iff there is a unique path between any pair of its vertices.

Outline

This lesson

Two efficient greedy algorithms for the minimum spanning tree problem.

Kruskal's algorithm

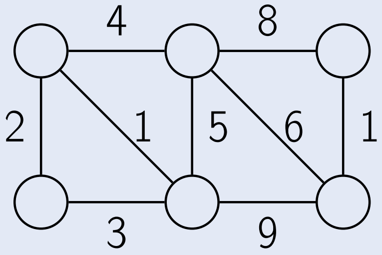
repeatedly add the next lightest edge if this doesn't produce a cycle

Prim's algorithm

repeatedly attach a new vertex to the current tree by a lightest edge

Kruskal's algorithm

repeatedly add the next lightest edge if this doesn't produce a cycle

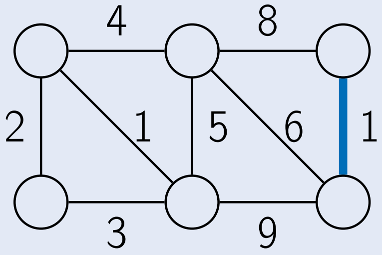


Prim's algorithm

repeatedly attach a new vertex to the current tree by a lightest edge

Kruskal's algorithm

repeatedly add the next lightest edge if this doesn't produce a cycle

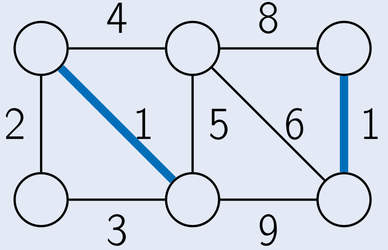


Prim's algorithm

repeatedly attach a new vertex to the current tree by a lightest edge

Kruskal's algorithm

repeatedly add the next lightest edge if this doesn't produce a cycle

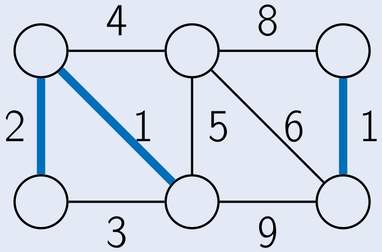


Prim's algorithm

repeatedly attach a new vertex to the current tree by a lightest edge

Kruskal's algorithm

repeatedly add the next lightest edge if this doesn't produce a cycle

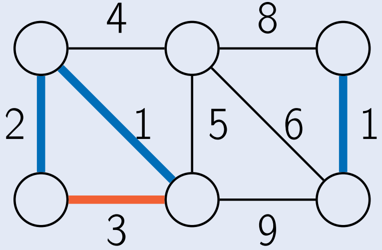


Prim's algorithm

repeatedly attach a new vertex to the current tree by a lightest edge

Kruskal's algorithm

repeatedly add the next lightest edge if this doesn't produce a cycle

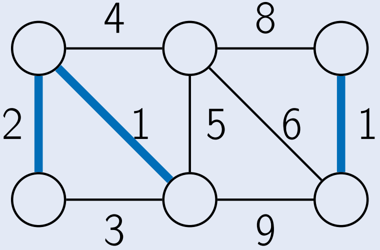


Prim's algorithm

repeatedly attach a new vertex to the current tree by a lightest edge

Kruskal's algorithm

repeatedly add the next lightest edge if this doesn't produce a cycle

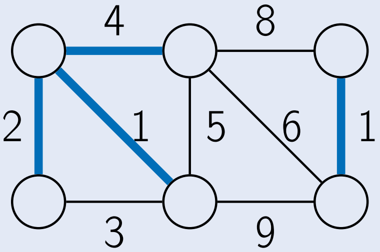


Prim's algorithm

repeatedly attach a new vertex to the current tree by a lightest edge

Kruskal's algorithm

repeatedly add the next lightest edge if this doesn't produce a cycle

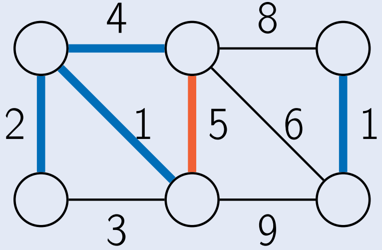


Prim's algorithm

repeatedly attach a new vertex to the current tree by a lightest edge

Kruskal's algorithm

repeatedly add the next lightest edge if this doesn't produce a cycle

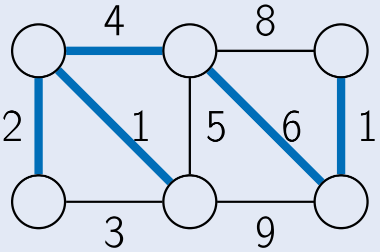


Prim's algorithm

repeatedly attach a new vertex to the current tree by a lightest edge

Kruskal's algorithm

repeatedly add the next lightest edge if this doesn't produce a cycle

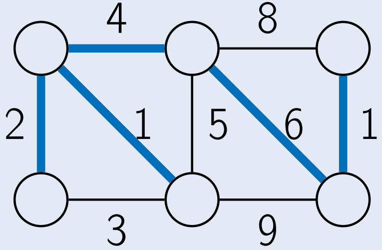


Prim's algorithm

repeatedly attach a new vertex to the current tree by a lightest edge

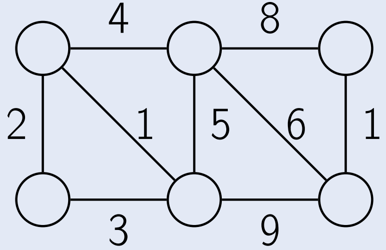
Kruskal's algorithm

repeatedly add the next lightest edge if this doesn't produce a cycle



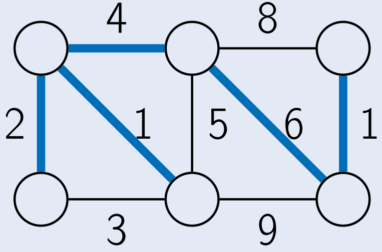
Prim's algorithm

repeatedly attach a new vertex to the current tree by a lightest edge



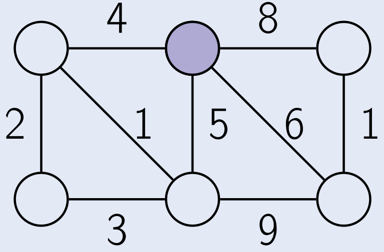
Kruskal's algorithm

repeatedly add the next lightest edge if this doesn't produce a cycle



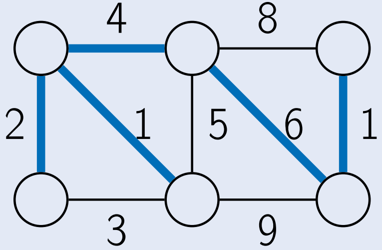
Prim's algorithm

repeatedly attach a new vertex to the current tree by a lightest edge



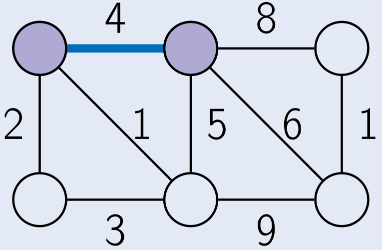
Kruskal's algorithm

repeatedly add the next lightest edge if this doesn't produce a cycle



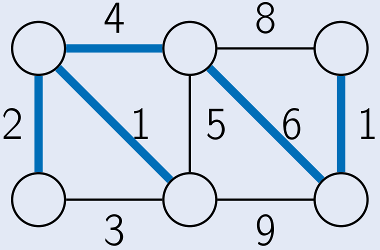
Prim's algorithm

repeatedly attach a new vertex to the current tree by a lightest edge



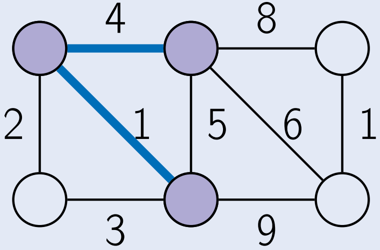
Kruskal's algorithm

repeatedly add the next lightest edge if this doesn't produce a cycle



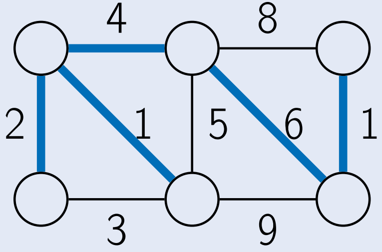
Prim's algorithm

repeatedly attach a new vertex to the current tree by a lightest edge



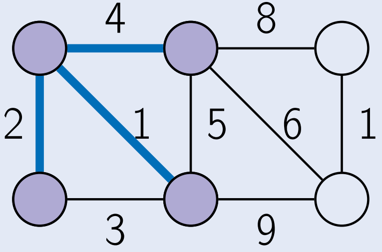
Kruskal's algorithm

repeatedly add the next lightest edge if this doesn't produce a cycle



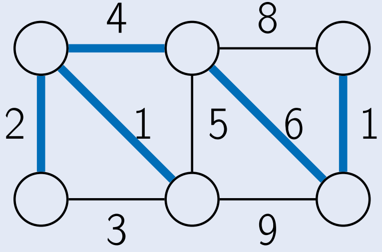
Prim's algorithm

repeatedly attach a new vertex to the current tree by a lightest edge



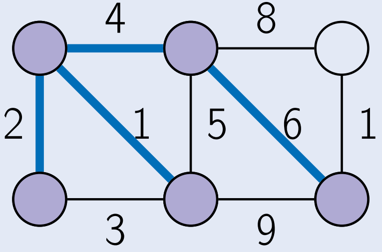
Kruskal's algorithm

repeatedly add the next lightest edge if this doesn't produce a cycle



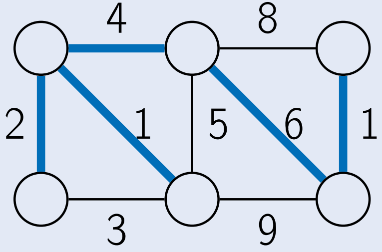
Prim's algorithm

repeatedly attach a new vertex to the current tree by a lightest edge



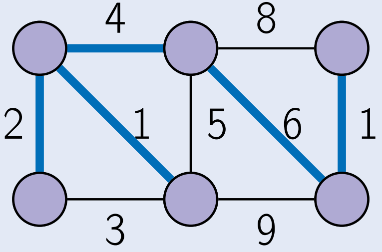
Kruskal's algorithm

repeatedly add the next lightest edge if this doesn't produce a cycle



Prim's algorithm

repeatedly attach a new vertex to the current tree by a lightest edge



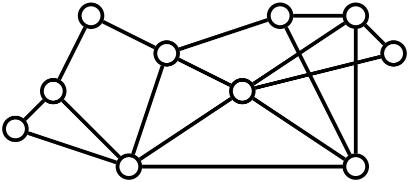
Outline

Cut property

Let $X \subseteq E$ be a part of a MST of $G(V, E)$, $S \subseteq V$ be such that no edge of X crosses between S and $V - S$, and $e \in E$ be a lightest edge across this partition. Then $X + \{e\}$ is a part of some MST.

Cut property

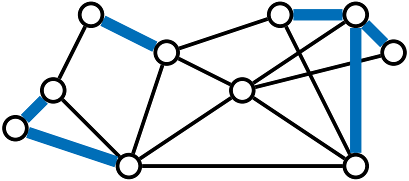
Let $X \subseteq E$ be a part of a MST of $G(V, E)$, $S \subseteq V$ be such that no edge of X crosses between S and $V - S$, and $e \in E$ be a lightest edge across this partition. Then $X + \{e\}$ is a part of some MST.



graph G

Cut property

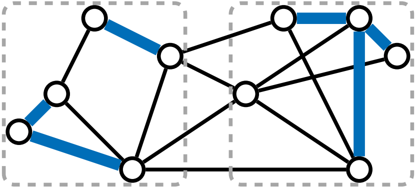
Let $X \subseteq E$ be a part of a MST of $G(V, E)$, $S \subseteq V$ be such that no edge of X crosses between S and $V - S$, and $e \in E$ be a lightest edge across this partition. Then $X + \{e\}$ is a part of some MST.



subset $X \subseteq E$ of some MST

Cut property

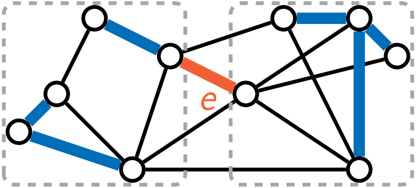
Let $X \subseteq E$ be a part of a MST of $G(V, E)$, $S \subseteq V$ be such that no edge of X crosses between S and $V - S$, and $e \in E$ be a lightest edge across this partition. Then $X + \{e\}$ is a part of some MST.



partition of V into S and $V - S$

Cut property

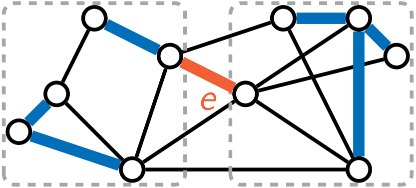
Let $X \subseteq E$ be a part of a MST of $G(V, E)$, $S \subseteq V$ be such that no edge of X crosses between S and $V - S$, and $e \in E$ be a lightest edge across this partition. Then $X + \{e\}$ is a part of some MST.



lightest edge e between S and $V - S$

Cut property

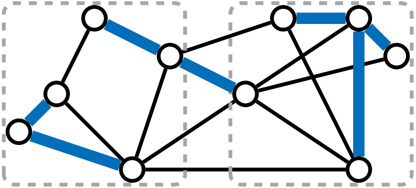
Let $X \subseteq E$ be a part of a MST of $G(V, E)$, $S \subseteq V$ be such that no edge of X crosses between S and $V - S$, and $e \in E$ be a lightest edge across this partition. Then $X + \{e\}$ is a part of some MST.



cut property states that $X + \{e\}$ is also a part of some MST

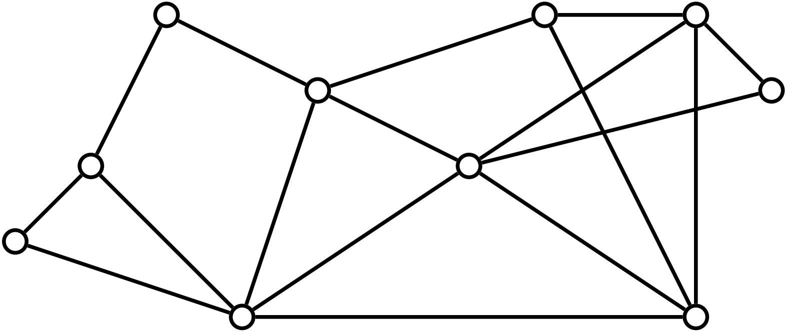
Cut property

Let $X \subseteq E$ be a part of a MST of $G(V, E)$, $S \subseteq V$ be such that no edge of X crosses between S and $V - S$, and $e \in E$ be a lightest edge across this partition. Then $X + \{e\}$ is a part of some MST.



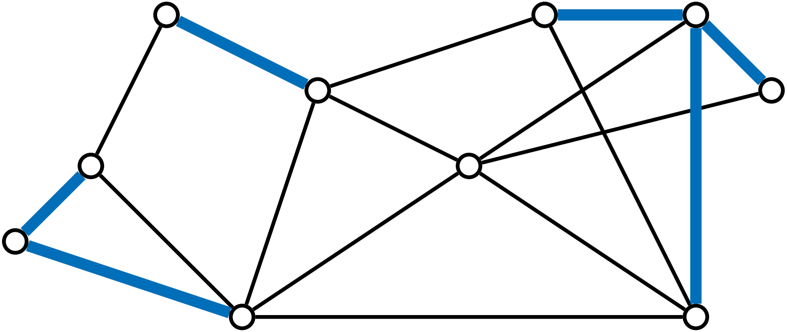
cut property states that $X + \{e\}$ is also a part of some MST

Proof



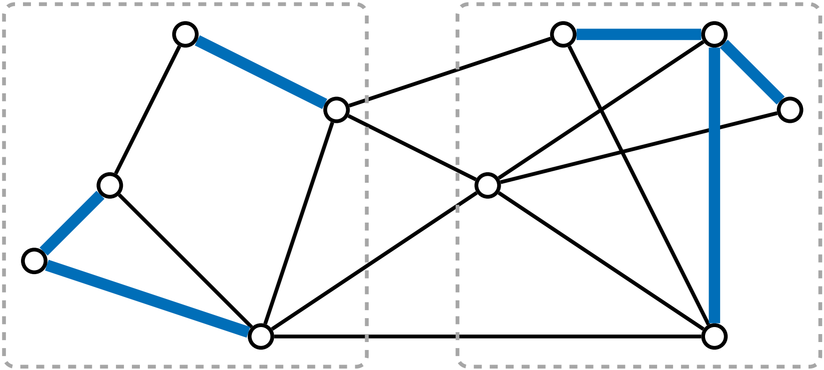
graph G

Proof



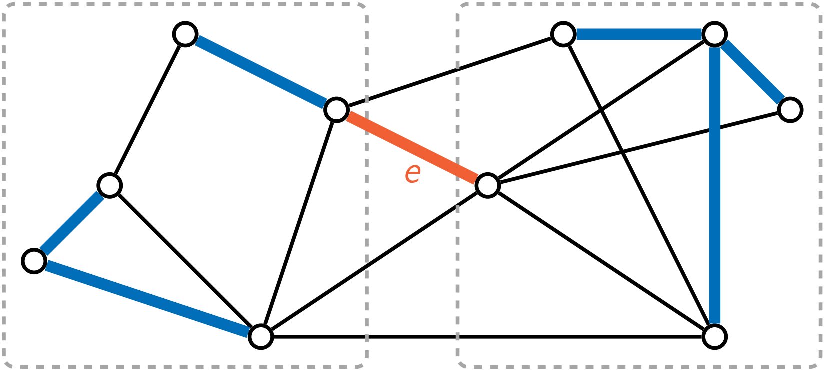
subset $X \subseteq E$ of some MST T

Proof



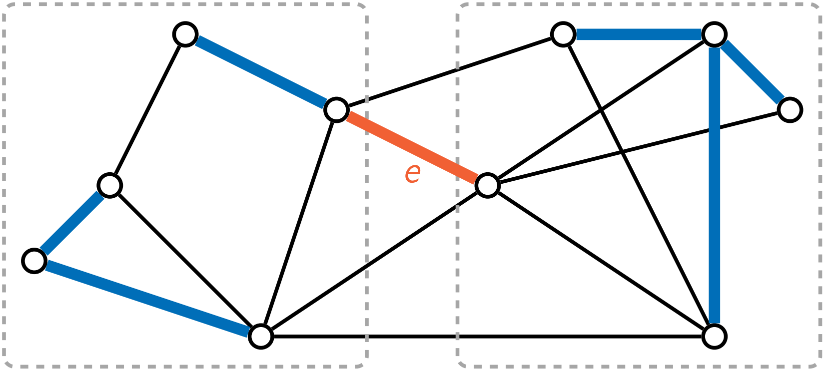
partition of V into S and $V - S$

Proof



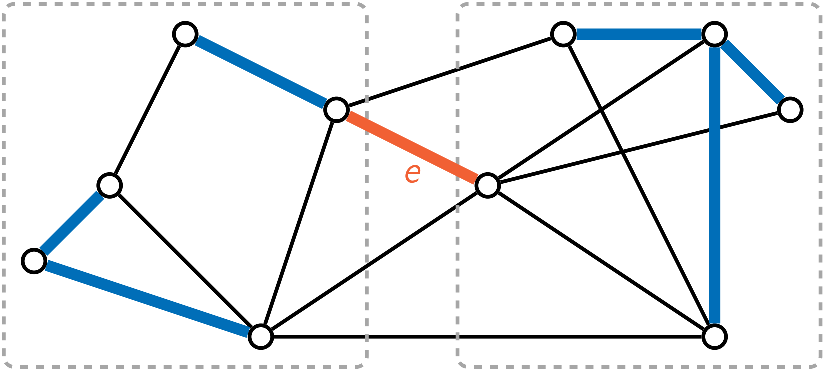
lightest edge e between S and $V - S$

Proof



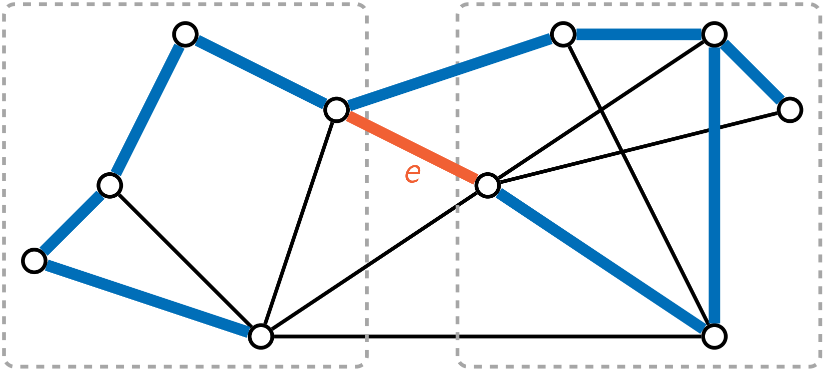
we know that X is a part of some MST T and need to show that $X + \{e\}$ is also a part of a (possibly different) MST

Proof



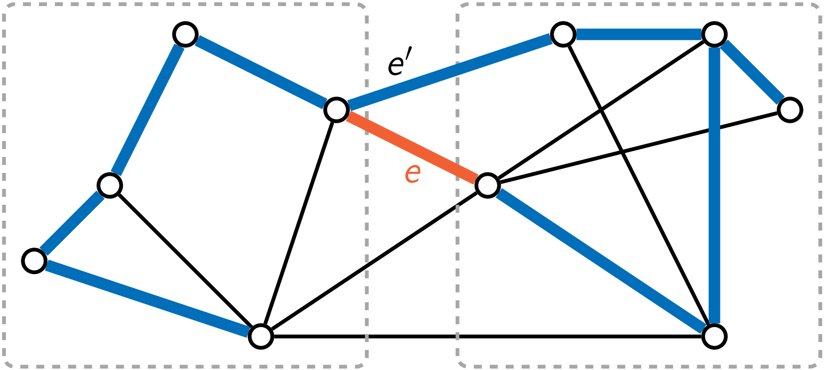
if $e \in T$ then there is nothing to prove; so assume that $e \notin T$

Proof



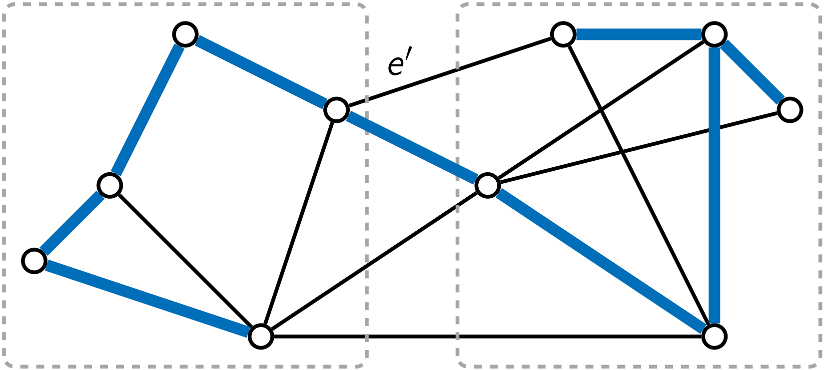
consider the tree T

Proof



adding e to T creates a cycle; let e' be an edge of this cycle that crosses S and $V - S$

Proof



then $T' = T - \{e'\} + \{e\}$ is an MST containing $X + \{e\}$: it is a tree, and $w(T') \leq w(T)$ since $w(e) \leq w(e')$

Outline

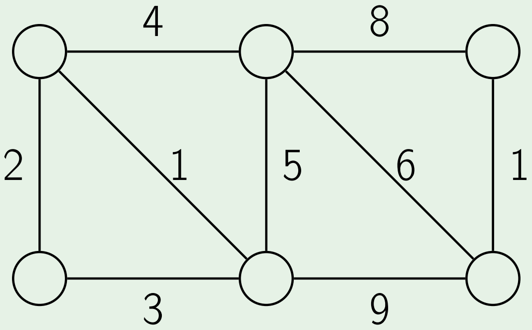
Kruskal's Algorithm

- Algorithm: repeatedly add to X the next lightest edge e that doesn't produce a cycle
- At any point of time, the set X is a forest, that is, a collection of trees
- The next edge e connects two different trees—say, T_1 and T_2
- The edge e is the lightest between T_1 and $V - T_1$, hence adding e is safe

Implementation Details

- use disjoint sets data structure
- initially, each vertex lies in a separate set
- each set is the set of vertices of a connected component
- to check whether the current edge $\{u, v\}$ produces a cycle, we check whether u and v belong to the same set

Example



Kruskal(G)

```
for all u ∈ V:
    MakeSet(v)
X ← empty set
sort the edges E by weight
for all {u, v} ∈ E in non-decreasing
    weight order:
    if Find(u) ≠ Find(v):
        add {u, v} to X
        Union(u, v)
return X
```

Running Time

- Sorting edges:

$$O(|E|\log|E|) = O(|E|\log|V|^2) = O(2|E|\log|V|) = O(|E|\log|V|)$$

- Processing edges:

$$2|E| \cdot T(\texttt{Find}) + |V| \cdot T(\texttt{Union}) = O((|E|+|V|)\log|V|) = O(|E|\log|V|)$$

- Total running time: $O(|E|\log|V|)$

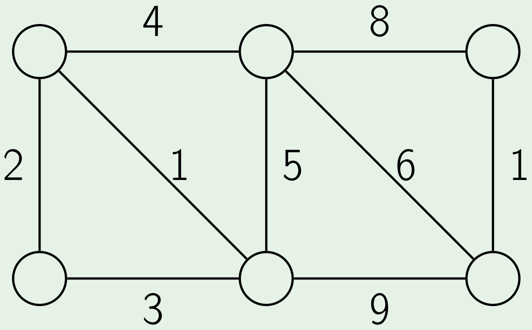
Outline

1. Building a Network
2. Greedy Algorithms
3. Cut Property
4. Kruskal's Algorithm
5. Prim's Algorithm

Prim's Algorithm

- X is always a subtree, grows by one edge at each iteration
- we add a lightest edge between a vertex of the tree and a vertex not in the tree
- very similar to Dijkstra's algorithm

Example



Prim's Algorithm

Prim(G)

```
for all u ∈ V:
    cost[u] ← ∞,  parent[u] ← nil
pick any initial vertex u_0
cost[u_0] ← 0
PrioQ ← MakeQueue(V)      {priority is cost}
while PrioQ is not empty:
    v ← ExtractMin(PrioQ)
    for all {v, z} ∈ E:
        if z ∈ PrioQ and cost[z] > w(v, z):
            cost[z] ← w(v, z),  parent[z] ← v
            ChangePriority(PrioQ, z, cost[z])
```

Running Time

- the running time is

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

Running Time

- the running time is

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

- for array-based implementation, the running time is $O(|V|^2)$

Running Time

- the running time is

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

- for array-based implementation, the running time is $O(|V|^2)$

- for binary heap-based implementation, the running time is
  $O((|V| + |E|) \log |V|) = O(|E| \log |V|)$

Summary

Kruskal: repeatedly add the next lightest edge if this doesn't produce a cycle; use disjoint sets to check whether the current edge joins two vertices from different components

Prim: repeatedly attach a new vertex to the current tree by a lightest edge; use priority queue to quickly find the next lightest edge