14 - Graph Algorithms (part 1)

[KOMS119602] & [KOMS120403]

Design and Analysis of Algorithm (2021/2022)

Dewi Sintiari

Prodi S1 Ilmu Komputer
Universitas Pendidikan Ganesha

Week 18-21 April 2022

Table of contents

The contents of this lecture is extracted from the slides of Robert Sedgewick (Analysis of Algorithms)

Graphs

Graph is a mathematical data structure that consists of set of nodes/vertices connected by edges.

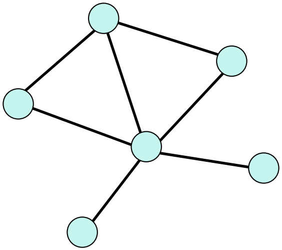


Figure: A (undirected) graph

Graphs

Vertices in the graph can have an orientation, and the graph is called a directed graph.

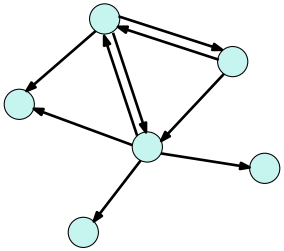


Figure: A directed graph

Graphs

Some terminology:

- Degree (of a vertex): # edges incident to that node
- In-degree (for directed graph): # edges coming to the vertex
- Out-degree: # edges going from the vertex
- Path: sequence of vertices/edges from one vertex to another
- Connected: if between two vertices, there is a path
- Cycle: a path that starts and ends at the same vertex
- Tree: connected graph that contains no cycle (acyclic)

Minimum Spanning Tree

What is a tree?

A tree is a connected acyclic graph.

- Connected means there is a path between two vertices in the graph.
- Acyclic means does not contain a cycle.

A spanning tree of an undirected graph G is a subgraph T that is a tree (connected, acyclic graph) and span G (includes all $V(G)$).

Minimum spanning tree (MST)

Problem: given an undirected graph G with positive edge weights. Find a minimum weight spanning tree

Theorem (Cayley, 1889)

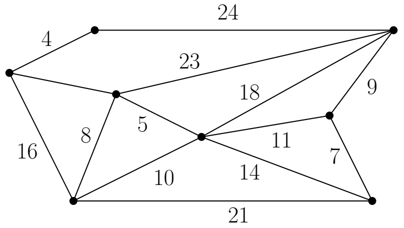
There are n^{n-2} spanning trees on the complete graph on n vertices.

Minimum spanning tree (MST)

Problem: given an undirected graph G with positive edge weights. Find a minimum weight spanning tree

Theorem (Cayley, 1889)

There are n^{n-2} spanning trees on the complete graph on n vertices.



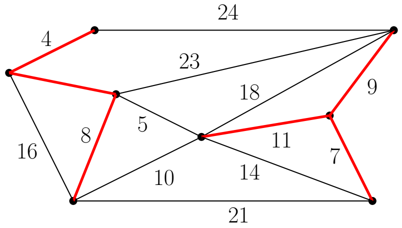
graph G with weighted edges

Minimum spanning tree (MST)

Problem: given an undirected graph G with positive edge weights. Find a minimum weight spanning tree

Theorem (Cayley, 1889)

There are n^{n-2} spanning trees on the complete graph on n vertices.



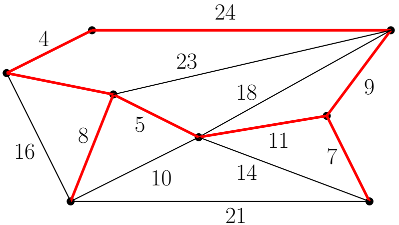
not connected

Minimum spanning tree (MST)

Problem: given an undirected graph G with positive edge weights. Find a minimum weight spanning tree

Theorem (Cayley, 1889)

There are n^{n-2} spanning trees on the complete graph on n vertices.



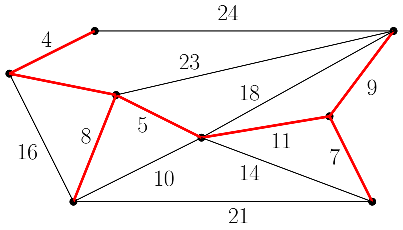
not acyclic

Minimum spanning tree (MST)

Problem: given an undirected graph G with positive edge weights. Find a minimum weight spanning tree

Theorem (Cayley, 1889)

There are n^{n-2} spanning trees on the complete graph on n vertices.



spanning tree T with cost $50 = 4 + 6 + 8 + 5 + 11 + 9 + 7$

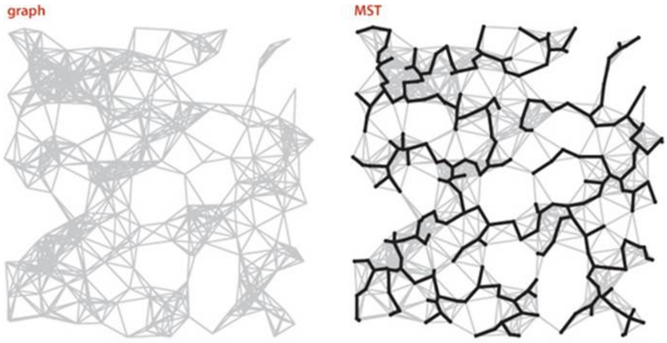
Minimum spanning tree (MST)

The origin of MST

- Problem of finding an efficient coverage in Western Moravia in Brno. To find the most economical construction of electrical power network.
- Czech scientist Otakar Borüvka developed the first known algorithm for finding a minimum spanning tree, in 1926.

MST in real life (1)

A large graph and its MST

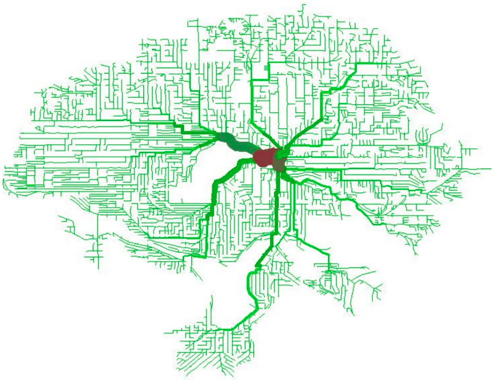


A 250-vertex Euclidean graph (with 1,273 edges) and its MST

source: `https://apprize.best/science/algorithms_2/algorithms_2.files/image091.jpg`

MST in real life (2)

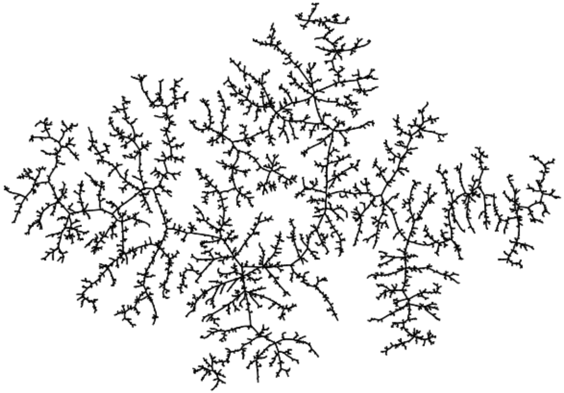
MST of bicycle routes in North Seattle



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

MST in real life (3)

MST of a random graph



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

Applications of MST

- Network design
    - *telephone, electrical, hydraulic, TV cable, computer, or road networks in satellite*
- Cluster analysis
- Real-time face verification
- Approximation algorithms for NP-hard problems
    - *example: the Traveling Salesman Problem*

Solving MST with brute-force approach

How do we solve MST with brute-force approach?

Brute-force algorithm:

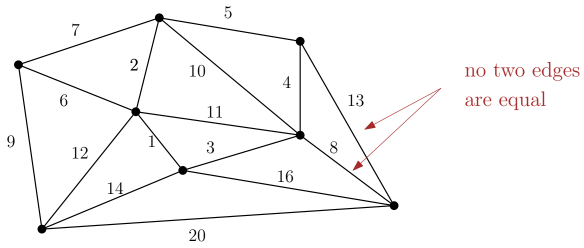
1. List all possible spanning tree
2. Compute the weight of every spanning tree
3. Take the spanning tree that has the minimum weight

Solving MST with greedy algorithm (1)

1. For simplification, we assume the followings:

- The graph is connected
- Edge weights are distinct
- Edge weights are not necessarily distances

Consequence: with the assumption, an MST exists and it is unique (the only one)

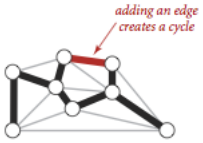


Solving MST with greedy algorithm (2)

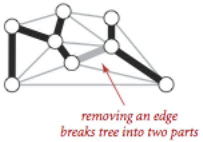
2. Underlying principle:

The following properties hold for every spanning tree in a graph

- Adding an edge that connects two vertices in a tree creates a unique cycle.



- Removing an edge from a tree breaks it into two separate subtrees.

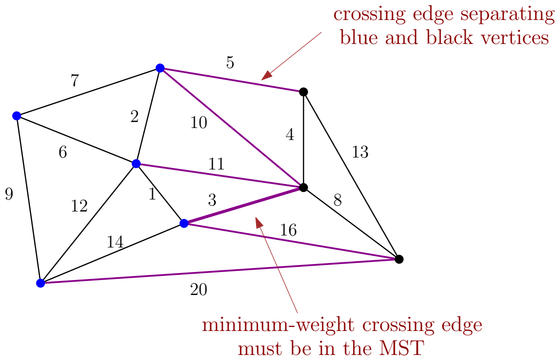


Solving MST with greedy algorithm (3)

3. Cut property

A cut in G is a partition of $V(G)$ into two (nonempty) sets A and B.

A crossing edge connects a vertex in A and a vertex in B.


crossing edge separating blue and black vertices
5
7
2
10
4
13
6
11
9
1
3
8
12
16
14
20
minimum-weight crossing edge must be in the MST

Solving MST with greedy algorithm (4)

3. Cut property

A cut in G is a partition of $V(G)$ into two (nonempty) sets A and B.
A crossing edge connects a vertex in A and a vertex in B.

Lemma (Cut property)

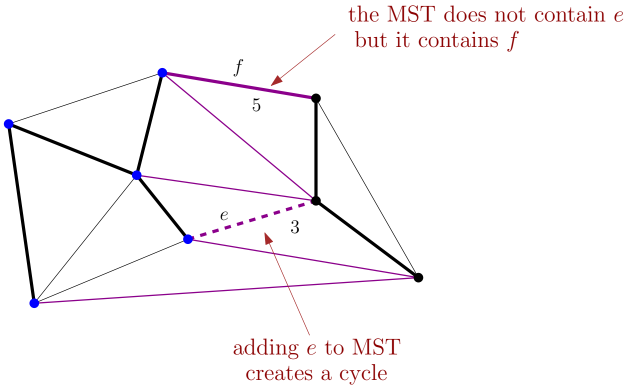
Given any cut, the crossing edge of minimum weight is in the MST.

Proof. Suppose minimum-weight crossing edge e is not in the MST T.

- Adding e to T creates a cycle.
- Some other edge f in the cycle must be a crossing edge.
- $T' = T - \{e\} + \{f\}$ is also a spanning tree, and $\text{cost}(T') < \text{cost}(T)$.
- Contradiction

Solving MST with greedy algorithm (5)

3. Cut property



the MST does not contain e but it contains f
f
5
e
3
adding e to MST creates a cycle

Greedy MST algorithm

Input: G: undirected graph, w: weight function
Output: a set of edges that forms a MST

- Start with all edges colored gray (gray color indicates the edges are not in the solution);
- Find cut with no black crossing edges; color its minimum-weight edge black (black color indicates that the edge is included in the solution);
- Repeat until $|V| - 1$ edges are colored black (all black edges are the MST).

Greedy MST algorithm

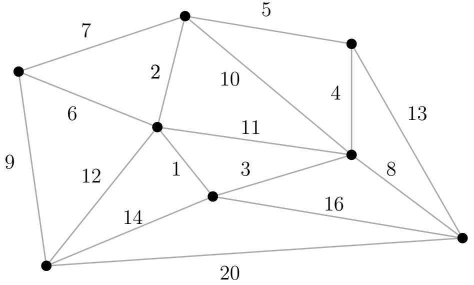


Figure: The input graph *G*

Greedy MST algorithm

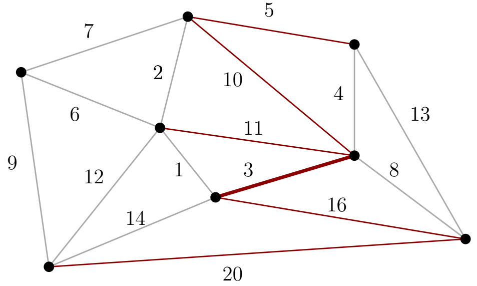


Figure: The red edges form a cut of G

Greedy MST algorithm

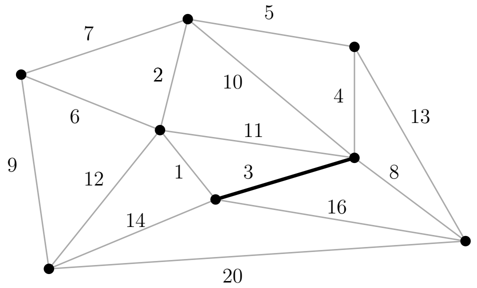


Figure: Take the minimum-weight edge in the cut and color it black

Greedy MST algorithm

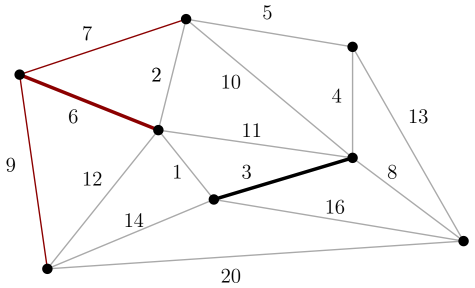


Figure: Choose another cut

Greedy MST algorithm

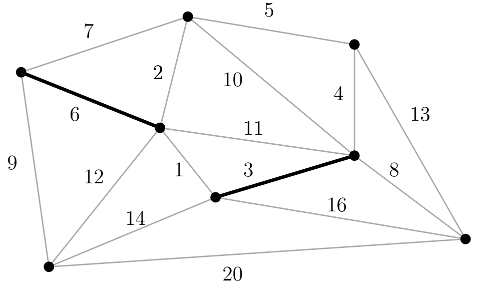


Figure: Take the minimum-weight edge in the cut and color it black

Greedy MST algorithm

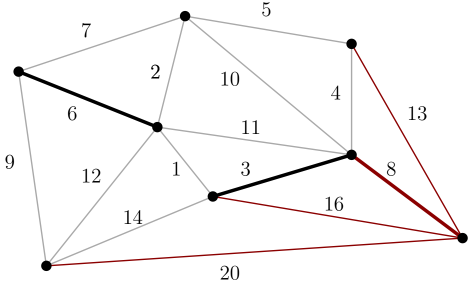


Figure: Choose another cut

Greedy MST algorithm

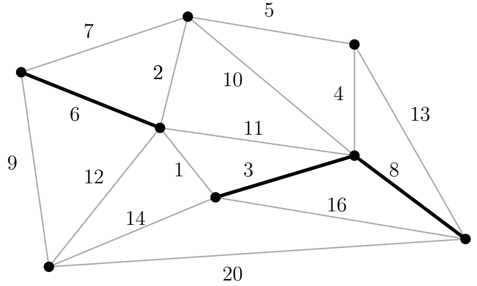


Figure: Take the minimum-weight edge in the cut and color it black

Greedy MST algorithm

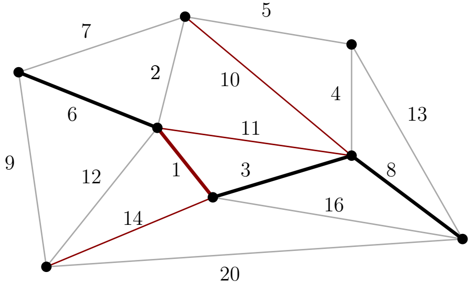


Figure: Choose another cut

Greedy MST algorithm

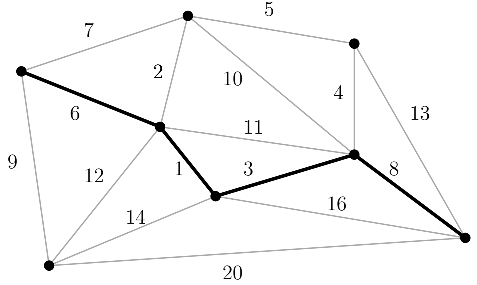


Figure: Take the minimum-weight edge in the cut and color it black

Greedy MST algorithm

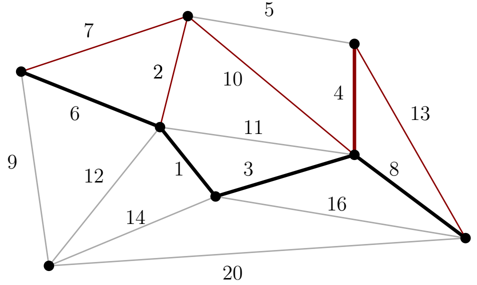


Figure: Choose another cut

Greedy MST algorithm

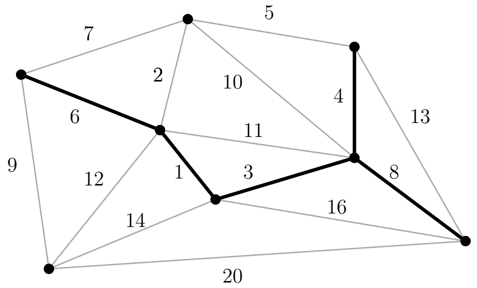


Figure: Take the minimum-weight edge in the cut and color it black

Greedy MST algorithm

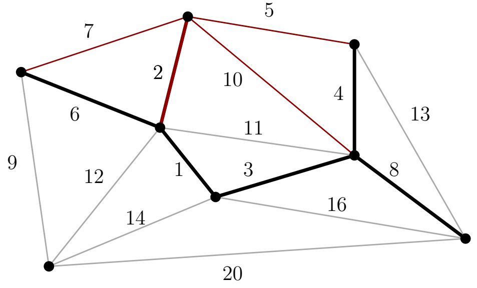


Figure: Choose another cut

Greedy MST algorithm

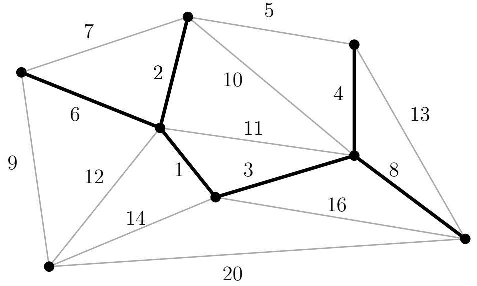


Figure: Take the minimum-weight edge in the cut and color it black

Greedy MST algorithm

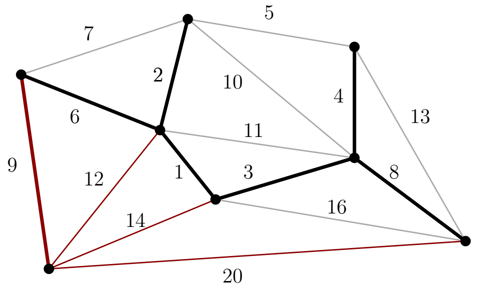


Figure: Choose another cut

Greedy MST algorithm

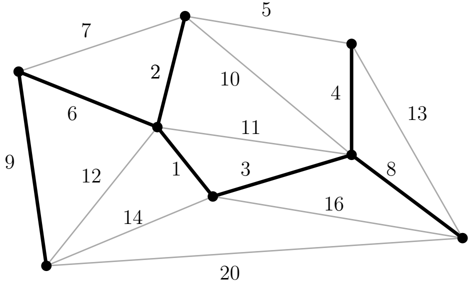


Figure: Take the minimum-weight edge in the cut and color it black

Greedy MST algorithm: correctness proof

Theorem

The greedy algorithm computes the MST.

Proof.

Greedy MST algorithm: correctness proof

Theorem

The greedy algorithm computes the MST.

Proof.

- Any edge colored black is in the MST (by the cut property).

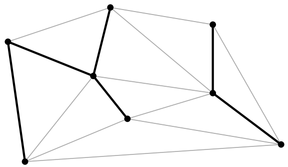
Greedy MST algorithm: correctness proof

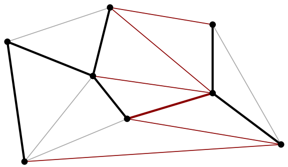
Theorem

The greedy algorithm computes the MST.

Proof.

- Any edge colored black is in the MST (by the cut property).
- Fewer than $|V| - 1$ black edges $\Rightarrow$ cut with no black crossing edges. (consider cut whose vertices are one connected component)


fewer than $|V| - 1$ edges colored black


a cut with no black crossing edges

Kruskal's and Prim's algorithms

Greedy MST algorithm: more efficient implementations

The greedy algorithm that is based on the "Cut property" is difficult to implement, and the efficiency if the algorithm can still be improved. We ask the following questions:

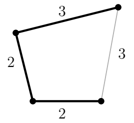
- How to efficiently choose the cut?
- How to efficiently find the minimum-weight edge?

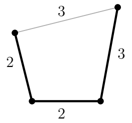
1. Kruskal's algorithm
2. Prim's algorithm
3. Borüvka's algorithm (this will not be discussed here)

Without the simplifying assumption...

What if edge weights are not distinct?

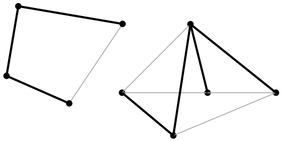
The MST is not unique. But the Greedy MST algorithm still works (*similar proof*)





What if the graph is not connected?

Get Minimum Spanning Forest (MST for each connected component)



Kruskal's algorithm

Kruskal's algorithm

This is to find a minimum spanning tree of an undirected edge-weighted graph. Note that it finds a minimum spanning forest if the graph is not connected).

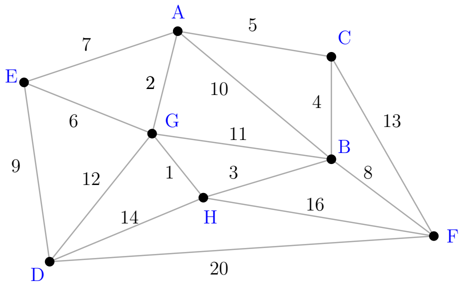
This algorithm uses Greedy technique, and it first appeared in Proceedings of the AMS, in 1956, and was written by Joseph Kruskal.

Input: a weighted graph G
Output: a set of vertices that forms an MST of G

Algorithm.

- Consider edges in ascending order of weight;
- Add next edge to tree T unless doing so would create a cycle.

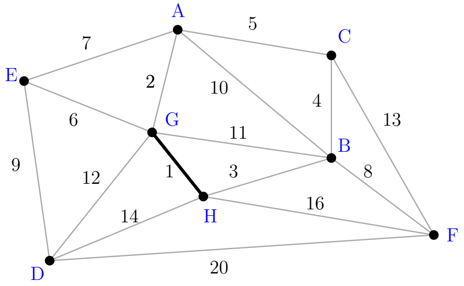
Kruskal's MST algorithm



Edge	Weight
GH	1
AG	2
BH	3
BC	4
AC	5
EG	6
AE	7
BF	8
DE	9
AB	10
BG	11
DG	12
CF	13
DH	14
FH	16
DF	20

Figure: The edges are ordered based on their weights

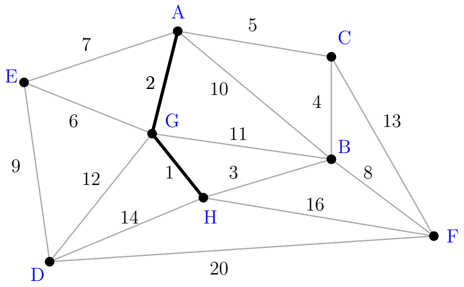
Kruskal's MST algorithm



Edge	**Weight**
GH	1

Figure: Edge *GH* is included in the MST

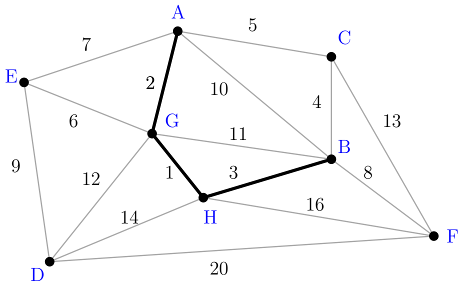
Kruskal's MST algorithm



Edge	Weight
GH	1
AG	2

Figure: Edge *AG* is included in the MST

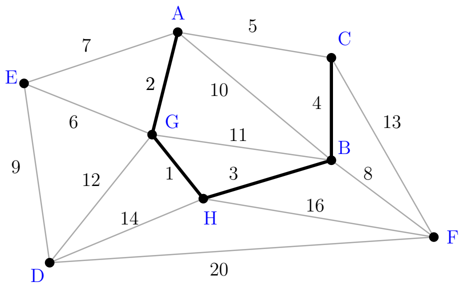
Kruskal's MST algorithm



Edge	**Weight**
GH	1
AG	2
BH	3

Figure: Edge *BH* is included in the MST

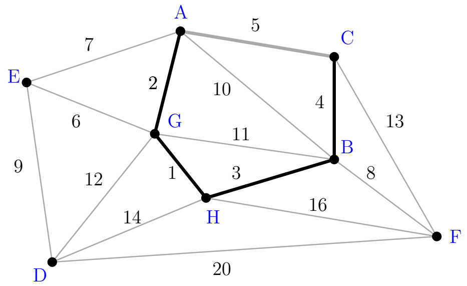
Kruskal's MST algorithm



Edge	Weight
GH	1
AG	2
BH	3
BC	4

Figure: Edge *BC* is included in the MST

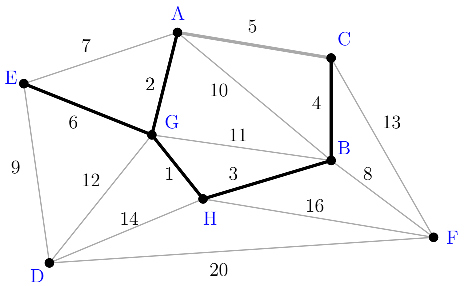
Kruskal's MST algorithm



Edge	**Weight**
GH	1
AG	2
BH	3
BC	4
AC	5

Figure: Edge *AC* is **not** included in the MST, otherwise it would create a cycle

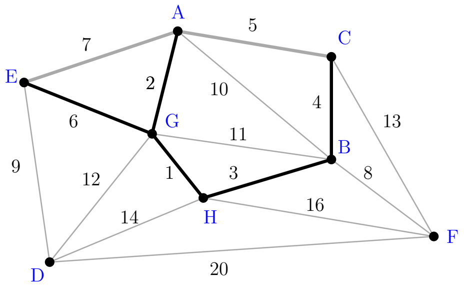
Kruskal's MST algorithm



Edge	Weight
GH	1
AG	2
BH	3
BC	4
AC	5
EG	6

Figure: Edge *EG* is included in the MST

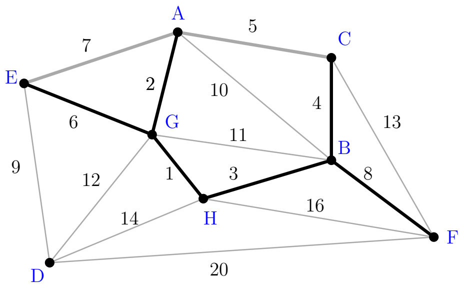
Kruskal's MST algorithm



Edge	Weight
GH	1
AG	2
BH	3
BC	4
AC	5
EG	6
AE	7

Figure: Edge *AE* is **not** included in the MST, otherwise it would create a cycle

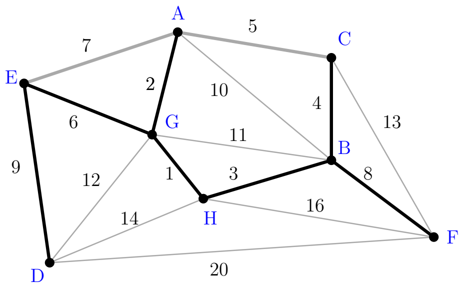
Kruskal's MST algorithm



Edge	Weight
GH	1
AG	2
BH	3
BC	4
AC	5
EG	6
AE	7
BF	8

Figure: Edge *BF* is included in the MST

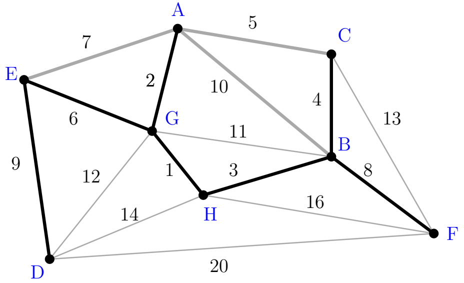
Kruskal's MST algorithm



Edge	Weight
GH	1
AG	2
BH	3
BC	4
AC	5
EG	6
AE	7
BF	8
DE	9

Figure: Edge *DE* is included in the MST

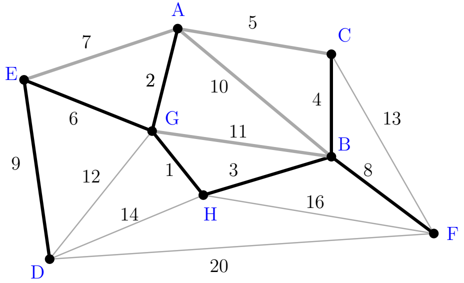
Kruskal's MST algorithm



Edge	Weight
GH	1
AG	2
BH	3
BC	4
AC	5
EG	6
AE	7
BF	8
DE	9
AB	10

Figure: Edge *AB* is **not** included in the MST, otherwise it would create a cycle

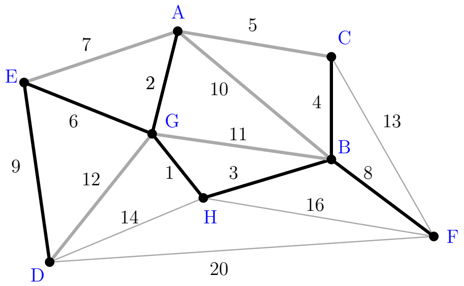
Kruskal's MST algorithm



Edge	Weight
GH	1
AG	2
BH	3
BC	4
AC	5
EG	6
AE	7
BF	8
DE	9
AB	10
BG	11

Figure: Edge *BG* is **not** included in the MST, otherwise it would create a cycle

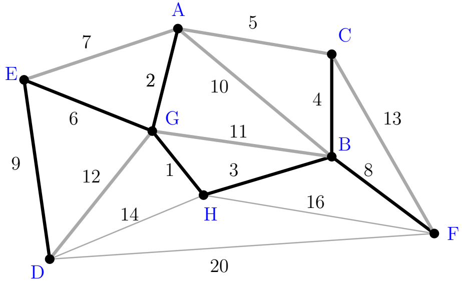
Kruskal's MST algorithm



Edge	**Weight**
GH	1
AG	2
BH	3
BC	4
AC	5
EG	6
AE	7
BF	8
DE	9
AB	10
BG	11
DG	12

Figure: Edge *DG* is **not** included in the MST, otherwise it would create a cycle

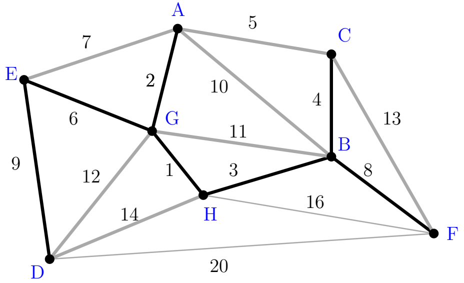
Kruskal's MST algorithm



Edge	Weight
GH	1
AG	2
BH	3
BC	4
AC	5
EG	6
AE	7
BF	8
DE	9
AB	10
BG	11
DG	12
CF	13

Figure: Edge *CF* is **not** included in the MST, otherwise it would create a cycle

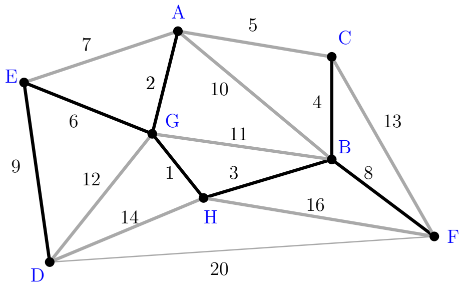
Kruskal's MST algorithm



Edge	**Weight**
GH	1
AG	2
BH	3
BC	4
AC	5
EG	6
AE	7
BF	8
DE	9
AB	10
BG	11
DG	12
CF	13
DH	14

Figure: Edge *DH* is **not** included in the MST, otherwise it would create a cycle

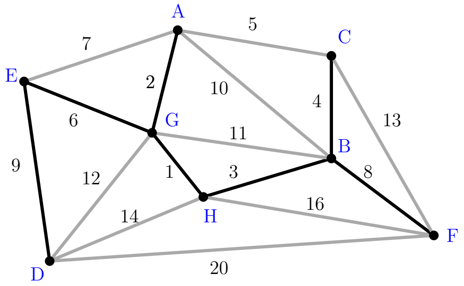
Kruskal's MST algorithm



Edge	Weight
GH	1
AG	2
BH	3
BC	4
AC	5
EG	6
AE	7
BF	8
DE	9
AB	10
BG	11
DG	12
CF	13
DH	14
FH	16

Figure: Edge *FH* is **not** included in the MST, otherwise it would create a cycle

Kruskal's MST algorithm



Edge	Weight
GH	1
AG	2
BH	3
BC	4
AC	5
EG	6
AE	7
BF	8
DE	9
AB	10
BG	11
DG	12
CF	13
DH	14
FH	16
DF	20

Figure: Edge *DF* is **not** included in the MST, otherwise it would create a cycle

Proof of correctness of Kruskal's MST algorithm

Theorem (Kruskal, 1956)

Kruskal's algorithm computes the MST.

Proof. Let T be the graph output by Kruskal's algorithm.

- T is a spanning tree. Follows from the algorithm (we do not choose an edge if it creates a cycle with the edges already chosen. T is connected, oth. the edge of smallest weight that crosses two components of T would have been chosen.
- T has the minimal weight among the other spanning trees. Follows from the **cut property**: given any cut, the crossing edge of minimum weight is in the MST.

Proof of correctness of Kruskal's MST algorithm

Question. How to check if adding an edge to T would create a cycle?

Alternative 1. Use DFS algorithm.

- $\mathcal{O}(n)$-time per cycle check, with $n = |V|$.
- $\mathcal{O}(mn)$-time overall, with $n = |V|$ adn $m = |E|$.

Alternative 2. Use the union-find data structure.

- Maintain a set for each connected component.
- If v and w are in same component, then adding edge vw creates a cycle.
- To add vw to T, merge sets containing v and w.

Prim's algorithm

Prim's algorithm

Prims algorithm is also a Greedy algorithm.

Algorithm

- It starts with an empty spanning tree;
- The idea is to maintain two sets of vertices;
- The first set contains the vertices already included in the MST, the other set contains the vertices not yet included;
- At every step, it considers all the edges that connect the two sets, and picks the minimum weight edge from these edges;
- After picking the edge, it moves the other endpoint of the edge to the set containing MST.

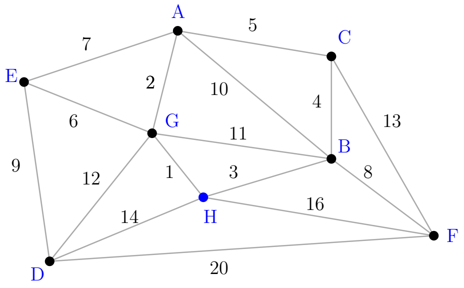
Prim's MST algorithm

Input: a weighted graph G
Output: a set of vertices that forms an MST of G

Algorithm:

- Initialize the solution set $T = \emptyset$;
- Start with an arbitrary vertex and greedily grow T;
- Add to T the minimum weight edge with exactly on endpoint in T;
- Repeat until T has size $|V| - 1$.

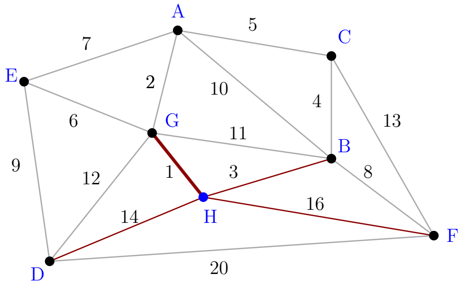
Prim's MST algorithm



Edge	Weight
GH	1
AG	2
BH	3
BC	4
AC	5
EG	6
AE	7
BF	8
DE	9
AB	10
BG	11
DG	12
CF	13
DH	14
FH	16
DF	20

Start with the weighted graph. Choose a random vertex, say H.

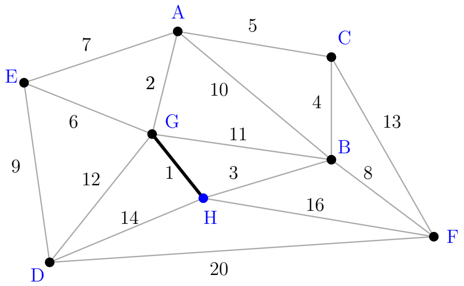
Prim's MST algorithm



Edge	**Weight**
GH	1

Look at the edges incident to *H*. *GH* is the edge of minimum weight.

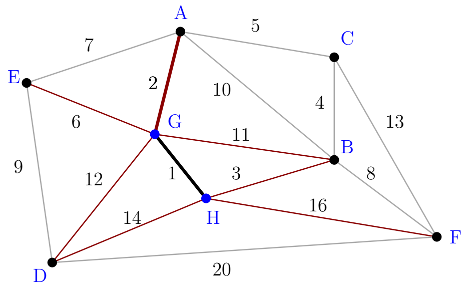
Prim's MST algorithm



Edge	**Weight**
GH	1

Include *GH* to the solution set.

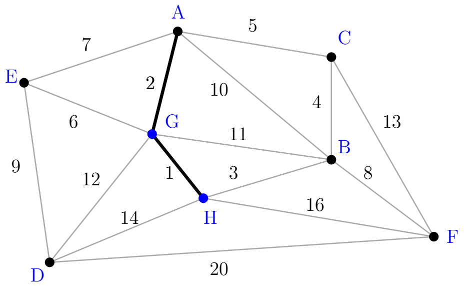
Prim's MST algorithm



Edge	Weight
GH	1
AG	2

Consider vertex $\{H, G\}$. Look at the edges incident to G or H. AG is the edge of minimum weight.

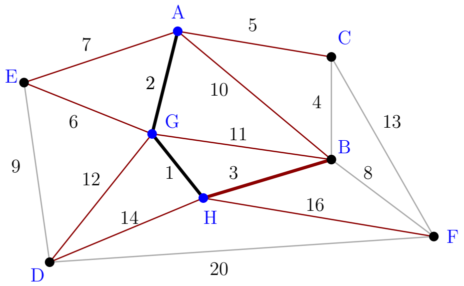
Prim's MST algorithm



Edge	Weight
GH	1
AG	2

Include *AG* to the solution set.

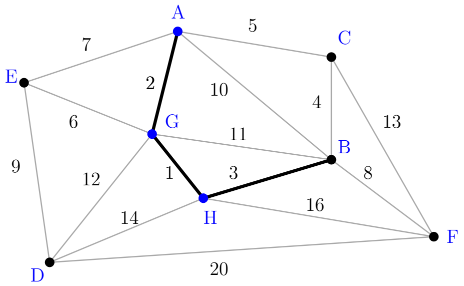
Prim's MST algorithm



Edge	Weight
GH	1
AG	2
BH	3

BH is the minimum-weight edge incident to $\{H, G, A\}$.

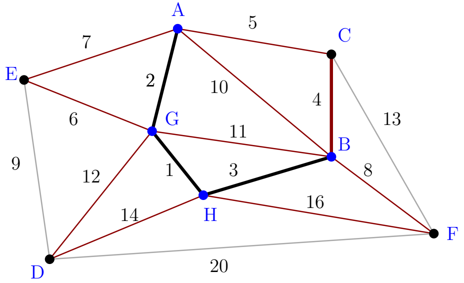
Prim's MST algorithm



Edge	**Weight**
GH	1
AG	2
BH	3

Include *BH* to the solution set.

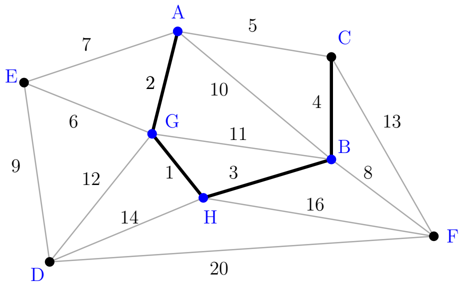
Prim's MST algorithm



Edge	**Weight**
GH	1
AG	2
BH	3
BC	4

BC is the minimum-weight edge incident to $\{H, G, A, B\}$.

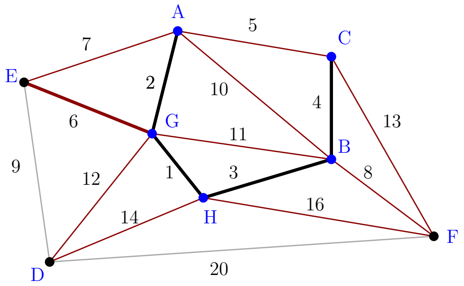
Prim's MST algorithm



Edge	Weight
GH	1
AG	2
BH	3
BC	4

Include *BC* to the solution set.

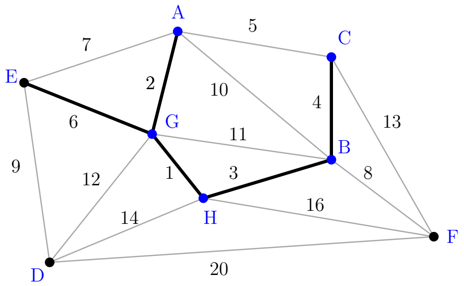
Prim's MST algorithm



Edge	Weight
GH	1
AG	2
BH	3
BC	4
AC	5
EG	6

EG is the minimum-weight edge incident to $\{H, G, A, B, C\}$
(AC cannot be chosen since it has two neighbors in the current solution)

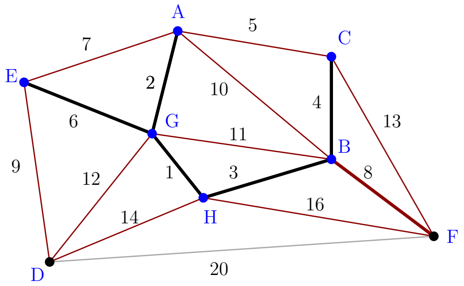
Prim's MST algorithm



Edge	Weight
GH	1
AG	2
BH	3
BC	4
AC	5
EG	6

Include *GE* to the solution set.

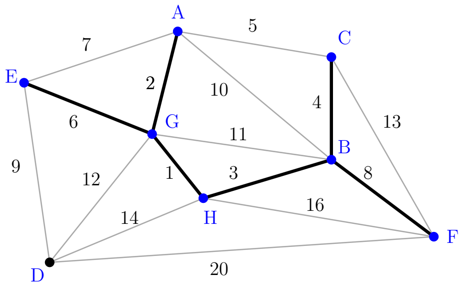
Prim's MST algorithm



Edge	Weight
GH	1
AG	2
BH	3
BC	4
AC	5
EG	6
AE	7
BF	8

BF is the minimum-weight edge incident to $\{H, G, A, B, C, E\}$
(*AE* cannot be chosen since it has two neighbors in the current solution)

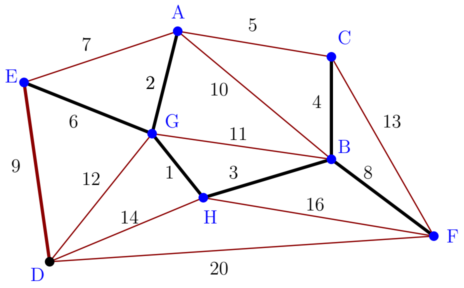
Prim's MST algorithm



Edge	Weight
GH	1
AG	2
BH	3
BC	4
AC	5
EG	6
AE	7
BF	8

Include *BF* to the solution set

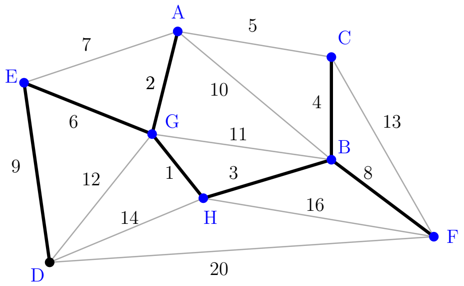
Prim's MST algorithm



Edge	Weight
GH	1
AG	2
BH	3
BC	4
AC	5
EG	6
AE	7
BF	8
DE	9

DE is the minimum-weight edge incident to $\{H, G, A, B, C, E, F\}$

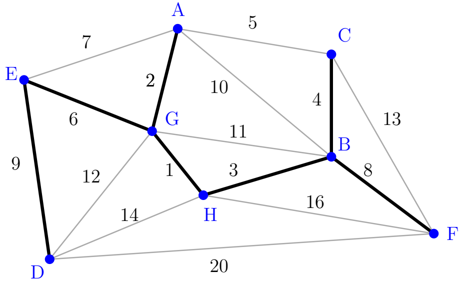
Prim's MST algorithm



Edge	Weight
GH	1
AG	2
BH	3
BC	4
AC	5
EG	6
AE	7
BF	8
DE	9

Include *DE* to the solution set

Prim's MST algorithm



Edge	**Weight**
GH	1
AG	2
BH	3
BC	4
AC	5
EG	6
AE	7
BF	8
DE	9

$\{GH, AG, BH, BC, EG, BF, DE\}$ is the solution of the MST

Prim's MST algorithm: proof of correctness

Theorem (Jarnik 1930, Djikstra 1957, Prim 1959)

Prim's algorithm computes the MST.

Proof. Let T be the graph output by Prim's algorithm

- T is a spanning tree. At each step, we add a vertex in $G \setminus T$ that has one neighbor in T. T is spanning because T contains $|V| - 1$ edges, i.e. $|V(T)| = |V(G)|$.
- T is an MST. Follows from the **cut property**: given any cut, the crossing edge of minimum weight is in the MST.

Kruskal's MST algorithm: proof of correctness

Question. How to check if adding an edge to T would create a cycle?

Alternative. Naive solution: use DFS.

- $\mathcal{O}(|V|)$-time per cycle check.
- $\mathcal{O}(|E||V|)$-time overall.

Question. How to find cheapest edge with exactly one endpoint in S?

Alternative. Brute force: try all edges.

- $\mathcal{O}(E)$-time per spanning tree edge.
- $\mathcal{O}(|E||V|)$-time overall.

Algorithm for MST in Euclidean space

Euclidean MST

Problem: given n points in the plane, find an MST connecting them, where *the distances between point pairs are their Euclidean distances*.

Euclidean distance. Given two vertices $a = (x_1, y_1)$ and $b = (x_2, y_2)$,

$$d_{(a,b)} = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

Naive algorithm: Compute $\Theta(n^2)$ distances and run Prim's algorithm.

Improvement. Exploit the geometric structure and do it in $\mathcal{O}(n \log n)$.

MST algorithm for graph in Euclidean space

Euclidean MST algorithm

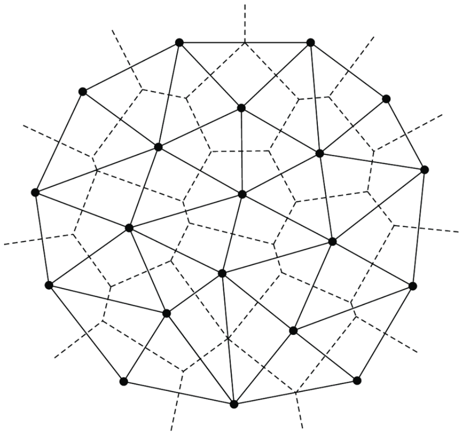
- Compute Voronoi diagram to get Delaunay triangulation.
- Run Kruskal's MST algorithm on Delaunay edges.

Complexity. $\mathcal{O}(n \log n)$

- The Delaunay triangulation contains $\leq 3n$ edges since it is planar.
- $\mathcal{O}(n \log n)$ for Voronoi.
- $\mathcal{O}(n \log n)$ for Kruskal

Lower bound. Any comparison-based Euclidean MST algorithm requires $\Omega(n \log n)$ comparisons.

MST algorithm for graph in Euclidean space



Application of Euclidean MST: k-clustering

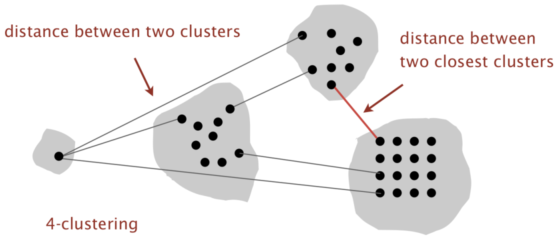
k-Clustering: Divide a set of objects classify into k coherent groups.

Distance function: Numeric value specifying "closeness" of two objects.

Goal: Divide into clusters so that objects in different clusters are far apart.

Single-link: Distance between two clusters equals the distance between the two closest objects (one in each cluster).

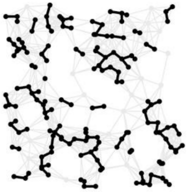
Single-link clustering: Given an integer k, find a k-clustering that maximizes the distance between two closest clusters.



Single-link clustering algorithm

"Well-known" algorithm in science literature for single-link k-clustering:

- Form $|V|$ clusters of one object each.
- Find the closest pair of objects such that each object is in a different cluster, and merge the two clusters.
- Repeat until there are exactly k clusters.



Observation: This is Kruskal's algorithm (stop when there are k connected components).

Alternative solution: Run Prim's algorithm and delete $k-1$ max weight edges.