CSC 226

Algorithms and Data Structures: II

Rich Little
rlittle@uvic.ca

Weighted Graphs

- A ***weighted graph*** is a graph model where we associate ***weights*** (or ***costs***) with each edge

- That is, for each edge e, we has a numeric label associated with it, denoted $w(e)$.

- We will look at algorithms for solving three general problems of weighed graphs
  - Minimum spanning trees
  - Shortest Paths
  - Network Flow

Weighted edge API

Edge abstraction needed for weighted edges.

public class Edge implements Comparable<Edge>

	Edge(int v, int w, double weight)	*create a weighted edge v-w*
int	either()	*either endpoint*
int	other(int v)	*the endpoint that's not v*
int	compareTo(Edge that)	*compare this edge to that edge*
double	weight()	*the weight*
String	toString()	*string representation*

Idiom for processing an edge e: int v = e.either(), w = e.other(v);

Weighted edge: Java implementation

```
public class Edge implements Comparable<Edge>
{
   private final int v, w;
   private final double weight;

   public Edge(int v, int w, double weight)
   {
      this.v = v;
      this.w = w;
      this.weight = weight;
   }

   public int either()
   {  return v;  }

   public int other(int vertex)
   {
      if (vertex == v) return w;
      else return v;
   }

   public int compareTo(Edge that)
   {
      if      (this.weight < that.weight) return -1;
      else if (this.weight > that.weight) return +1;
      else                                return  0;
   }
}
```

constructor

either endpoint

other endpoint

compare edges by weight

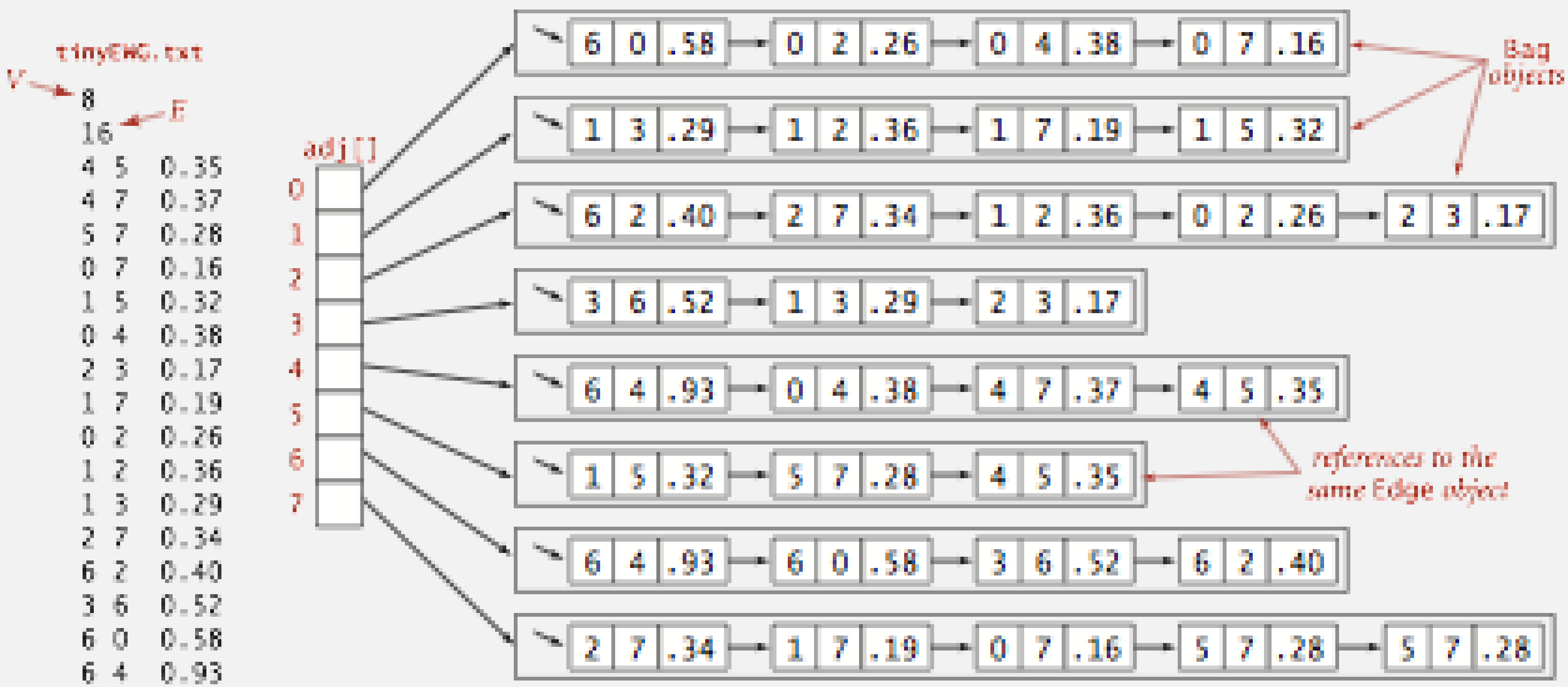
Edge-weighted graph API

public class EdgeWeightedGraph		
	EdgeWeightedGraph(int V)	*create an empty graph with V vertices*
	EdgeWeightedGraph(In in)	*create a graph from input stream*
void	addEdge(Edge e)	*add weighted edge e to this graph*
Iterable<Edge>	adj(int v)	*edges incident to v*
Iterable<Edge>	edges()	*all edges in this graph*
int	V()	*number of vertices*
int	E()	*number of edges*
String	toString()	*string representation*

Conventions. Allow self-loops and parallel edges.

Edge-weighted graph: adjacency-lists representation

Maintain vertex-indexed array of Edge lists.



Edge-weighted graph: adjacency-lists implementation

```
public class EdgeWeightedGraph
{
  private final int V;
  private final Bag<Edge>[] adj;

  public EdgeWeightedGraph(int V)
  {
    this.V = V;
    adj = (Bag<Edge>[]) new Bag[V];
    for (int v = 0; v < V; v++)
      adj[v] = new Bag<Edge>();
  }

  public void addEdge(Edge e)
  {
    int v = e.either(), w = e.other(v);
    adj[v].add(e);
    adj[w].add(e);
  }

  public Iterable<Edge> adj(int v)
  {  return adj[v];  }
}
```

same as Graph, but adjacency lists of Edges instead of integers

constructor

add edge to both adjacency lists

Minimum Spanning Tree Definition

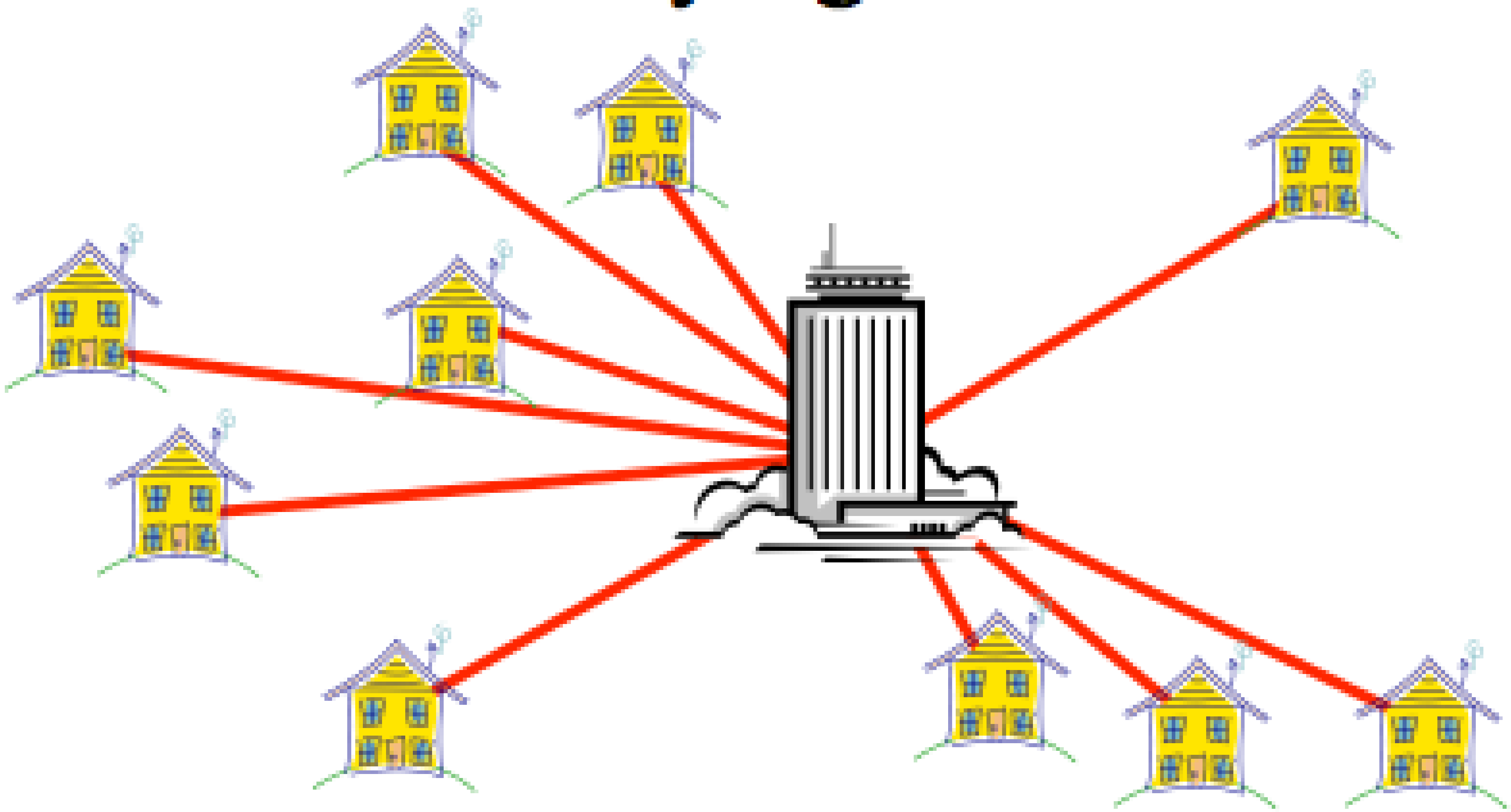
- *Input*: A weighted, connected graph $G = (V, E)$ consisting of vertices (or nodes), V, and edges, E, with edge weights

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

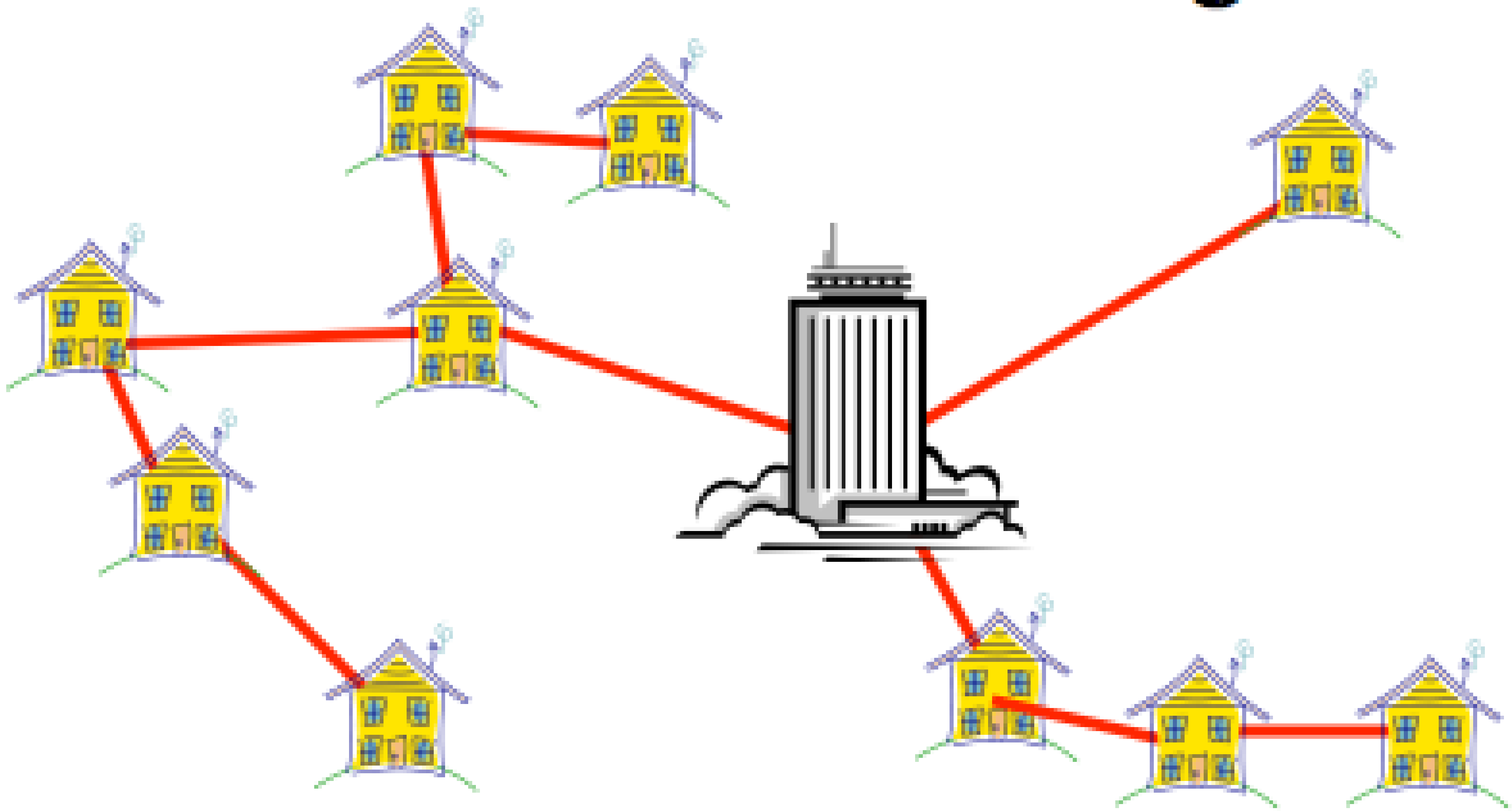
$$w(T) = \sum_{e \in E_T} w(e)$$

is minimized.

Problem: Laying Cable TV Wire

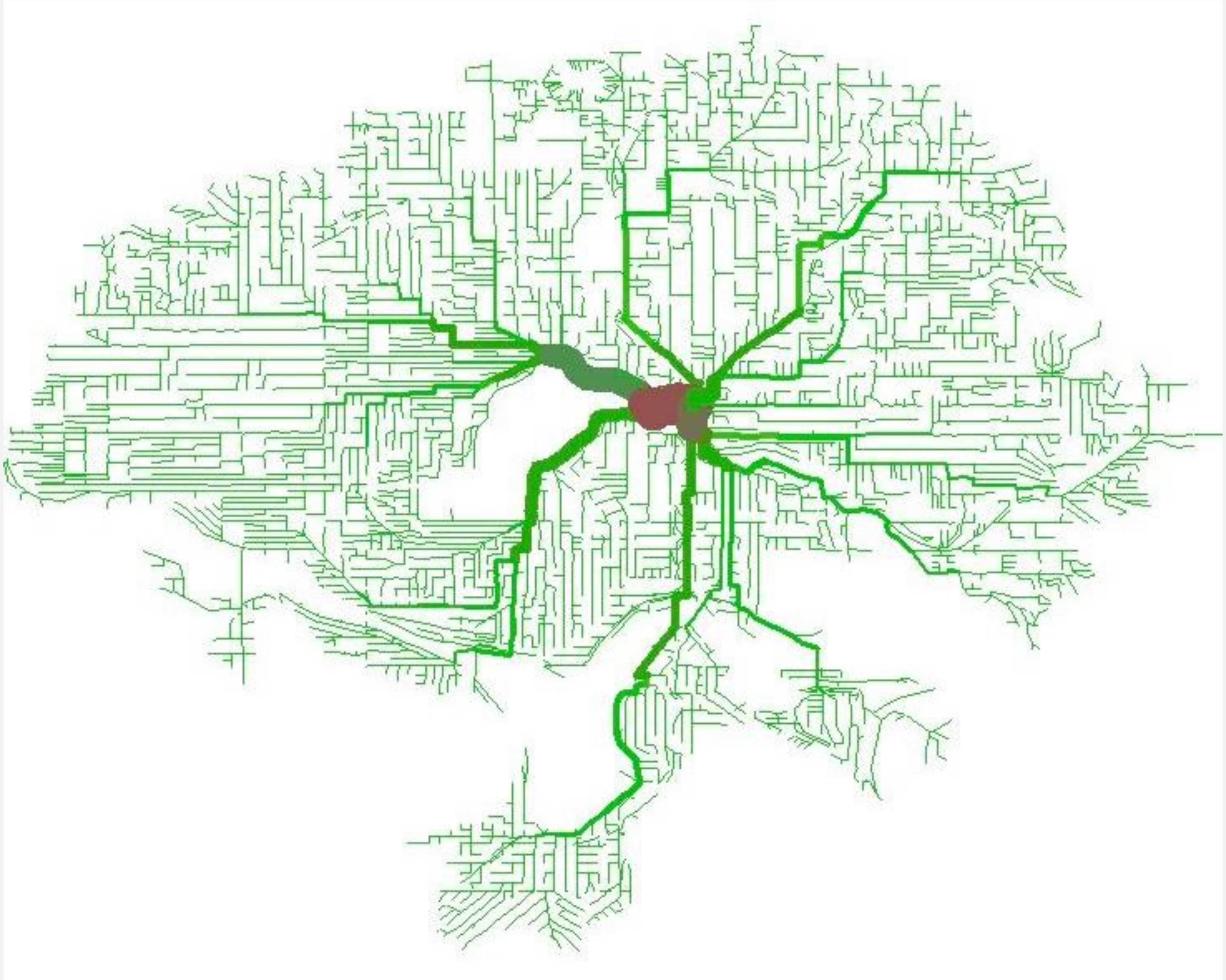


Minimize Wiring



Network design

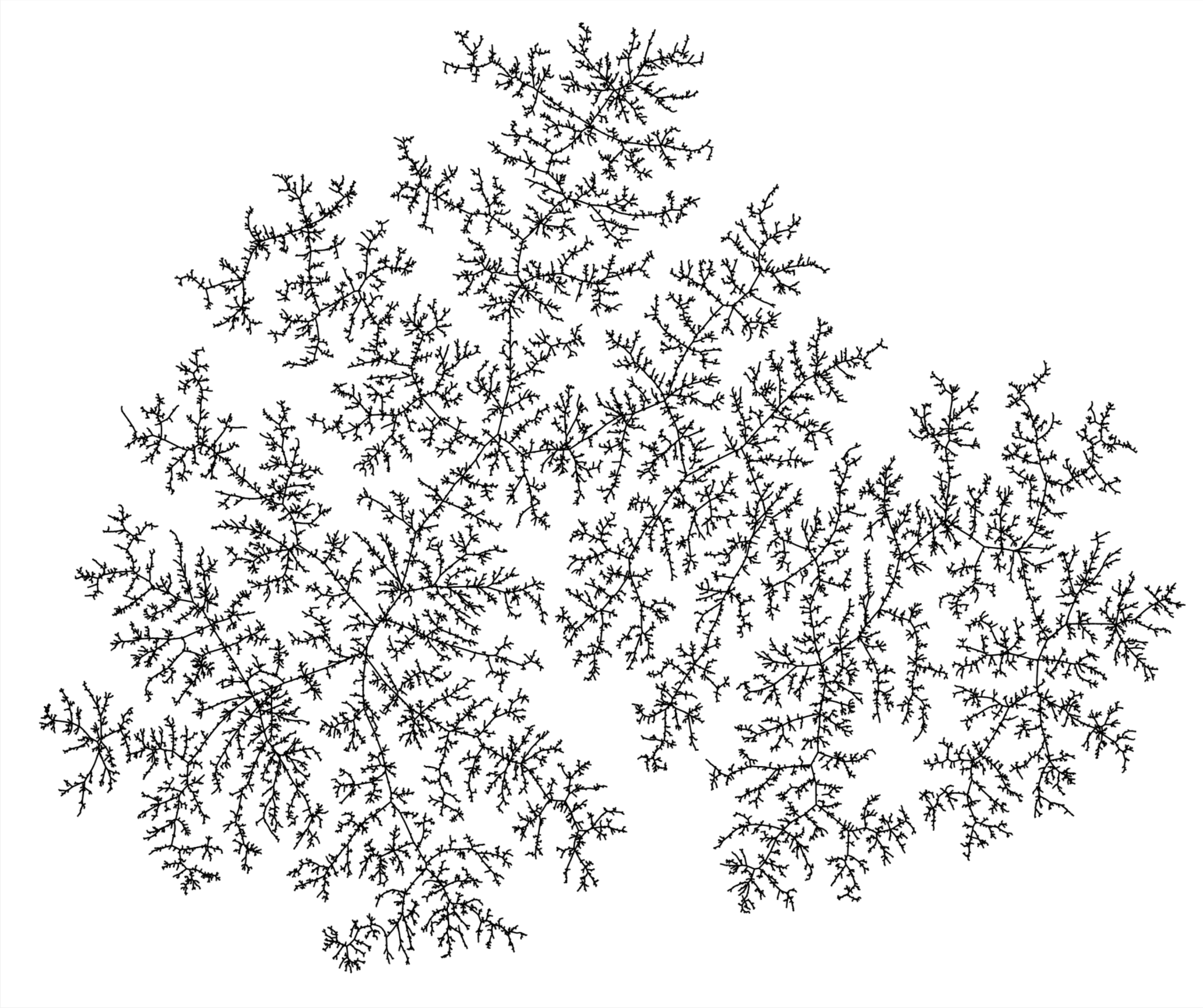
MST of bicycle routes in North Seattle



http://www.flickr.com/photos/ewedistrict/21980840

Models of nature

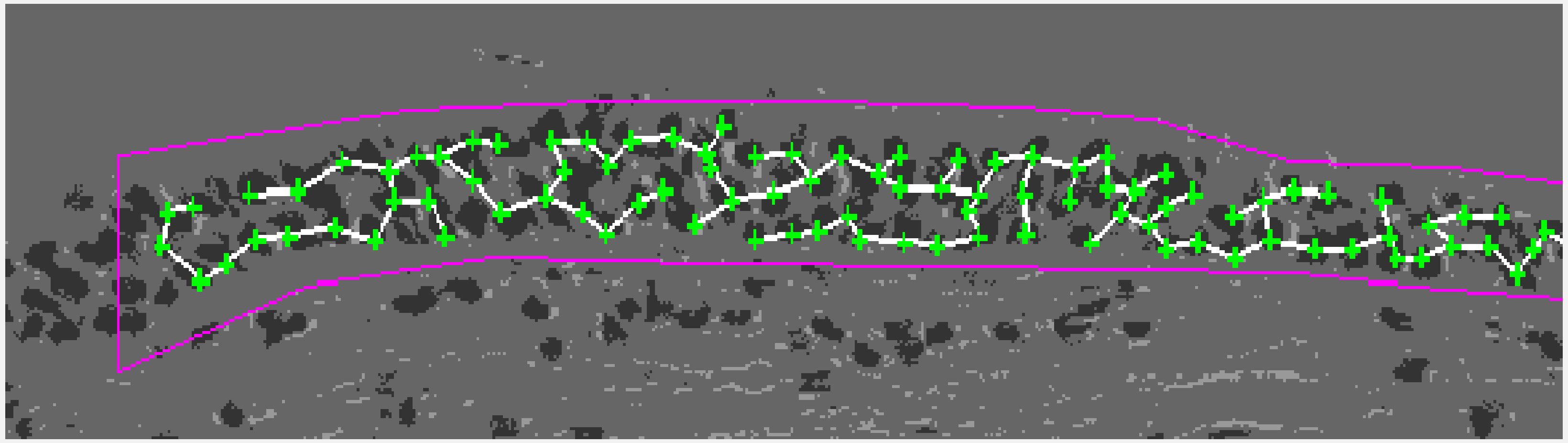
MST of random graph

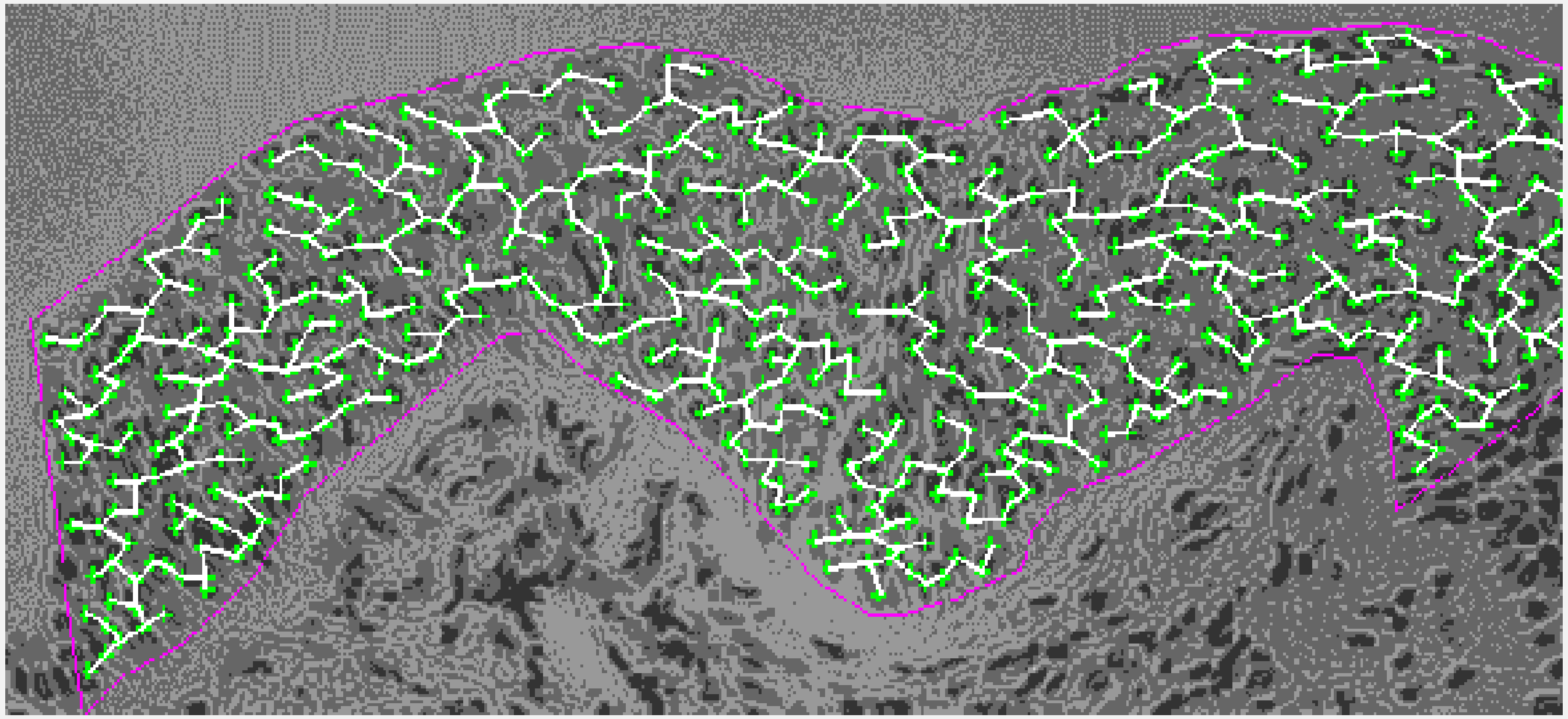


http://algo.inria.fr/broutin/gallery.html

Medical image processing

MST describes arrangement of nuclei in the epithelium for cancer research

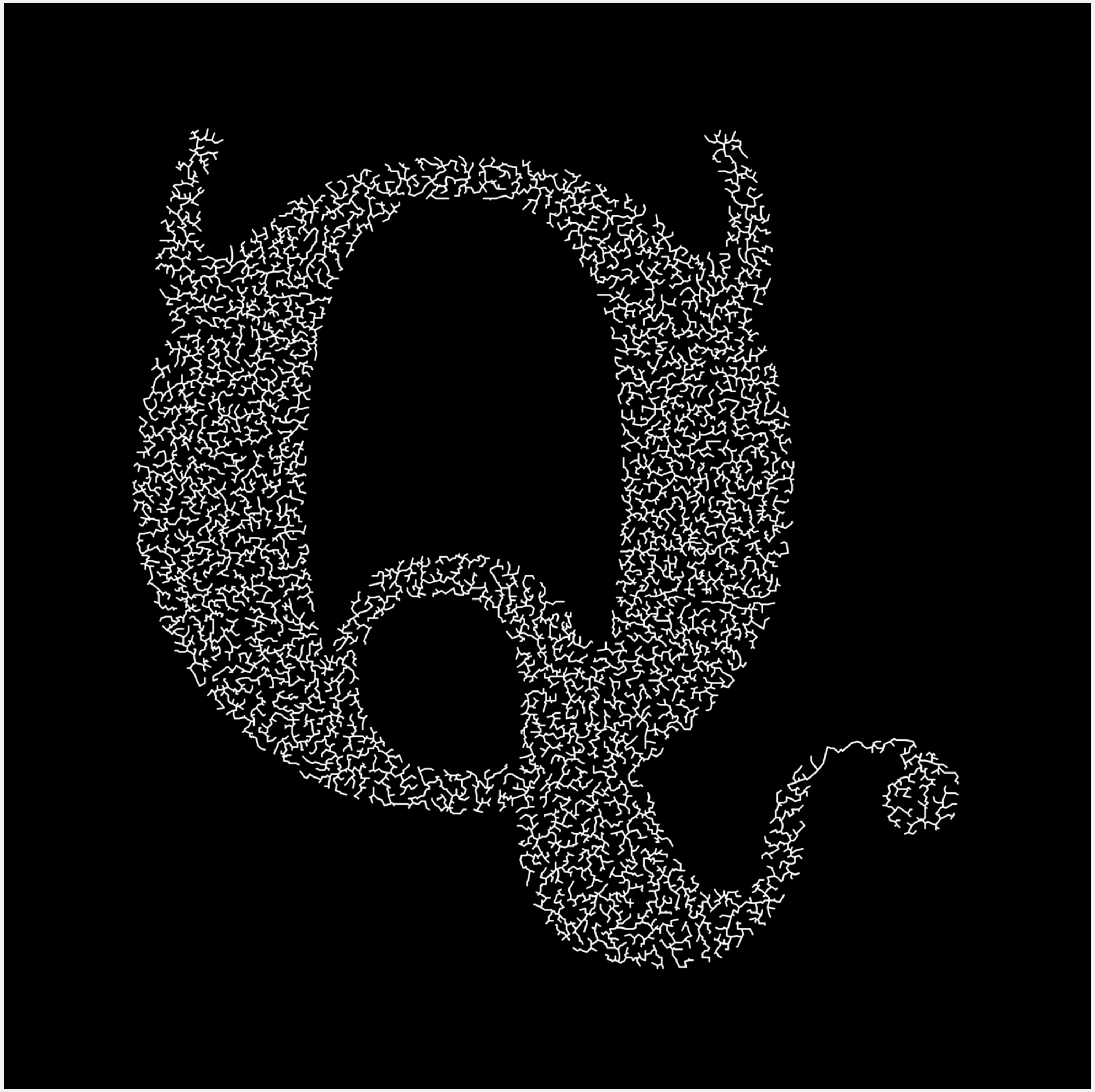




http://www.bccrc.ca/ci/ta01_archlevel.html

Dithering

MST dithering



http://www.flickr.com/photos/quasimondo/2695389651

Applications

MST is fundamental problem with diverse applications.

- Dithering.
- Cluster analysis.
- Max bottleneck paths.
- Real-time face verification.
- LDPC codes for error correction.
- Image registration with Renyi entropy.
- Find road networks in satellite and aerial imagery.
- Reducing data storage in sequencing amino acids in a protein.
- Model locality of particle interactions in turbulent fluid flows.
- Autoconfig protocol for Ethernet bridging to avoid cycles in a network.
- Approximation algorithms for NP-hard problems (e.g., TSP, Steiner tree).
- Network design (communication, electrical, hydraulic, computer, road).

http://www.ics.uci.edu/~eppstein/gina/mst.html

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 Formula [1889])

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

Minimum Spanning Tree algorithms

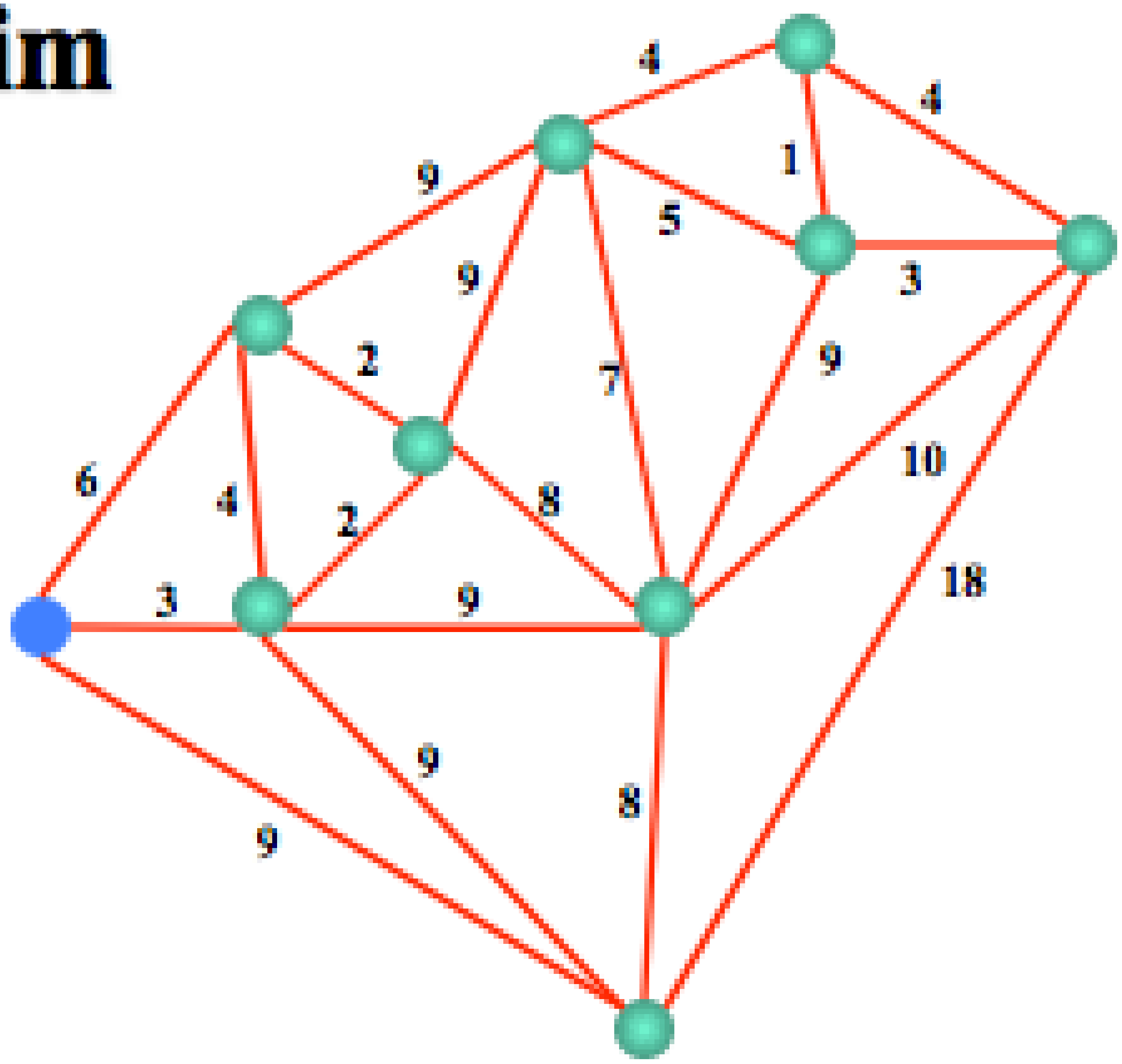
- 1926 Barůvka $O(m \log n)$
- 1930 Prim-Jarník's
  - 1930 Jarník
  - 1957 Dijkstra
  - 1959 Prim
  - 1964 with Heaps $O(m \log n)$
  - 1987 Fredman and Tarjan with Fibonacci Heaps $O(m+n \log n)$
- 1956 Kruskal's algorithm
  - 1956 Kruskal
  - 1974 Aho, Hopcroft and Ullman with Union-Find Disjoint Set $O(m \log n)$
- 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

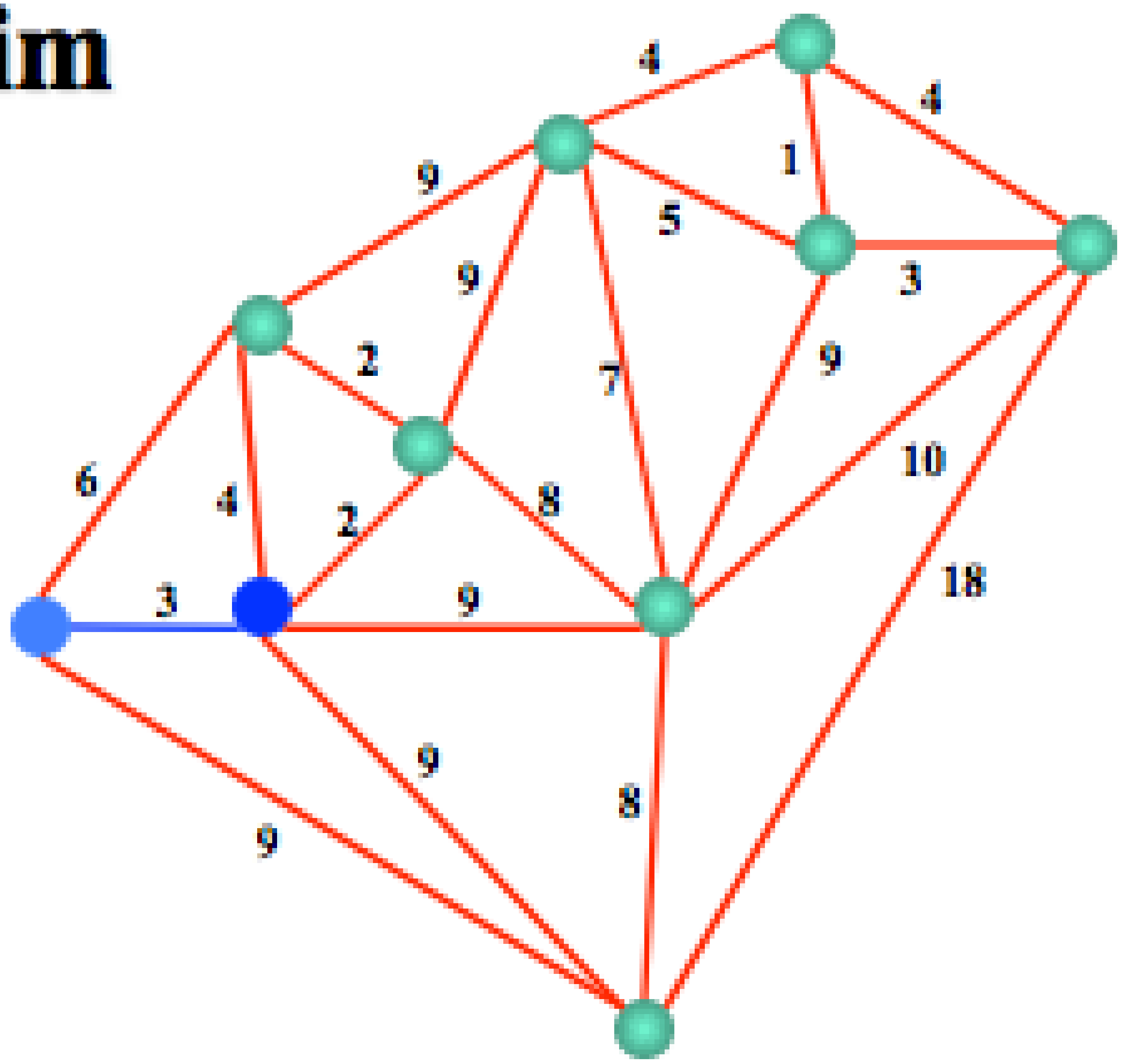
Prim’s Algorithm
Idea

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

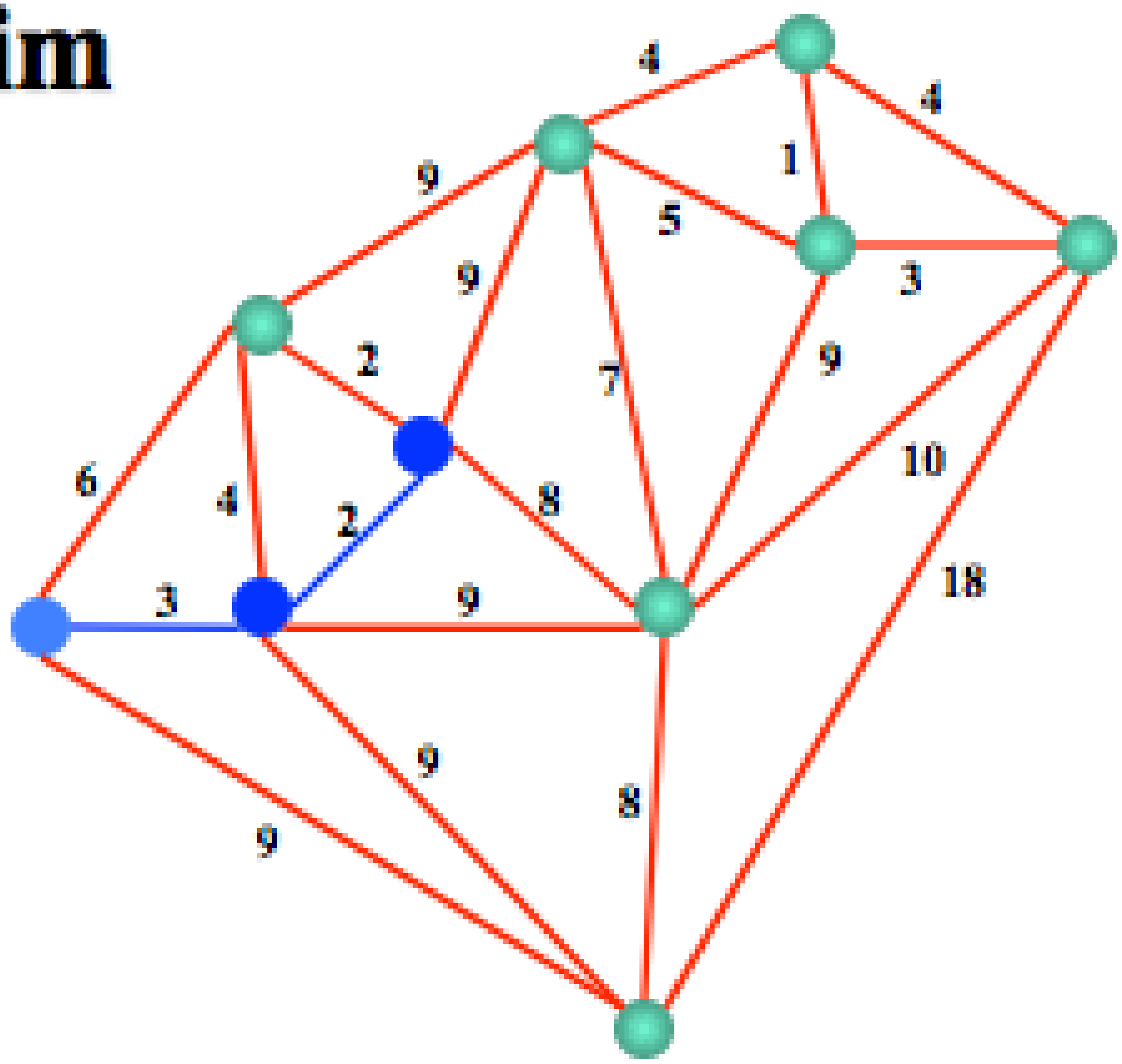
Prim



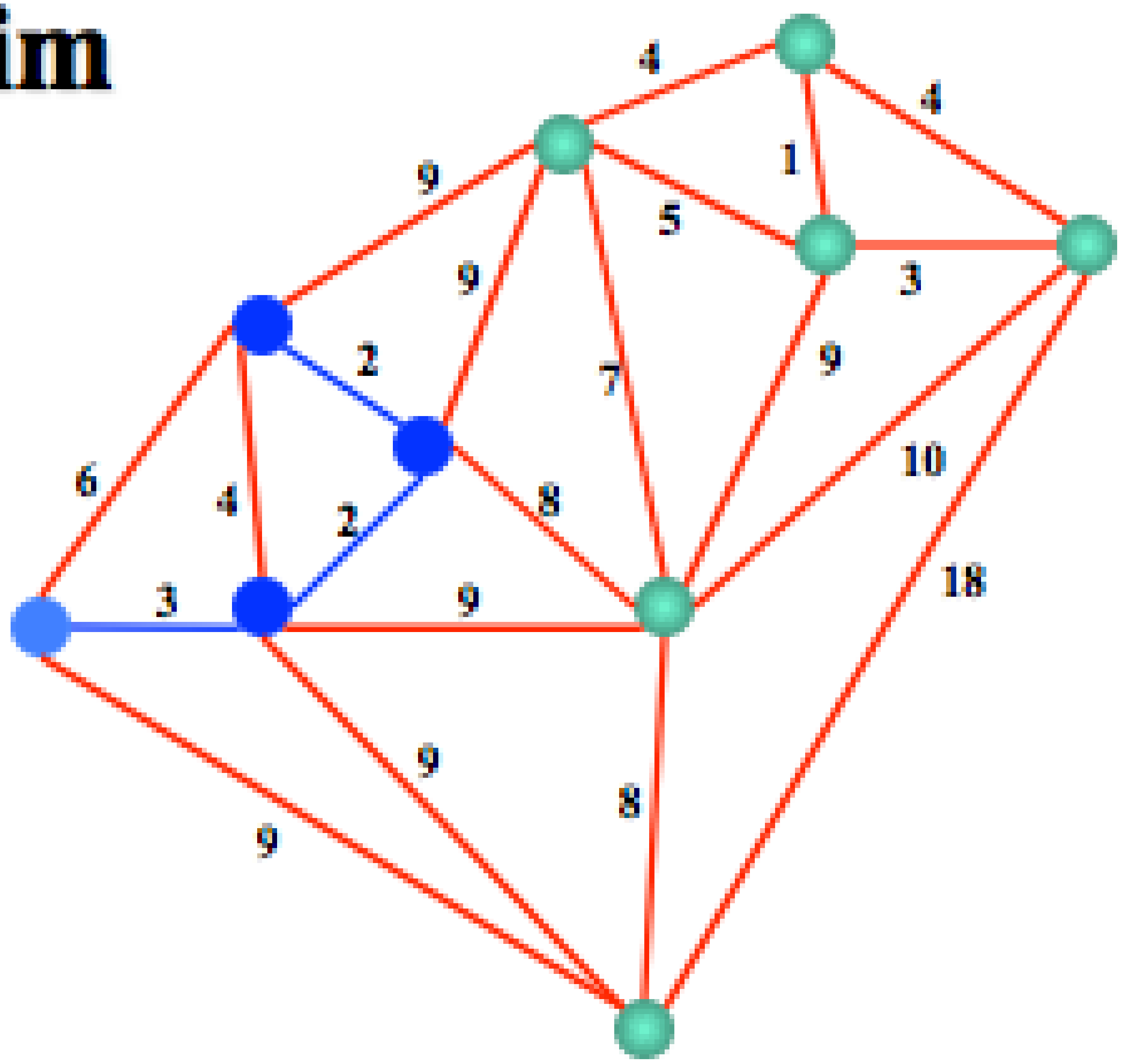
Prim



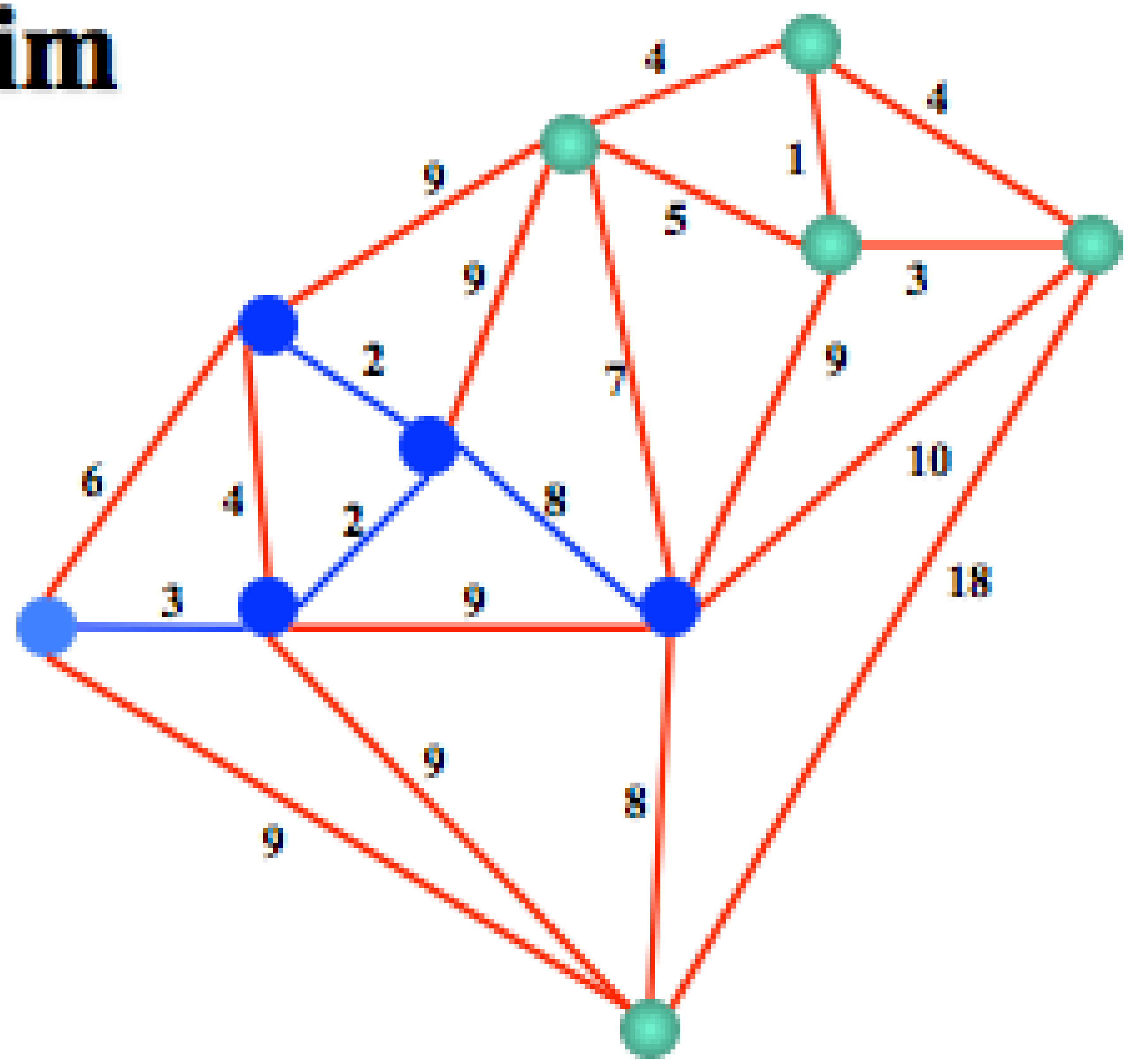
Prim



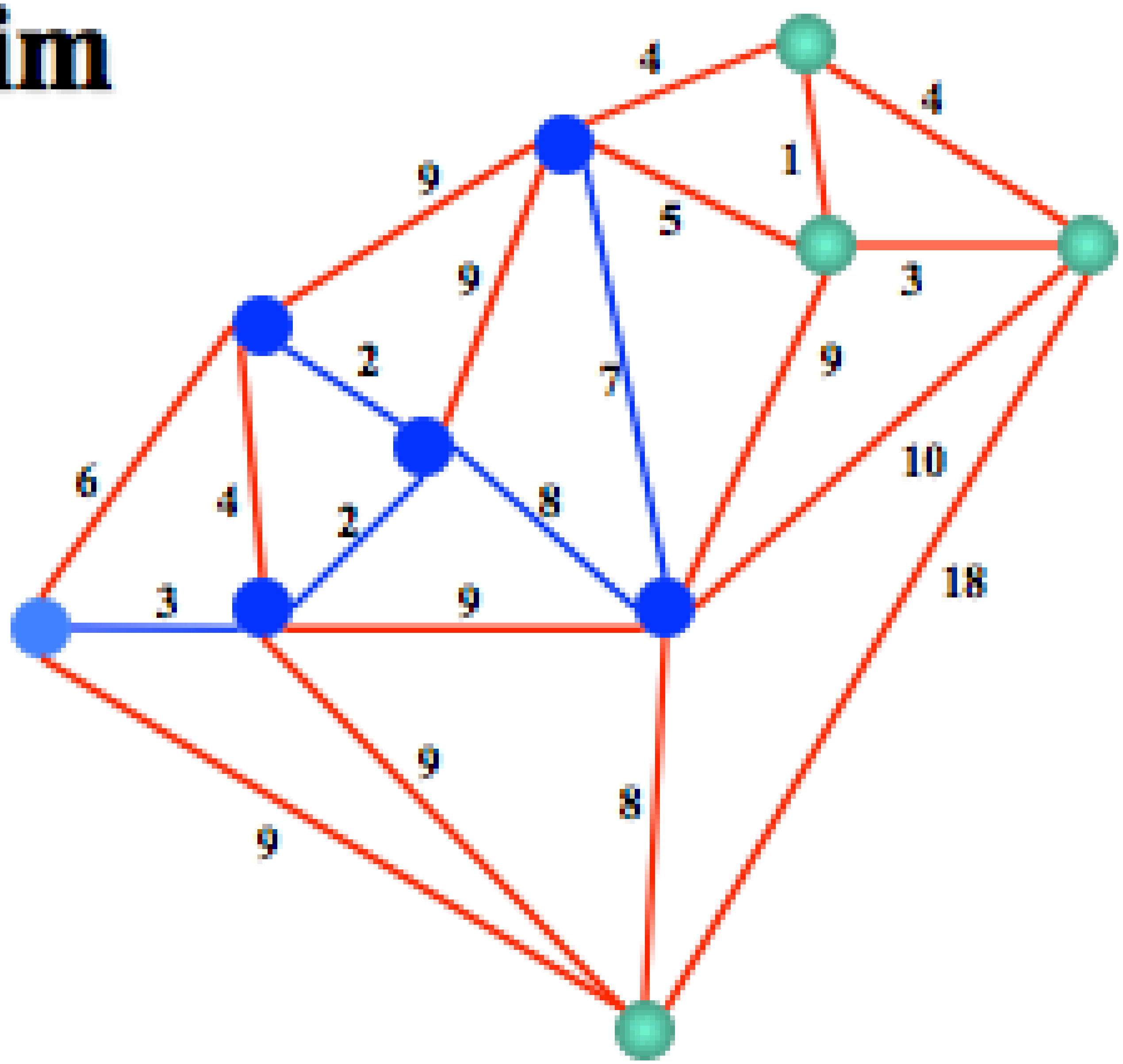
Prim



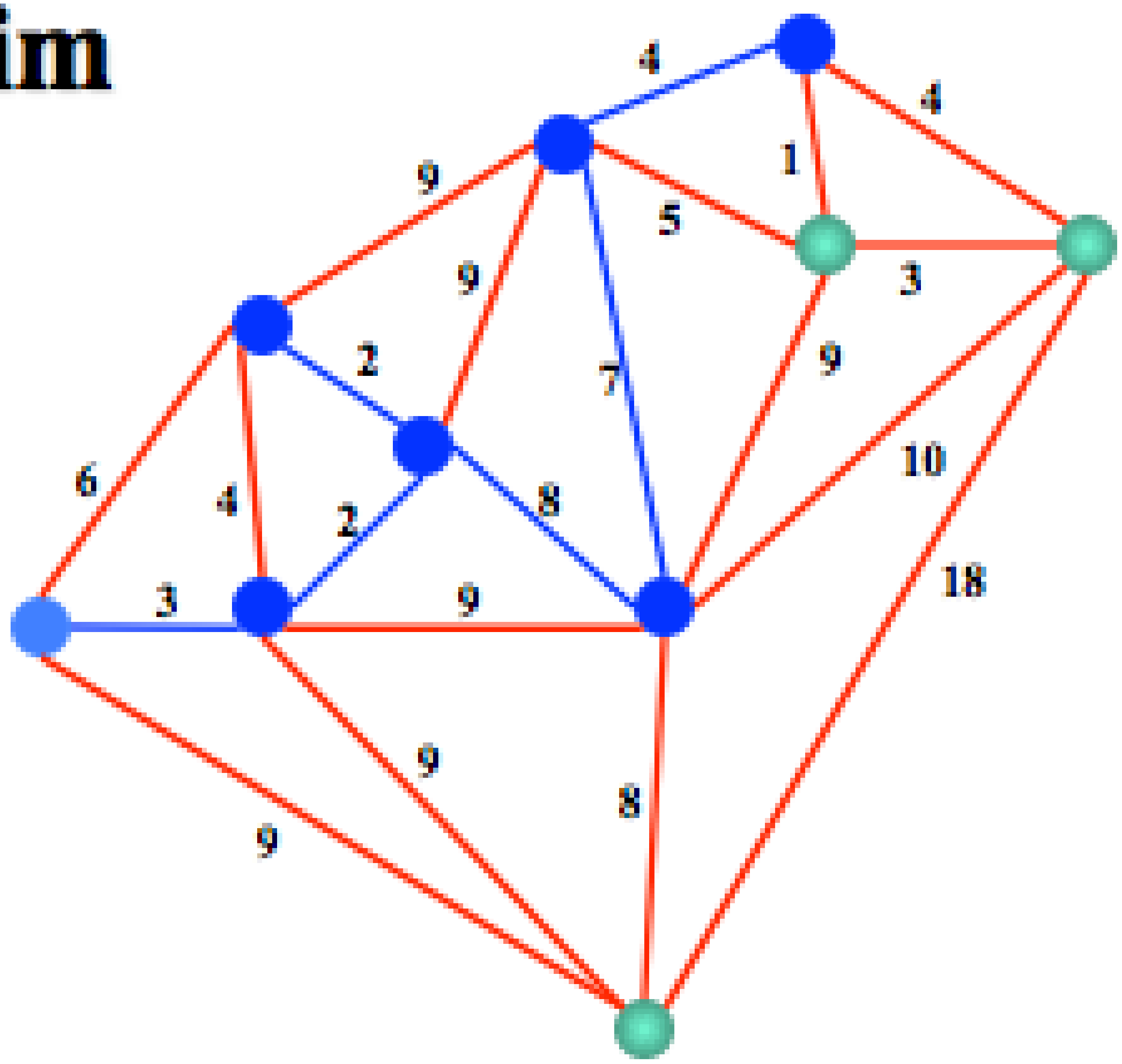
Prim



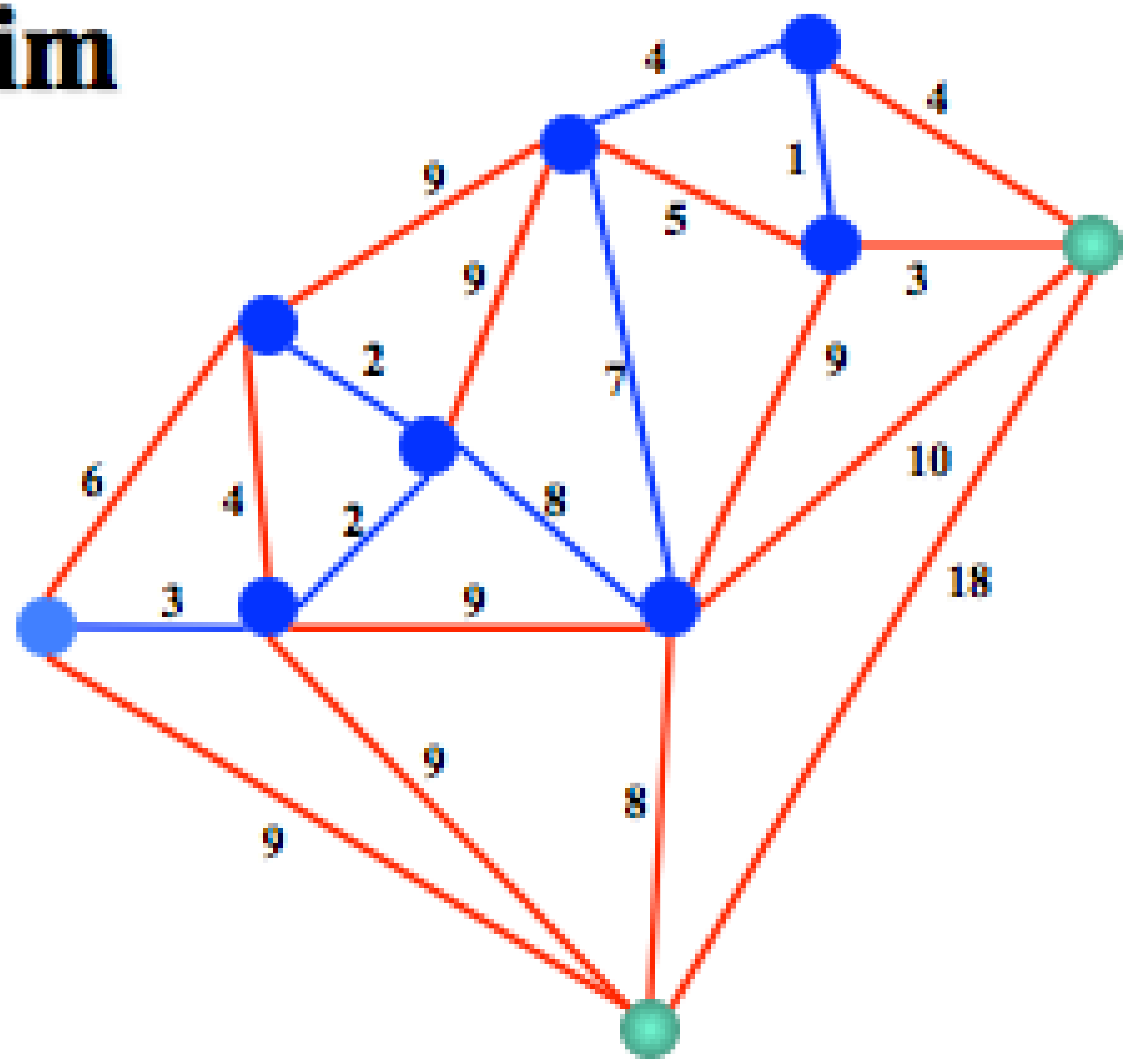
Prim



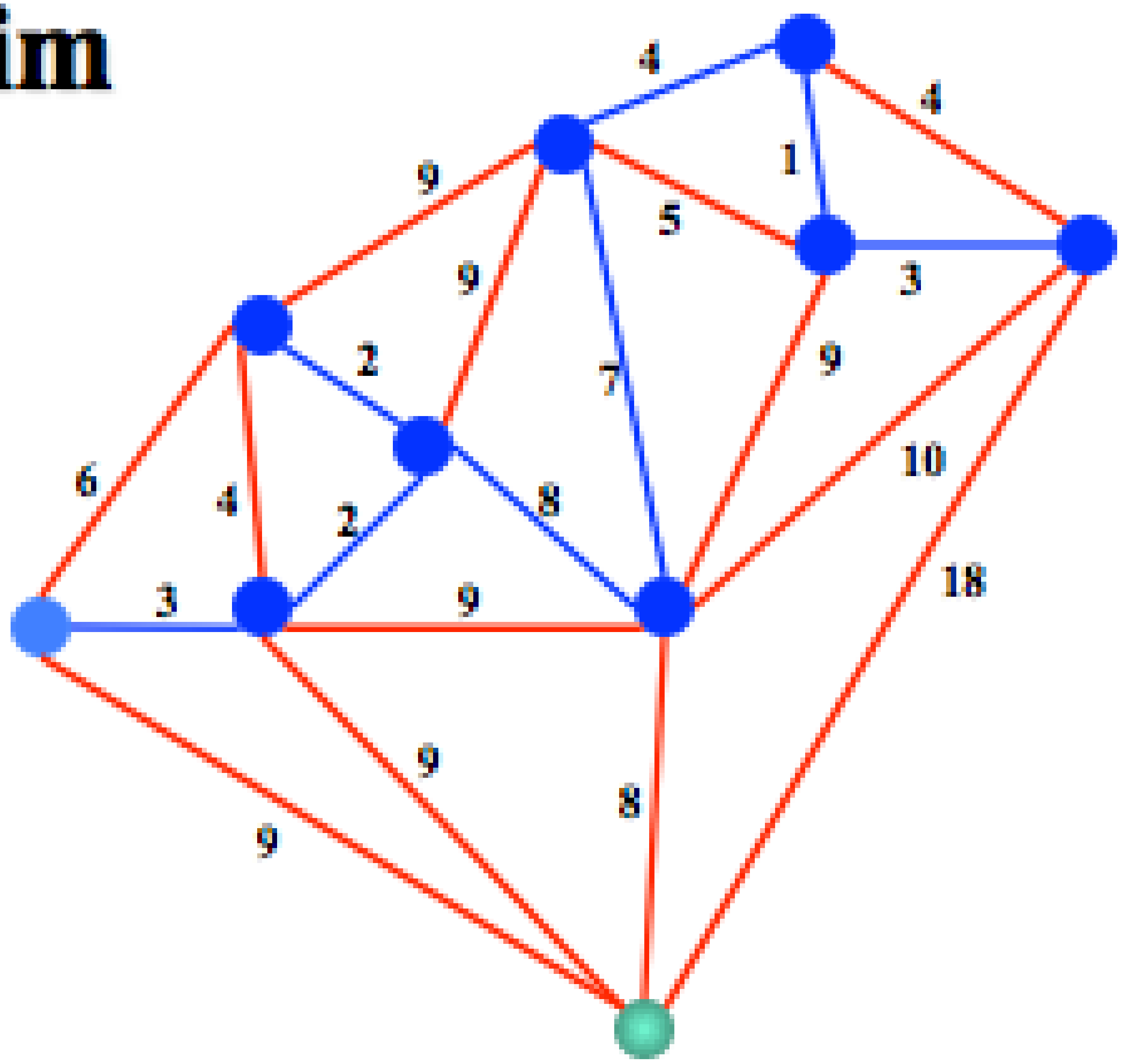
Prim



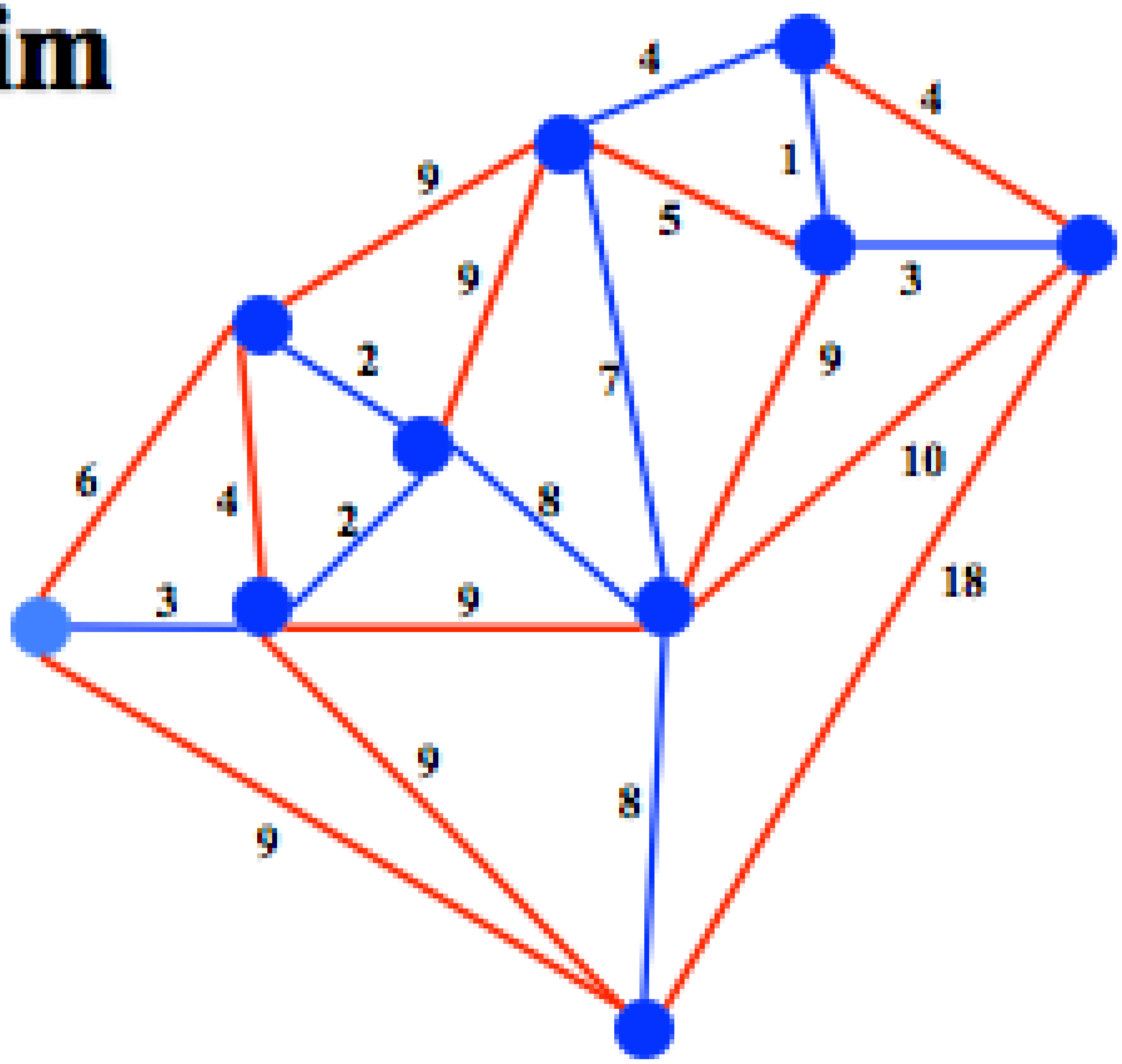
Prim



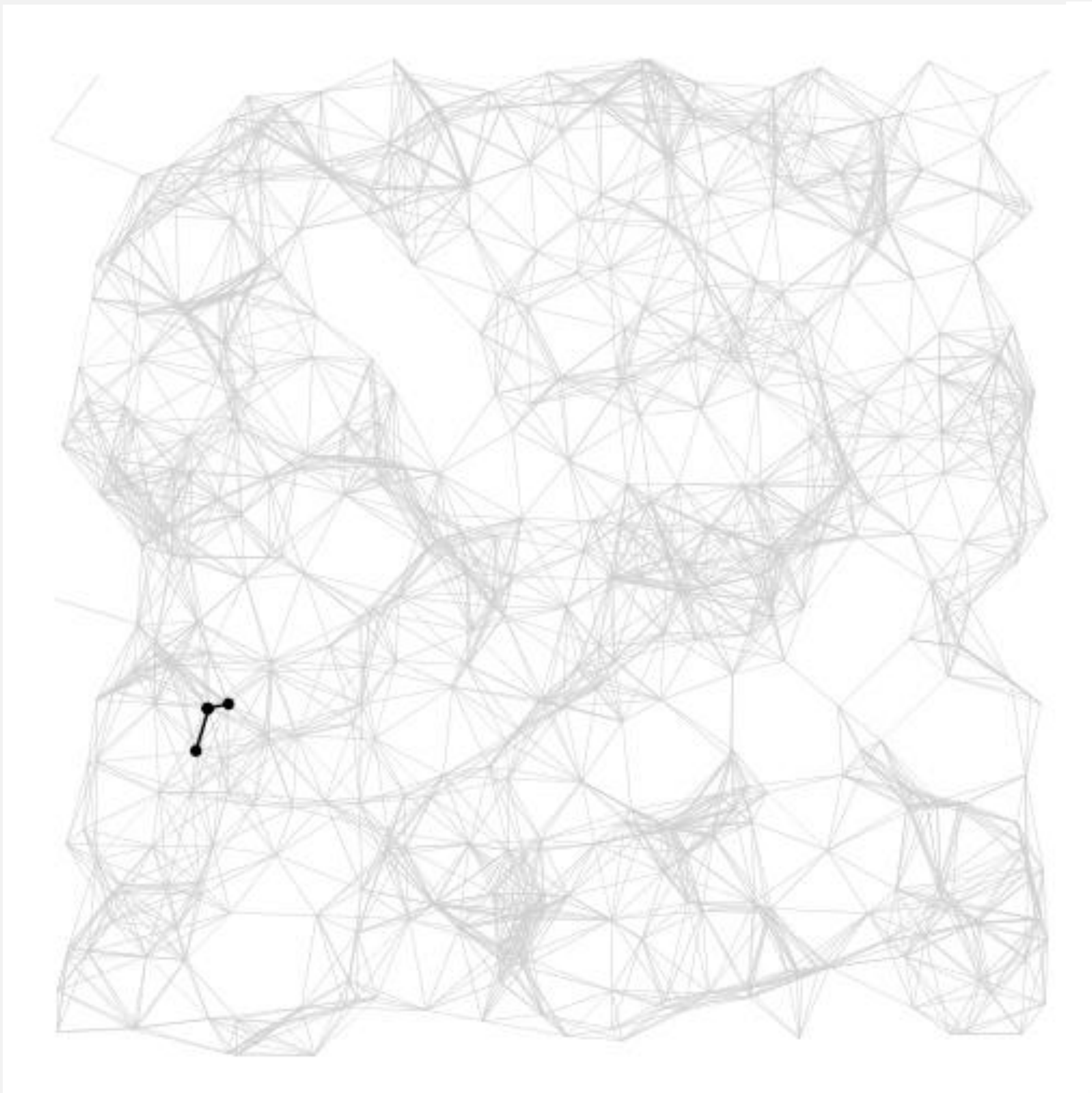
Prim



Prim



Prim's algorithm: visualization



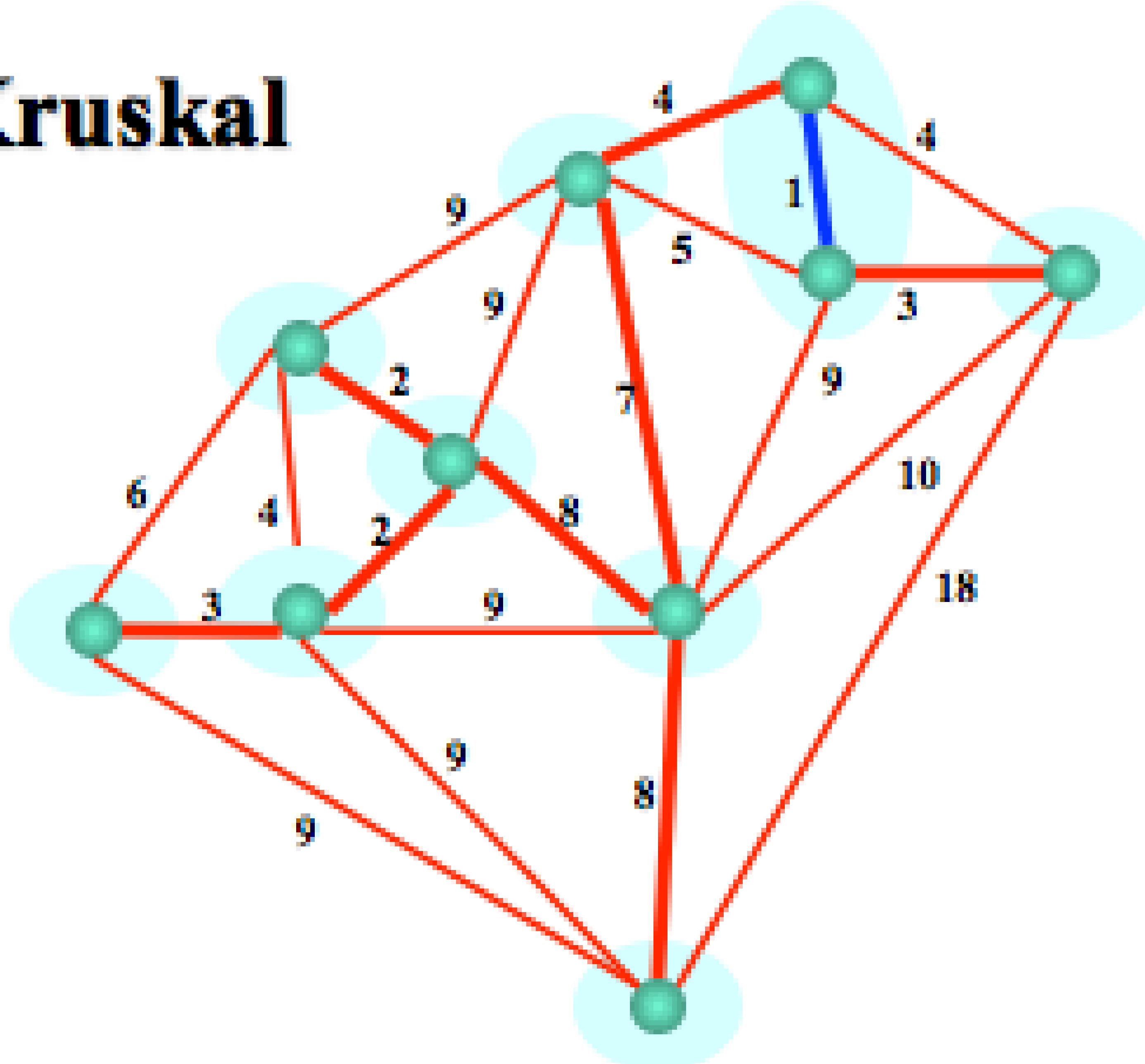
Prim’s Algorithm
Idea

- 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*

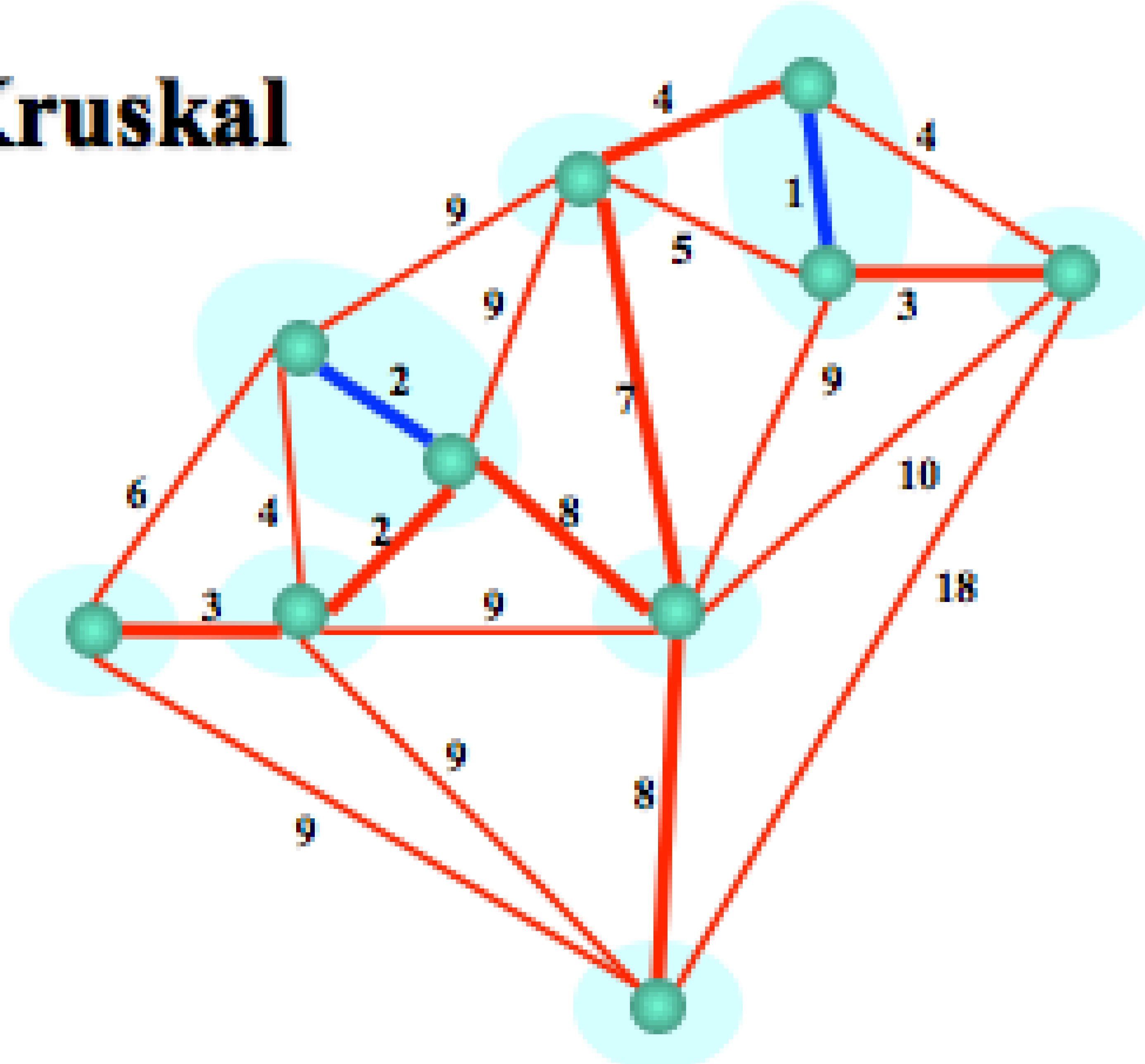
Kruskal



Sorted edge weights

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

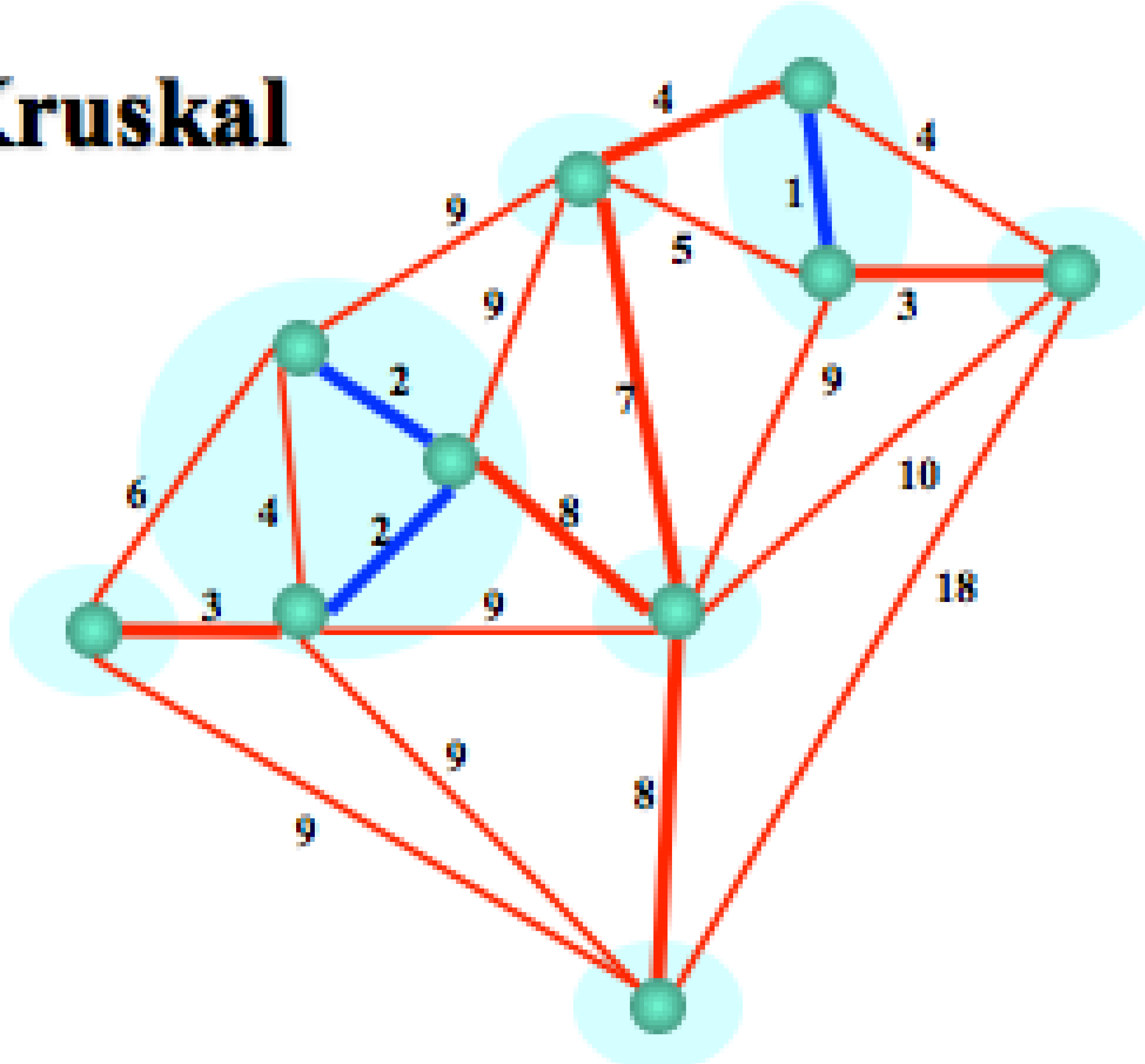
Kruskal



Sorted edge weights

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

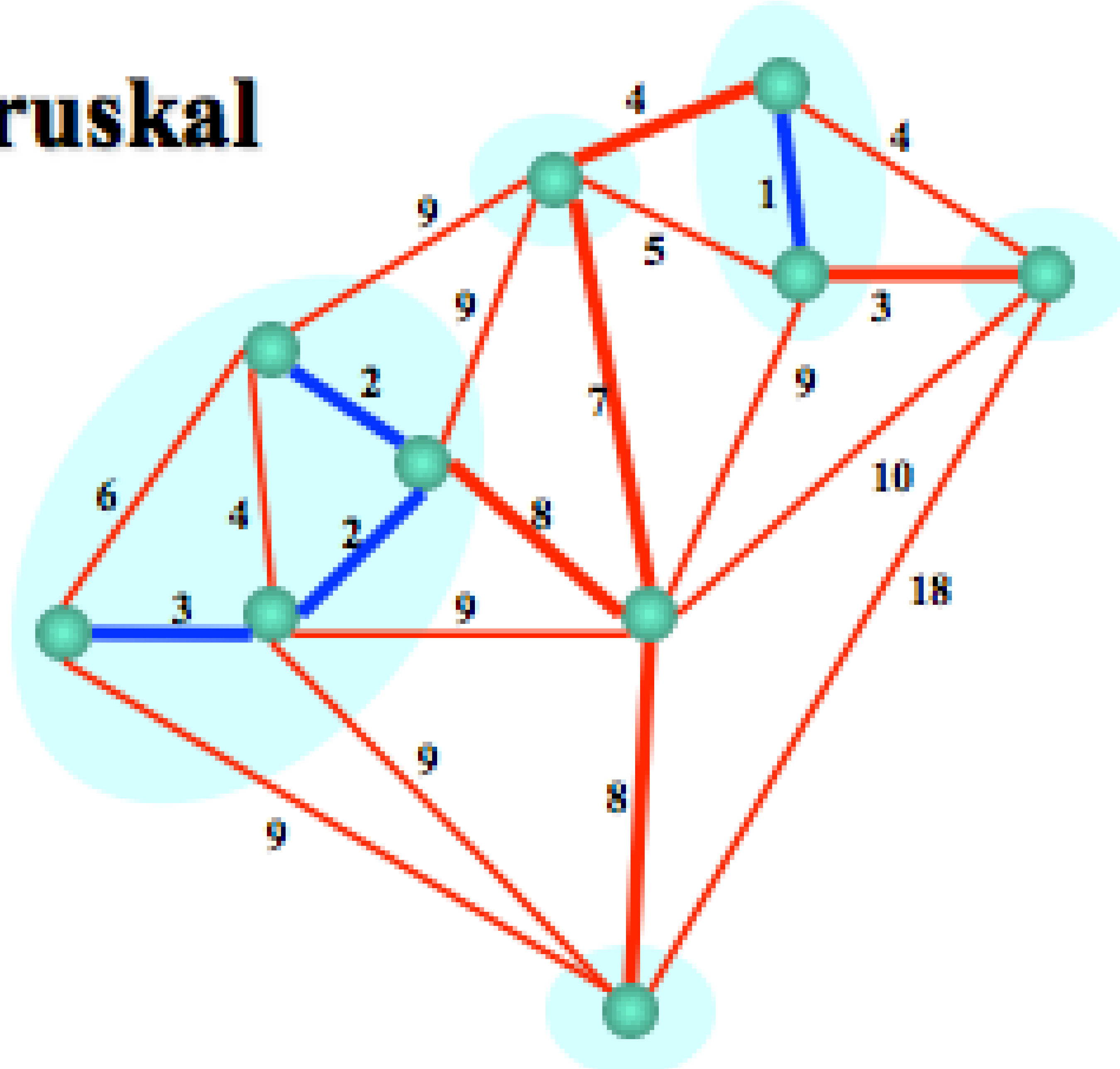
Kruskal



Sorted edge weights

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

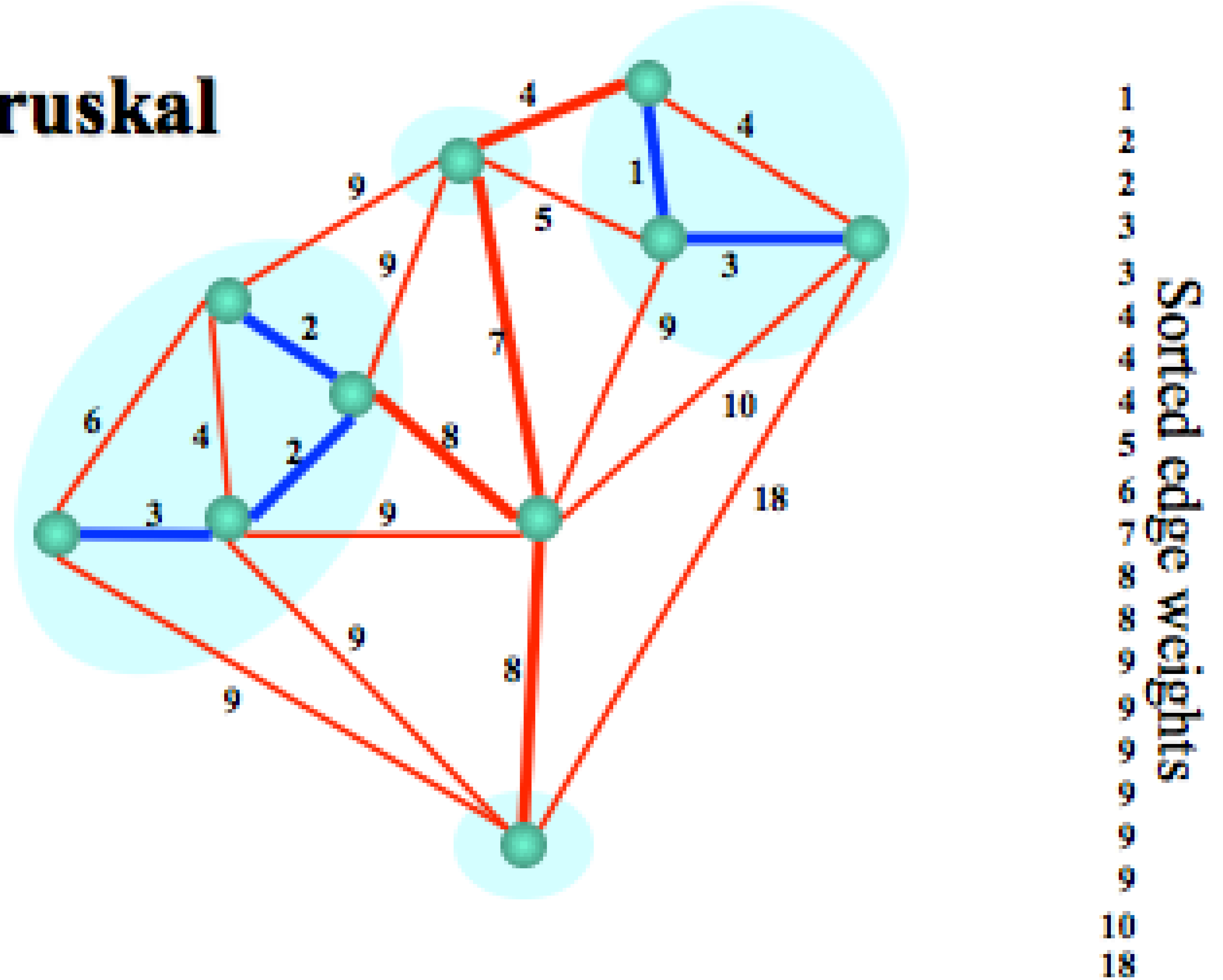
Kruskal



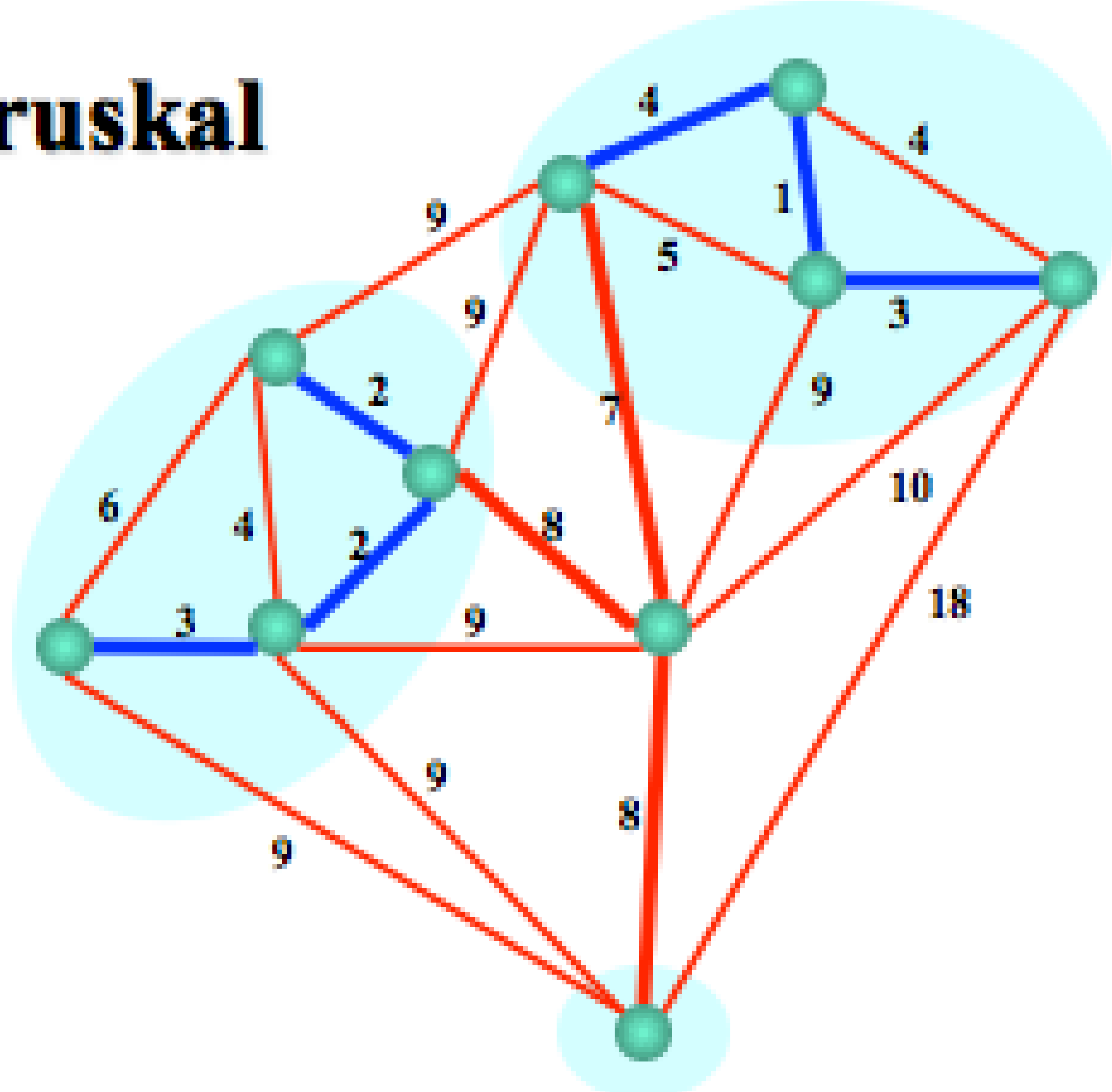
1
2
2
3
3
4
4
4
5
6
7
8
8
9
9
9
9
9
9
10
18

Sorted edge weights

Kruskal


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

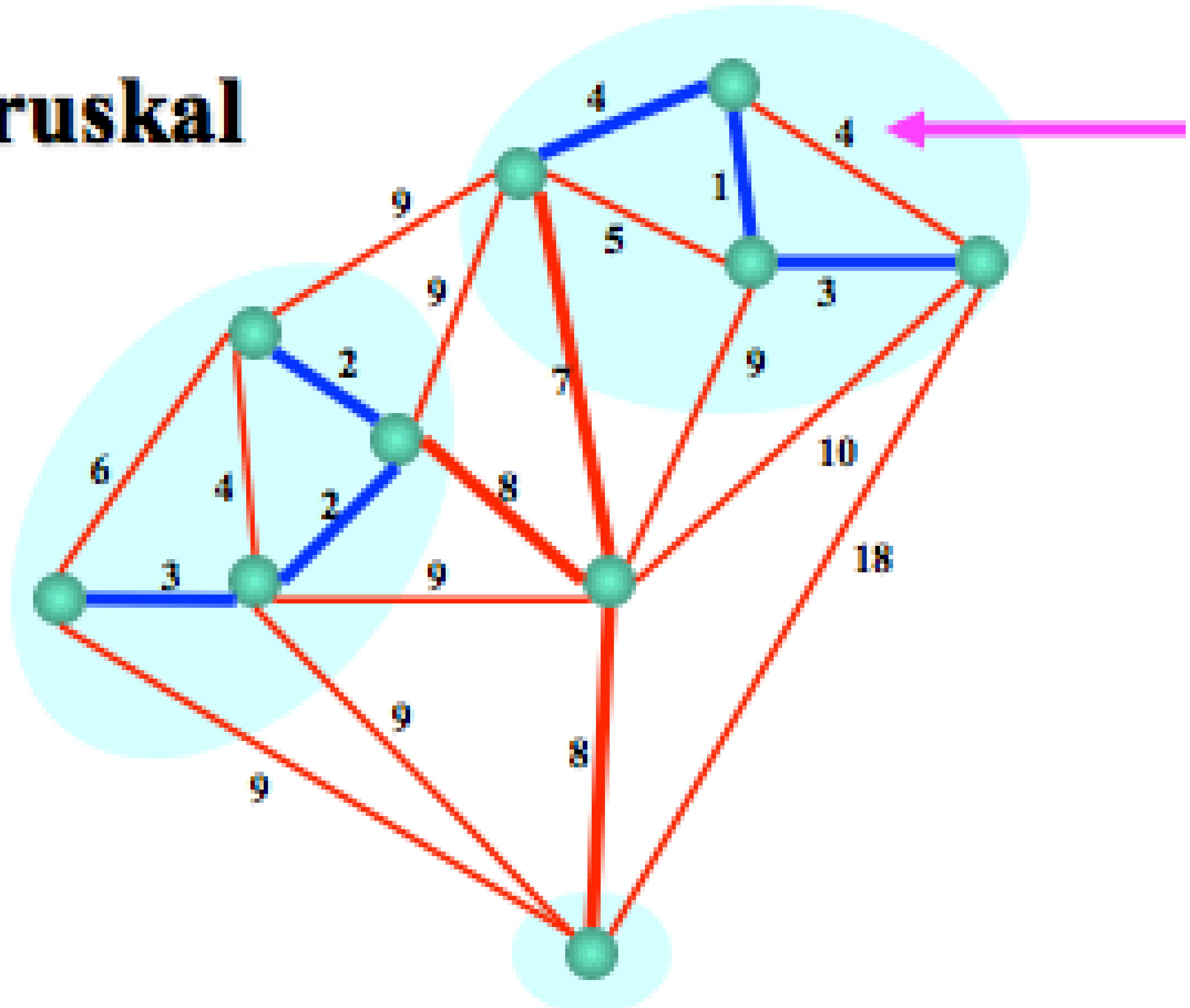
Kruskal



Sorted edge weights

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

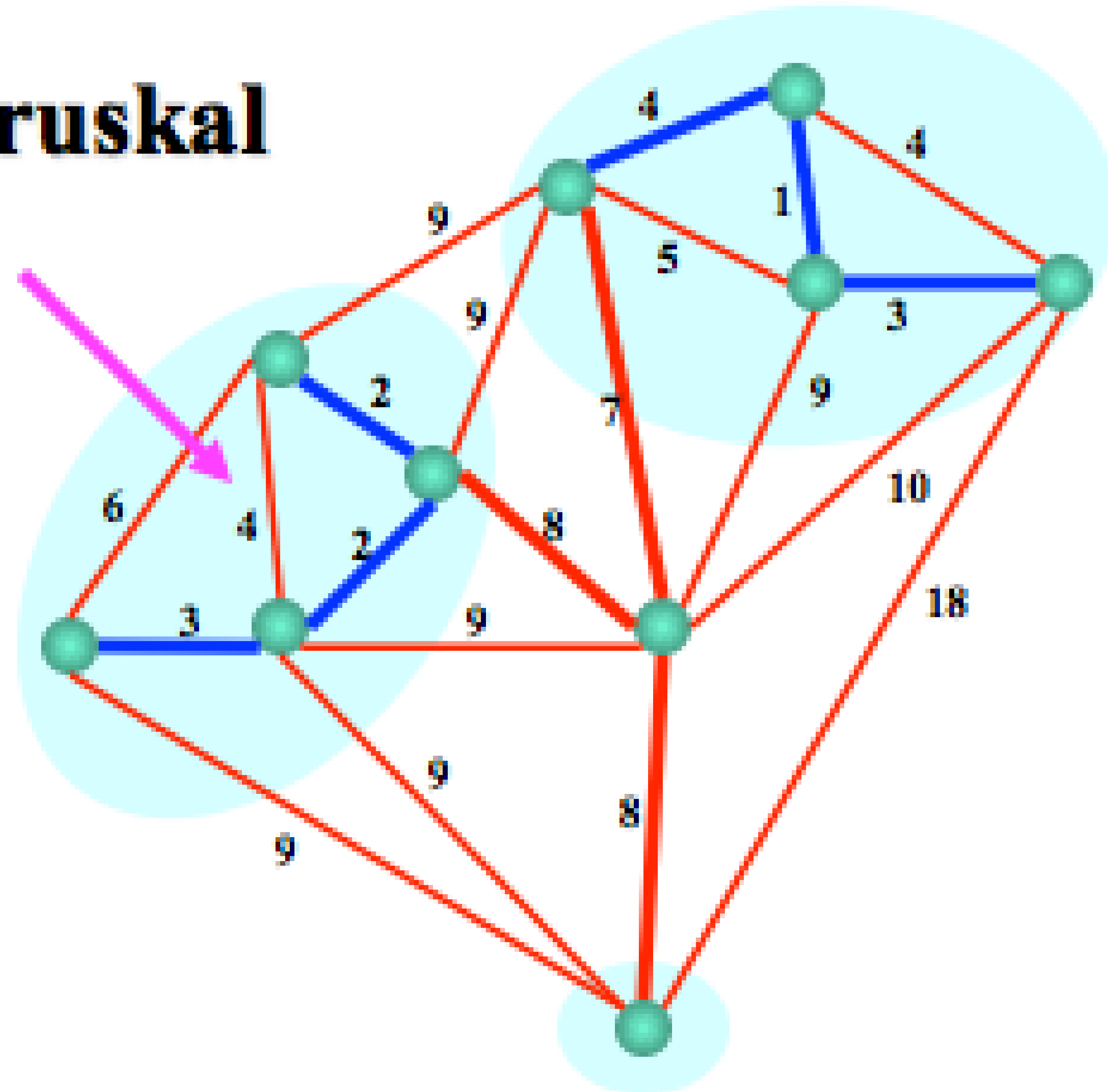
Kruskal



Sorted edge weights

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

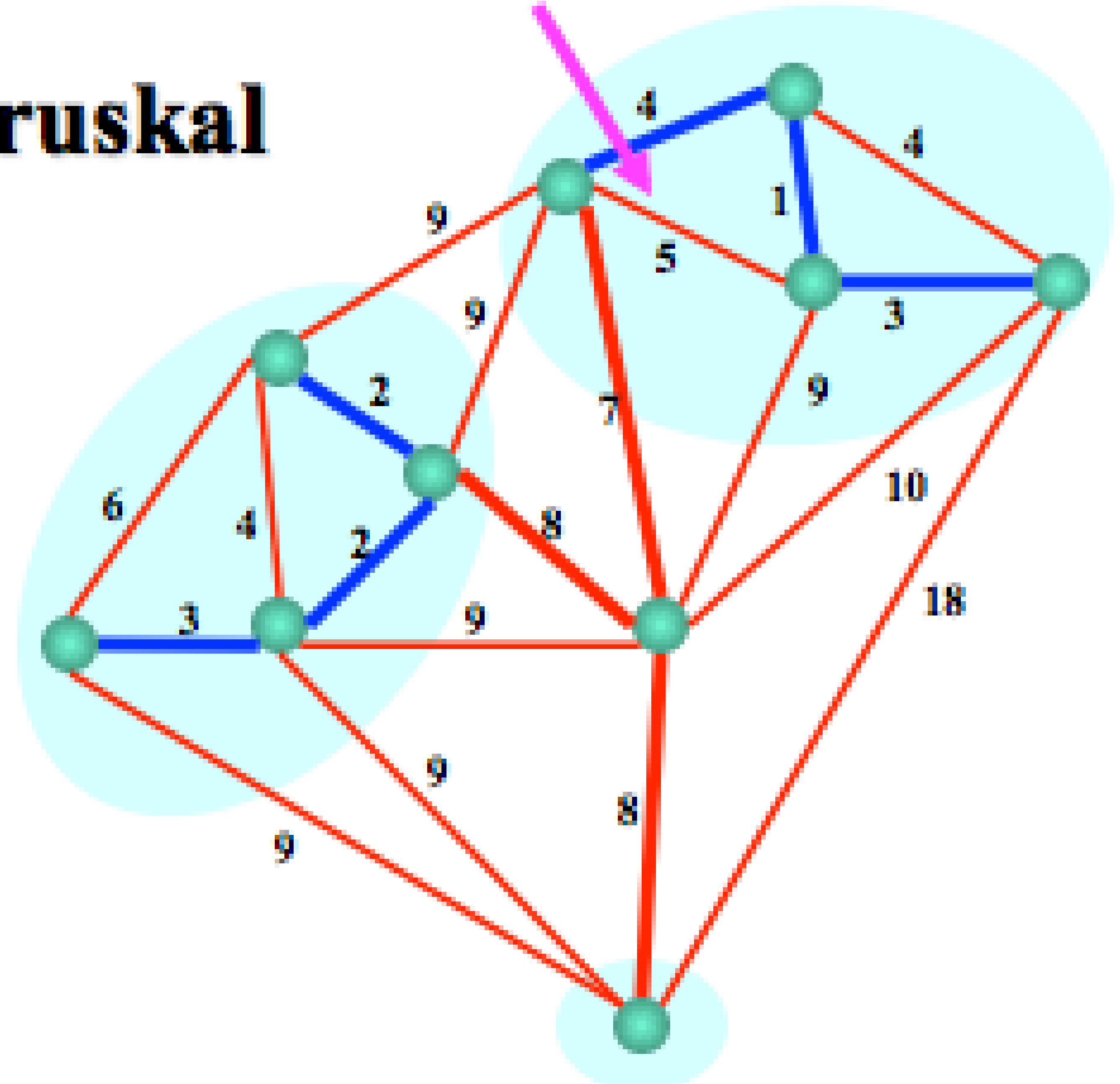
Kruskal



Sorted edge weights

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

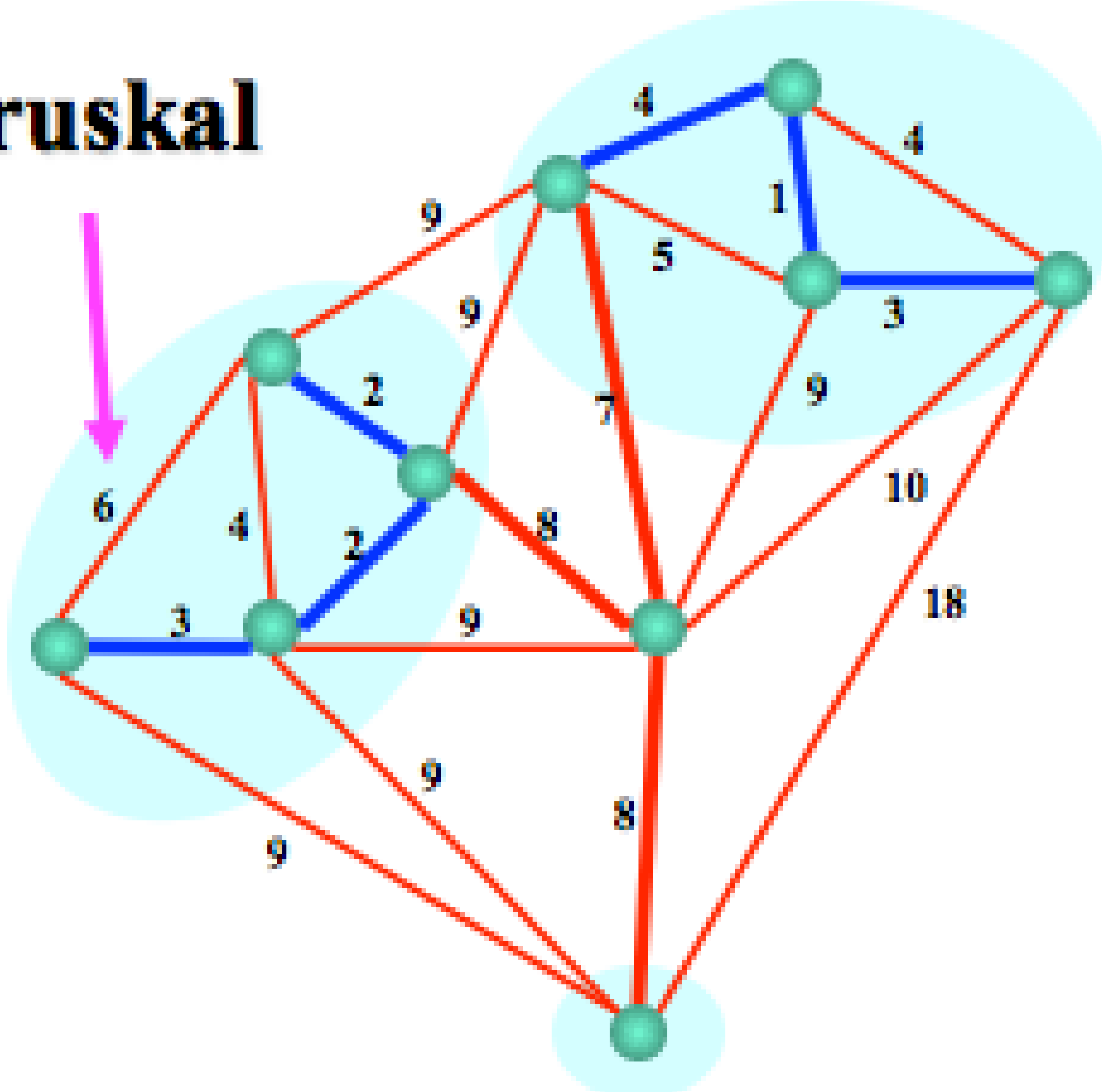
Kruskal



Sorted edge weights

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

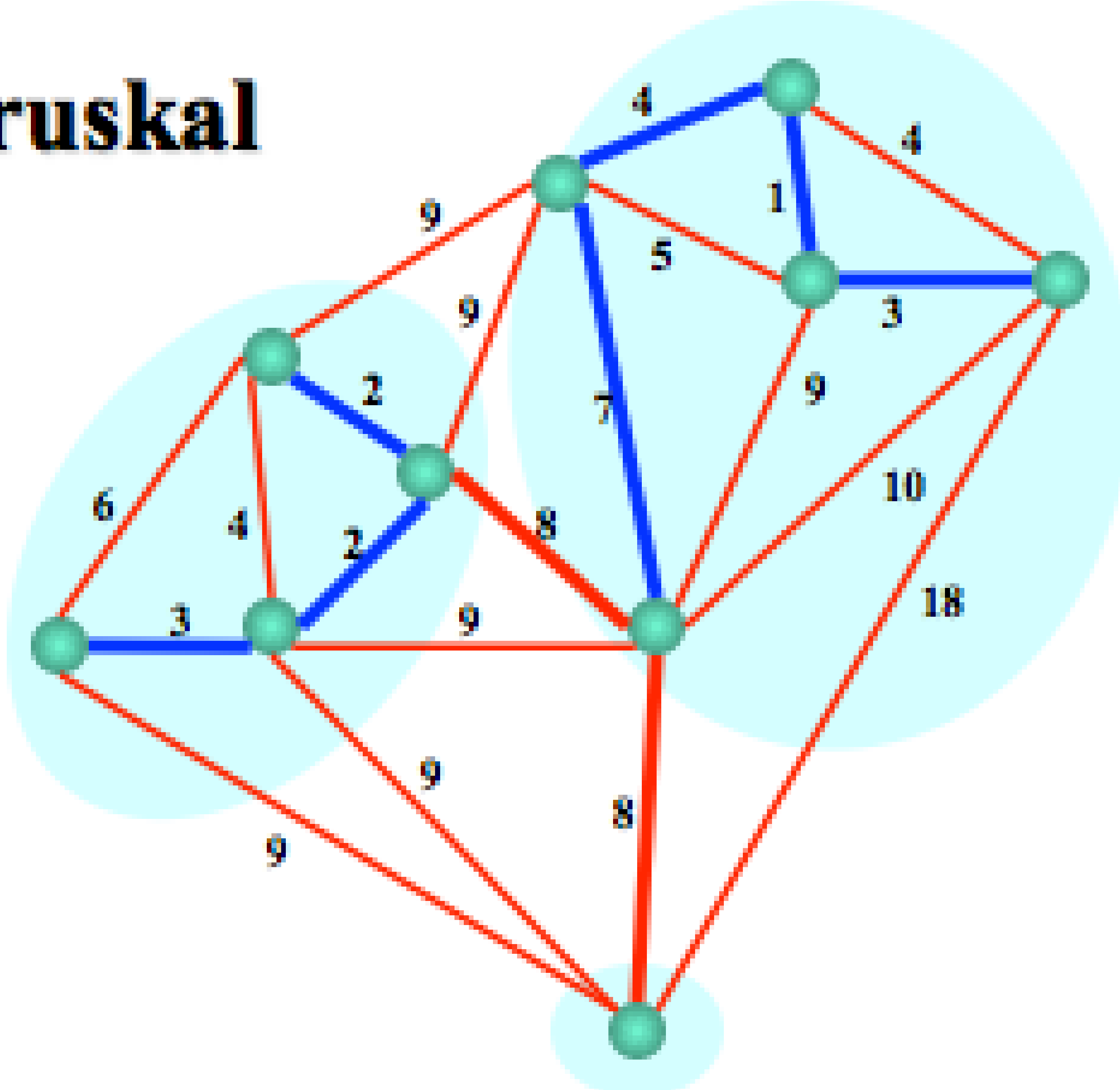
Kruskal



Sorted edge weights

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

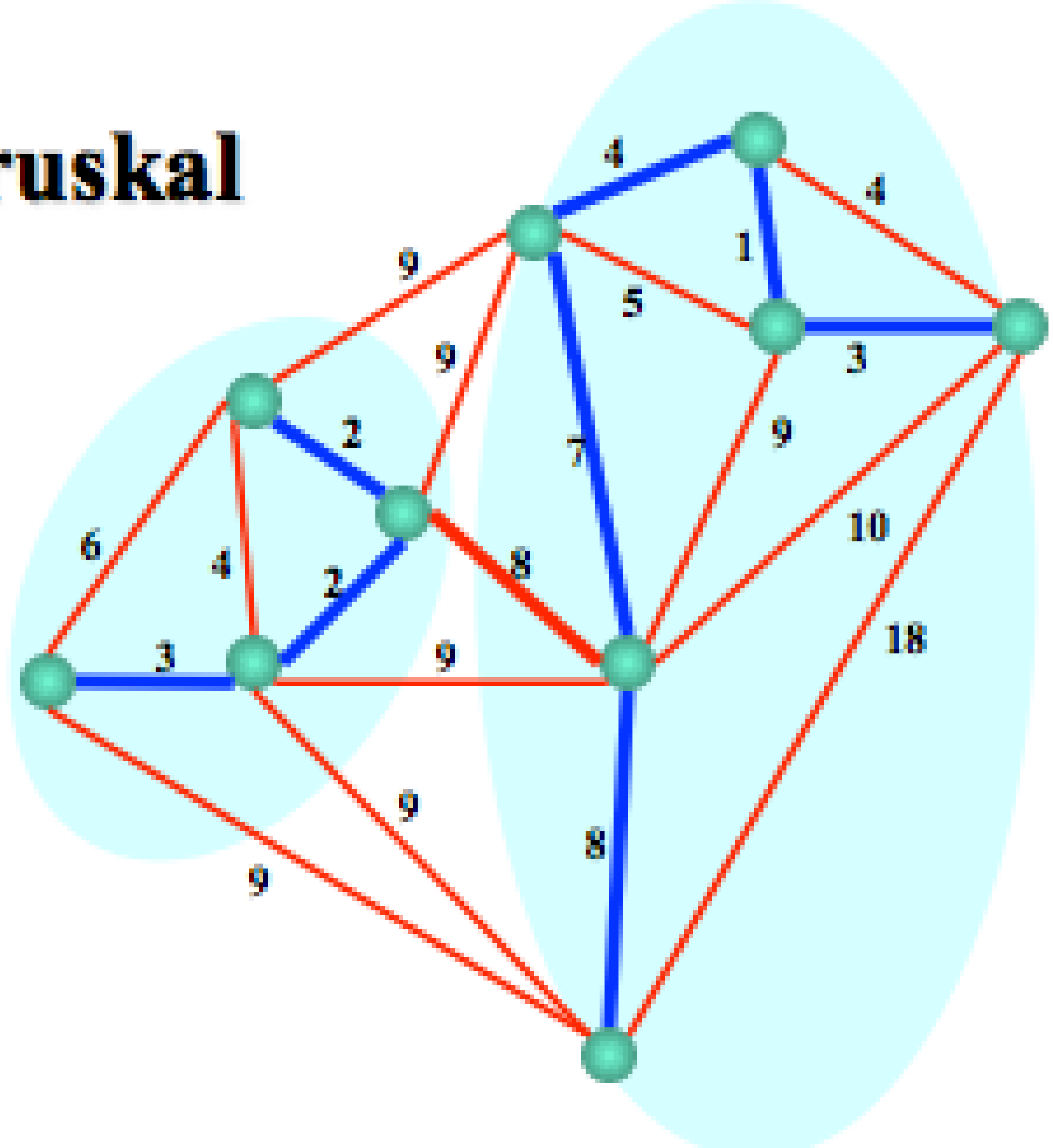
Kruskal



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

Sorted edge weights

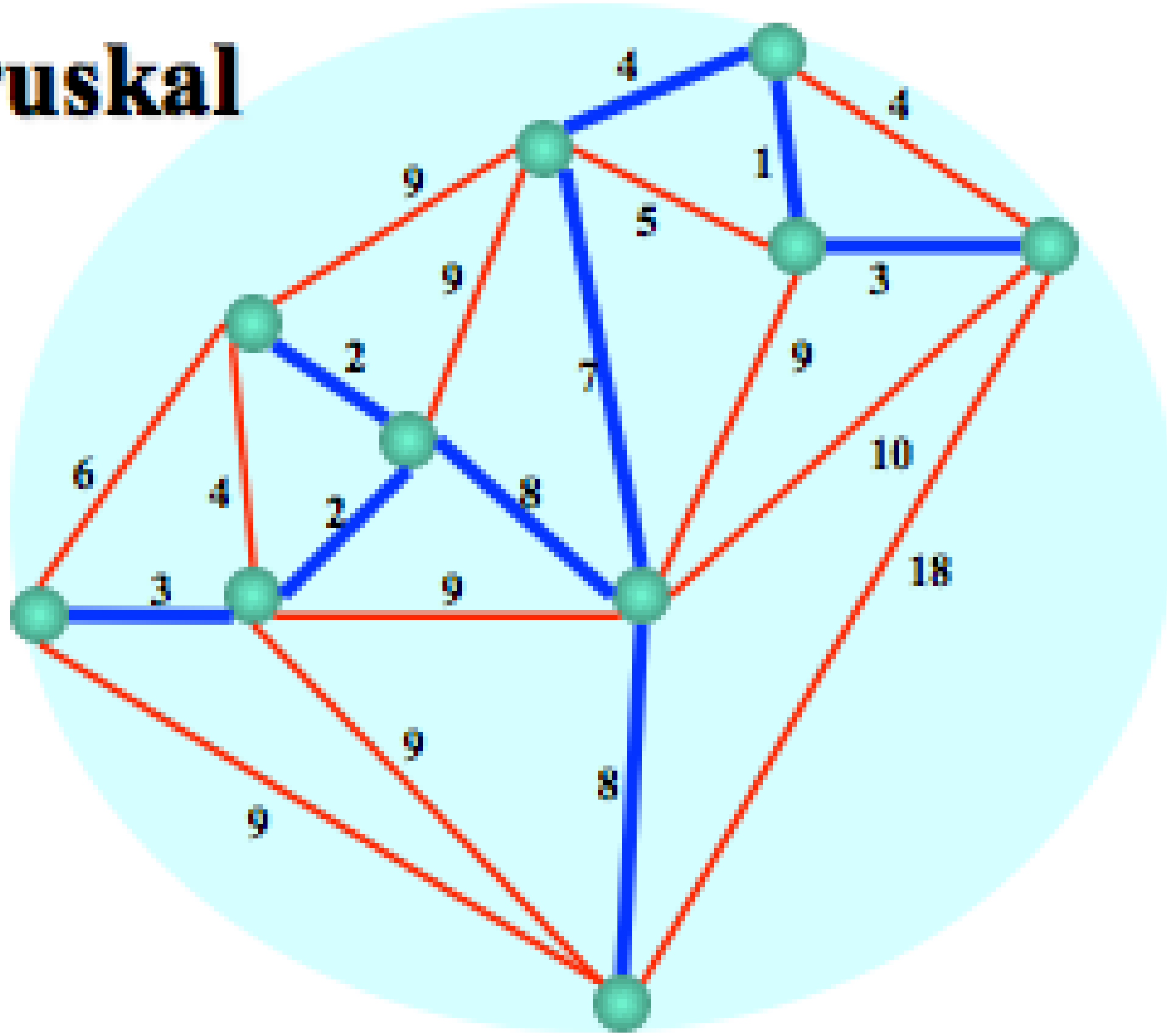
Kruskal



Sorted edge weights

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

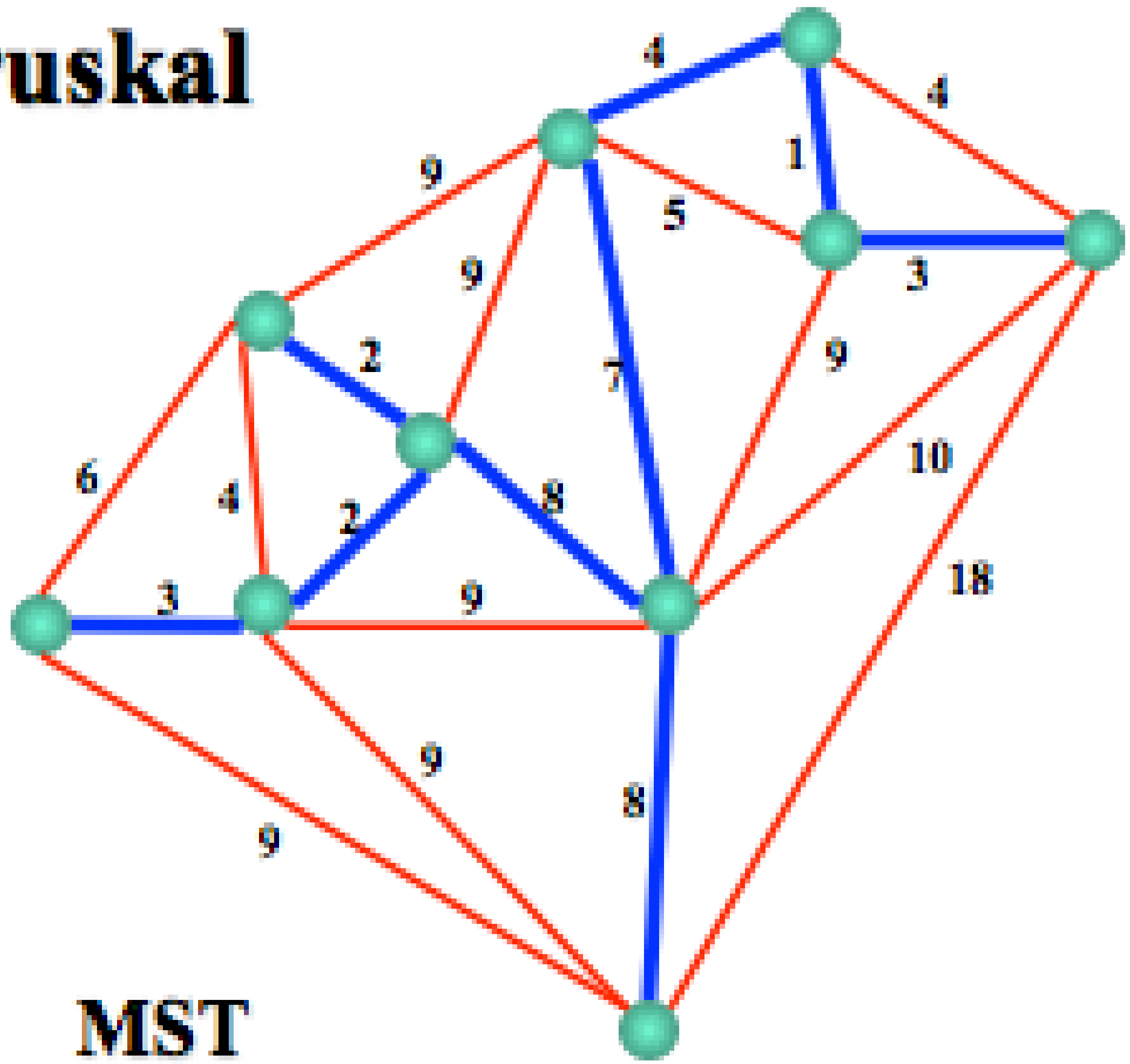
Kruskal



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

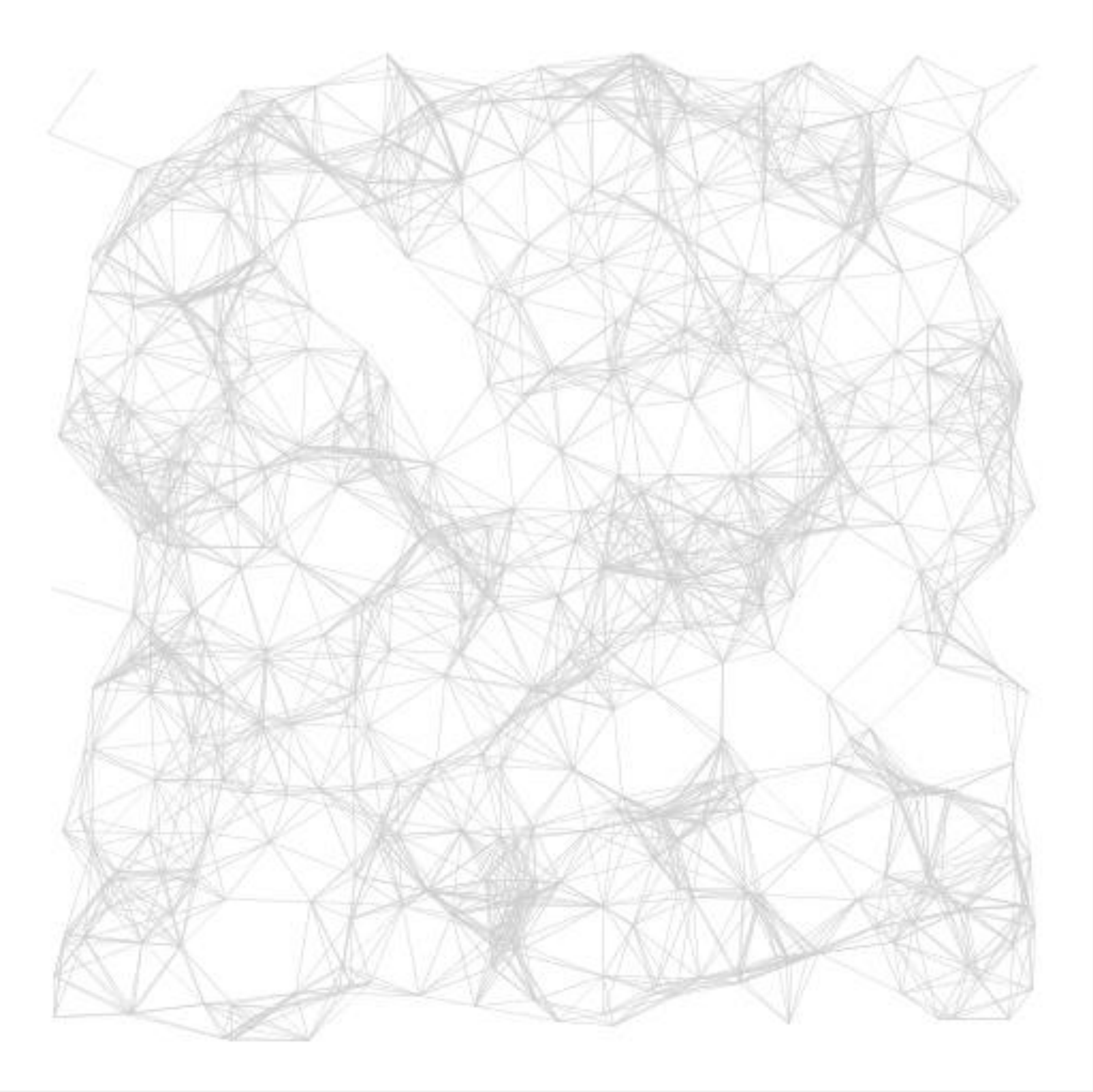


MST

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's algorithm: visualization



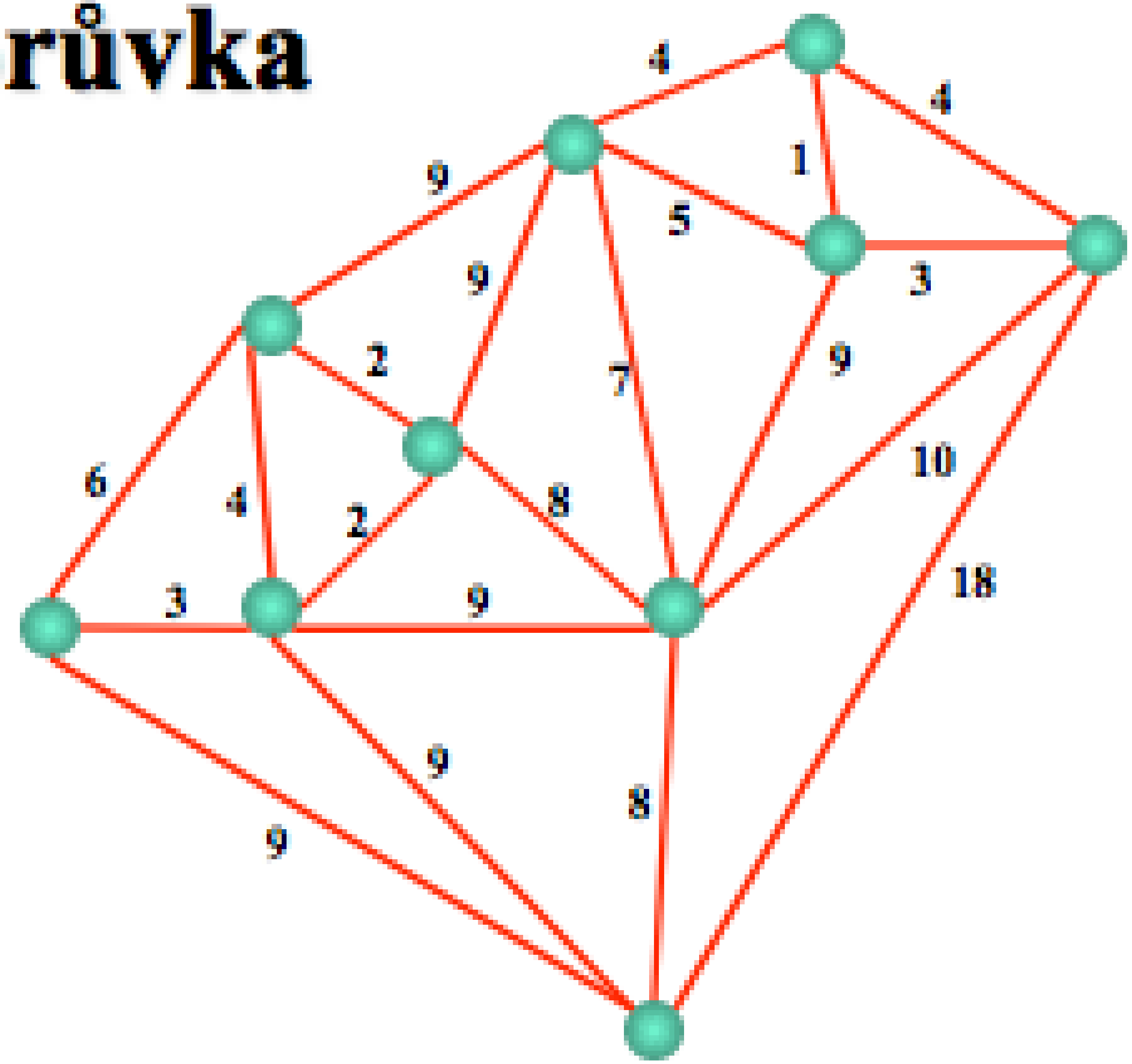
Kruskal’s Algorithm Idea

- 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

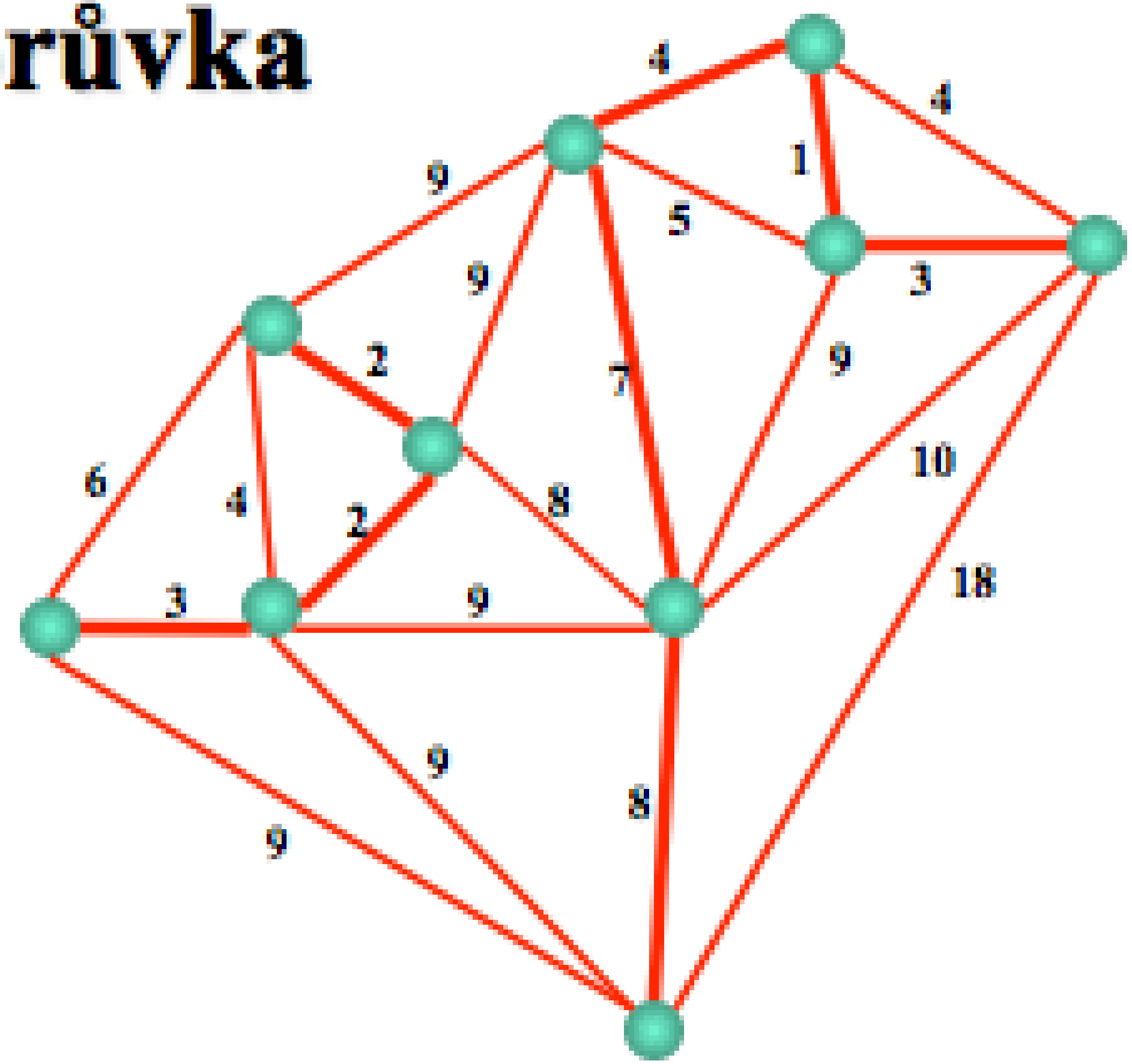
- Often assume every edge has a unique weight.
- Initially, each vertex is considered a separate component.
- The algorithm merges disjoint components as follows; repeating the step until only one component exists.
- In each step, every component is merged with some other using the cheapest outgoing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

Borůvka

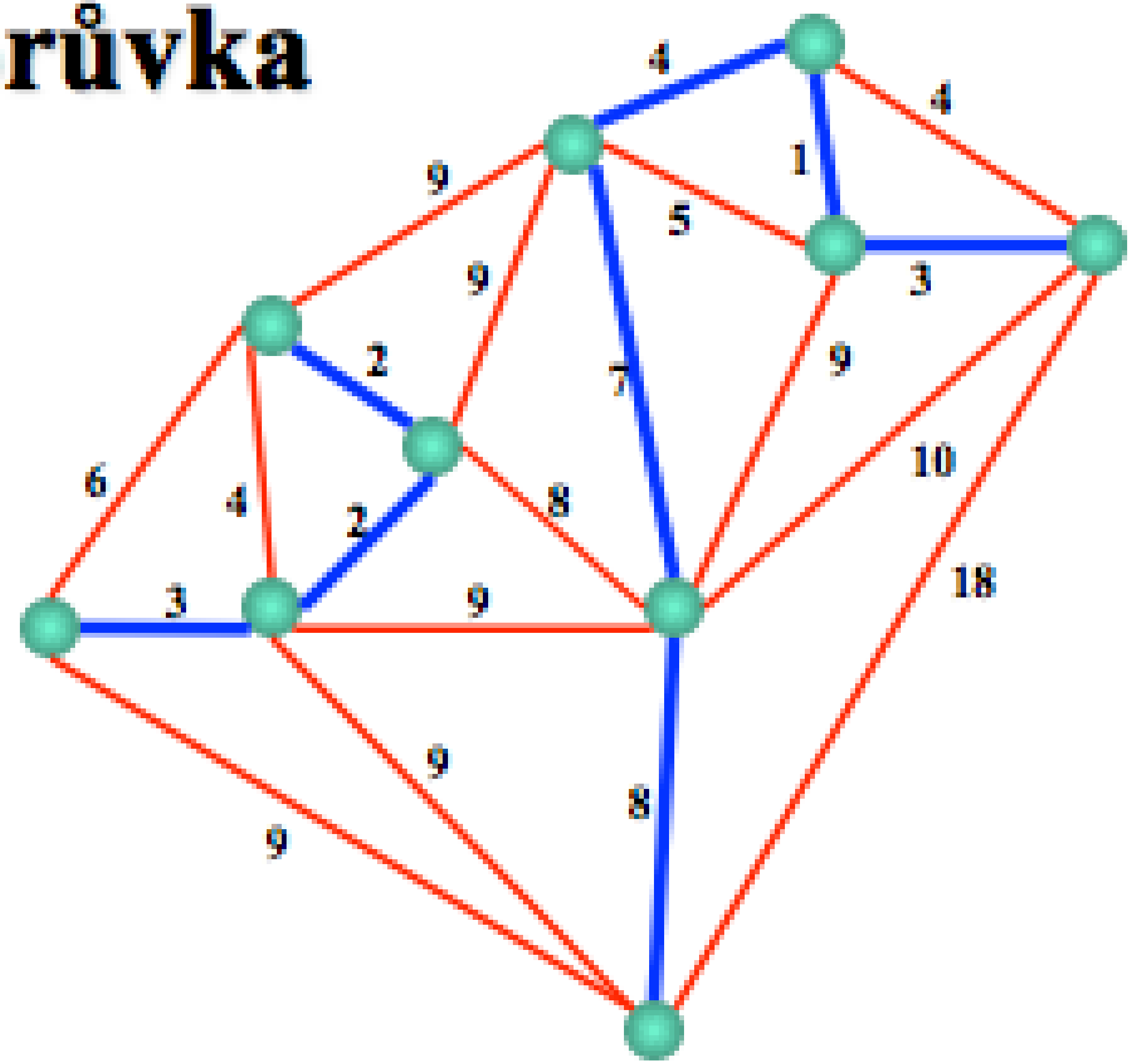


Every vertex is a tree

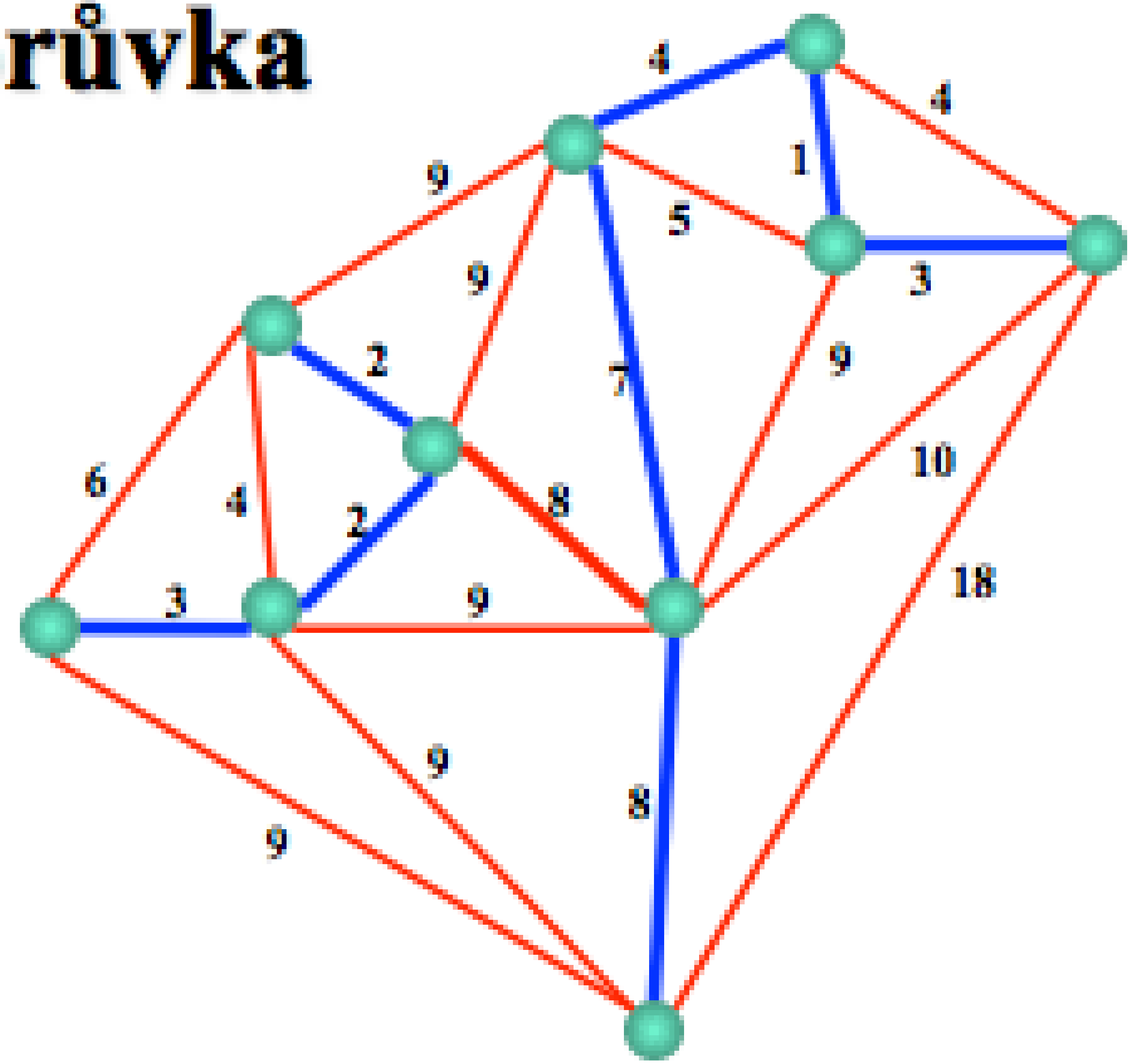
Borůvka



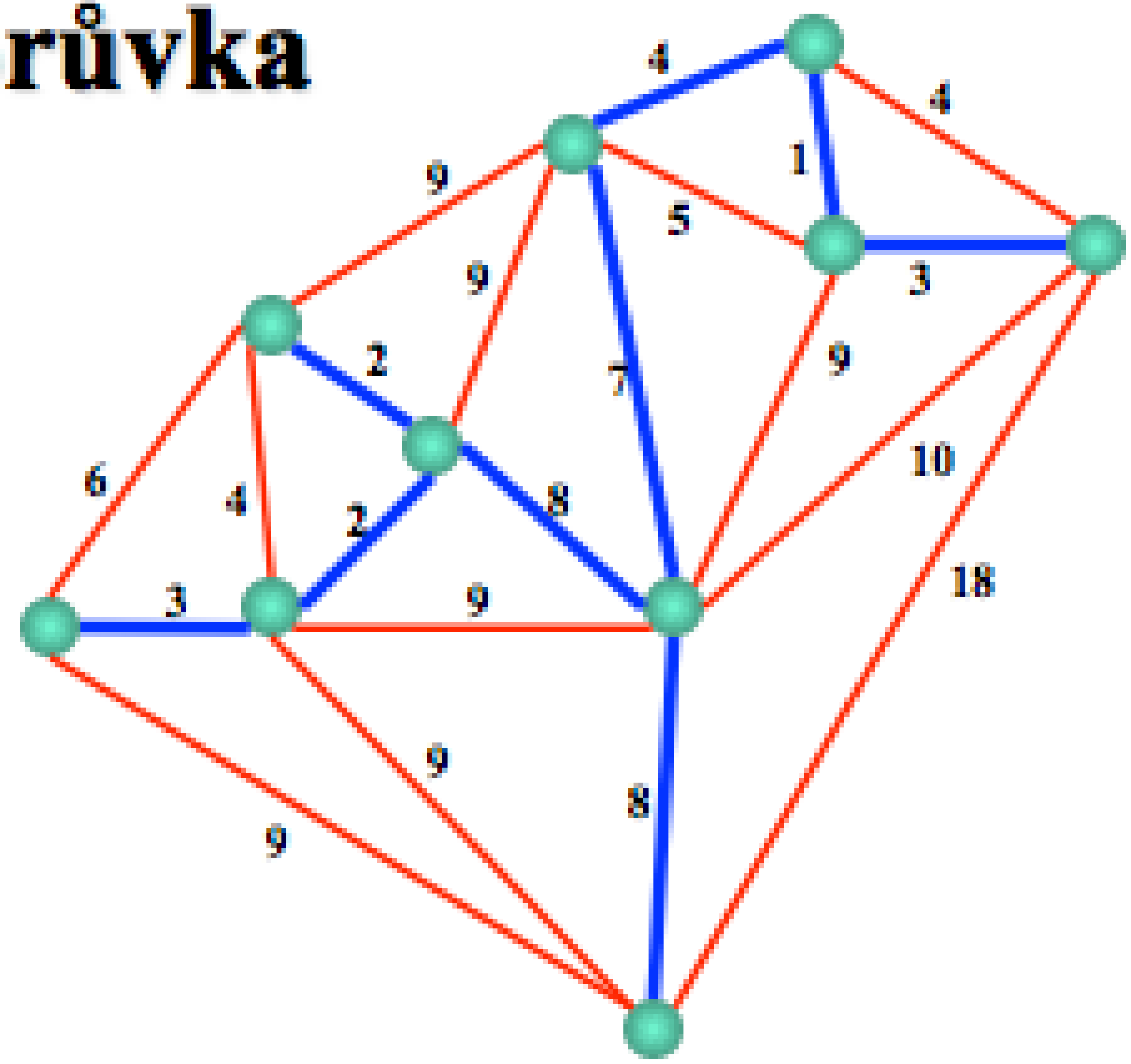
Borůvka



Borůvka



Borůvka



Borůvka's Algorithm Idea

- 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

- Why do these algorithm ideas work (and produce correct MSTs)?

- How do we implement these algorithms efficiently? What are good data structures?