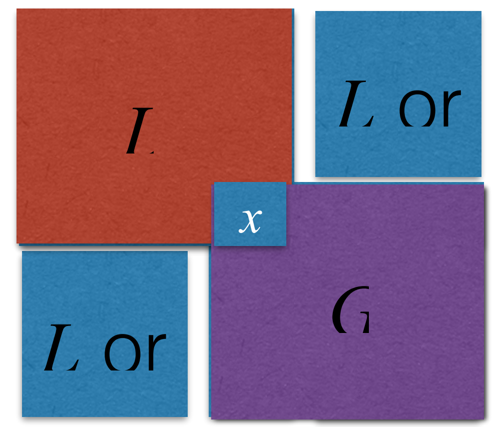
more exactly …

- How many of the $\lceil n/5 \rceil$ medians are smaller than x, how many larger?
- at least $\lceil \frac{n}{10} \rceil - 1$ each
- How many elements are at least in L (G)?
- $3\frac{n}{10}$
- Therefore we recurse—after split—on at most $7\frac{n}{10}$ elements



Running time of LinearSelect

$$T(n) \leq an + T(\frac{n}{5}) + T(\frac{7n}{10})$$ for not small n

$T(n)$ is constant for small n

We show that $T(n) \leq cn$ and therefore $T(n) \in O(n)$ for constant c, using induction.

Running time of LinearSelect

$$T(n) \leq an + T(\frac{n}{5}) + T(\frac{7n}{10})$$

$$= an + c\frac{n}{5} + 7c\frac{n}{10}$$

$$= an + 9c\frac{n}{10}$$

Running time of LinearSelect

- We can show that $an + 9c\frac{n}{10} \leq cn$ for $c \geq 10a$
- Therefore LinearSelect has indeed a linear time complexity!

Remarks

- When picking the pivot, instead of dividing into short sequences of size 5, one can also choose short sequences of, e.g., size 7
    - size 3 does not work! (Assignment question)
- LinearSelect is also called Linear Median Algorithm
- In practice, QuickSelect is typically the better choice
- LinearSelect requires huge inputs to show linear running time in practice

Graphs

- Minimum spanning trees
- Shortest Paths

Minimum Spanning Tree Definition

- *Input*: A weighted connected graph $G = (V, E)$ consisting of vertices (or nodes), V, and edges, E, with positive integer edge weights

- *Output*: A *minimum spanning tree* (*MST*) $T = (V, E_T)$, that is T is a connected subgraph of G ($E_T \subseteq E$) such that T is acyclic, and T is shortest

Definition

- Let $G = (V,E)$ be an undirected connected graph with edge weights that are *positive* integers
- T = (V, E_T) is a minimum spanning tree for G if

  (1) T is a subgraph of G

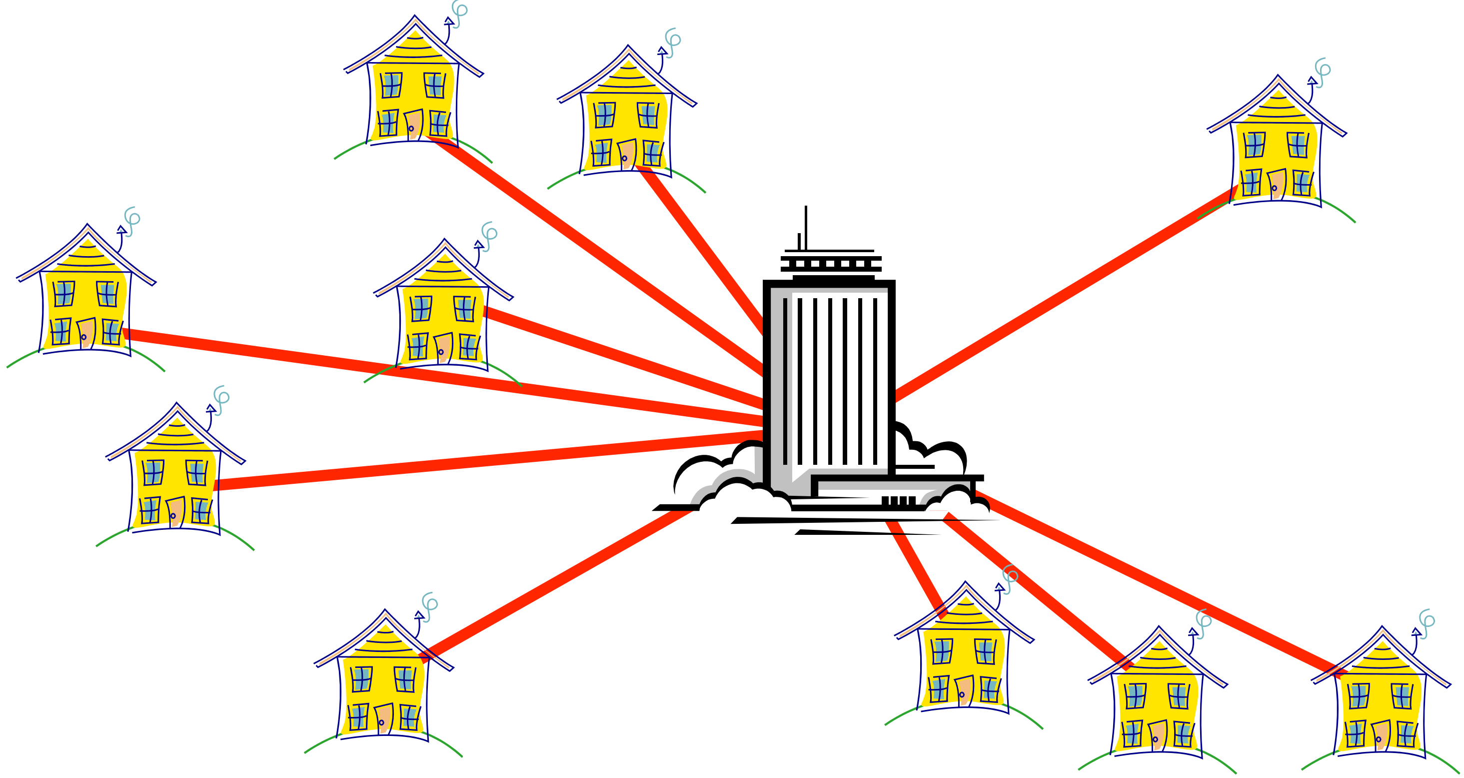
  (2) T is a tree

  (3) T is the *lightest* graph satisfying (1) and (2),

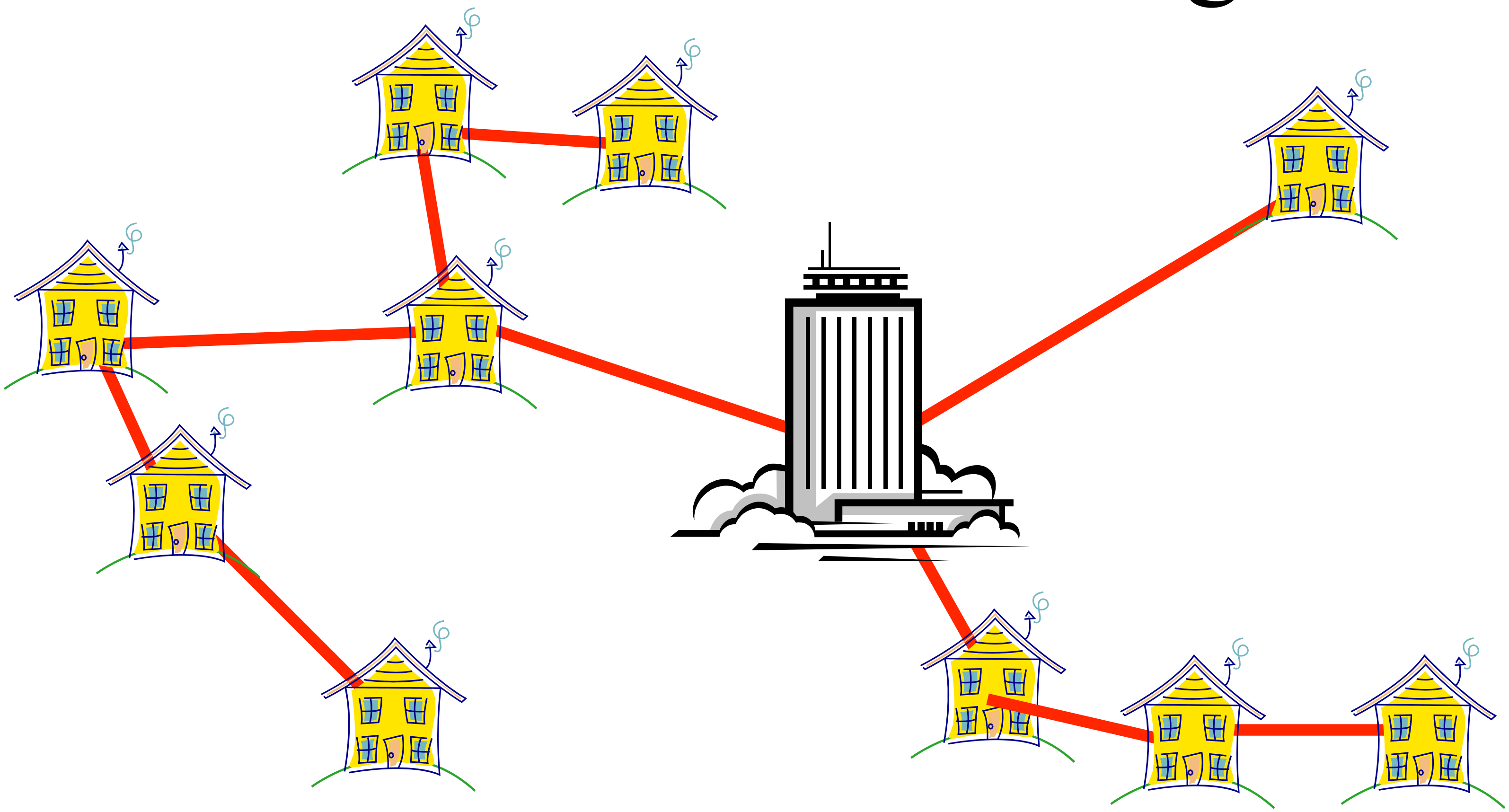
  i.e. ,
$$\sum_{e \in E_T} w(e) = \min_{\substack{T' \text{ is} \\ \text{spanning} \\ \text{tree for } G}} \sum_{e \in E_{T'}} w(e)$$

$\left.\begin{array}{l}(1)\\(2)\end{array}\right\}$ spanning tree

Problem: Laying Cable TV Wire



Minimize Wiring



Applications of Minimum Spanning Trees

- used in image processing (e.g., cancer research)
- clustering
- identifications of patterns in gene expressions
- routing in mobile networks

Determining MST by Brute Force

- Create all spanning trees
- Pick the lightest
- Not feasible!!
  - A complete graph (every pair of vertices is connected by an edge) has $|V|^{|V|-2}$ many spanning trees (Cayley's Theorem [1889])

Minimum Spanning Tree algorithms

- 1926 Barůvka $O(m \log n)$
- 1930 Prim-Jarník's
  - 1930 Jarník
  - 1957 Dijkstra
  - 1959 Prim
  - 1964 with Heaps $O(m \log n)$
  - 1987 Fredman and Tarjan with Fibonacci Heaps $O(m+n \log n)$
- 1956 Kruskal's algorithm
  - 1956 Kruskal
  - 1974 Aho, Hopcroft and Ullman with Union-Find Disjoint Set $O(m \log n)$
- 1975 Yao $O(m \log\log n)$
- 1976 Cheriton and Tarjan $O(m \log\log n)$
- 1995 Karger, Klein and Tarjan Randomized MST based on Barůvka and Kruskal $O(m)$
- 2000 Chazelle $O(m\ \alpha(m,n))$

n: number of vertices
m: number of edges

Prim's Algorithm
Idea

- Initialize tree with single chosen vertex
- Grow tree by finding lightest edge not yet in tree and connect it to tree; repeat until all vertices are in the tree
- *Example of greedy algorithm*

Reminder

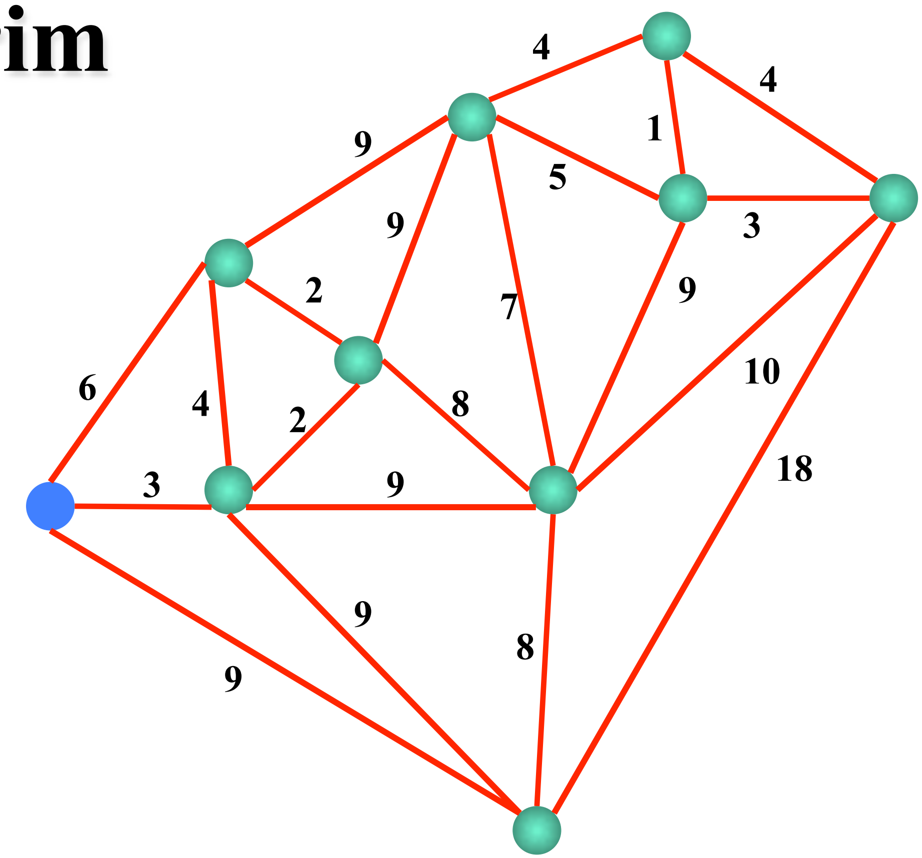
Greedy Algorithm Design Technique

- Applied to optimization problems
  - an objective function is *minimized* or *maximized*
- Characterized by the *greedy-choice property:*
  - a global optimal configuration can be reached by a series of locally optimal choices
  - starting from a well-defined configuration, optimal choices are choices that are best from among the possibilities available at the time

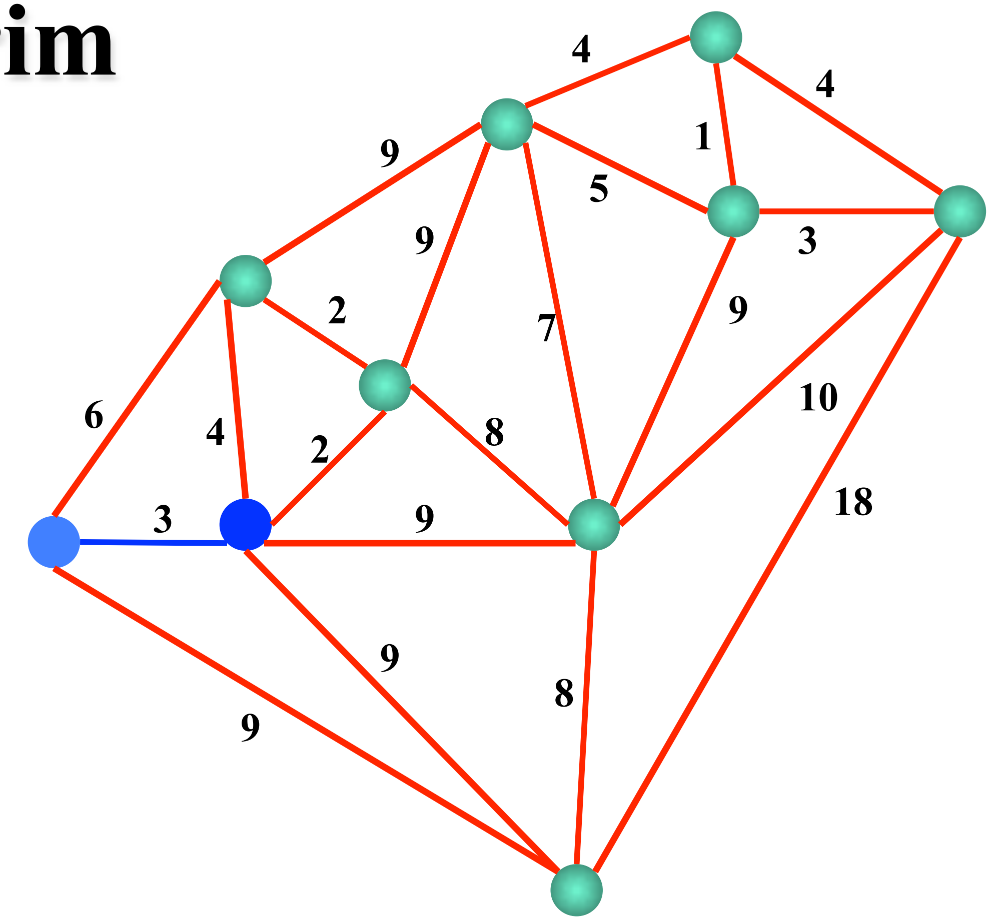
Prim’s Algorithm Idea

- Initialize tree with single chosen vertex
- Grow tree by finding lightest edge not yet in tree and connect it to tree; repeat until all vertices are in the tree
- *Example of greedy algorithm*

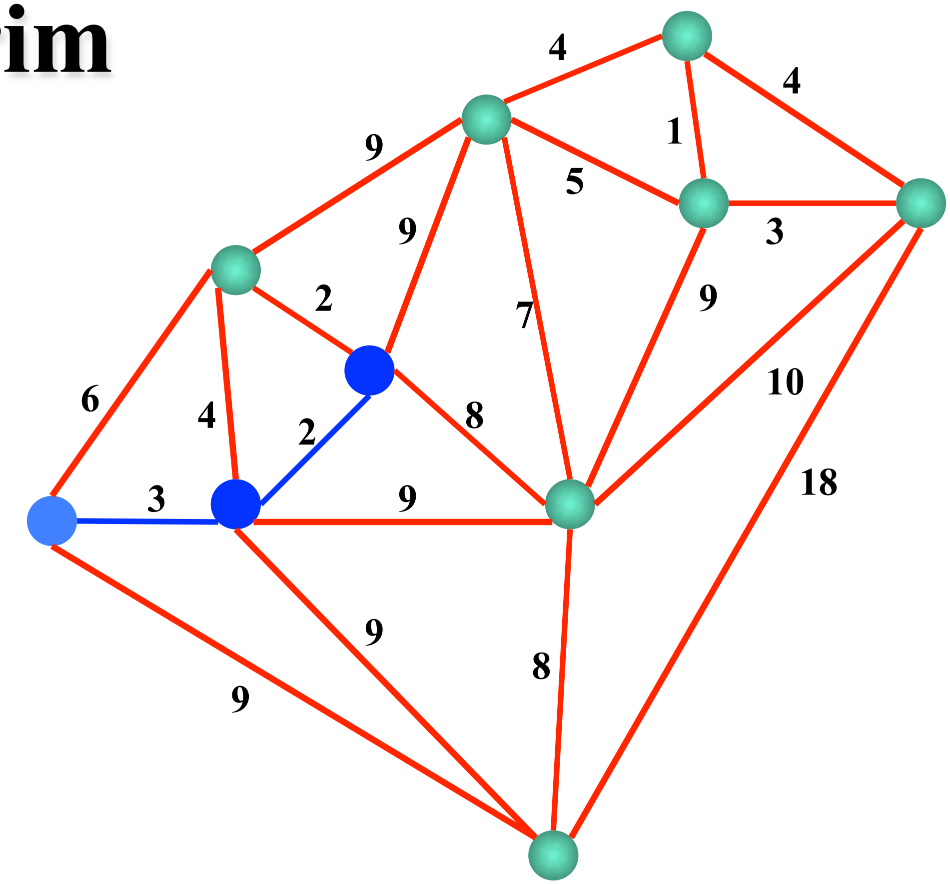
Prim



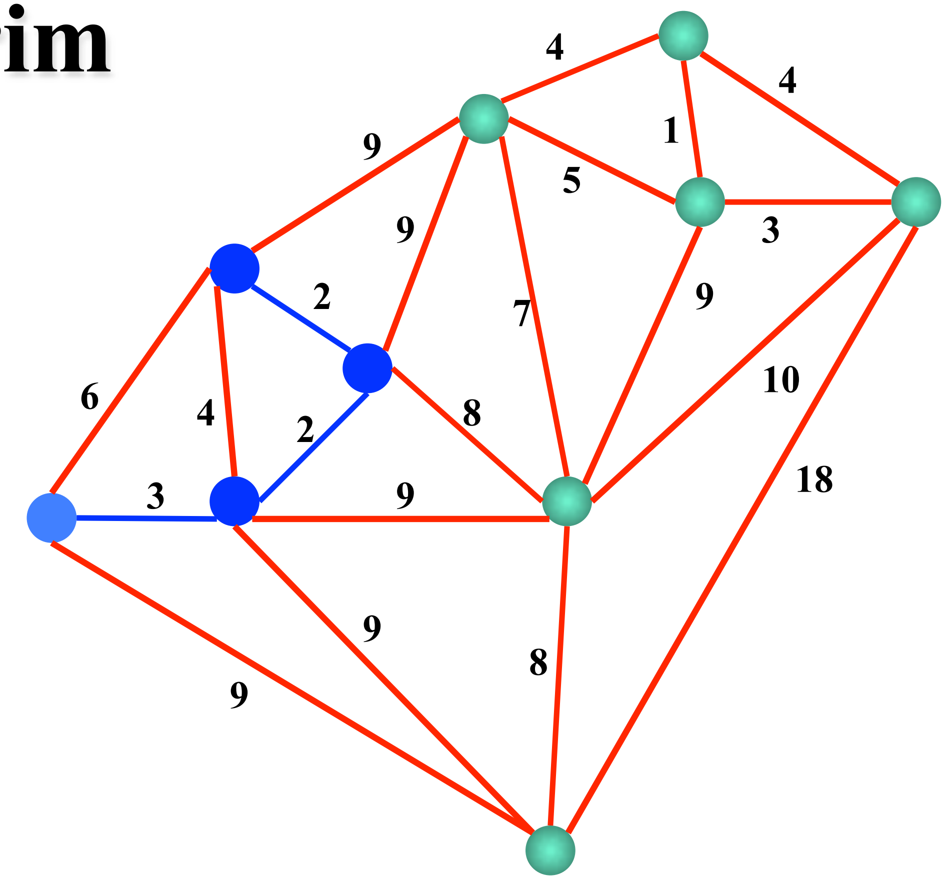
Prim



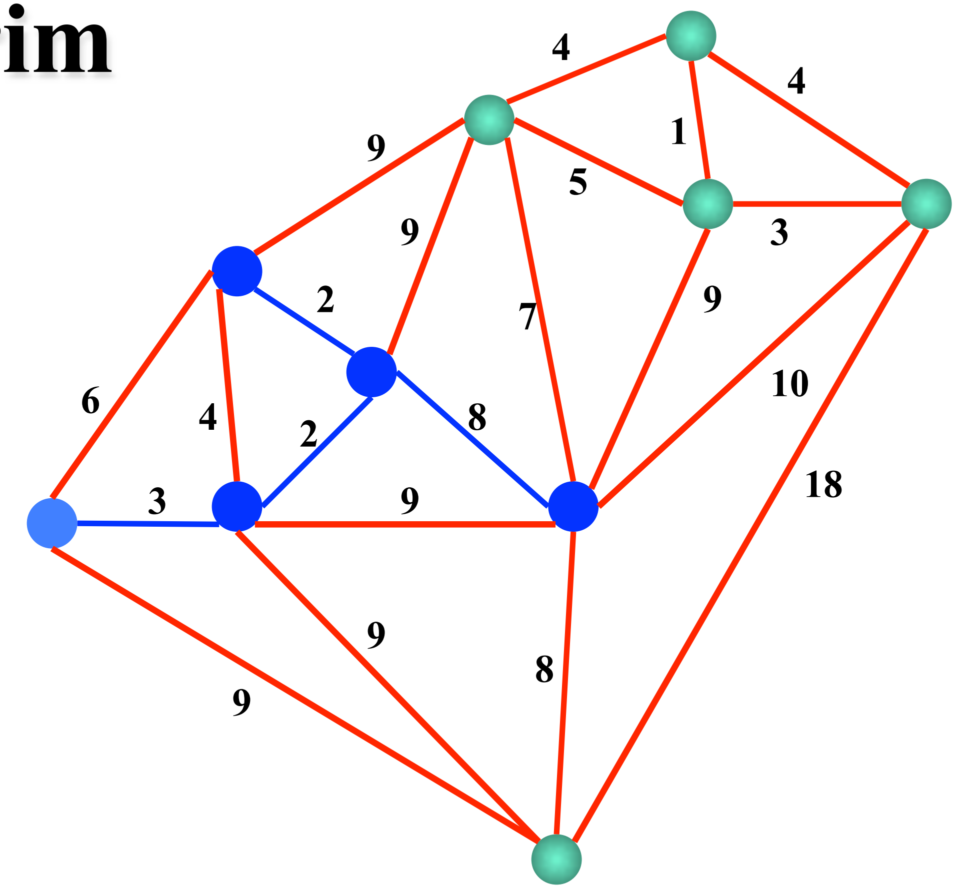
Prim



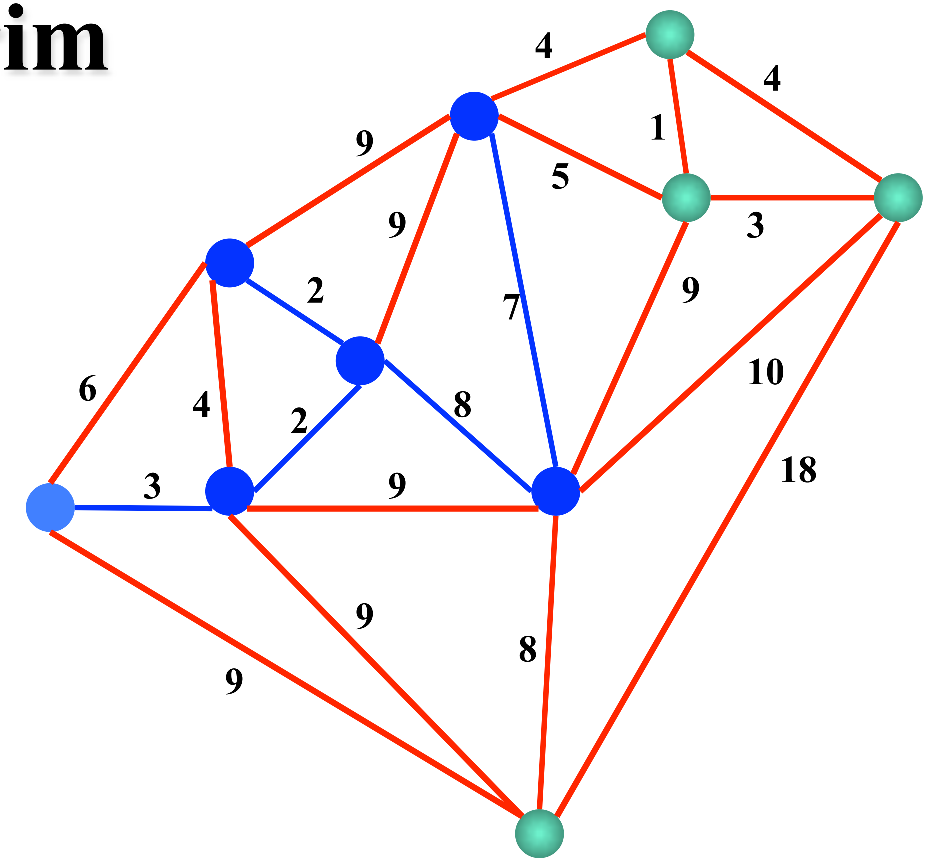
Prim



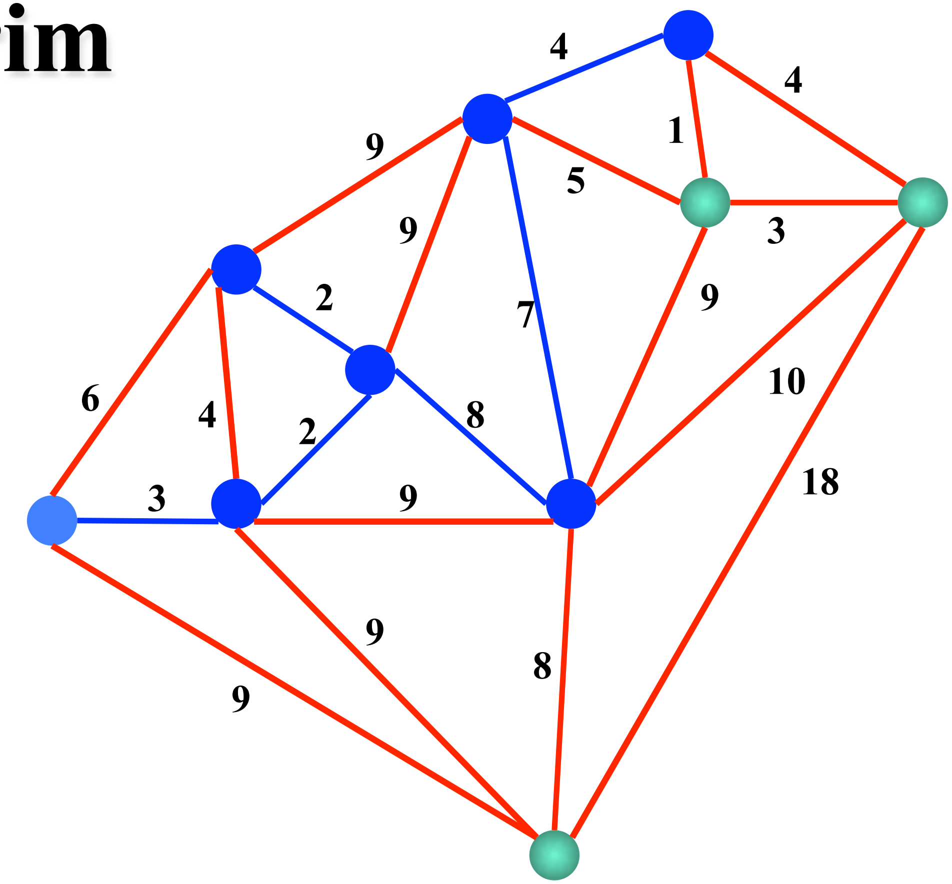
Prim



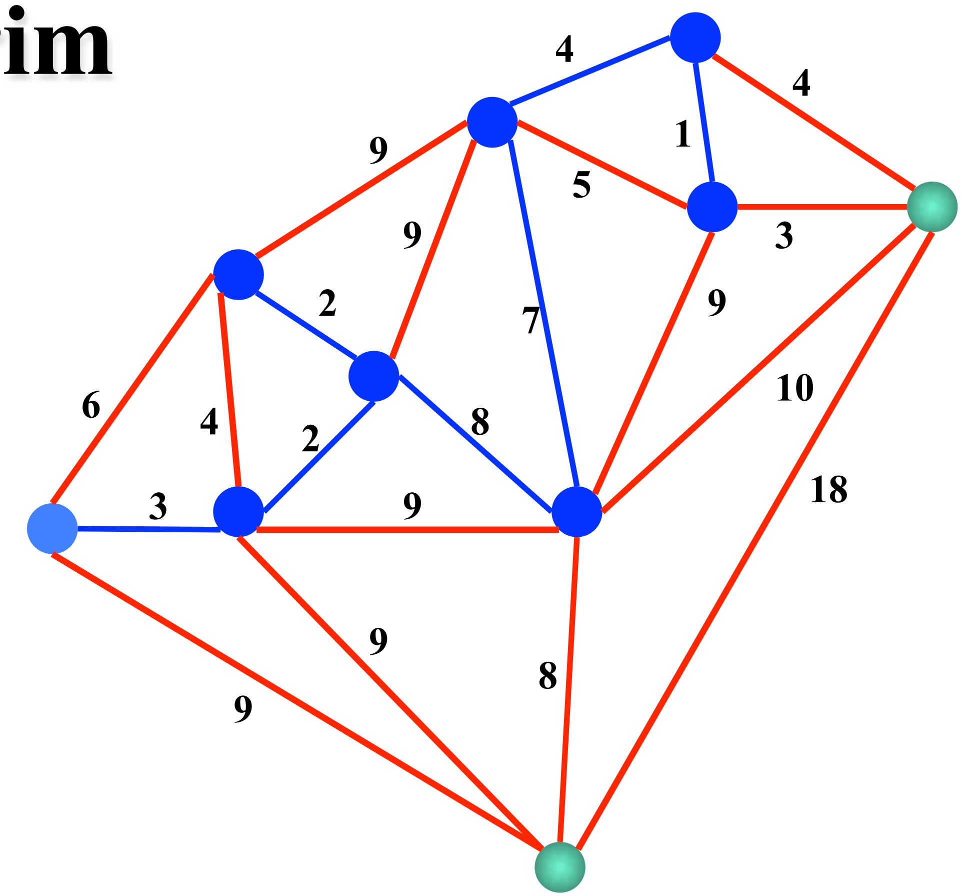
Prim



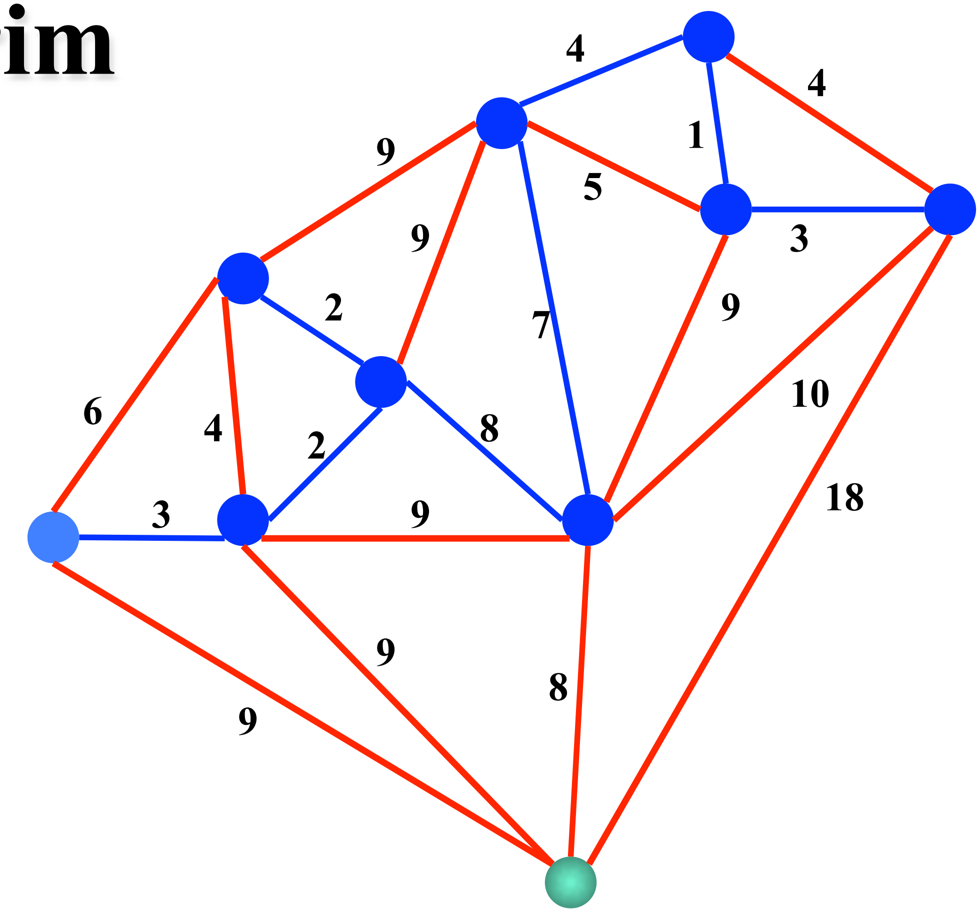
Prim



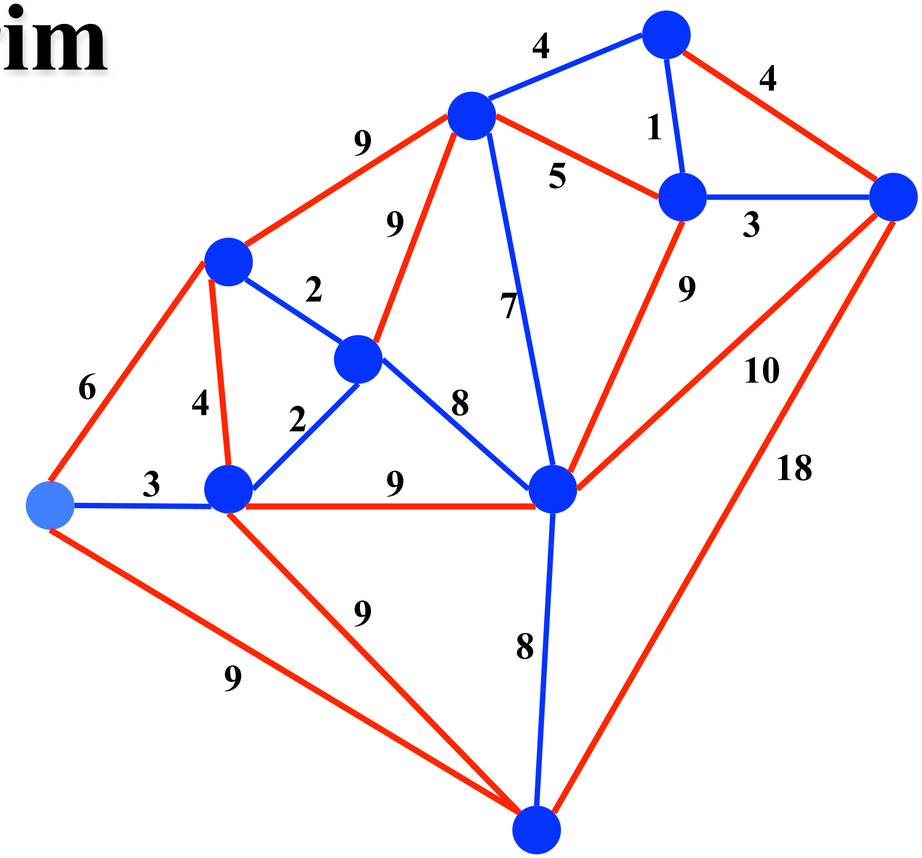
Prim



Prim



Prim



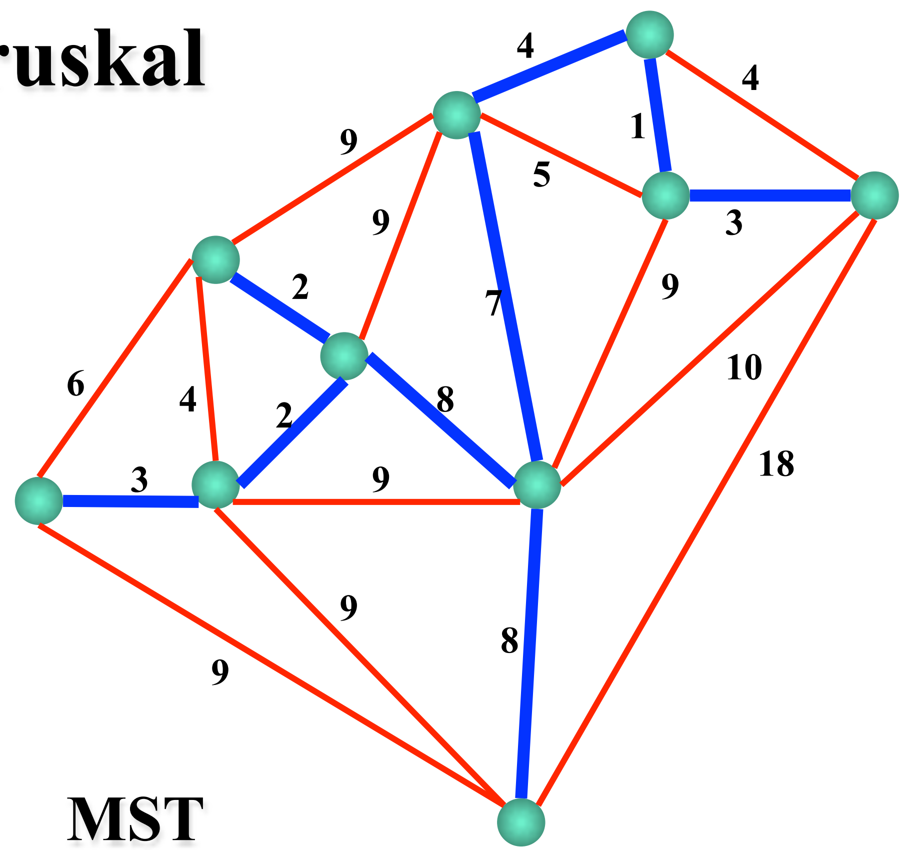
Prim's Algorithm
Idea

- Initialize tree with single chosen vertex
- Grow tree by finding lightest edge not yet in tree and connect it to tree; repeat until all vertices are in the tree
- *Example of greedy algorithm*

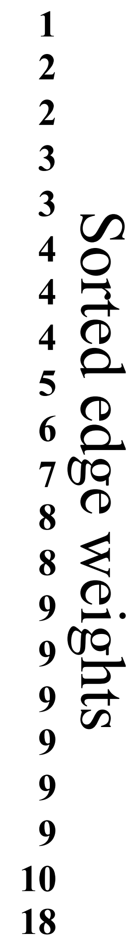
Kruskal's Algorithm Idea

- Initialize a forest consisting of all nodes
- Pick a (non-selected) minimum weight edge and, if it connects two different trees of the forest, select it, otherwise discard it; repeat
- *Example of greedy algorithm*

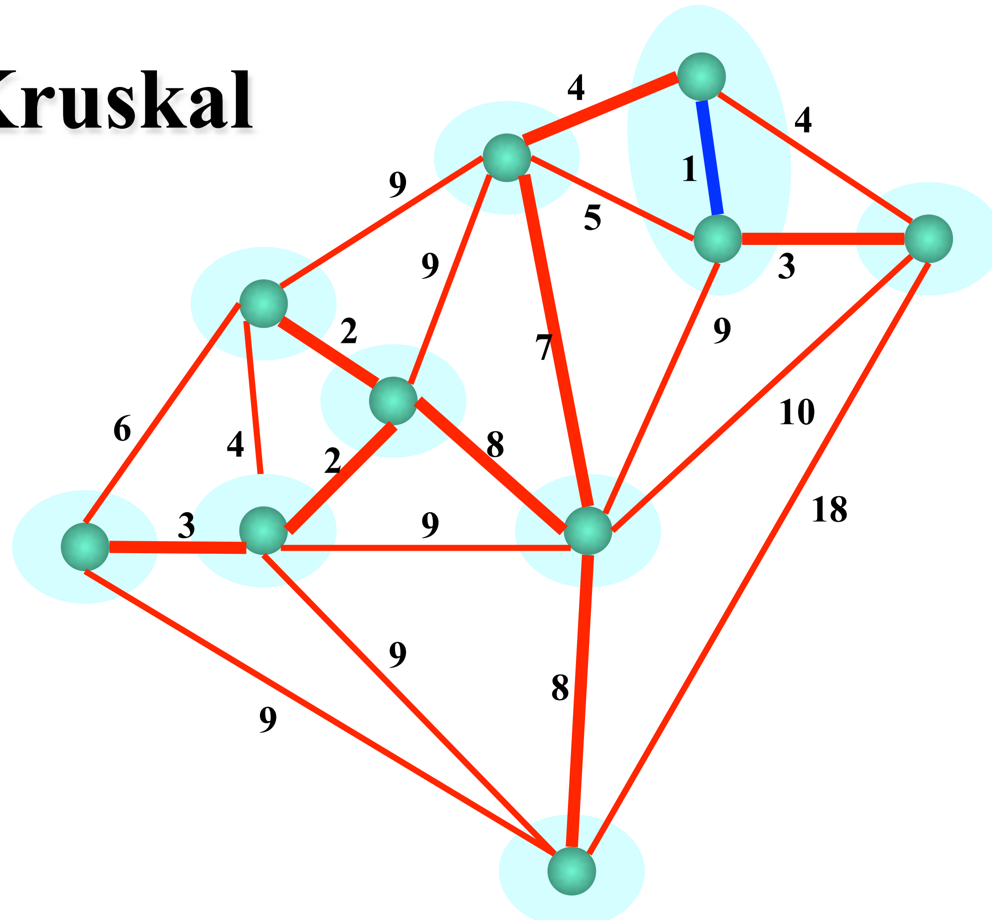
Kruskal


4
4
1
9
5
3
9
2
7
9
6
4
2
8
10
18
3
9
9
8
9

MST


1
2
2
3
3
4
4
4
5
6
7
8
8
9
9
9
9
9
9
10
18
Sorted edge weights

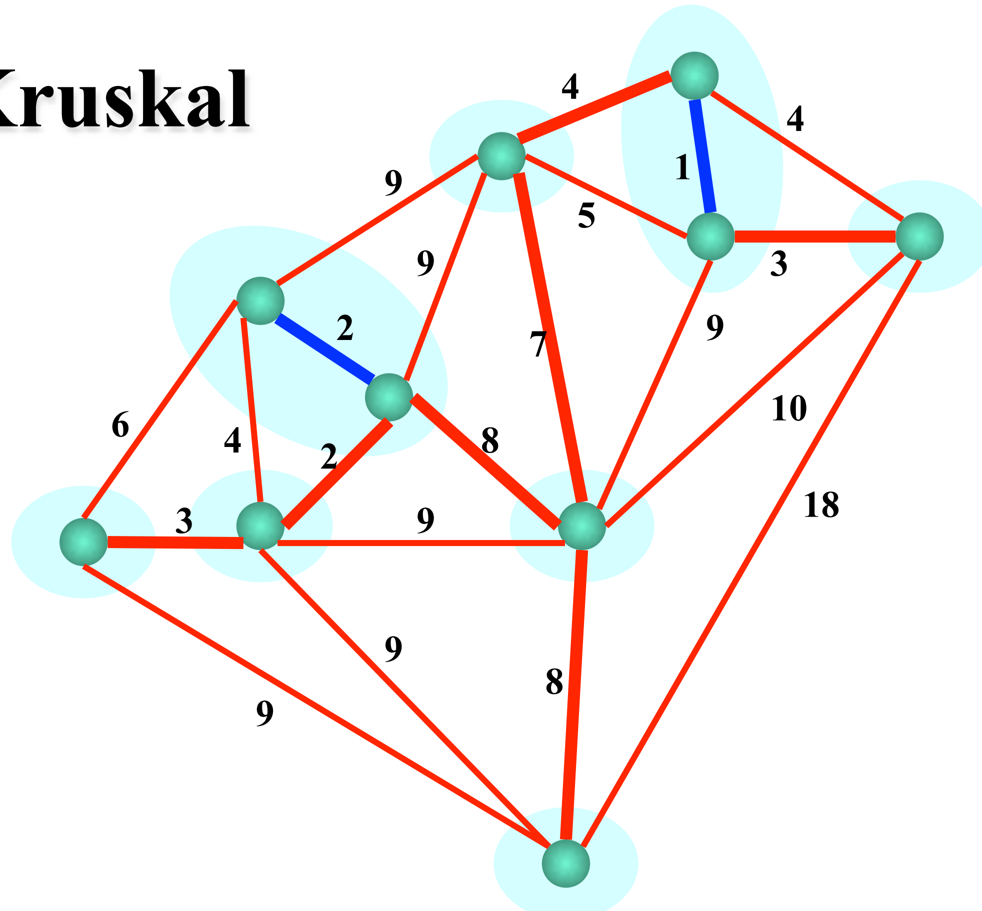
Kruskal



Sorted edge weights

1
2
2
3
3
4
4
4
5
6
7
8
8
9
9
9
9
9
9
10
18

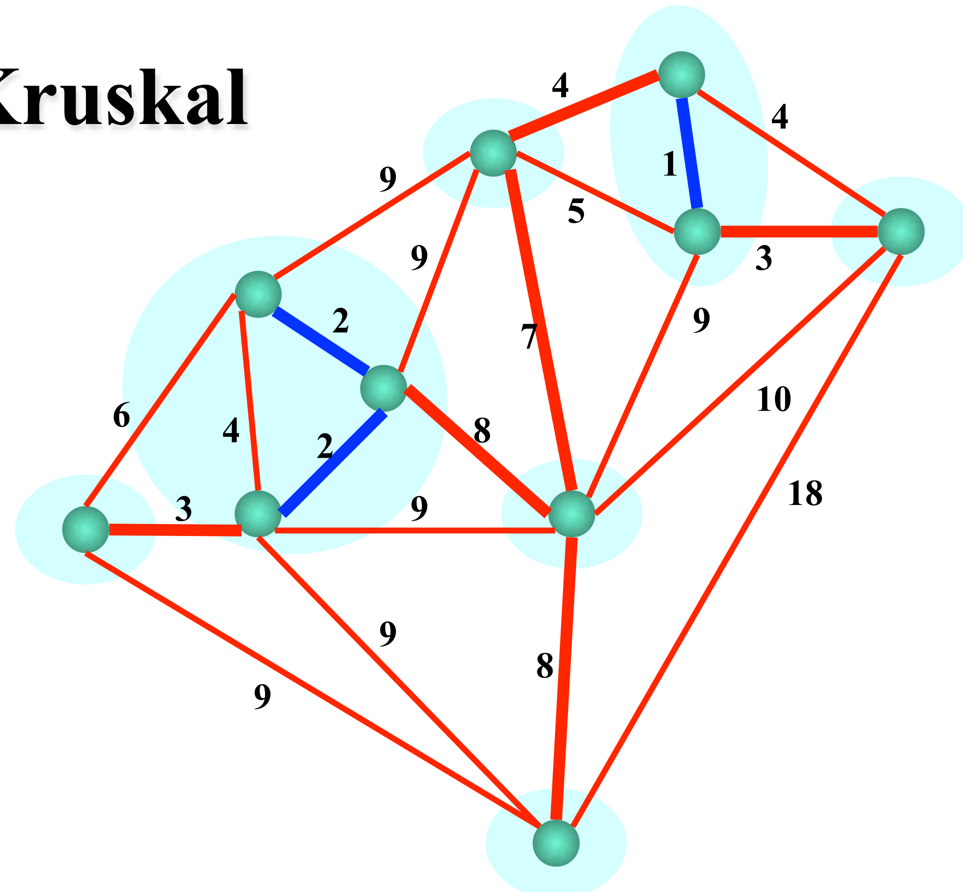
Kruskal



1
2
2
3
3
4
4
4
5
6
7
8
8
9
9
9
9
9
9
10
18

Sorted edge weights

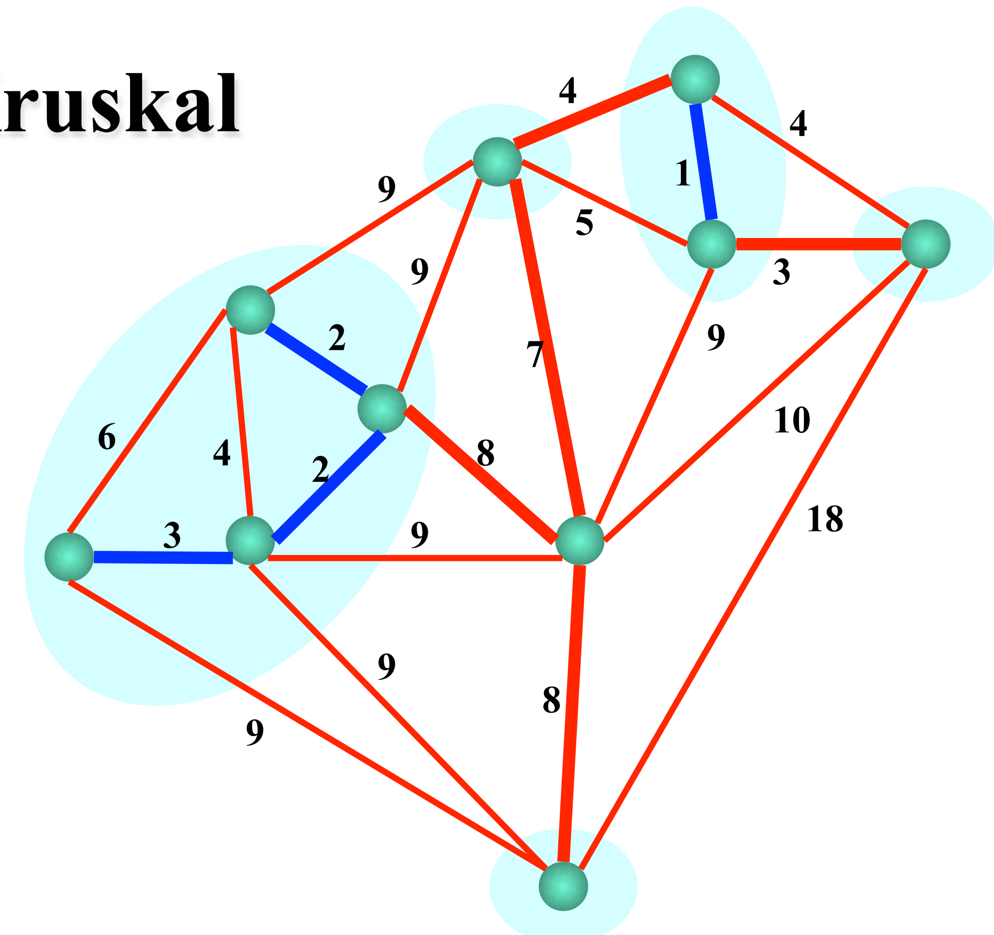
Kruskal



Sorted edge weights

1
2
2
3
3
4
4
4
5
6
7
8
8
9
9
9
9
9
9
10
18

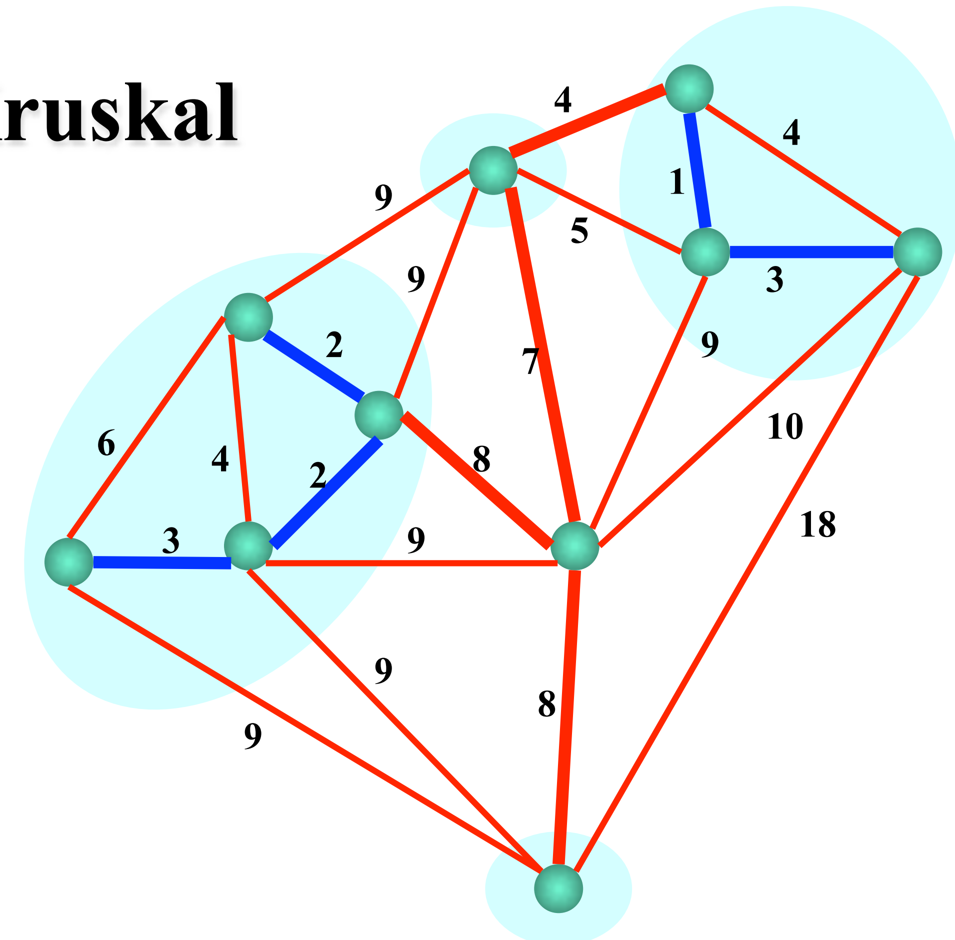
Kruskal



Sorted edge weights

1
2
2
3
3
4
4
4
5
6
7
8
8
9
9
9
9
9
9
10
18

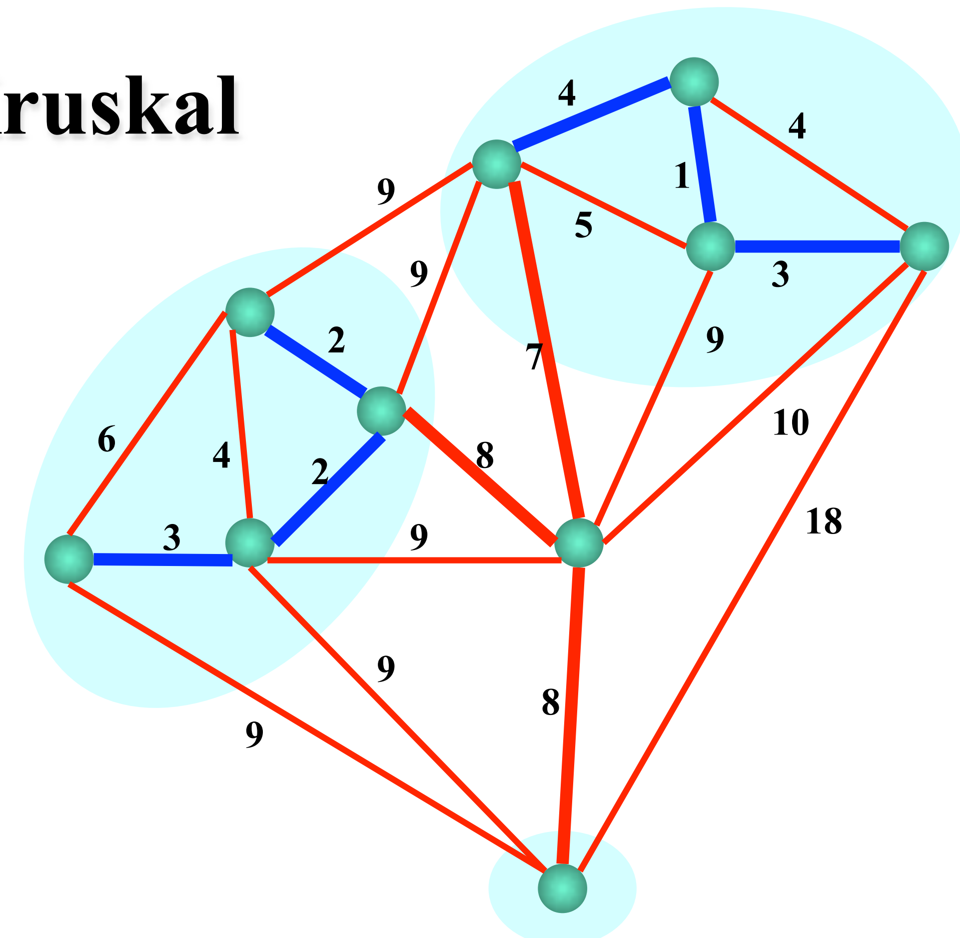
Kruskal



1
2
2
3
3
4
4
4
5
6
7
8
8
9
9
9
9
9
9
10
18

Sorted edge weights

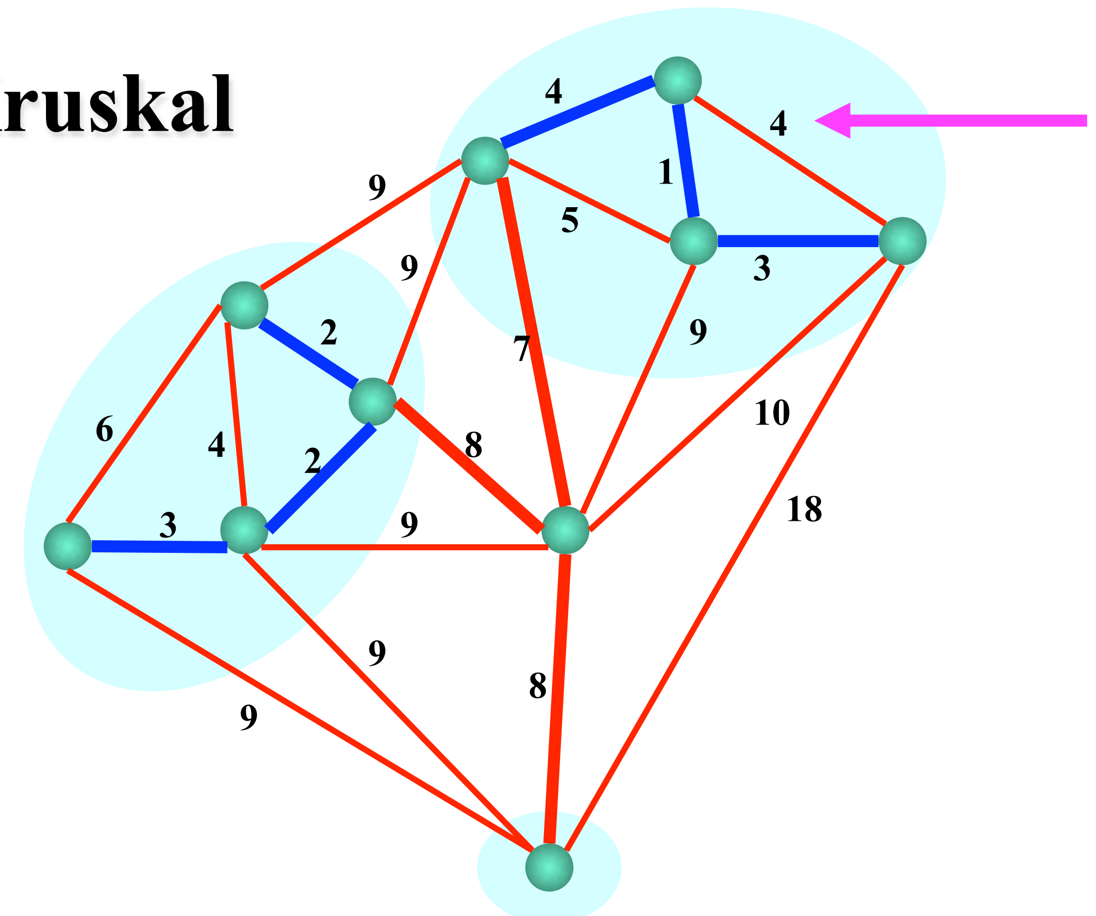
Kruskal



1
2
2
3
3
4
4
4
5
6
7
8
8
9
9
9
9
9
9
10
18

Sorted edge weights

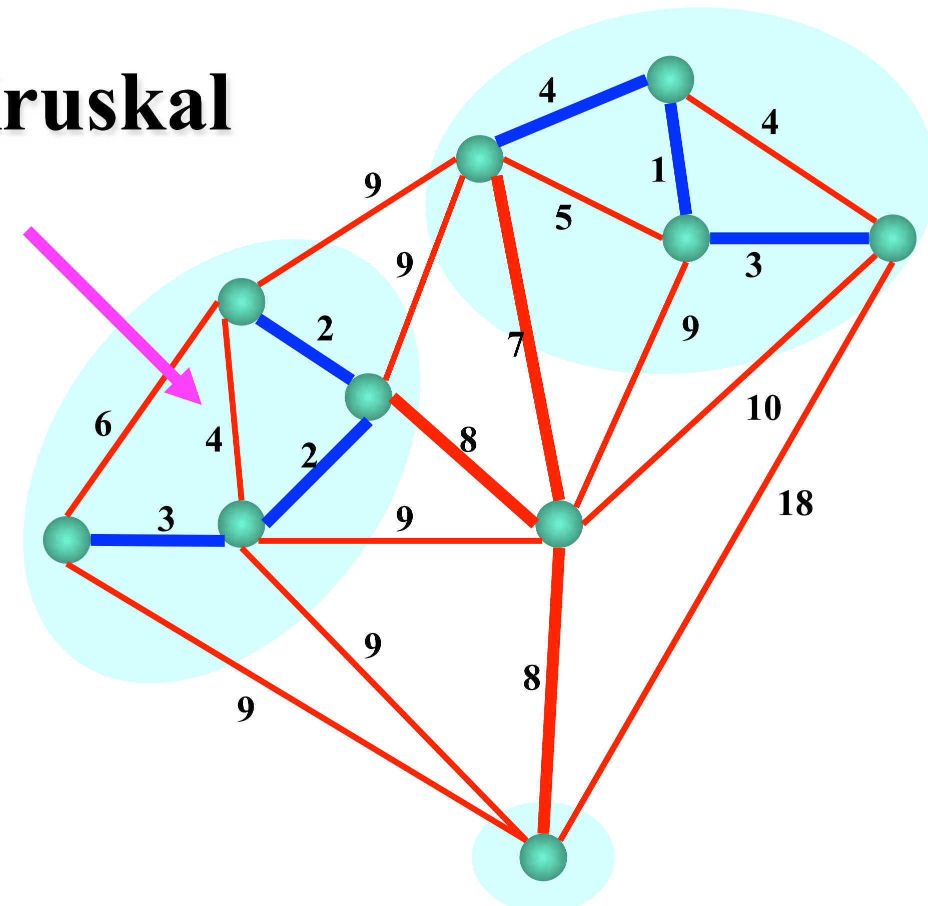
Kruskal



Sorted edge weights

1
2
2
3
3
4
4
4
5
6
7
8
8
9
9
9
9
9
9
10
18

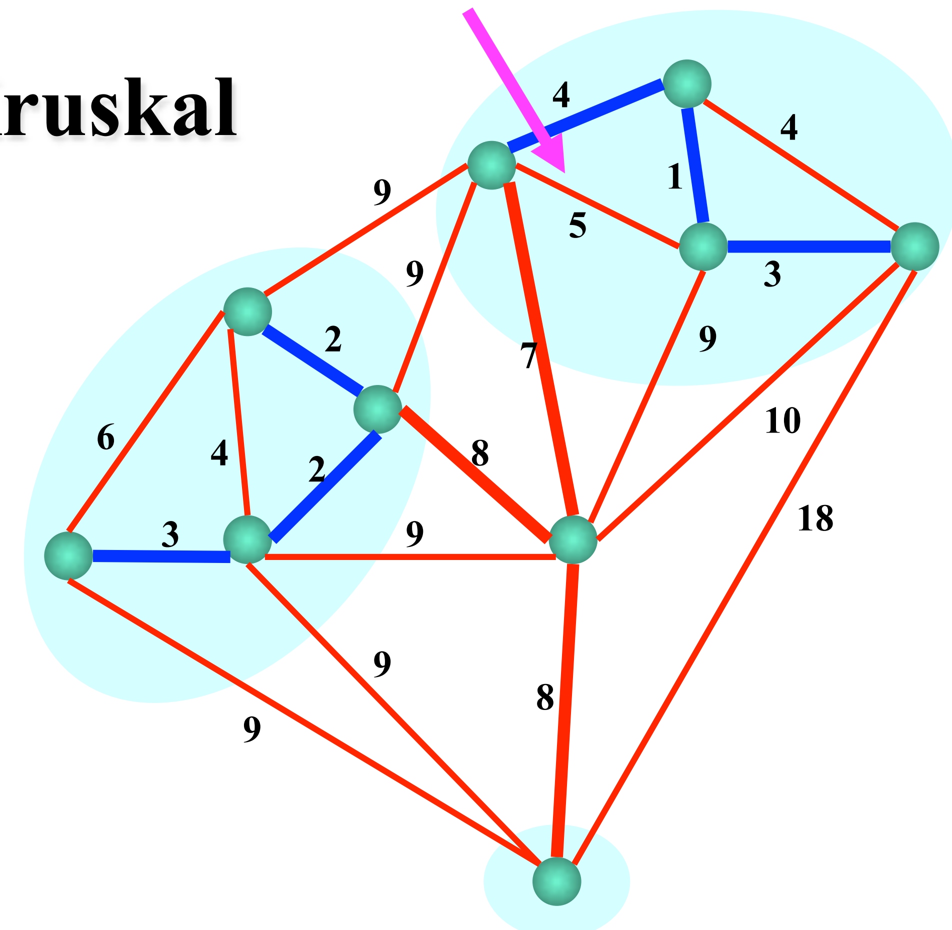
Kruskal



Sorted edge weights

1
2
2
3
3
4
4
4
5
6
7
8
8
9
9
9
9
9
9
10
18

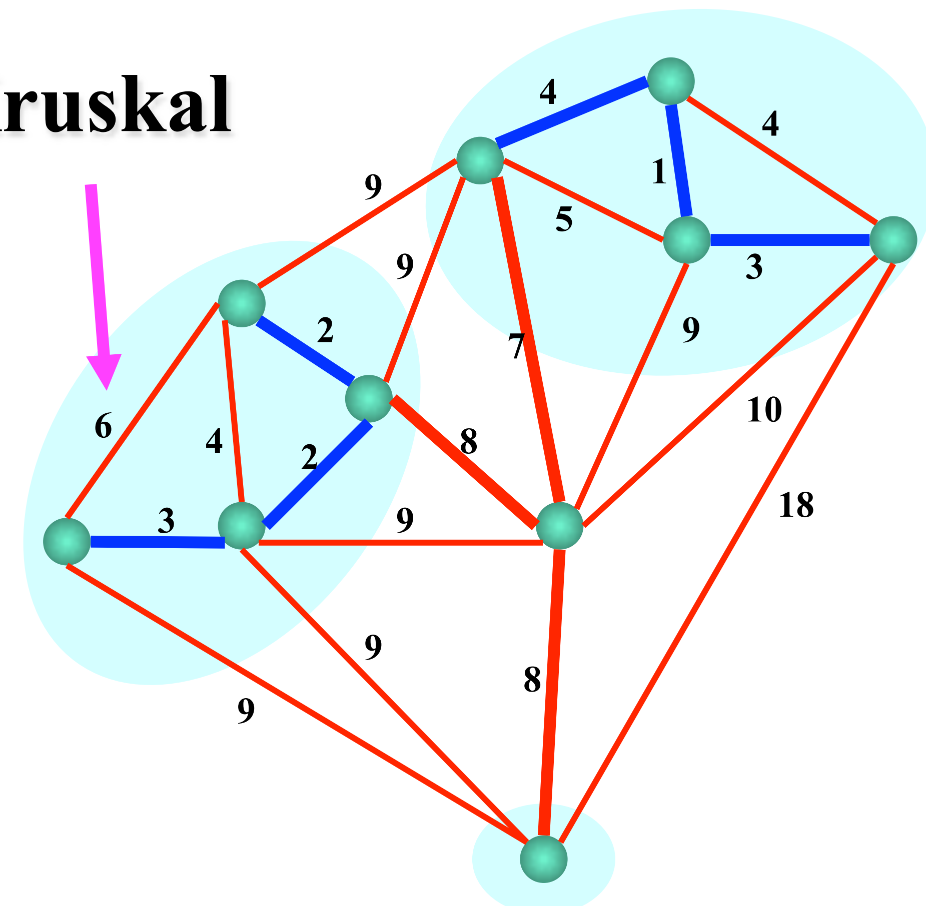
Kruskal



1
2
2
3
3
4
4
4
5
6
7
8
8
9
9
9
9
9
9
10
18

Sorted edge weights

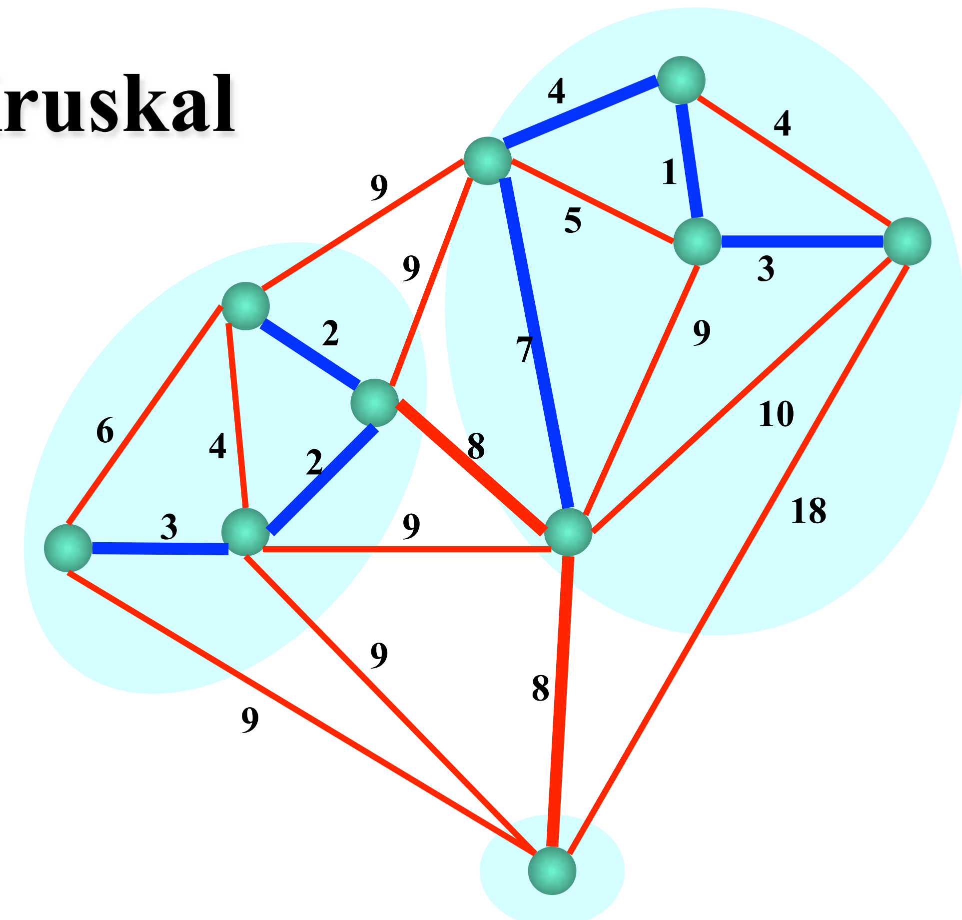
Kruskal



Sorted edge weights

1
2
2
3
3
4
4
4
5
6
7
8
8
9
9
9
9
9
9
10
18

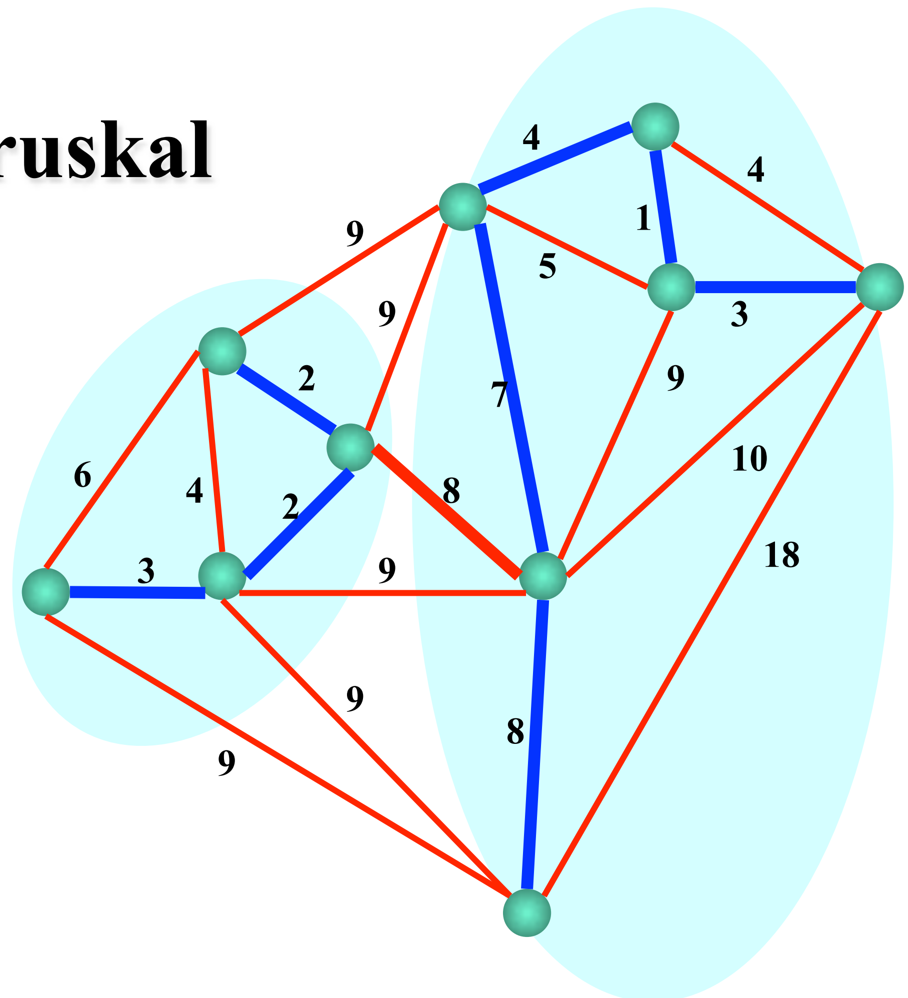
Kruskal



Sorted edge weights

1
2
2
3
3
4
4
4
5
6
7
8
8
9
9
9
9
9
9
10
18

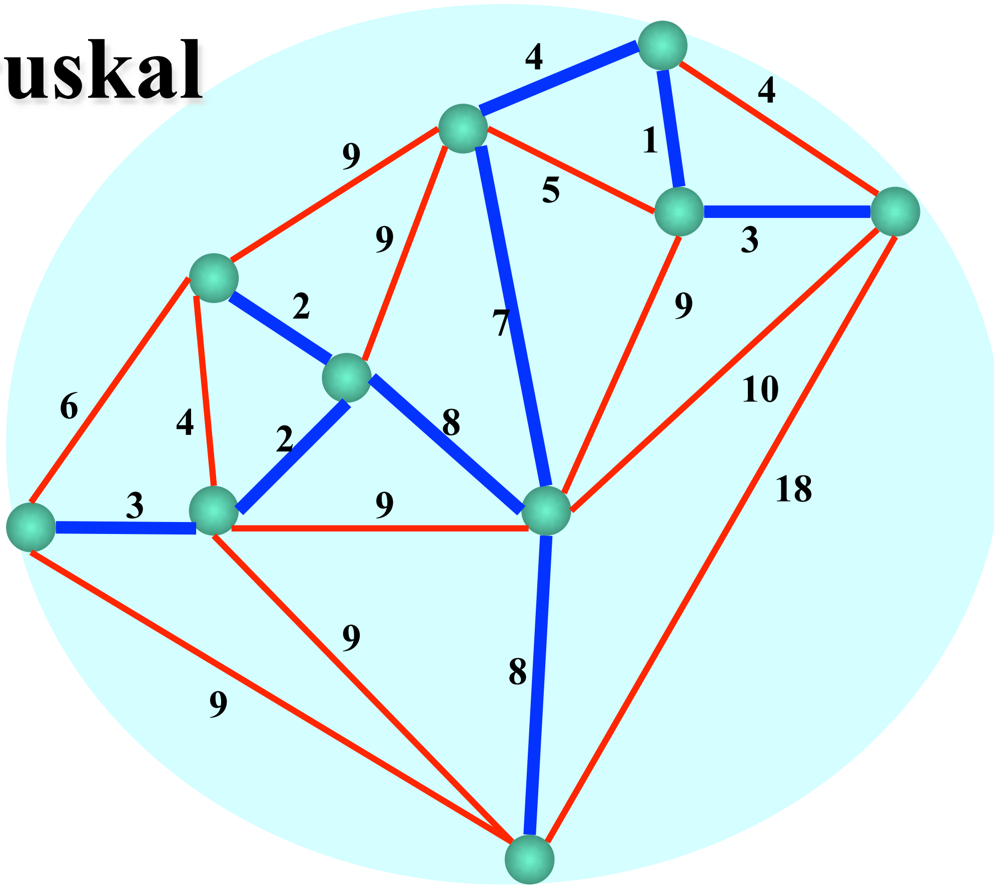
Kruskal



1
2
2
3
3
4
4
4
5
6
7
8
8
9
9
9
9
9
9
10
18

Sorted edge weights

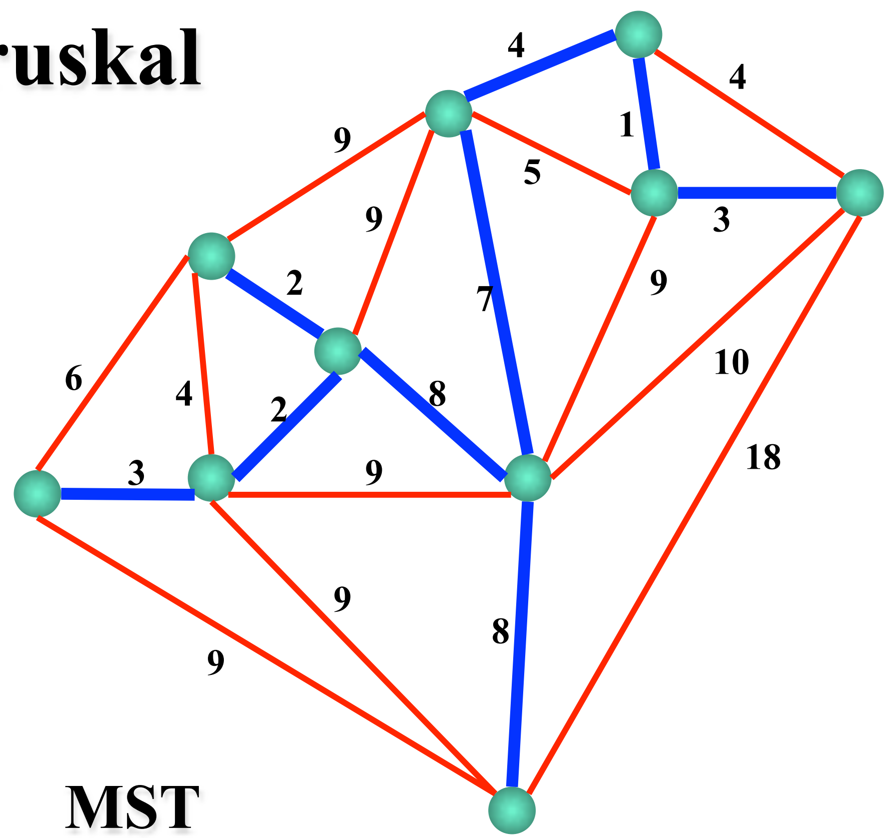
Kruskal



Sorted edge weights

1
2
2
3
3
4
4
4
5
6
7
8
8
9
9
9
9
9
9
10
18

Kruskal



MST

1
2
2
3
3
4
4
4
5
6
7
8
8
9
9
9
9
9
9
10
18

Sorted edge weights

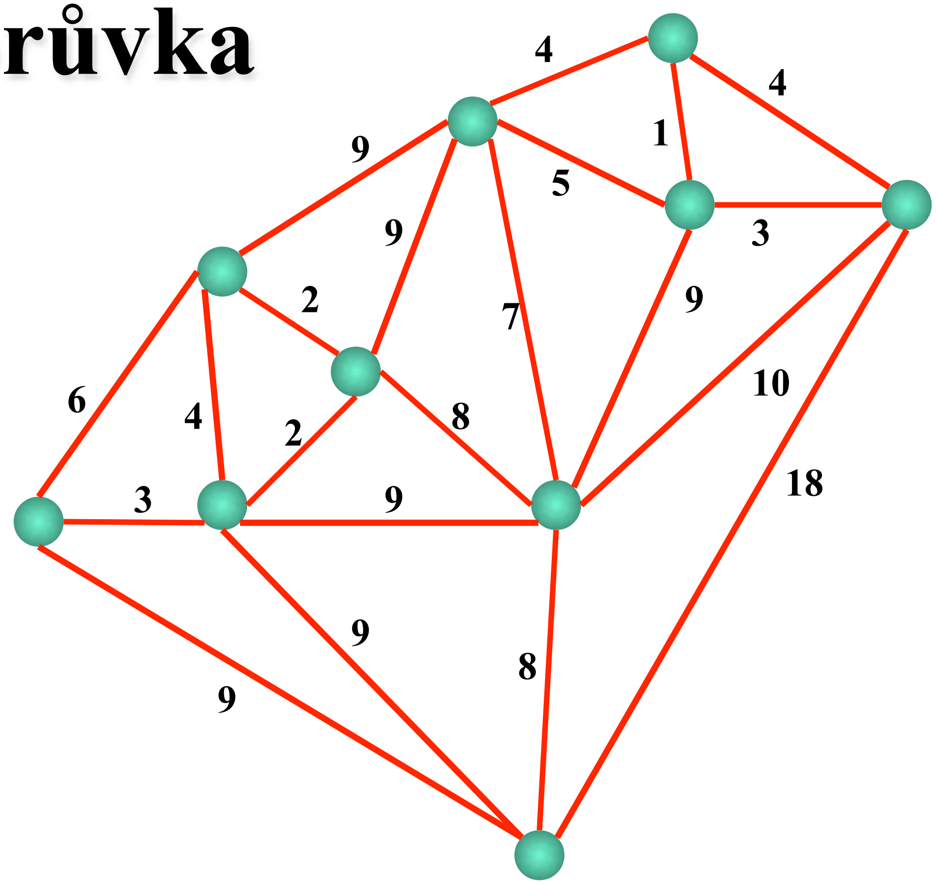
Kruskal's Algorithm Idea

- Initialize a forest consisting of all nodes
- Pick a (non-selected) minimum weight edge and, if it connects two different trees of the forest, select it, otherwise discard it; repeat
- *Example of greedy algorithm*

Borůvka's Algorithm Idea

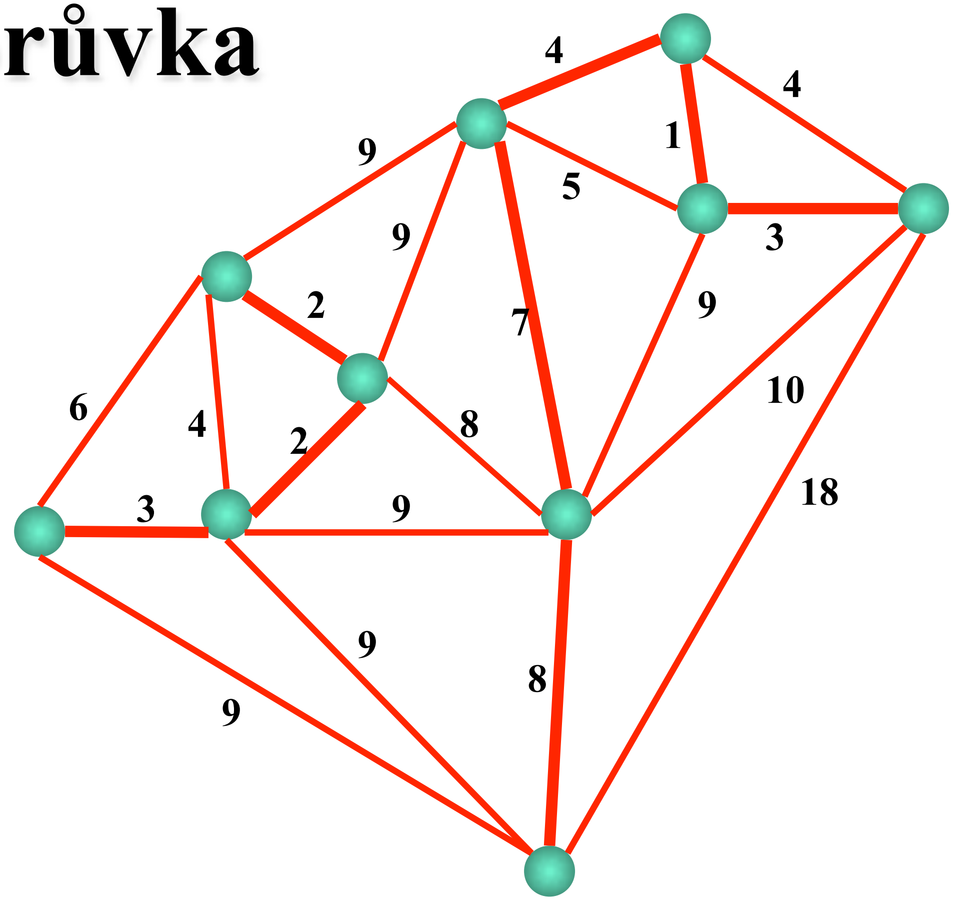
- Assume every edge has a unique weight.
- Initially, each vertex is considered a separate component.
- The algorithm merges disjoint components as follows; repeating the step until only one component exists.
- In each step, every component is merged with some other using the cheapest outgoing edge of the given component.

Borůvka

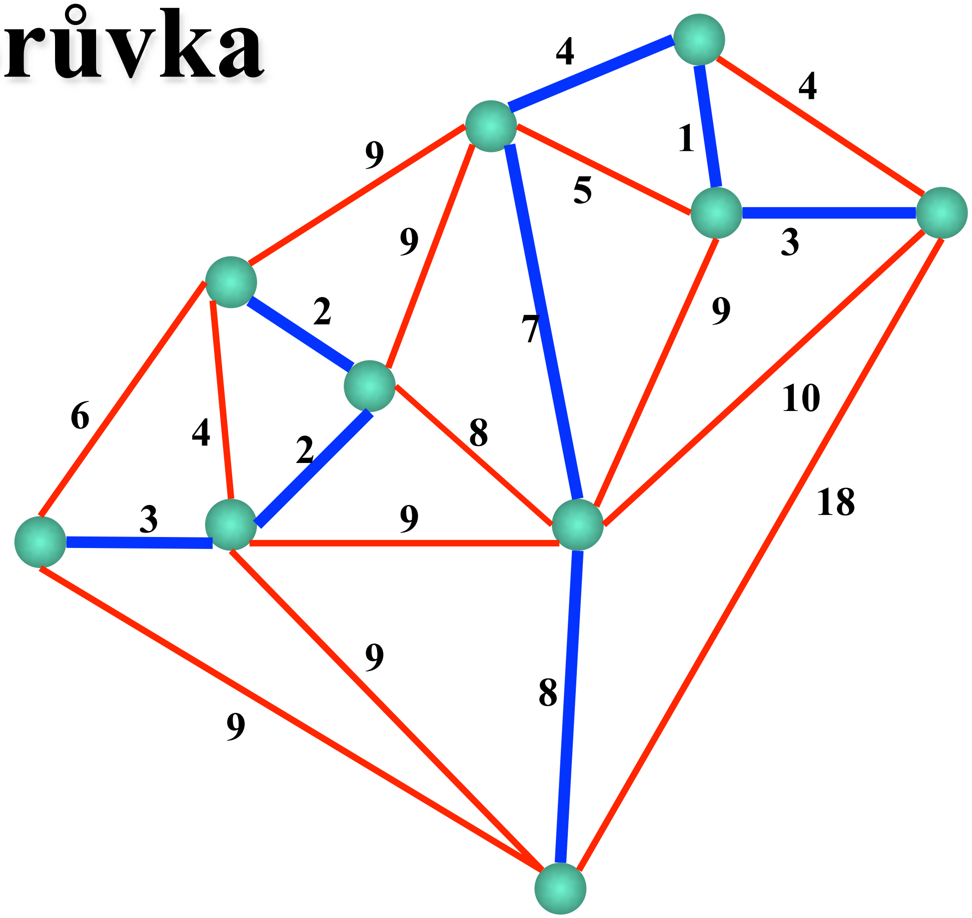


Every vertex is a tree

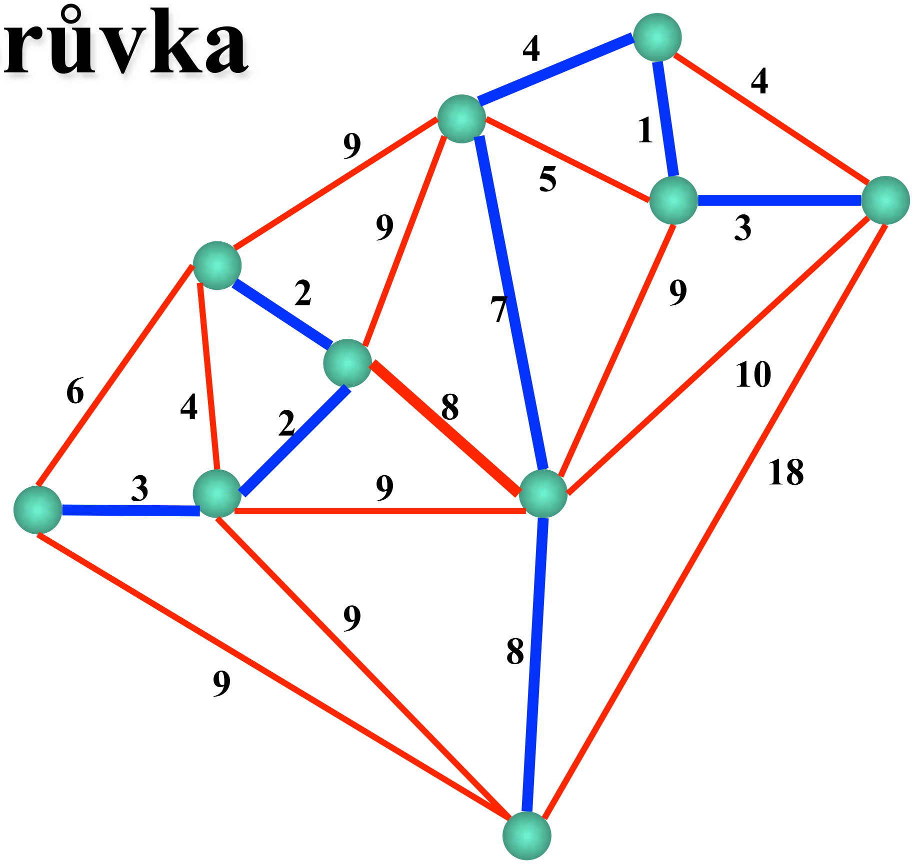
Borůvka



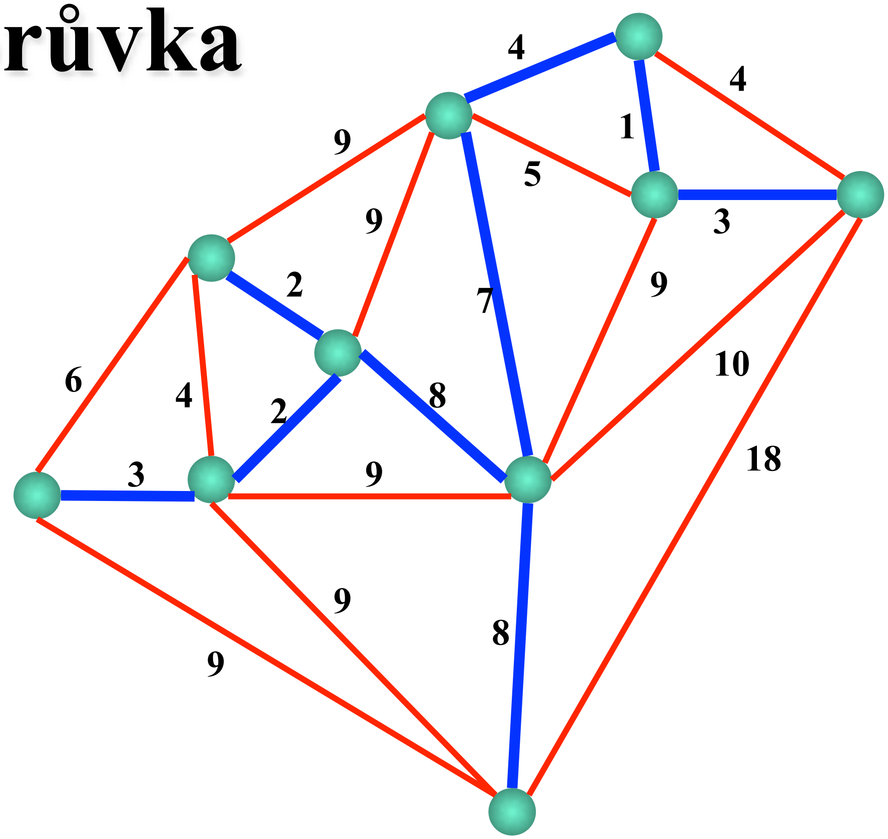
Borůvka



Borůvka



Borůvka



Borůvka's Algorithm Idea

- Assume every edge has a unique weight.
- Initially, each vertex is considered a separate component.
- The algorithm merges disjoint components as follows; repeating the step until only one component exists.
- In each step, every component is merged with some other using the cheapest outgoing edge of the given component.

To come

- Today: why do these algorithm ideas work (and produce correct MSTs)?
- Later: how do we implement these algorithms efficiently? What are good data structures?

Two basic properties for minimum spanning trees

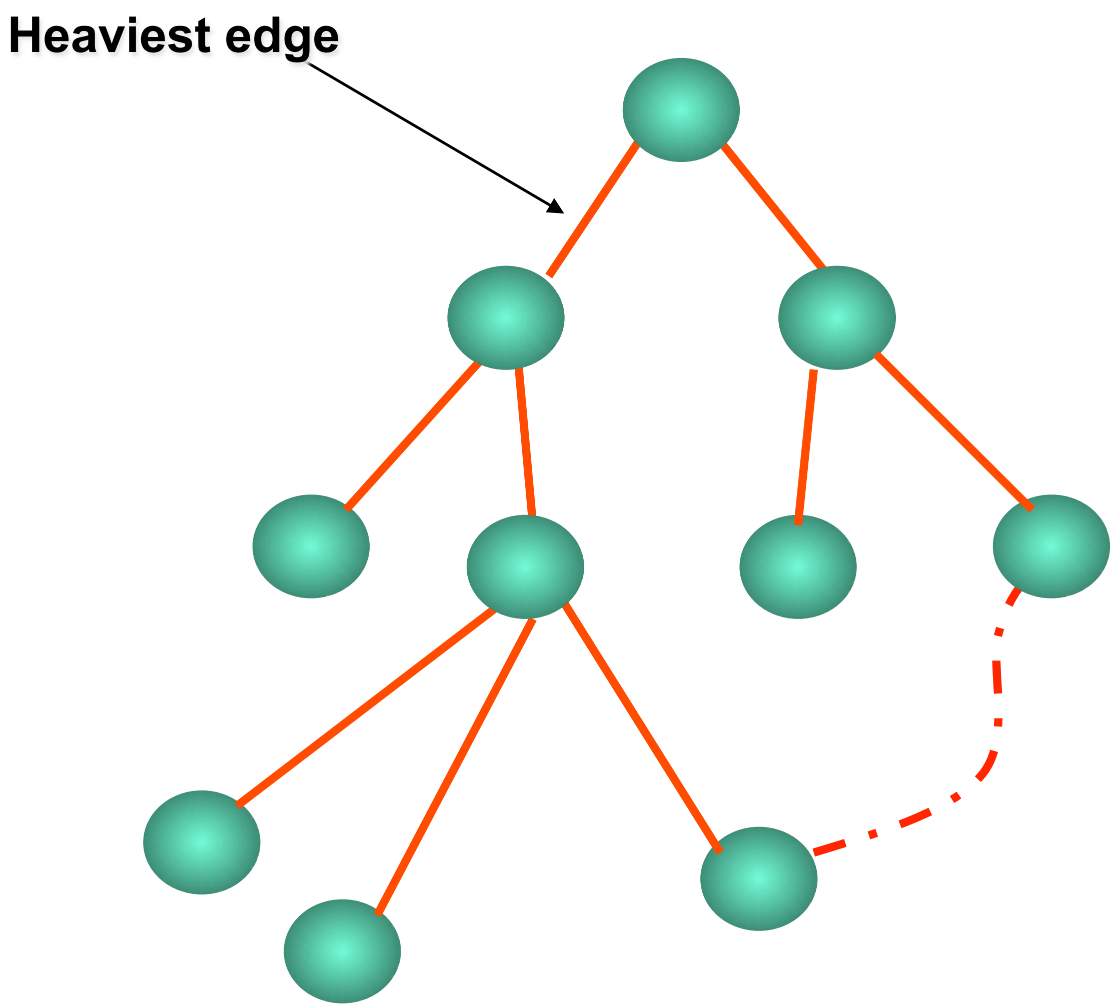
- Cycle property
- Cut property

Cycle property

- Let C be any cycle in graph G (that is a set of edges building a cycle). Let e be a heaviest edge in the cycle.

- Then there exists a minimum spanning tree for G that does not contain e.

Cycle property



Cycle property

- Let C be any cycle in graph G (that is a set of edges building a cycle). Let e be a heaviest edge in the cycle.

- Then there exists a minimum spanning tree for G that does not contain e.

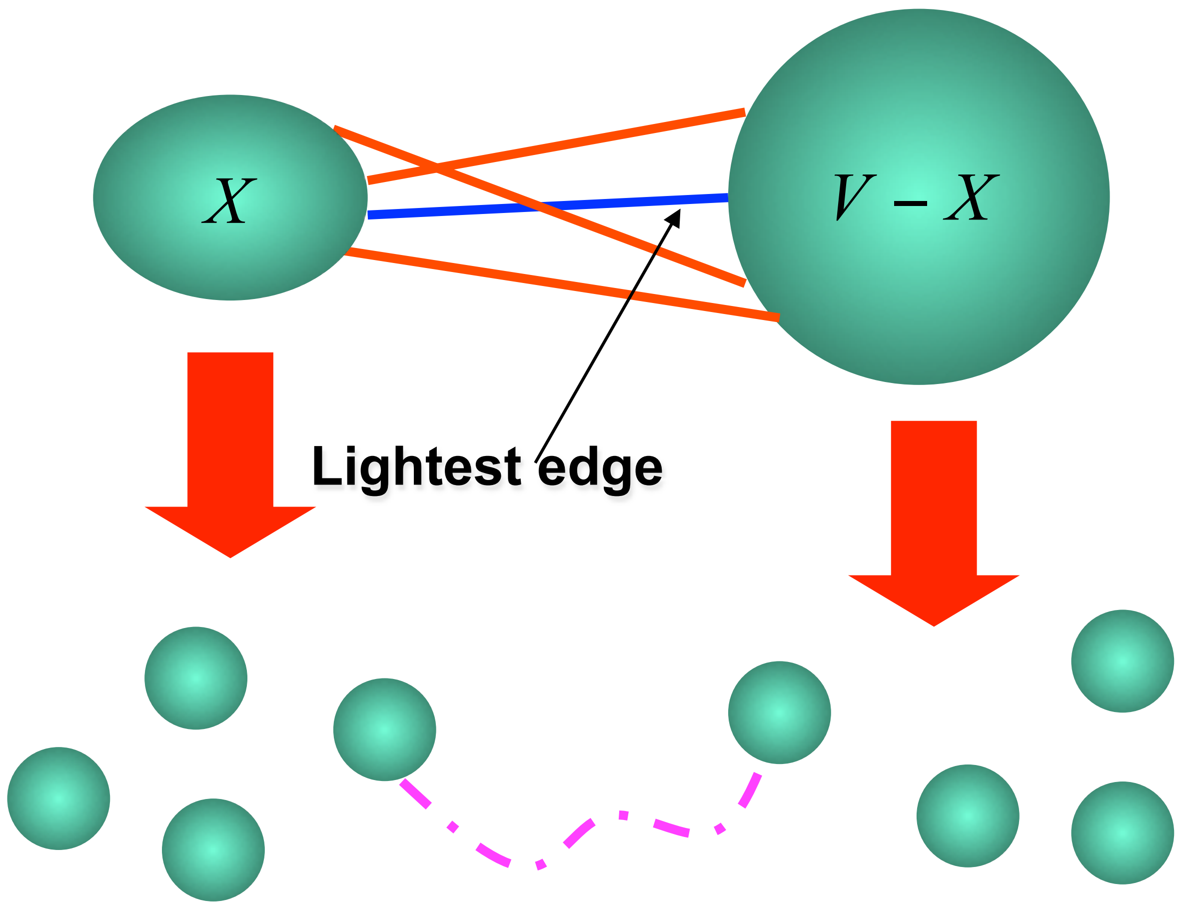
Proof. (Cycle property)

- For now, assume that all edges in the graph are of distinct weight
- We proof by contradiction: there is no MST T for G containing edge e
- Assume e does belong to MST T. Then deleting e from T disconnects T into two trees, T_1 and T_2.
- Consider cycle C. C consists of some vertices that belong to T_1 and the other vertices of C belong to T_2.
- There is an edge in C, say f, that connects a vertex from T_1 to a vertex T_2.
- Merge T_1 and T_2 using f to spanning tree T^*. The new tree, T^*, is lighter than T. A contradiction.

Cut Property

- Let V' be any subset of vertices in weighted graph $G = (V, E)$, and let e be a lightest edge that has exactly one endpoint in V'
- Then there is a minimum spanning tree T for G that contains e.

Cut Property



Cut Property

- Let V' be any subset of vertices in weighted graph $G = (V, E)$, and let e be a lightest edge that has exactly one endpoint in V'
- Then there is a minimum spanning tree T for G that contains e.

Proof (Cut property)

- For now, assume that all edges in the graph are of distinct weight
- We proof by contradiction: MST T for G contains edge e
- Assume it does not
- Add e to T creating cycle C
- Consider edge f in C that has exactly one endpoint in V'
- Create spanning tree T^* by replacing e with f, but T^* is heavier than T. Contradiction.

Prim's Algorithm Correctness of Idea

- Initialize tree with single chosen vertex

Cut property

- Grow tree by finding lightest edge not yet in tree and expanding the tree, and connect it to tree; repeat until all vertices are in the tree

- *Example of greedy algorithm*

Kruskal's Algorithm Correctness of Idea

Cut property

- Initialize a forest consisting of all nodes
- Pick a (non-selected) minimum weight edge and, if it connects two different trees of the forest, select it, otherwise discard it; repeat

Cycle property

- *Example of greedy algorithm*

Borůvka's Algorithm Correctness of Idea

- Assume every edge has a unique weight.
- Initially, each vertex is considered a separate component.
- The algorithm merges disjoint components as follows and repeating the step until only one component exists.
- In each step, every component is merged with some other using the cheapest outgoing edge of the given component.

Cut property

Pseudocode: Kruskal's Algorithm

1. Algorithm KruskalMSTNaive($G = (V,E)$)
    1. sort edges in E according to weight
    2. Initialize E_T as empty set
    3. pick lightest edge e in E; remove e from E
    4. If e does not build a cycle with edges in E_T then add e to E_T; else, discard e
    5. Repeat from 3. until E is empty

Running time KruskalMSTNaive