Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

http://algs4.cs.princeton.edu

4.3 MINIMUM SPANNING TREES

- *introduction*
- *greedy algorithm*
- *edge-weighted graph API*
- *Kruskal's algorithm*
- *Prim's algorithm*
- *context*

Last updated on Nov 19, 2015, 10:38 AM

http://algs4.cs.princeton.edu

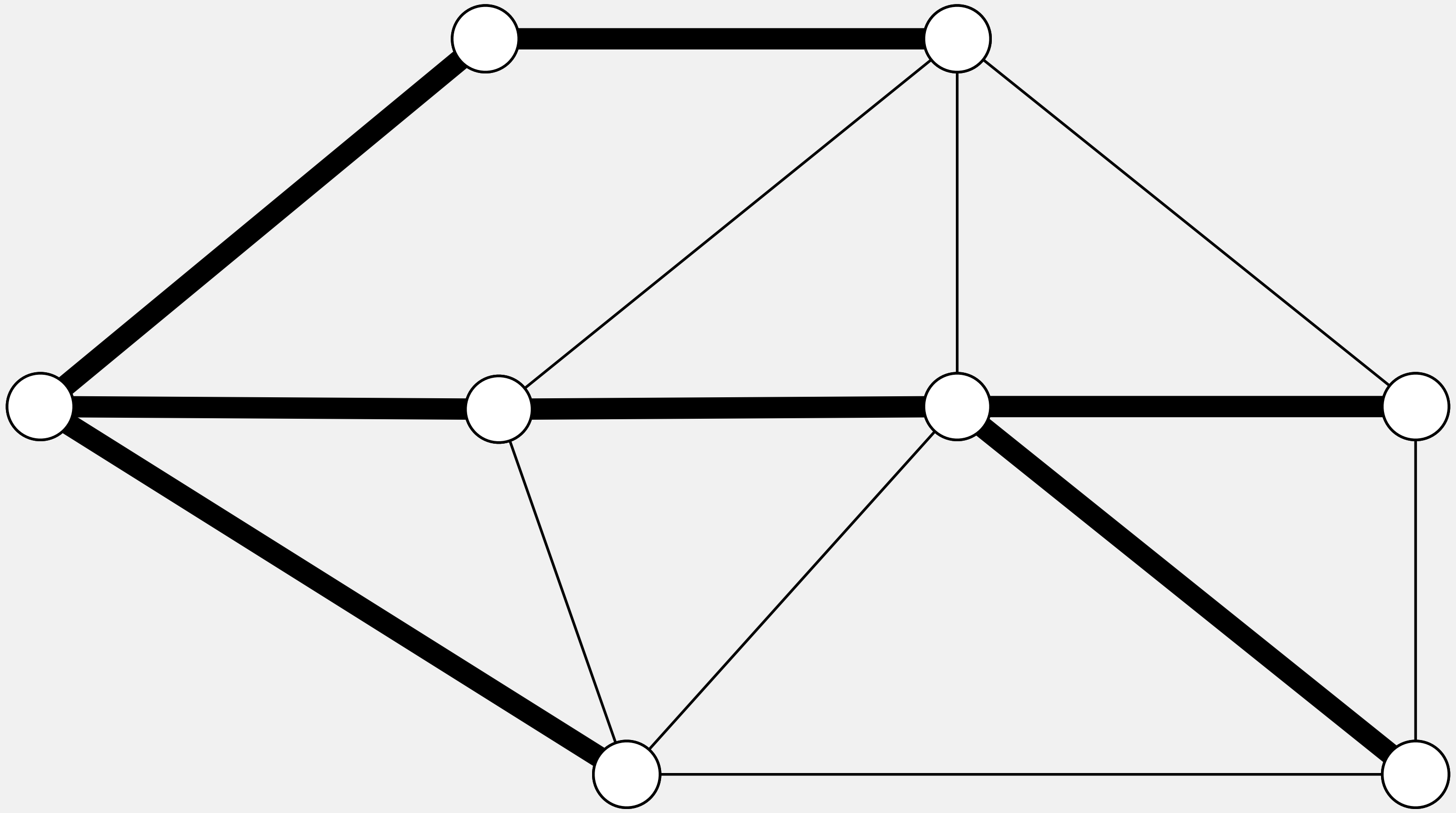
4.3 Minimum Spanning Trees

- *introduction*
- *greedy algorithm*
- *edge-weighted graph API*
- *Kruskal's algorithm*
- *Prim's algorithm*
- *context*

Spanning tree

Def. A spanning tree of an undirected graph G is a subgraph T that is:

- A tree: connected and acyclic.
- Spanning: includes all of the vertices.



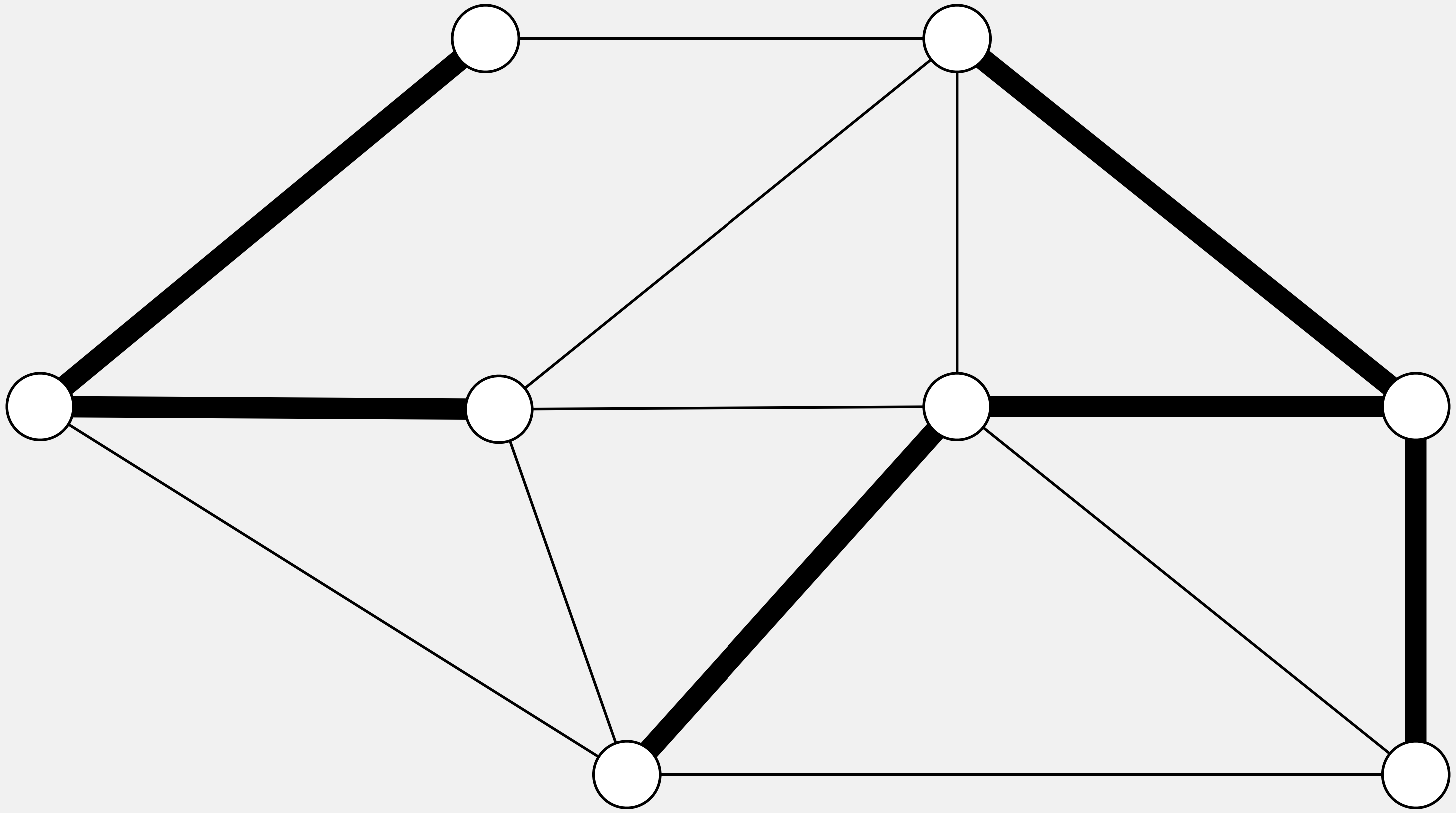
graph G

spanning tree T

Spanning tree

Def. A spanning tree of an undirected graph G is a subgraph T that is:

- A tree: connected and acyclic.
- Spanning: includes all of the vertices.

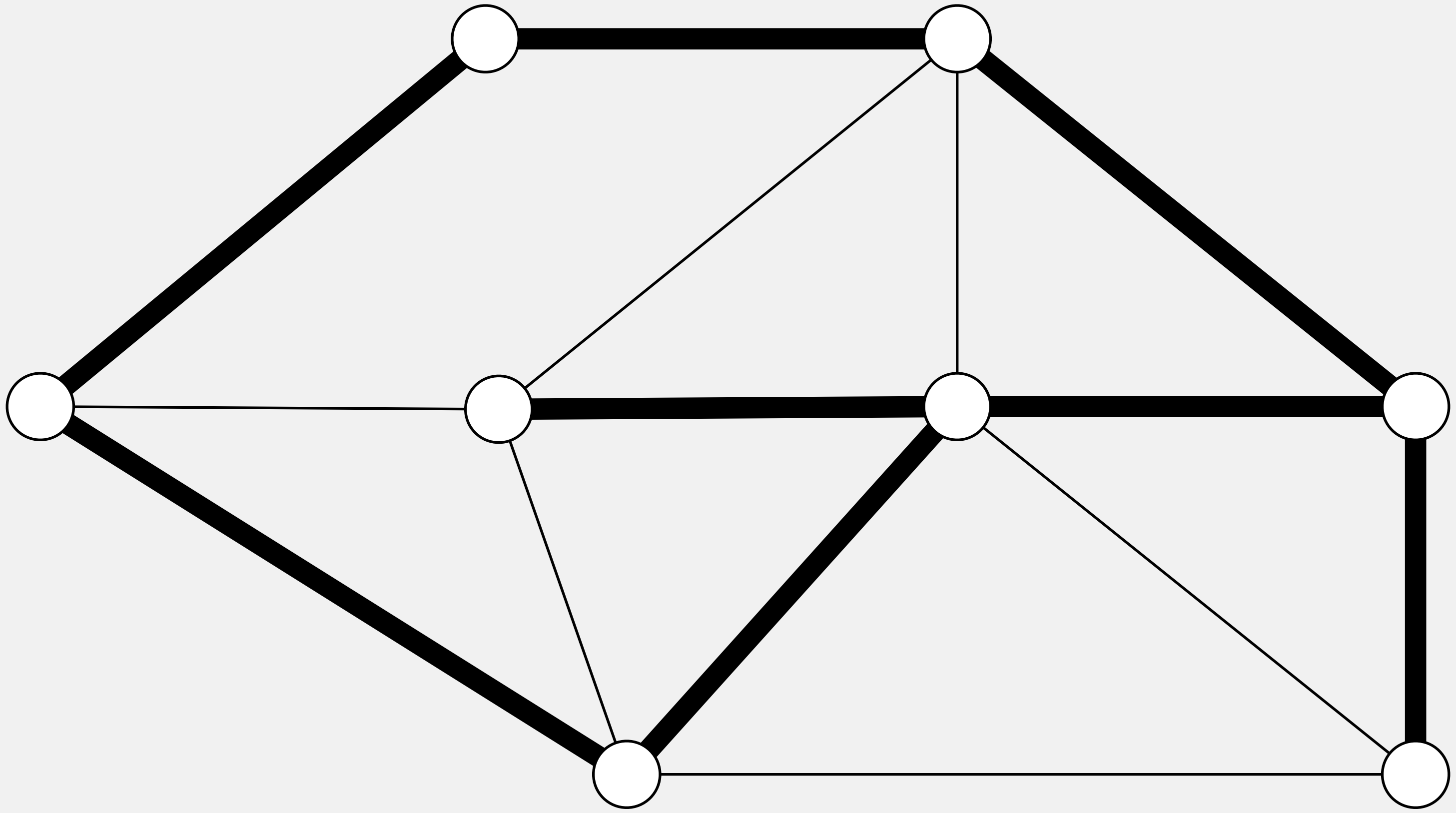


not connected

Spanning tree

Def. A spanning tree of an undirected graph G is a subgraph T that is:

- A tree: connected and acyclic.
- Spanning: includes all of the vertices.

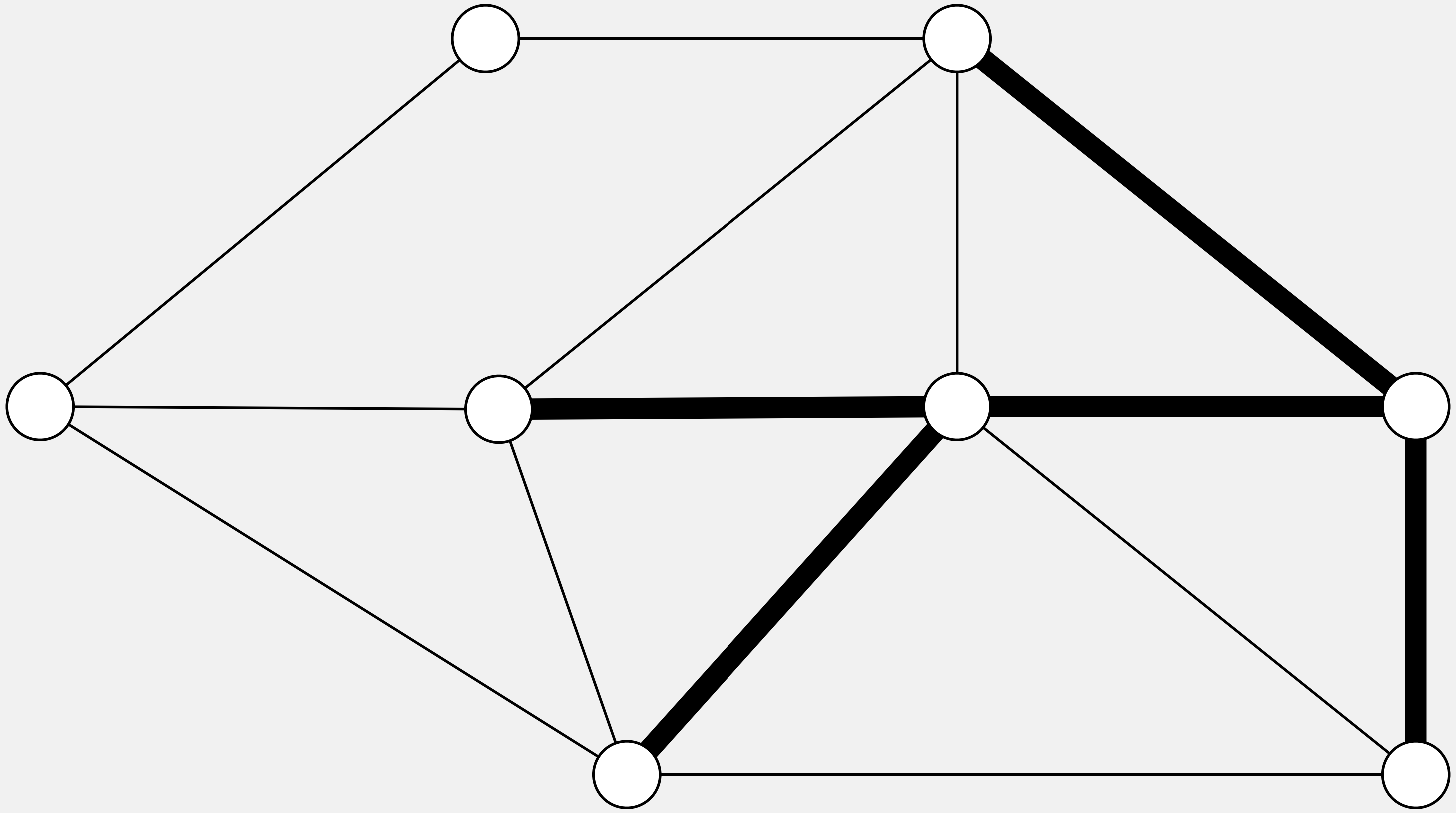


not a tree (cyclic)

Spanning tree

Def. A spanning tree of an undirected graph G is a subgraph T that is:

- A tree: connected and acyclic.
- Spanning: includes all of the vertices.



not spanning

Quiz 1: spanning trees

Let T be a spanning tree of a connected graph G with V vertices.

Which of the following statements are true?

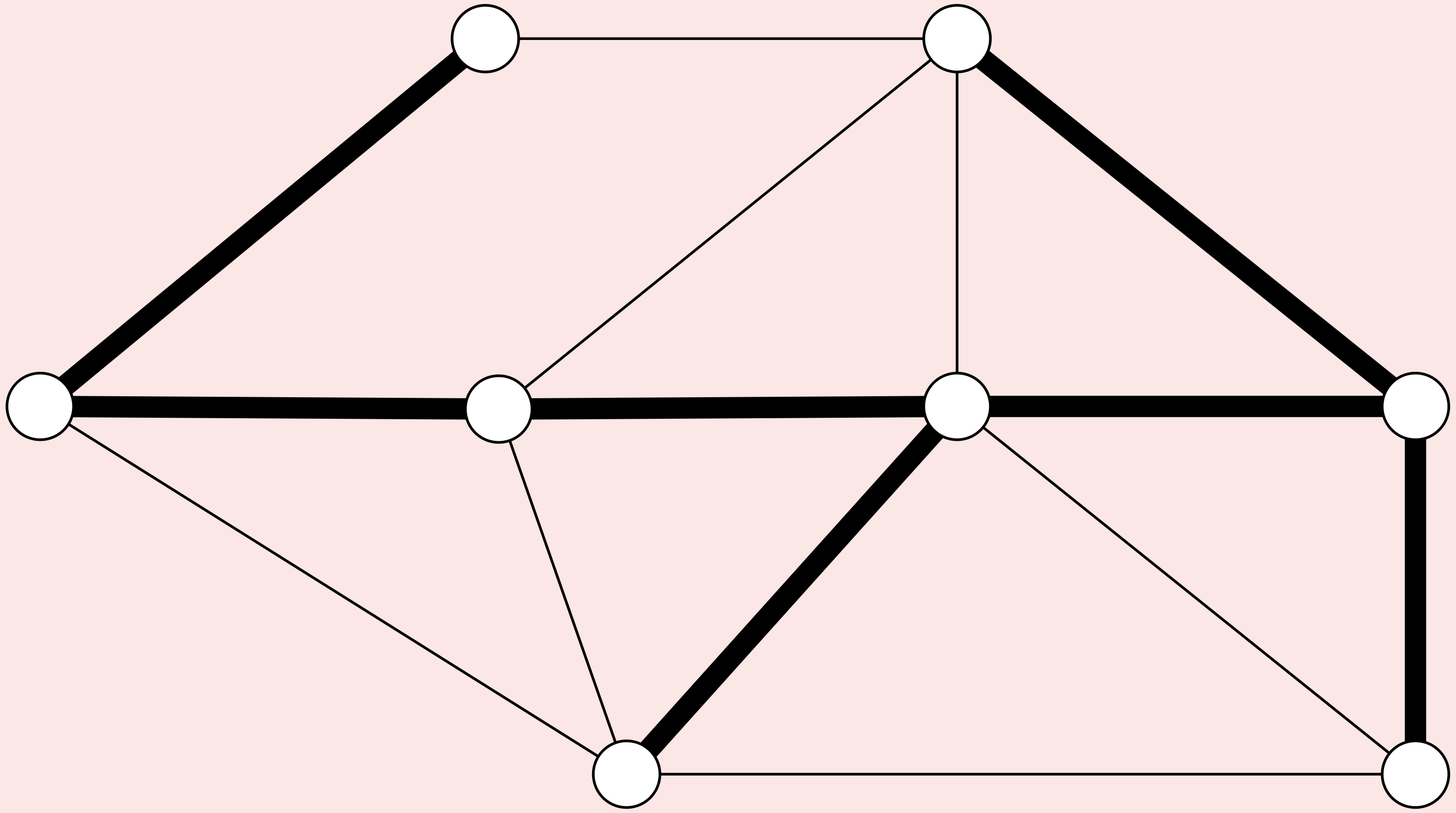
A. T contains exactly $V - 1$ edges.

B. Removing any edge from T disconnects it.

C. Adding any edge to T creates a cycle.

D. All of the above.

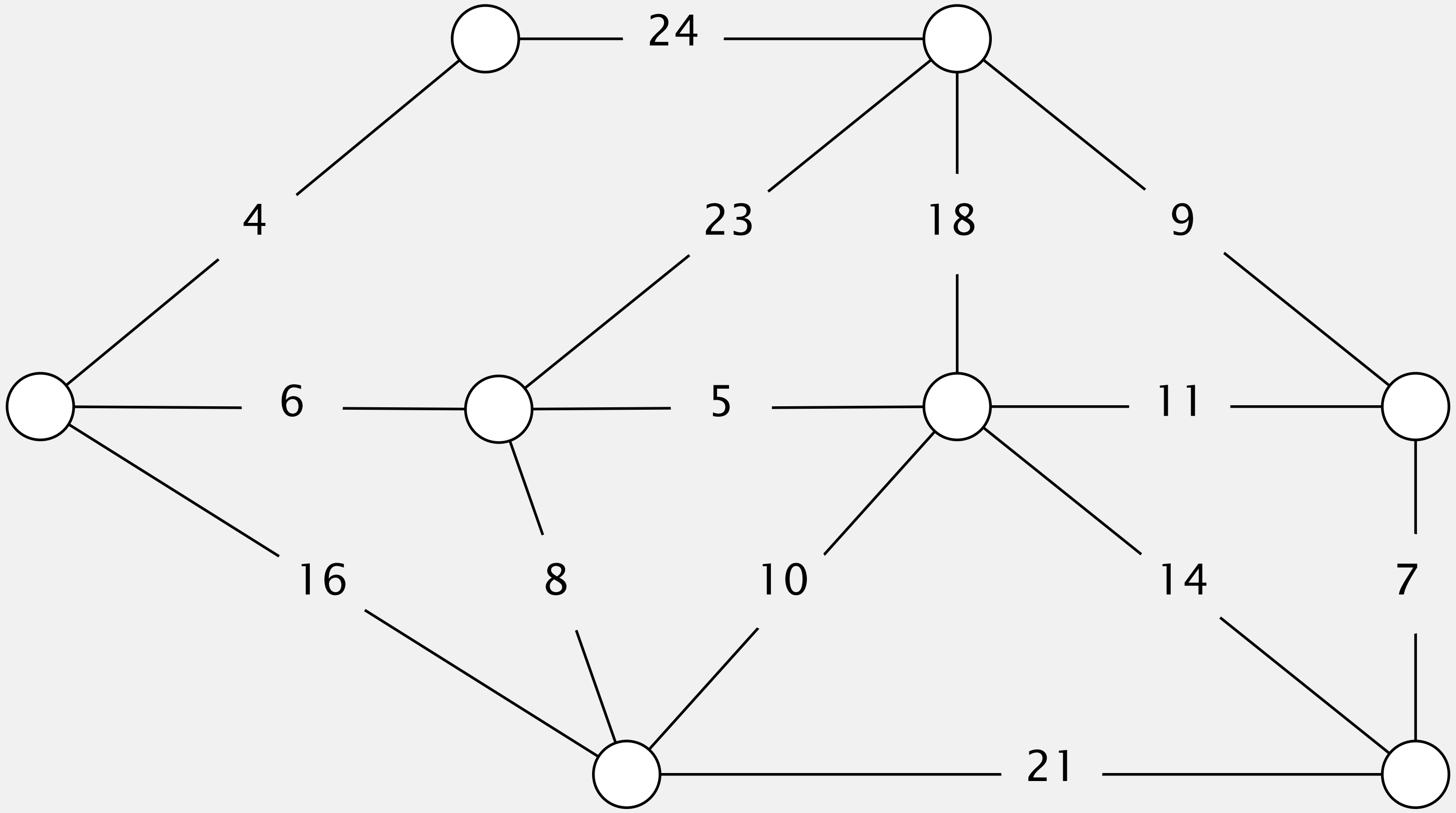
E. *I don't know.*



spanning tree T of graph G

Minimum spanning tree problem

Input. Connected, undirected graph G with positive edge weights.

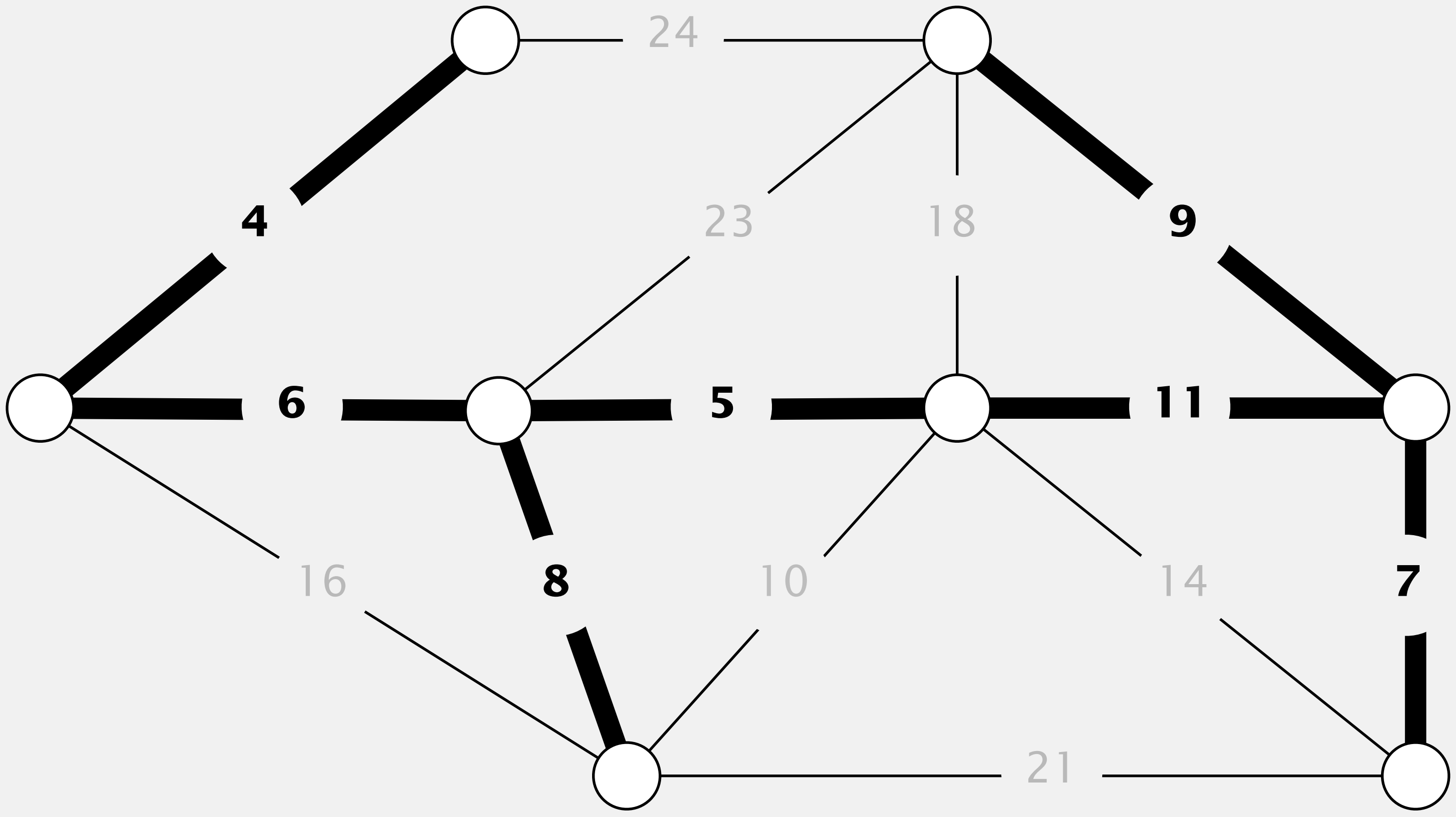


edge-weighted digraph G

Minimum spanning tree problem

Input. Connected, undirected graph G with positive edge weights.

Output. A spanning tree of minimum weight.

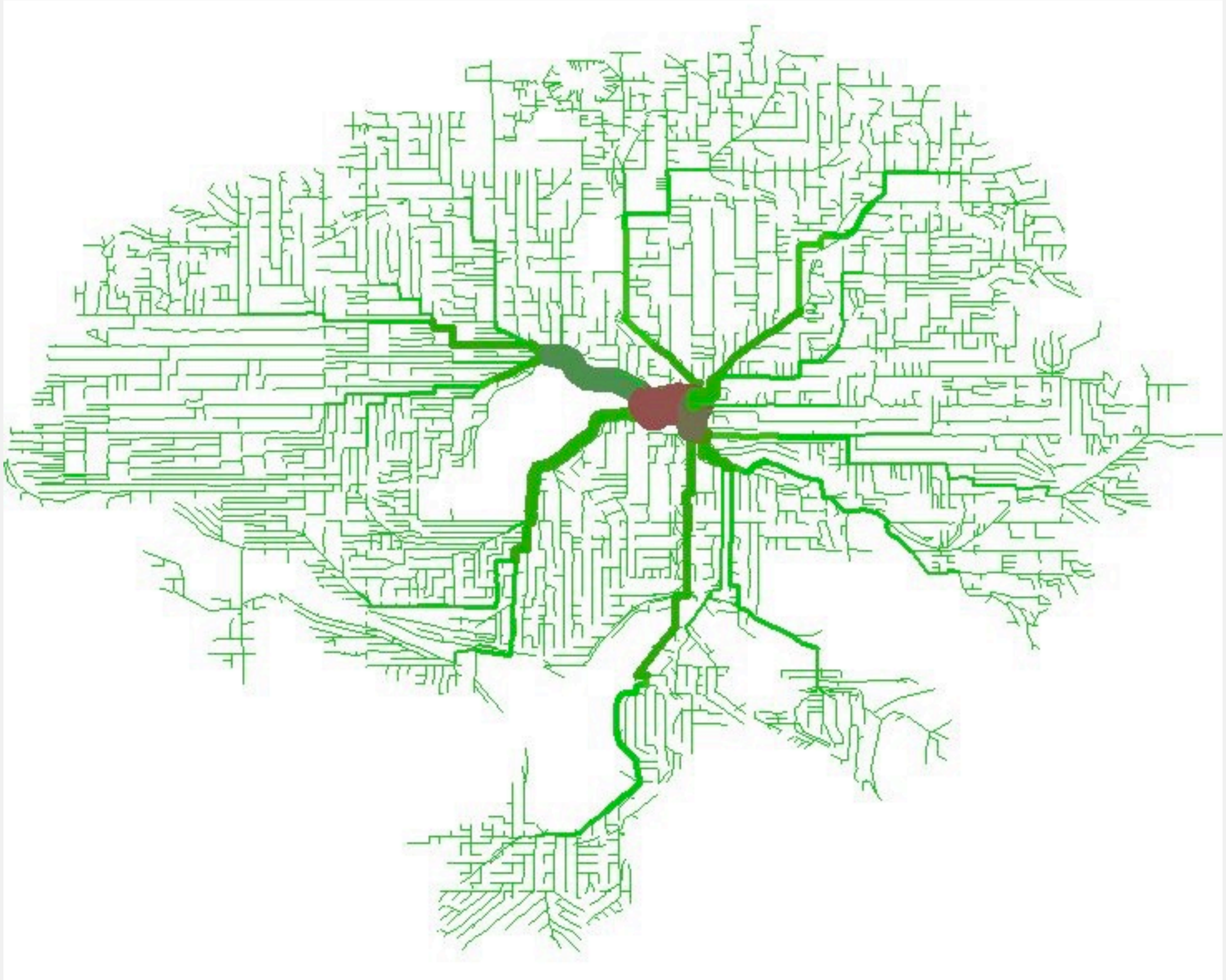


minimum spanning tree T
(weight = 50 = 4 + 6 + 8 + 5 + 11 + 9 + 7)

Brute force. Try all spanning trees? (Impractical.)

Network design

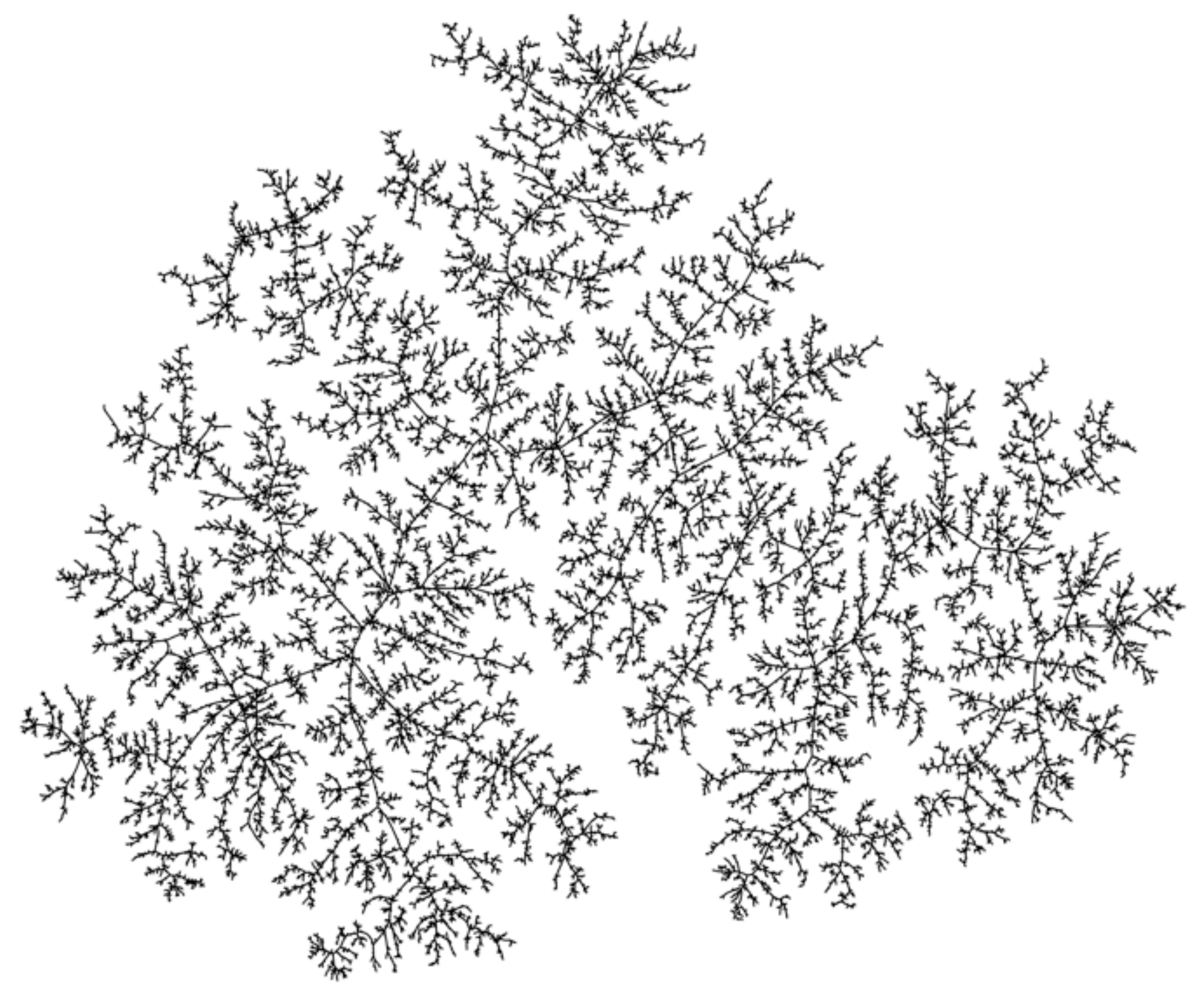
MST of bicycle routes in North Seattle



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

Models of nature

MST of random graph

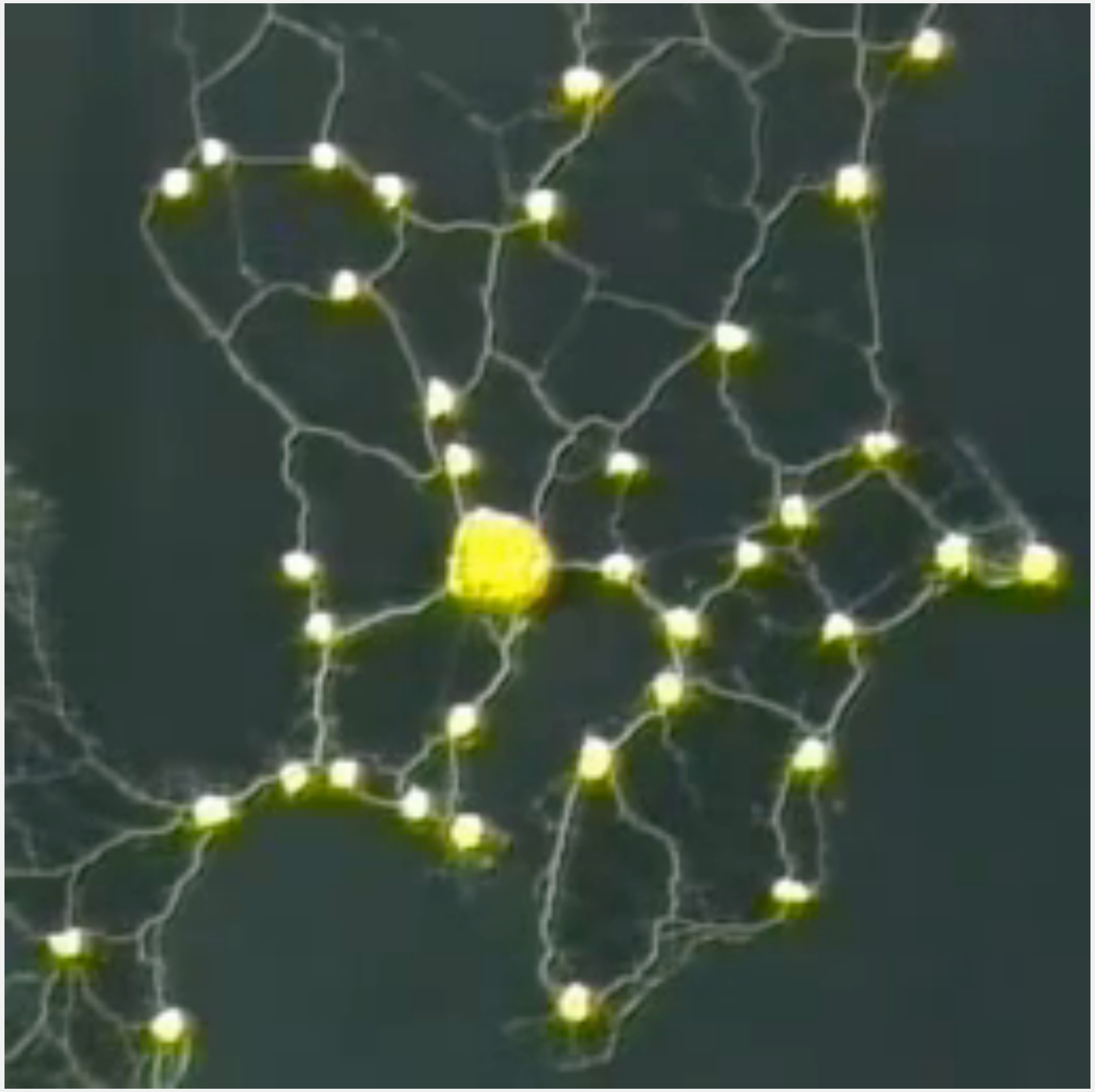


http://algo.inria.fr/broutin/gallery.html

Slime mold grows network just like Tokyo rail system

Rules for Biologically Inspired Adaptive Network Design

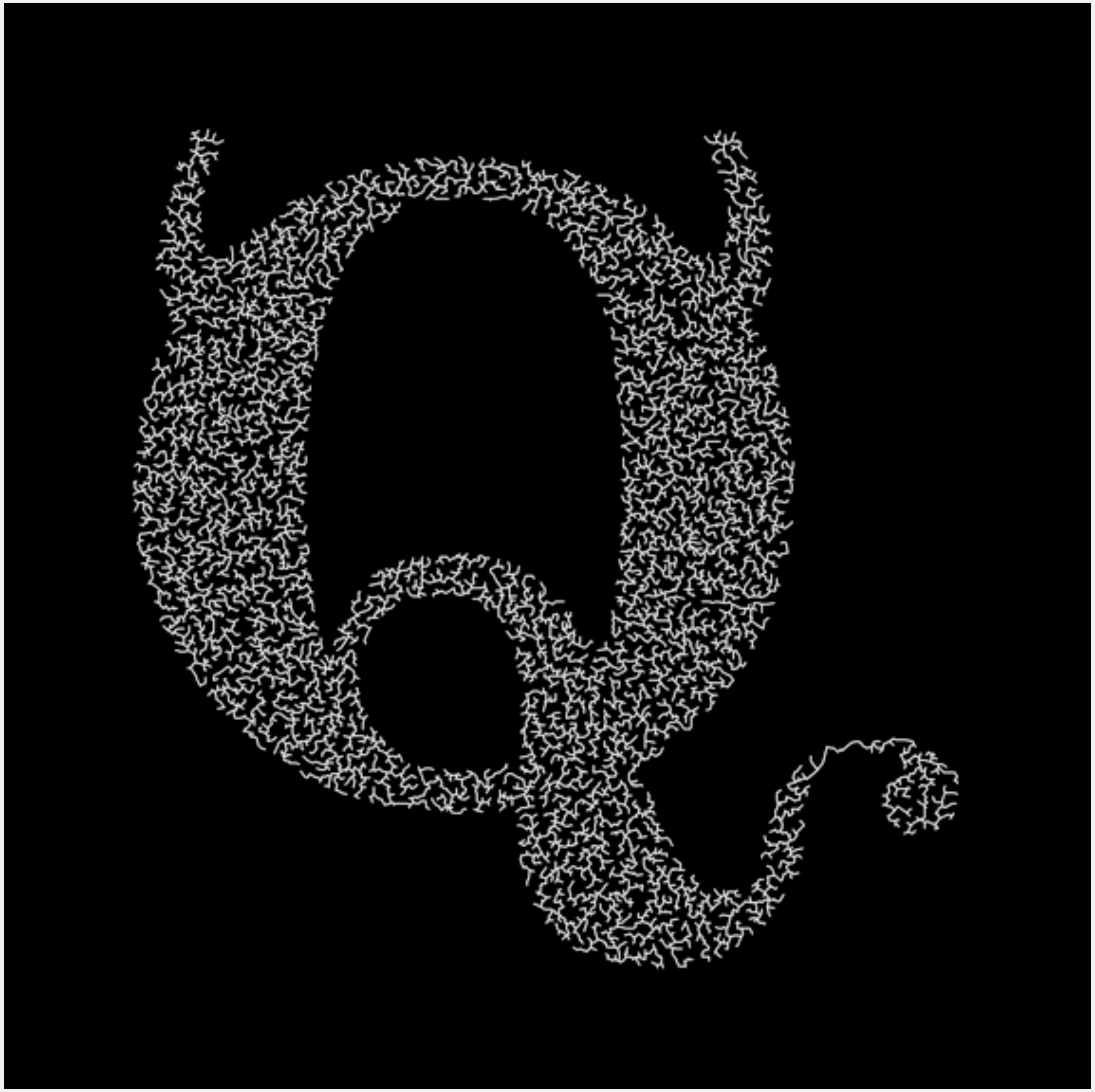
Atsushi Tero,[1,2] Seiji Takagi,[1] Tetsu Saigusa,[3] Kentaro Ito,[1] Dan P. Bebber,[4] Mark D. Fricker,[4]
Kenji Yumiki,[5] Ryo Kobayashi,[5,6] Toshiyuki Nakagaki[1,6]*



https://www.youtube.com/watch?v=GwKuFREOgmo

Image processing

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


Algorithms

Robert Sedgewick | Kevin Wayne

http://algs4.cs.princeton.edu

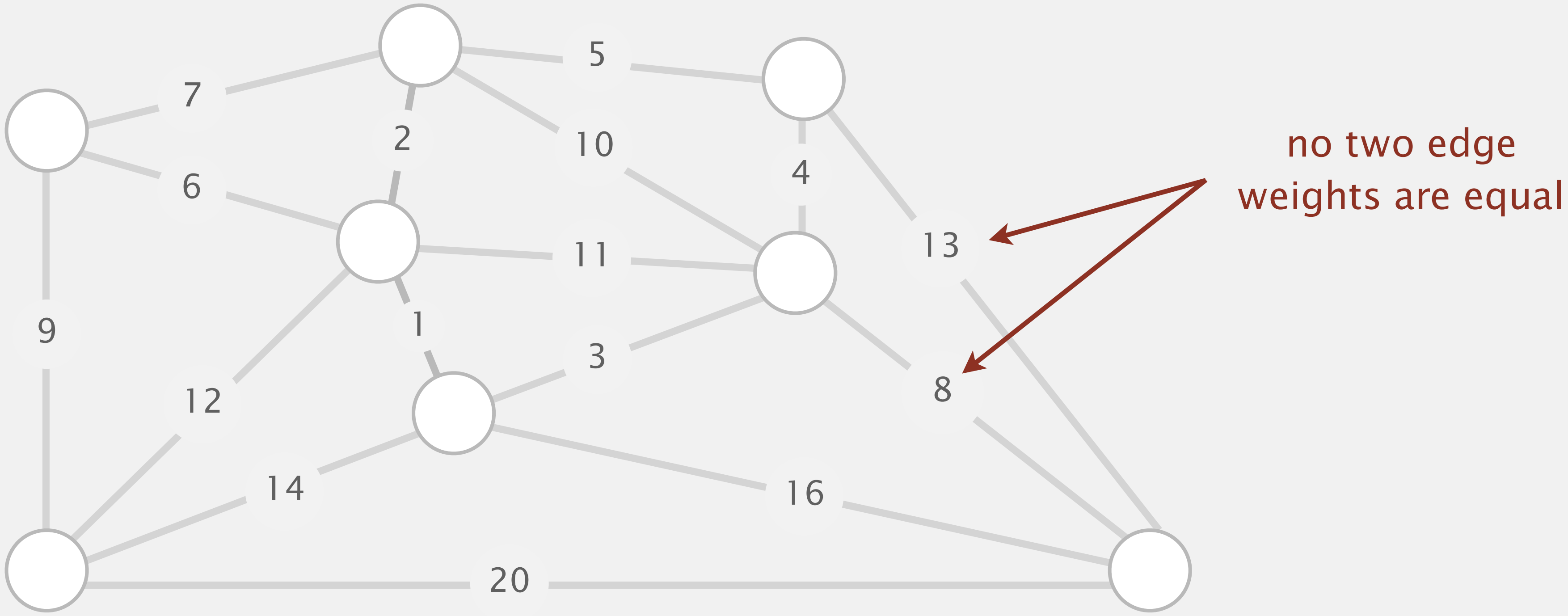
4.3 Minimum Spanning Trees

- *introduction*
- *greedy algorithm*
- *edge-weighted graph API*
- *Kruskal's algorithm*
- *Prim's algorithm*
- *context*

Simplifying assumptions

For simplicity, we assume

- The graph is connected. ⇒ MST exists.
- The edge weights are distinct. ⇒ MST is unique.

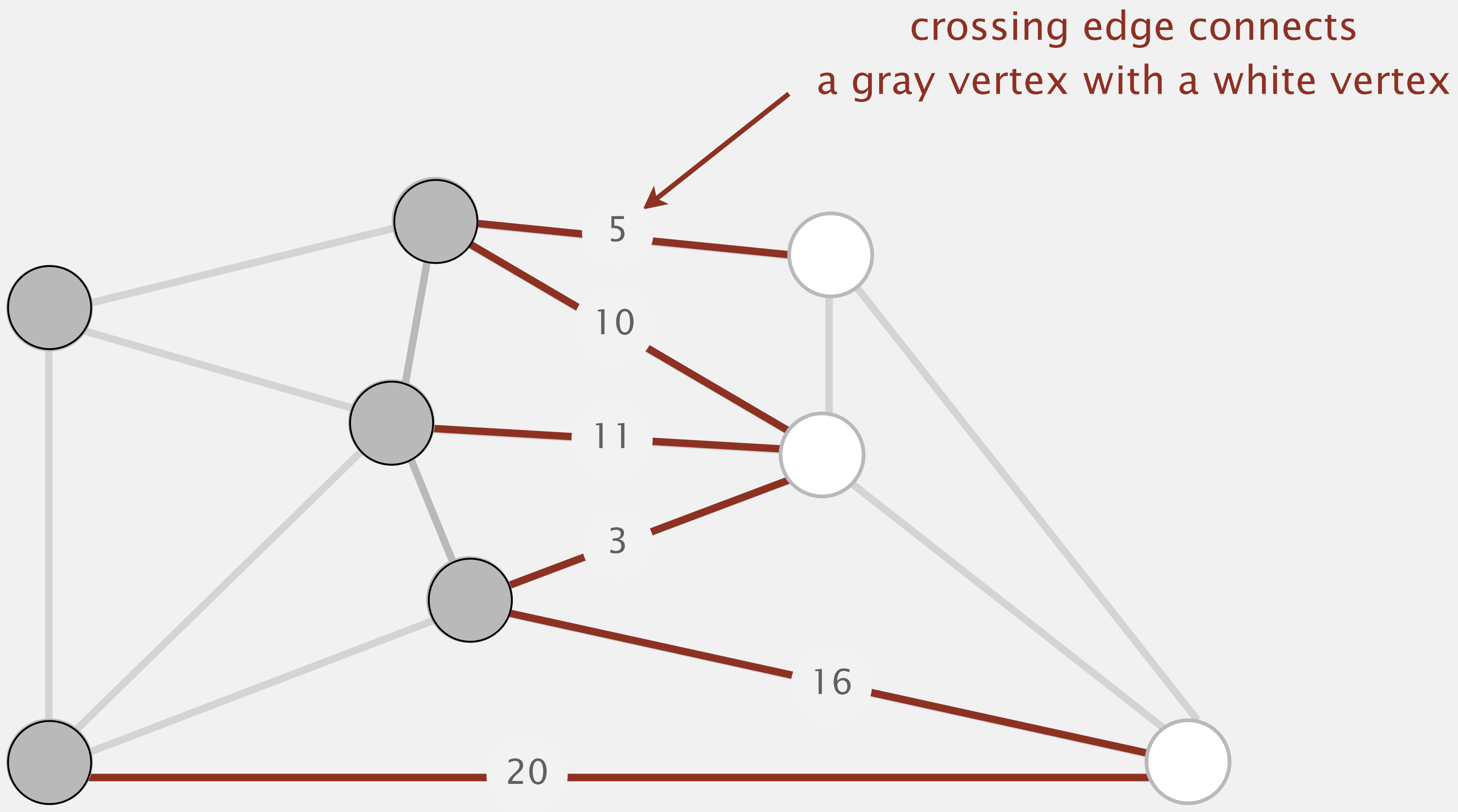


no two edge weights are equal

Cut property

Def. A cut in a graph is a partition of its vertices into two (nonempty) sets.

Def. A crossing edge connects a vertex in one set with a vertex in the other.


crossing edge connects
a gray vertex with a white vertex

Minimum spanning trees: quiz 2

Which is the min weight edge crossing the cut $\{ 2, 3, 5, 6 \}$?

A. 0–7 (0.16)

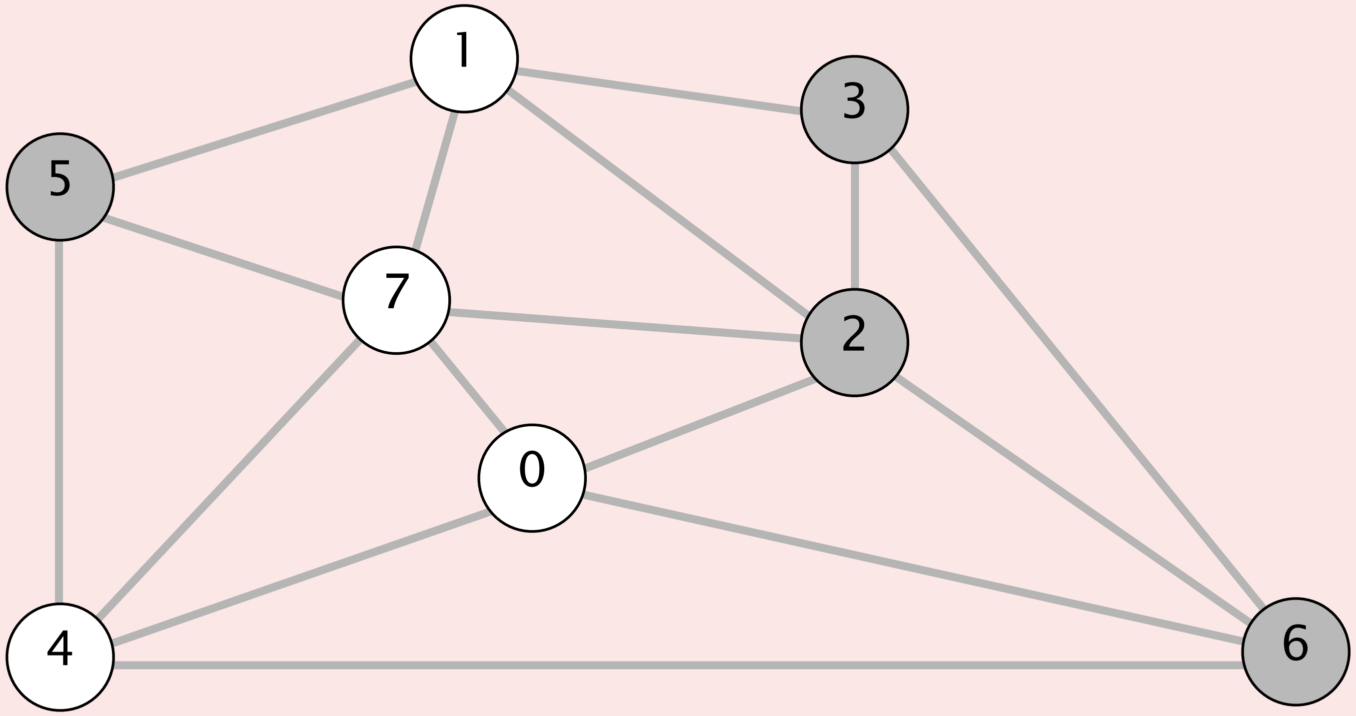
B. 2–3 (0.17)

C. 0–2 (0.26)

D. 5–7 (0.28)

E. *I don't know.*

0-7	0.16
2-3	0.17
1-7	0.19
0-2	0.26
5-7	0.28
1-3	0.29
1-5	0.32
2-7	0.34
4-5	0.35
1-2	0.36
4-7	0.37
0-4	0.38
6-2	0.40
3-6	0.52
6-0	0.58
6-4	0.93

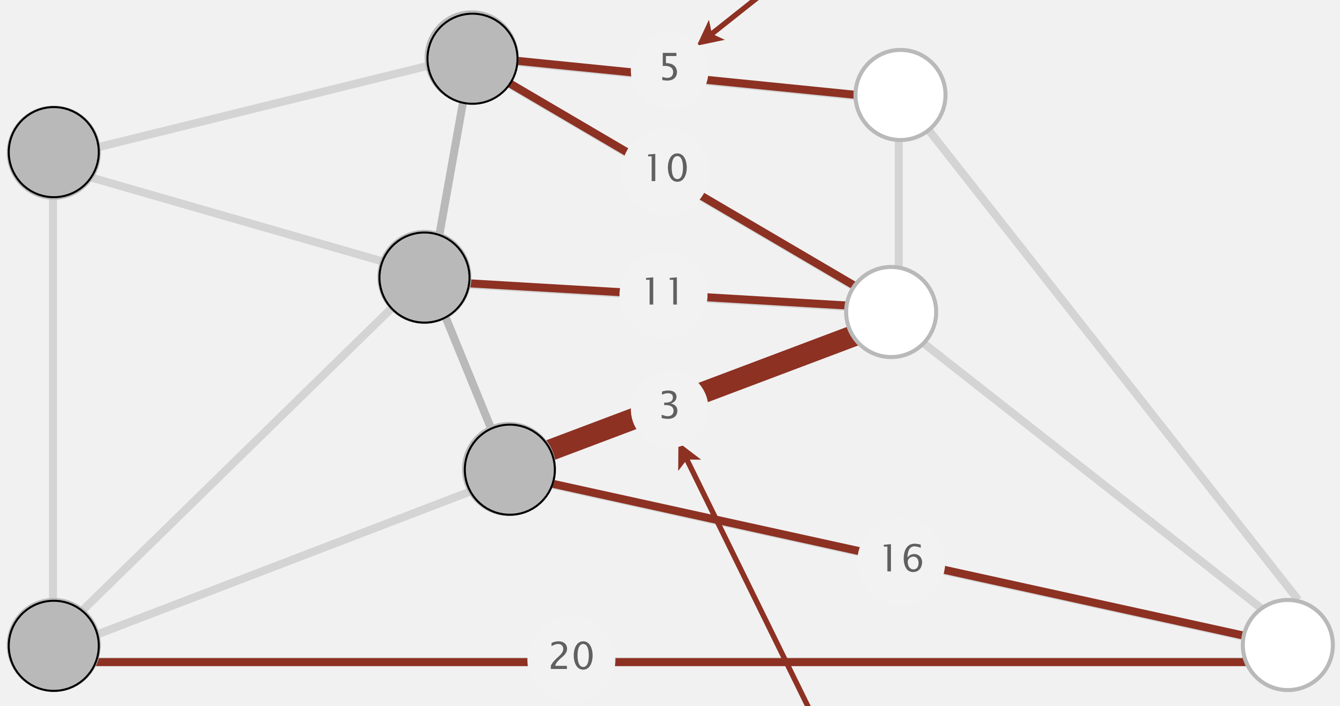


Cut property

Def. A cut in a graph is a partition of its vertices into two (nonempty) sets.

Def. A crossing edge connects a vertex in one set with a vertex in the other.

Cut property. Given any cut, the crossing edge of min weight is in the MST.



crossing edge connects
a gray vertex with a white vertex

minimum-weight crossing edge
must be in the MST

Cut property: correctness proof

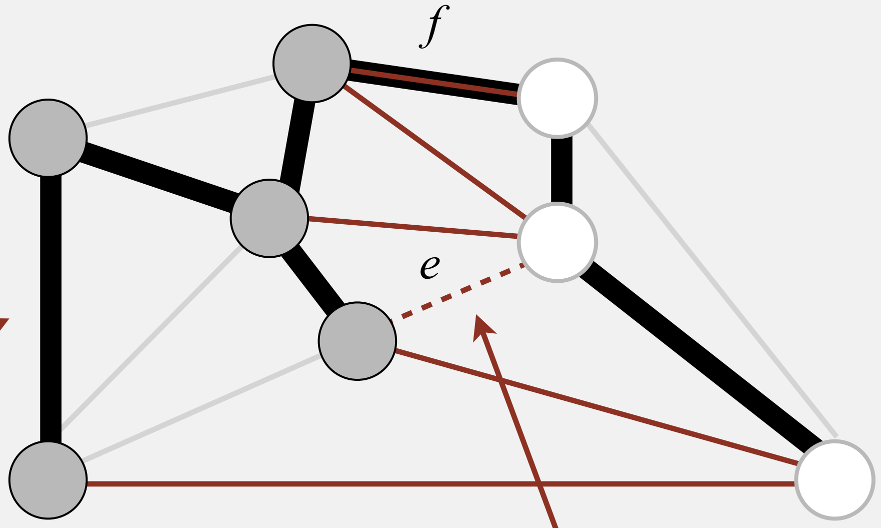
Def. A cut in a graph is a partition of its vertices into two (nonempty) sets.

Def. A crossing edge connects a vertex in one set with a vertex in the other.

Cut property. Given any cut, the crossing edge of min weight is in the MST.

Pf. Suppose min-weight crossing edge e is not in the MST.

- Adding e to the MST creates a cycle.
- Some other edge f in cycle must be a crossing edge.
- Removing f and adding e is also a spanning tree.
- Since weight of e is less than the weight of f, that spanning tree has lower weight.
- Contradiction. ▪

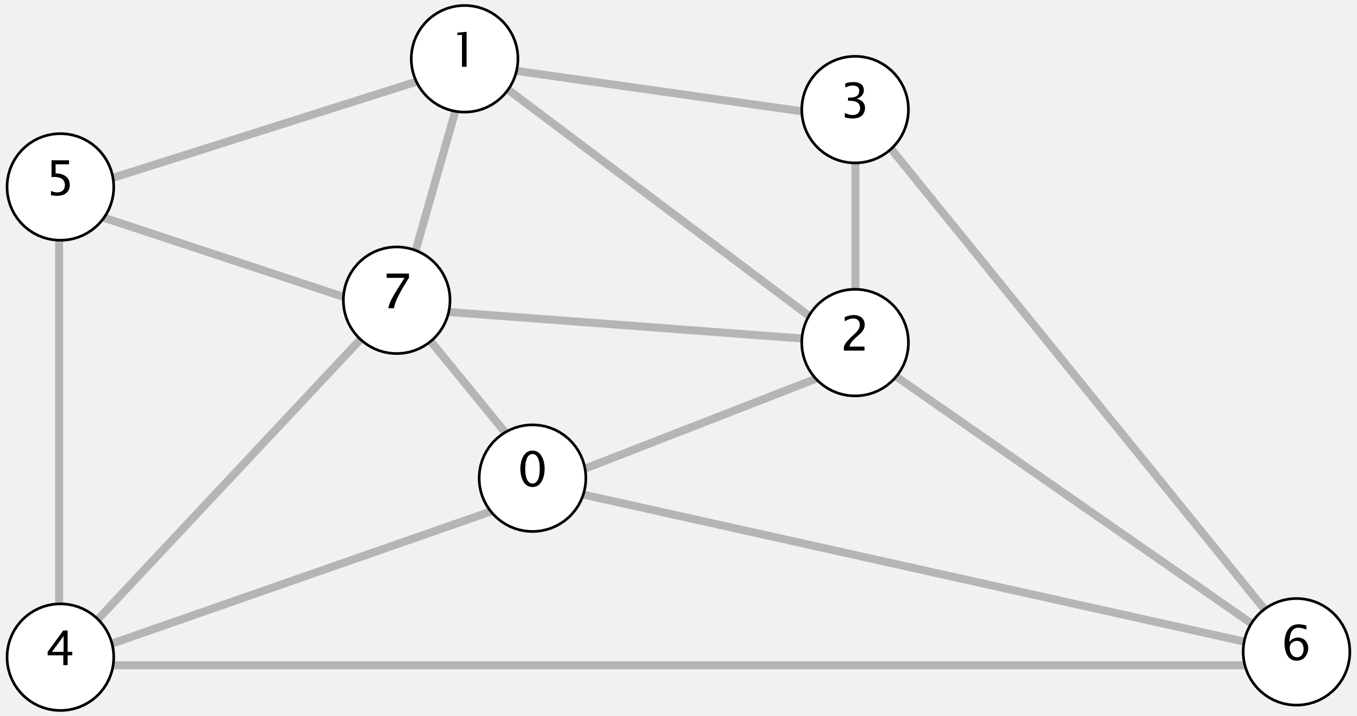


the MST does not contain e

adding e to MST creates a cycle

Greedy MST algorithm demo

- Start with all edges colored gray.
- Find cut with no black crossing edges; color its min-weight edge black.
- Repeat until $V - 1$ edges are colored black.



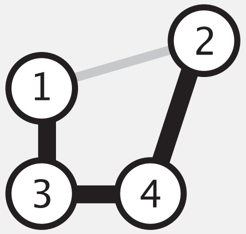
an edge-weighted graph

0-7	0.16
2-3	0.17
1-7	0.19
0-2	0.26
5-7	0.28
1-3	0.29
1-5	0.32
2-7	0.34
4-5	0.35
1-2	0.36
4-7	0.37
0-4	0.38
6-2	0.40
3-6	0.52
6-0	0.58
6-4	0.93

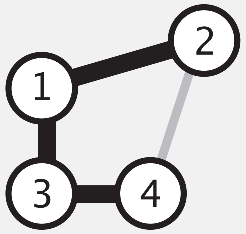
Removing two simplifying assumptions

Q. What if edge weights are not all distinct?

A. Greedy MST algorithm correct even if equal weights are present!



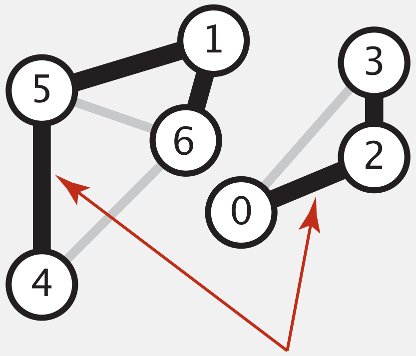
1	2	1.00
1	3	0.50
2	4	1.00
3	4	0.50



1	2	1.00
1	3	0.50
2	4	1.00
3	4	0.50

Q. What if graph is not connected?

A. Compute minimum spanning forest = one MST per component.



can independently compute MSTs of components

4	5	0.61
4	6	0.62
5	6	0.88
1	5	0.11
2	3	0.35
0	3	0.6
1	6	0.10
0	2	0.22

Greedy MST algorithm: efficient implementations

In practice: How to find cut? How to find min-weight edge?

Ex 1. Kruskal's algorithm. [stay tuned]

Ex 2. Prim's algorithm. [stay tuned]

Ex 3. Borüvka's algorithm.

http://algs4.cs.princeton.edu

4.3 Minimum Spanning Trees

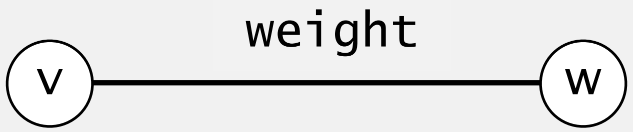
- *introduction*
- *greedy algorithm*
- *edge-weighted graph API*
- *Kruskal's algorithm*
- *Prim's algorithm*
- *context*

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

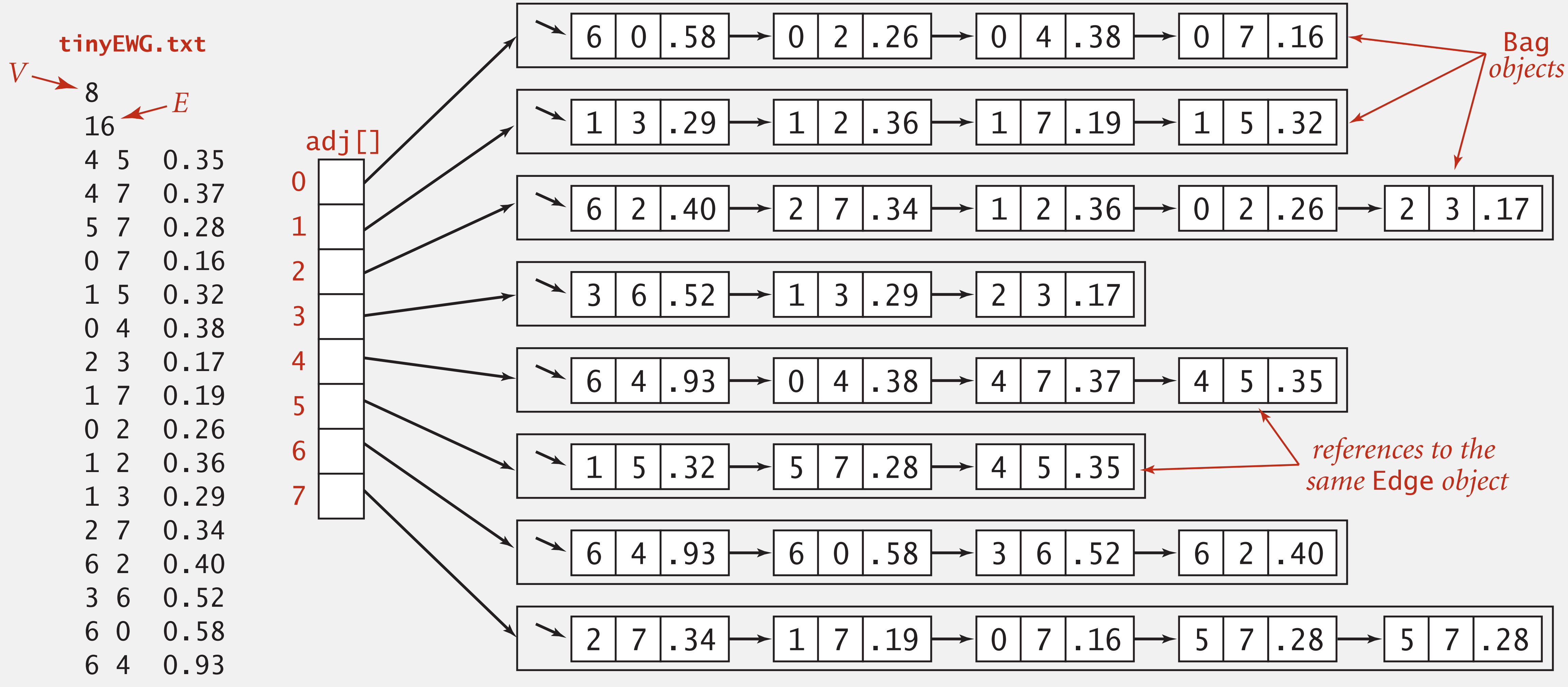
`public class EdgeWeightedGraph`

	`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.



tinyEWG.txt

```
8
16
4 5   0.35
4 7   0.37
5 7   0.28
0 7   0.16
1 5   0.32
0 4   0.38
2 3   0.17
1 7   0.19
0 2   0.26
1 2   0.36
1 3   0.29
2 7   0.34
6 2   0.40
3 6   0.52
6 0   0.58
6 4   0.93
```

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 API

Q. How to represent the MST?

	`public class MST`	
	`MST(EdgeWeightedGraph G)`	*constructor*
`Iterable<Edge>`	`edges()`	*edges in MST*
`double`	`weight()`	*weight of MST*

http://algs4.cs.princeton.edu

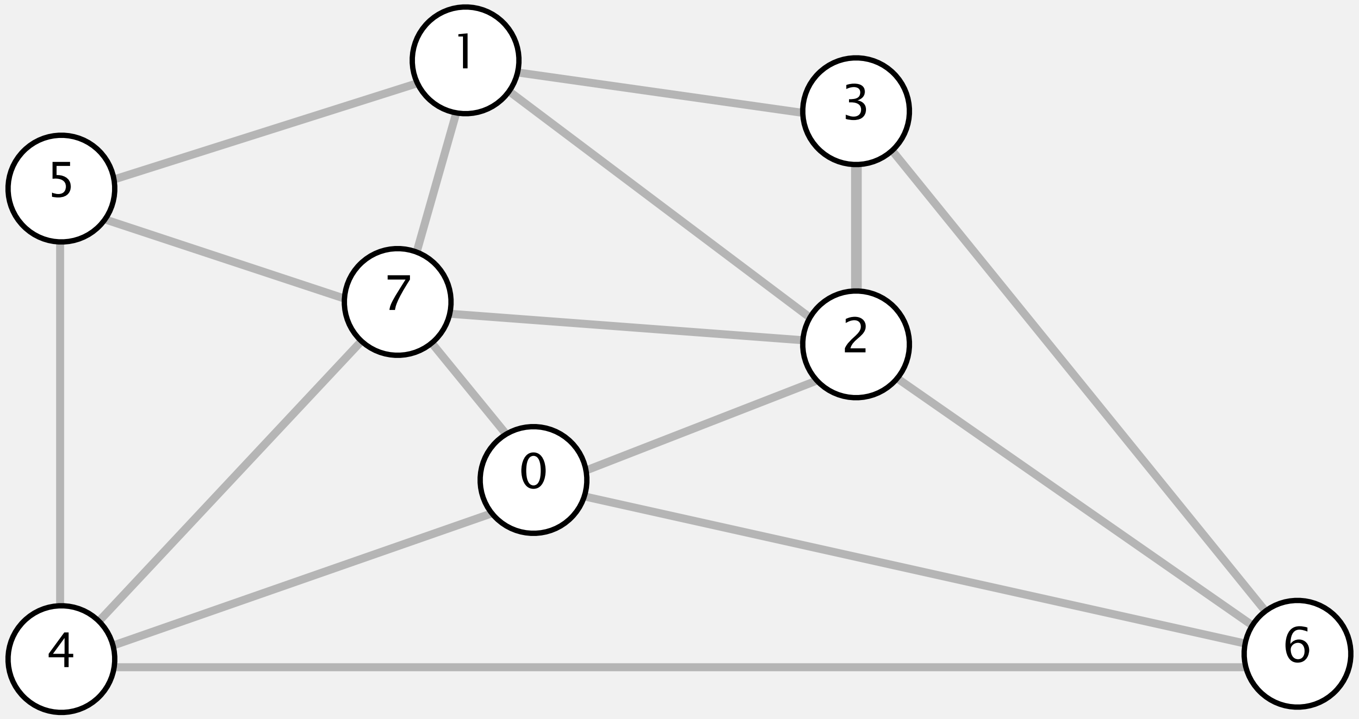
4.3 Minimum Spanning Trees

- *introduction*
- *greedy algorithm*
- *edge-weighted graph API*
- ***Kruskal's algorithm***
- *Prim's algorithm*
- *context*

Kruskal's algorithm demo

Consider edges in ascending order of weight.

- Add next edge to tree T unless doing so would create a cycle.



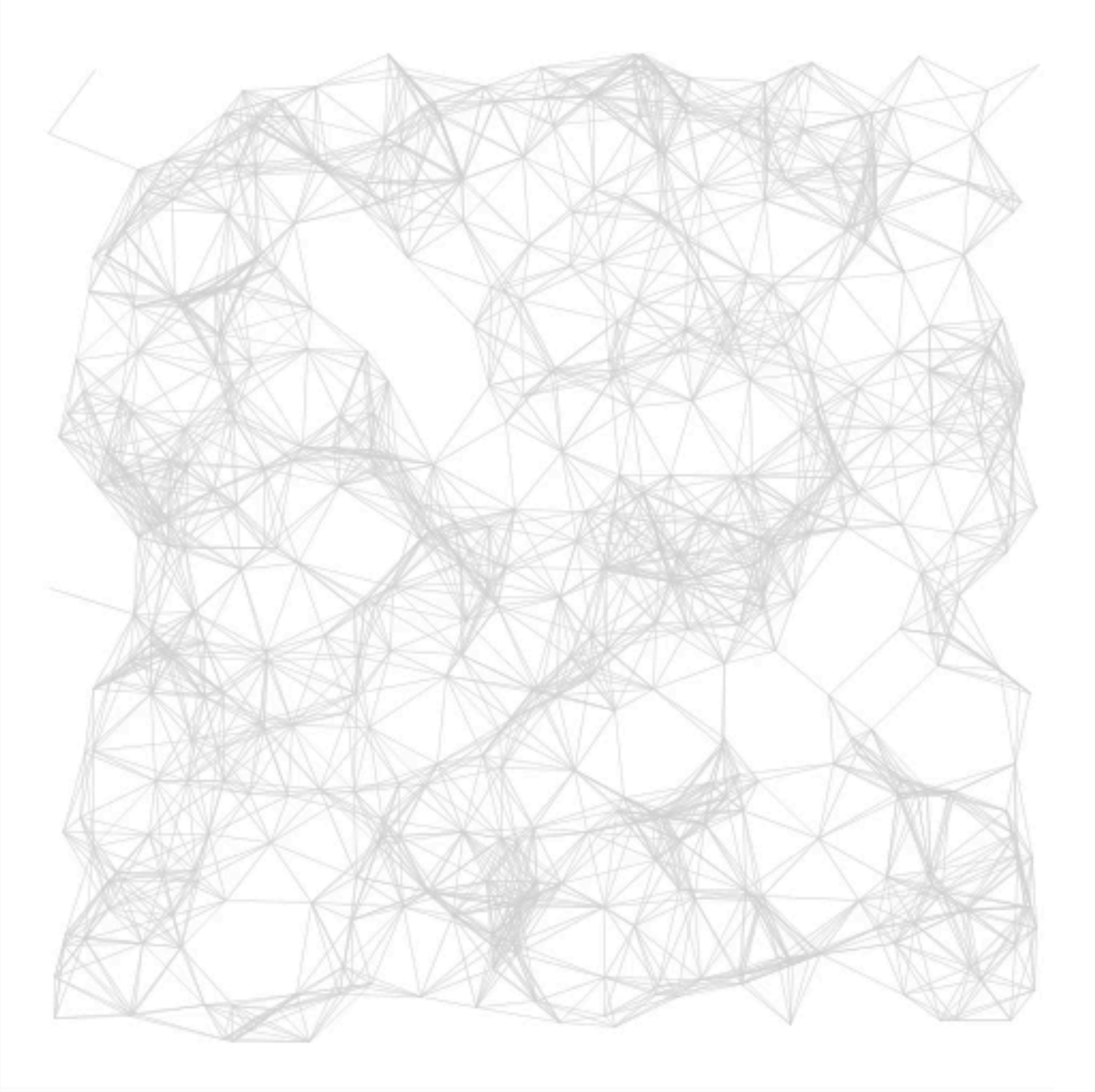
an edge-weighted graph

graph edges sorted by weight

↓

0-7	0.16
2-3	0.17
1-7	0.19
0-2	0.26
5-7	0.28
1-3	0.29
1-5	0.32
2-7	0.34
4-5	0.35
1-2	0.36
4-7	0.37
0-4	0.38
6-2	0.40
3-6	0.52
6-0	0.58
6-4	0.93

Kruskal's algorithm: visualization

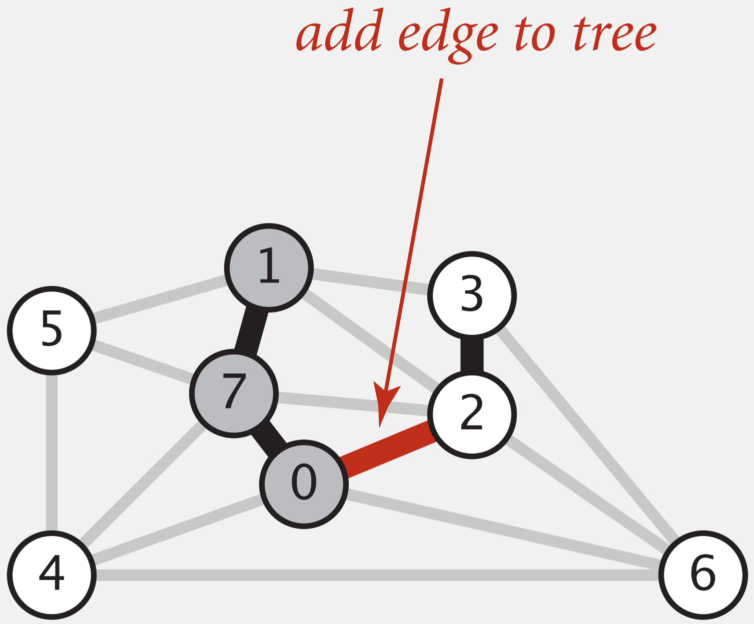


Kruskal's algorithm: correctness proof

Proposition. [Kruskal 1956] Kruskal's algorithm computes the MST.

Pf. Kruskal's algorithm is a special case of the greedy MST algorithm.

- Suppose Kruskal's algorithm colors the edge $e = v–w$ black.
- Cut = set of vertices connected to v in tree T.
- No crossing edge is black.
- No crossing edge has lower weight. Why?



Kruskal's algorithm: implementation challenge

Challenge. Would adding edge v–w to tree T create a cycle? If not, add it.

How difficult to implement?

A. $E + V$

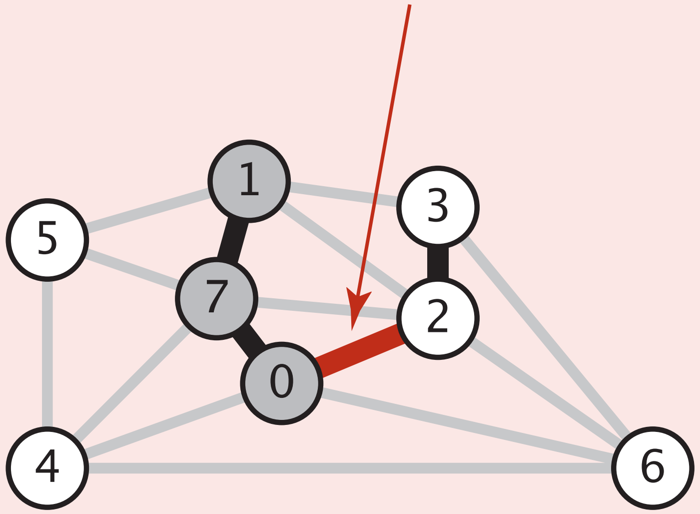
B. V

C. $\log V$ (or $\log^* V$)

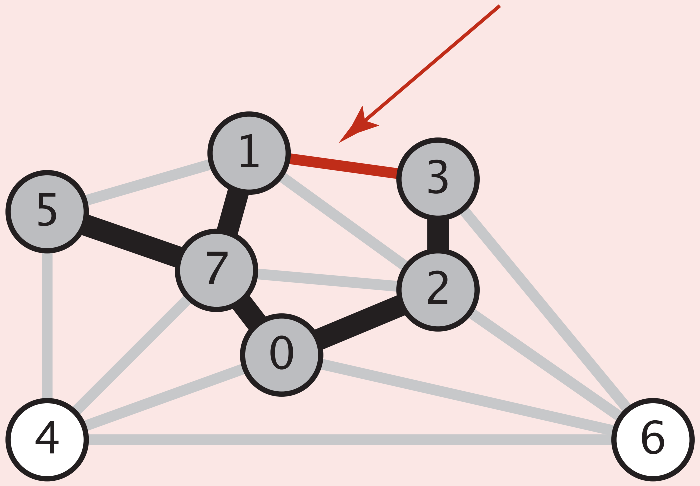
D. $\log E$ (or $\log^* E$)

E. 1

add edge to tree



adding edge to tree would create a cycle

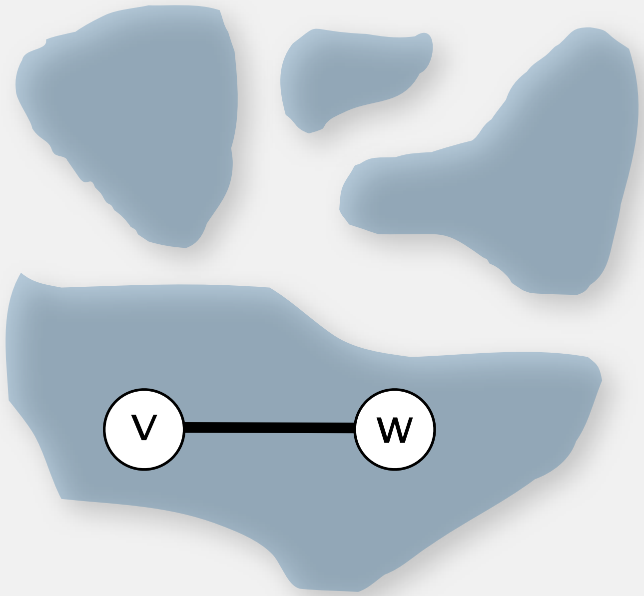


Kruskal's algorithm: implementation challenge

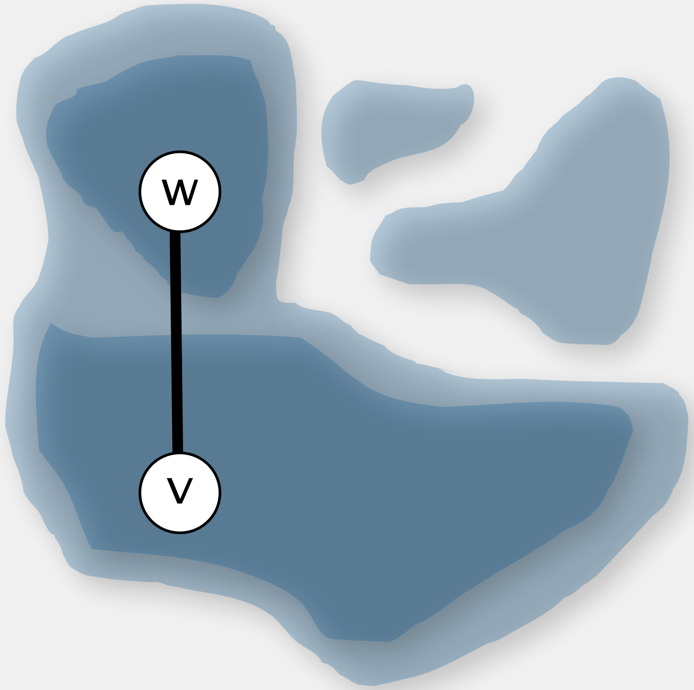
Challenge. Would adding edge v–w to tree T create a cycle? If not, add it.

Efficient solution. Use the union-find data structure.

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



Case 1: adding v–w creates a cycle



Case 2: add v–w to T and merge sets containing v and w

Kruskal's algorithm: Java implementation

```
public class KruskalMST
{
   private Queue<Edge> mst = new Queue<Edge>();

   public KruskalMST(EdgeWeightedGraph G)
   {
      MinPQ<Edge> pq = new MinPQ<Edge>(G.edges());

      UF uf = new UF(G.V());
      while (!pq.isEmpty() && mst.size() < G.V()-1)
      {
         Edge e = pq.delMin();
         int v = e.either(), w = e.other(v);
         if (!uf.connected(v, w))
         {
            uf.union(v, w);
            mst.enqueue(e);
         }
      }
   }

   public Iterable<Edge> edges()
   {  return mst;  }
}
```

build priority queue
(or sort)

greedily add edges to MST

edge v–w does not create cycle

merge connected components

add edge e to MST

Kruskal's algorithm: running time

Proposition. Kruskal's algorithm computes MST in time proportional to $E \log E$ (in the worst case).

Pf.

operation	frequency	time per op
build pq	1	E
delete-min	E	$\log E$
union	V	$\log^* V$ †
connected	E	$\log^* V$ †

← often called fewer than E times

† amortized bound using weighted quick union with path compression


Algorithms

Robert Sedgewick | Kevin Wayne

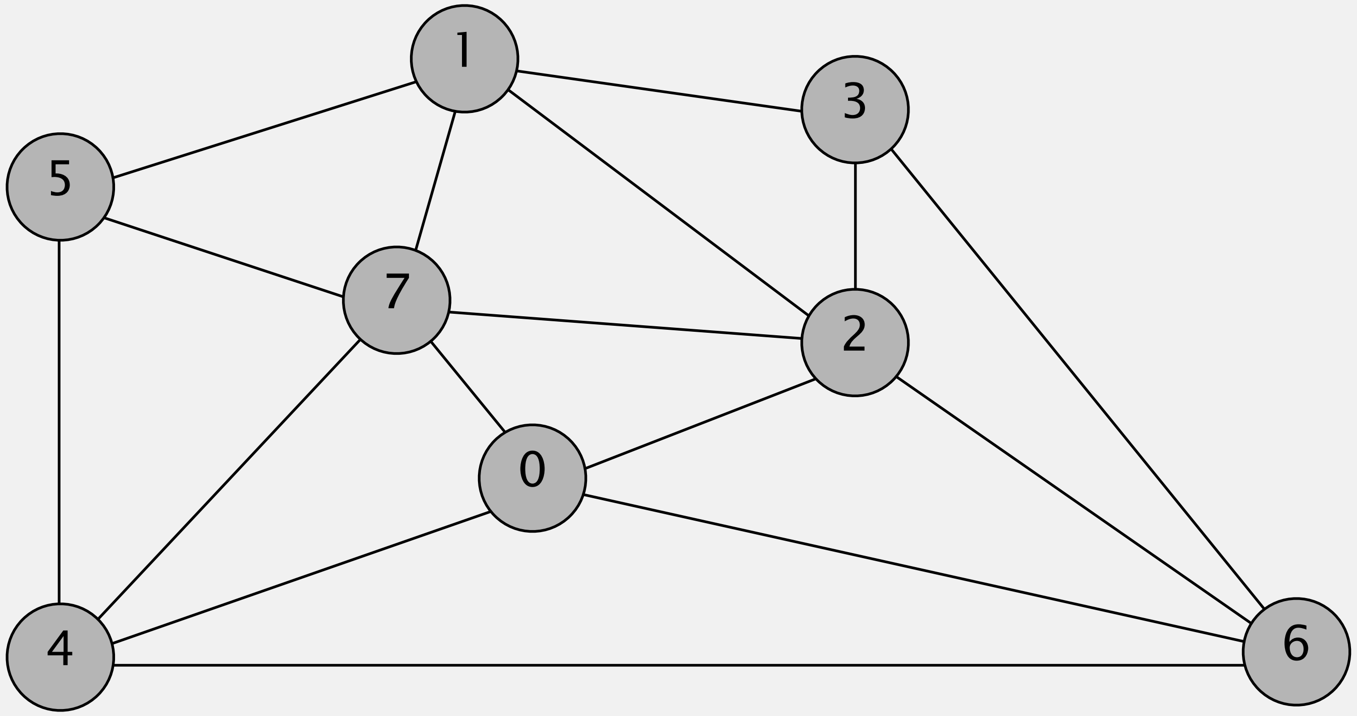
http://algs4.cs.princeton.edu

4.3 MINIMUM SPANNING TREES

- *introduction*
- *greedy algorithm*
- *edge-weighted graph API*
- *Kruskal's algorithm*
- *Prim's algorithm*
- *context*

Prim's algorithm demo

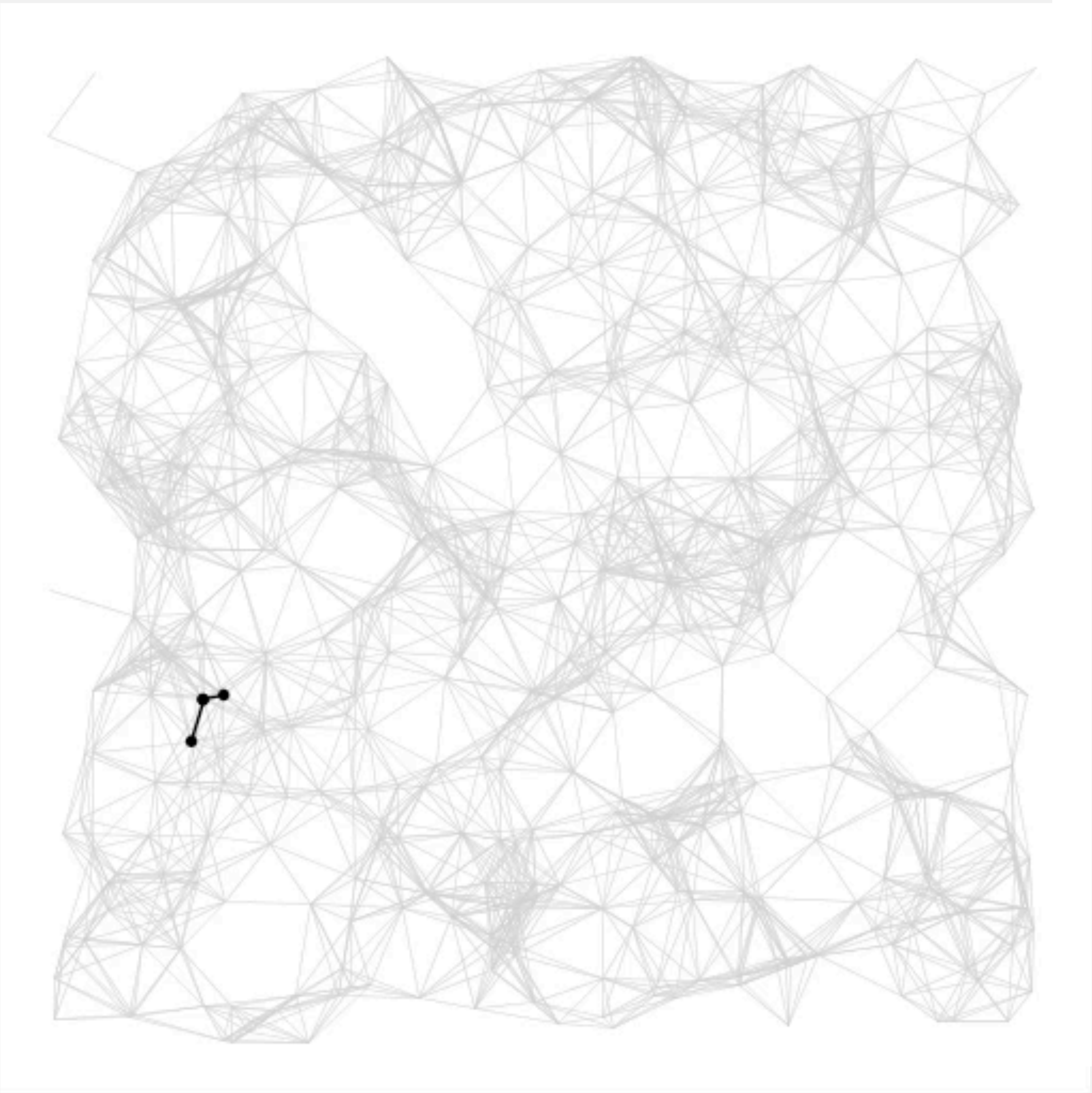
- Start with vertex 0 and greedily grow tree T.
- Add to T the min weight edge with exactly one endpoint in T.
- Repeat until $V - 1$ edges.



an edge-weighted graph

0-7	0.16
2-3	0.17
1-7	0.19
0-2	0.26
5-7	0.28
1-3	0.29
1-5	0.32
2-7	0.34
4-5	0.35
1-2	0.36
4-7	0.37
0-4	0.38
6-2	0.40
3-6	0.52
6-0	0.58
6-4	0.93

Prim's algorithm: visualization



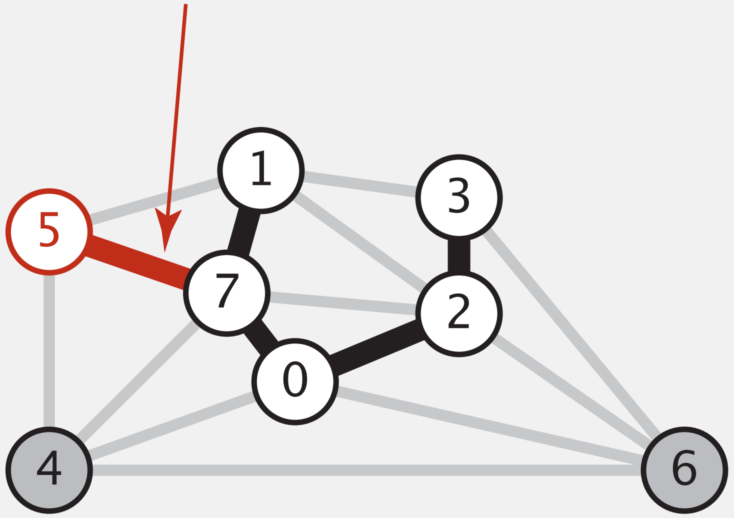
Prim's algorithm: proof of correctness

Proposition. [Jarník 1930, Dijkstra 1957, Prim 1959]
Prim's algorithm computes the MST.

Pf. Prim's algorithm is a special case of the greedy MST algorithm.

- Suppose edge *e* = min weight edge connecting a vertex on the tree to a vertex not on the tree.
- Cut = set of vertices connected on tree.
- No crossing edge is black.
- No crossing edge has lower weight.

edge e = 7-5 added to tree



Prim's algorithm: implementation challenge

Challenge. Find the min weight edge with exactly one endpoint in T.

How difficult?

A. E

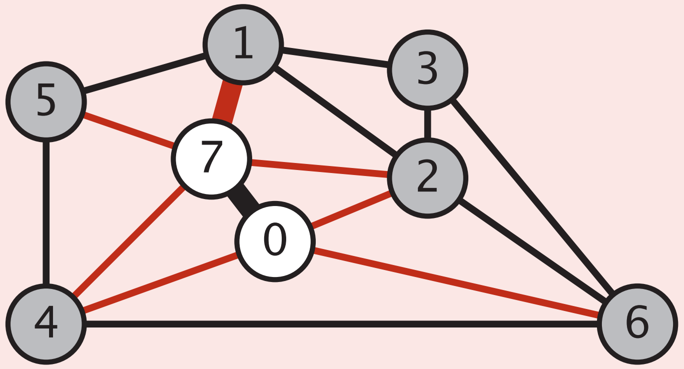
B. V

C. $\log E$

D. 1

E. *I don't know.*

1-7 is min weight edge with exactly one endpoint in T



1-7	0.19
0-2	0.26
5-7	0.28
2-7	0.34
4-7	0.37
0-4	0.38
6-0	0.58

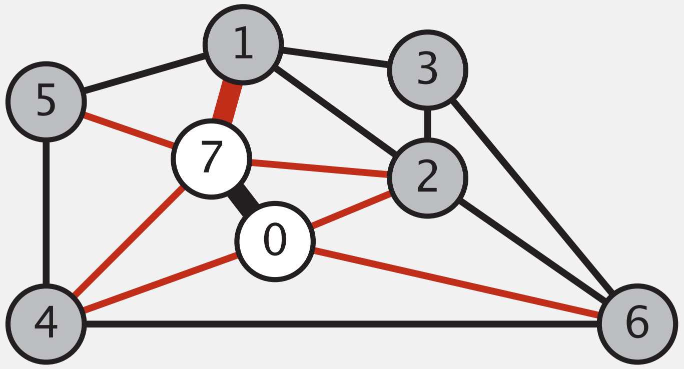
Prim's algorithm: lazy implementation

Challenge. Find the min weight edge with exactly one endpoint in T.

Lazy solution. Maintain a PQ of edges with (at least) one endpoint in T.

- Key = edge; priority = weight of edge.
- Delete-min to determine next edge $e = v–w$ to add to T.
- Disregard if both endpoints v and w are marked (both in T).
- Otherwise, let w be the unmarked vertex (not in T):
  - add e to T and mark w
  - add to PQ all edges incident to w (assuming other endpoint not in T)

1-7 is min weight edge with exactly one endpoint in T

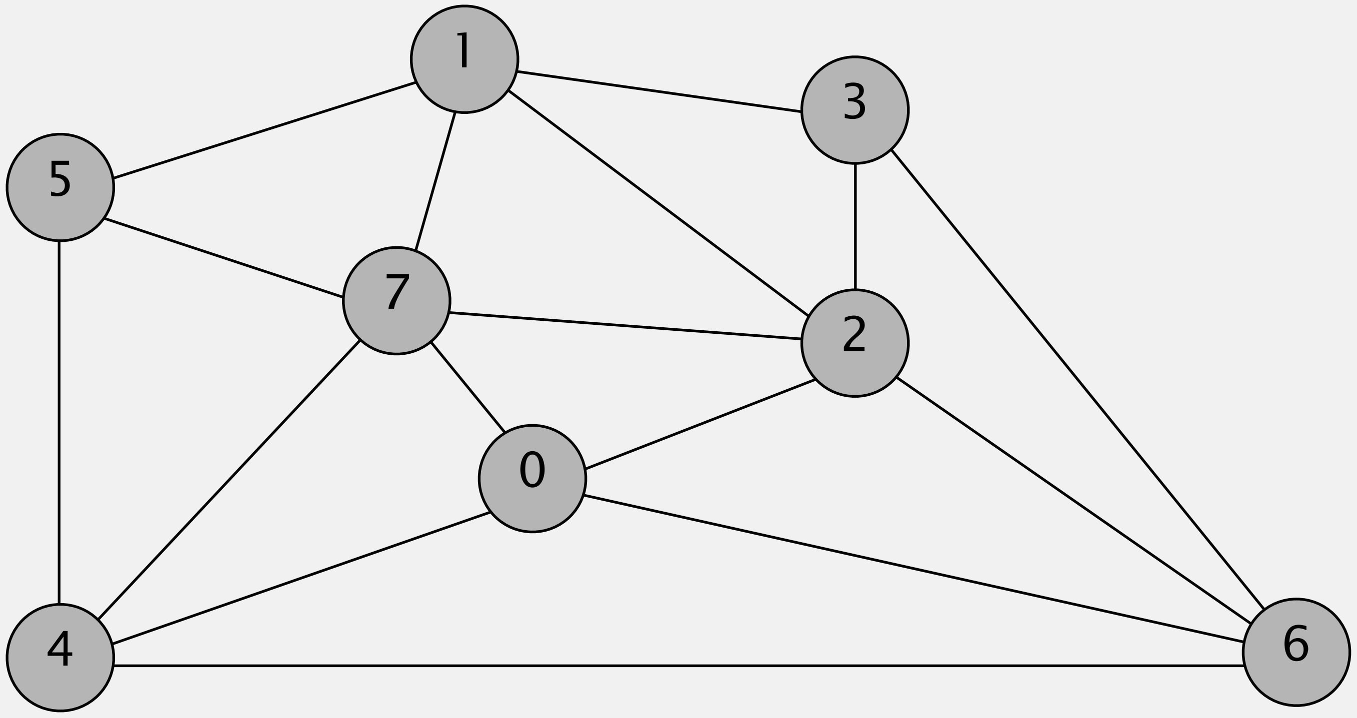


priority queue of crossing edges

1-7	0.19
0-2	0.26
5-7	0.28
2-7	0.34
4-7	0.37
0-4	0.38
6-0	0.58

Prim's algorithm: lazy implementation demo

- Start with vertex 0 and greedily grow tree T.
- Add to T the min weight edge with exactly one endpoint in T.
- Repeat until $V - 1$ edges.



an edge-weighted graph

0-7	0.16
2-3	0.17
1-7	0.19
0-2	0.26
5-7	0.28
1-3	0.29
1-5	0.32
2-7	0.34
4-5	0.35
1-2	0.36
4-7	0.37
0-4	0.38
6-2	0.40
3-6	0.52
6-0	0.58
6-4	0.93

Prim's algorithm: lazy implementation

```
public class LazyPrimMST
{
   private boolean[] marked;    // MST vertices
   private Queue<Edge> mst;     // MST edges
   private MinPQ<Edge> pq;      // PQ of edges

    public LazyPrimMST(WeightedGraph G)
    {
        pq = new MinPQ<Edge>();
        mst = new Queue<Edge>();
        marked = new boolean[G.V()];
        visit(G, 0);

        while (!pq.isEmpty() && mst.size() < G.V() - 1)
        {
           Edge e = pq.delMin();
           int v = e.either(), w = e.other(v);
           if (marked[v] && marked[w]) continue;
           mst.enqueue(e);
           if (!marked[v]) visit(G, v);
           if (!marked[w]) visit(G, w);
        }
   }
}
```

assume G is connected

repeatedly delete the
min weight edge e = v–w from PQ

ignore if both endpoints in T

add edge e to tree

add either v or w to tree

Prim's algorithm: lazy implementation

```
private void visit(WeightedGraph G, int v)
{
   marked[v] = true;
   for (Edge e : G.adj(v))
      if (!marked[e.other(v)])
         pq.insert(e);
}

public Iterable<Edge> mst()
{  return mst;  }
```

add v to T

for each edge e = v–w, add to PQ if w not already in T

Lazy Prim's algorithm: running time

Proposition. Lazy Prim's algorithm computes the MST in time proportional to $E \log E$ and extra space proportional to E (in the worst case).

↖ minor defect

Pf.

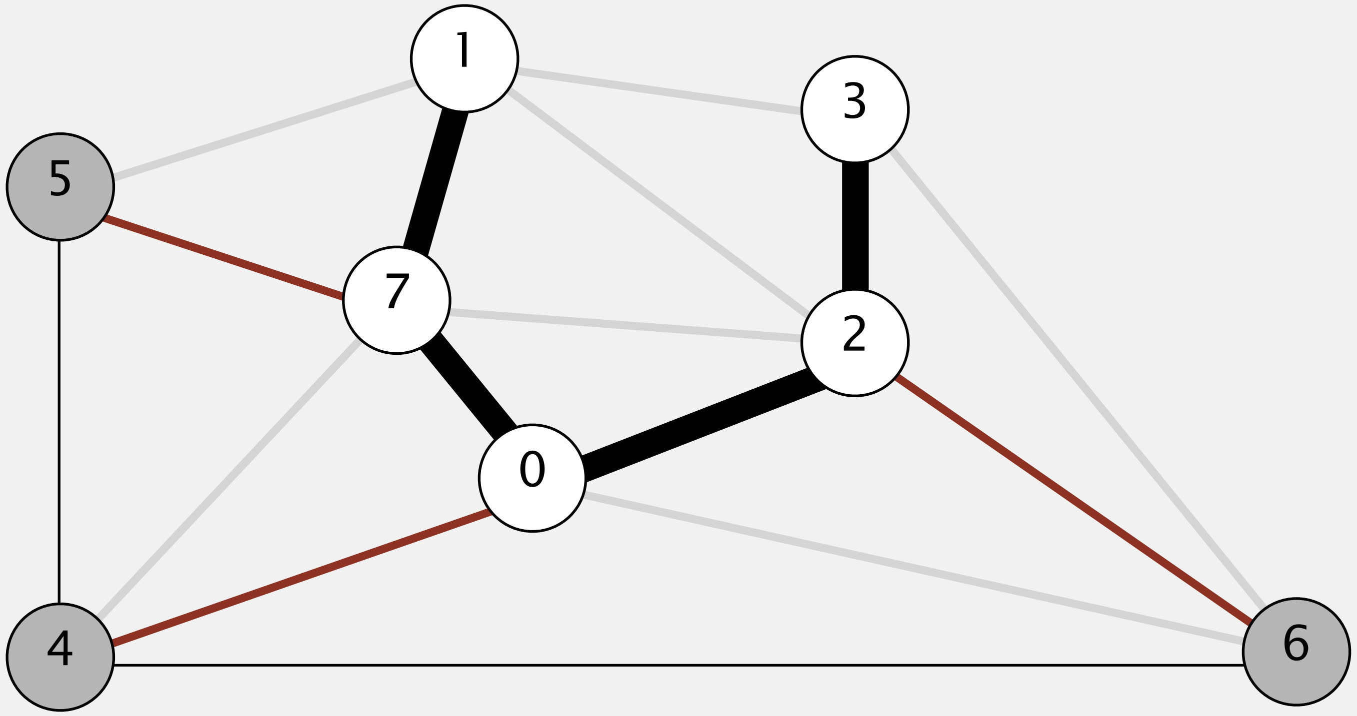
operation	frequency	binary heap
delete min	E	$\log E$
insert	E	$\log E$

Prim's algorithm: eager implementation

Challenge. Find min weight edge with exactly one endpoint in T.

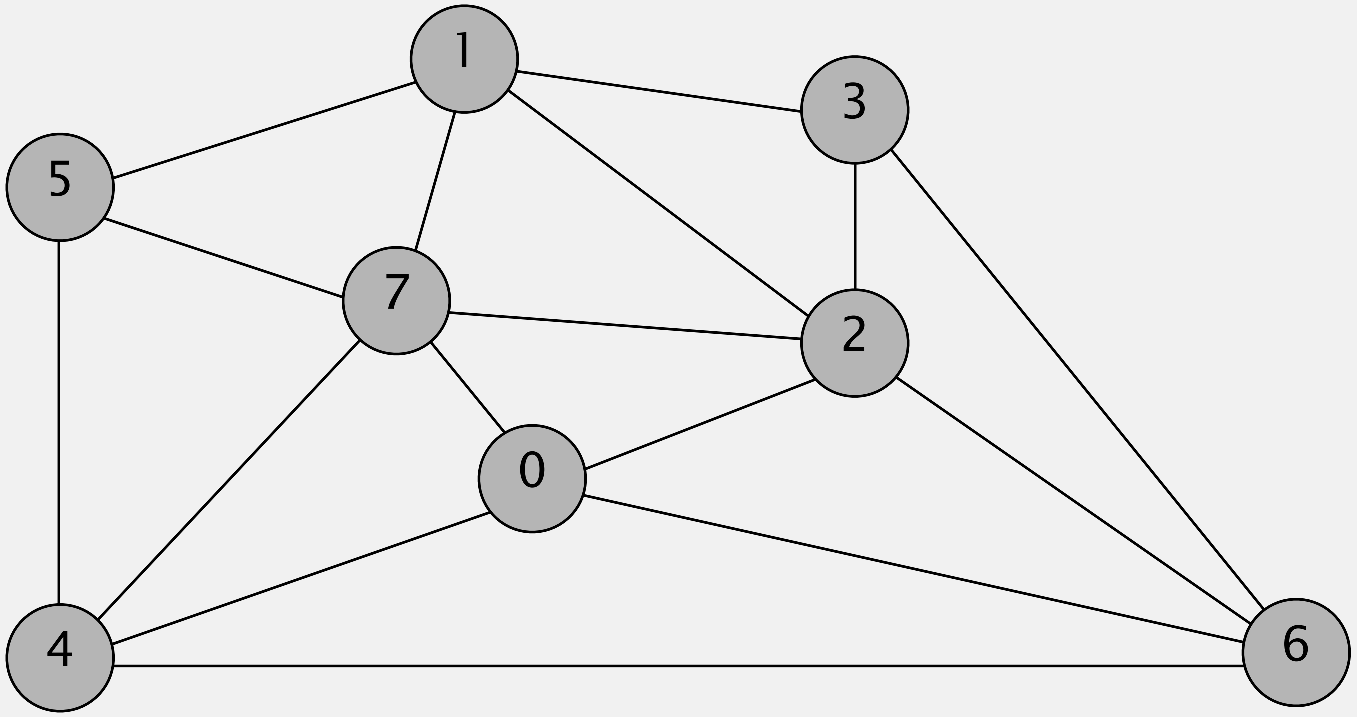
Observation. For each vertex v, need only lightest edge connecting v to T.

- MST includes at most one edge connecting v to T. Why?
- If MST includes such an edge, it must take lightest such edge. Why?



Prim's algorithm: eager implementation demo

- Start with vertex 0 and greedily grow tree T.
- Add to T the min weight edge with exactly one endpoint in T.
- Repeat until $V-1$ edges.

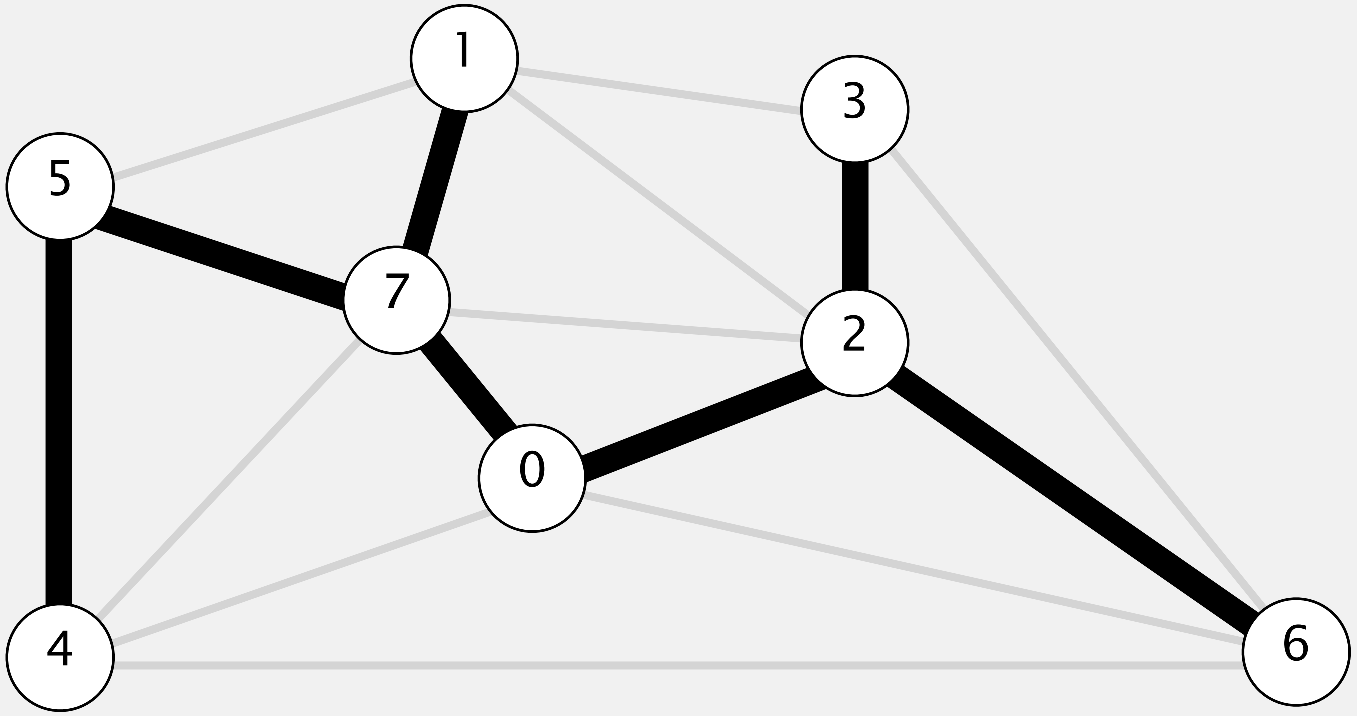


an edge-weighted graph

0-7	0.16
2-3	0.17
1-7	0.19
0-2	0.26
5-7	0.28
1-3	0.29
1-5	0.32
2-7	0.34
4-5	0.35
1-2	0.36
4-7	0.37
0-4	0.38
6-2	0.40
3-6	0.52
6-0	0.58
6-4	0.93

Prim's algorithm: eager implementation demo

- Start with vertex 0 and greedily grow tree T.
- Add to T the min weight edge with exactly one endpoint in T.
- Repeat until $V - 1$ edges.



v	edgeTo[]	distTo[]
0	-	-
7	0–7	0.16
1	1–7	0.19
2	0–2	0.26
3	2–3	0.17
5	5–7	0.28
4	4–5	0.35
6	6–2	0.40

MST edges

0-7 1-7 0-2 2-3 5-7 4-5 6-2

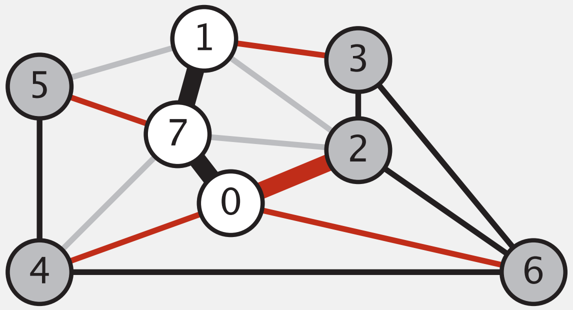
Prim's algorithm: eager implementation

Challenge. Find min weight edge with exactly one endpoint in T.

PQ has at most one entry per vertex

Eager solution. Maintain a PQ of vertices connected by an edge to T, where priority of vertex v = weight of lightest edge connecting v to T.

- Delete min vertex v and add its associated edge $e = v–w$ to T.
- Update PQ by considering all edges $e = v–x$ incident to v
  - ignore if x is already in T
  - add x to PQ if not already on it
  - decrease priority of x if $v–x$ becomes lightest edge connecting x to T



```
0
1 1-7 0.19
2 0-2 0.26
3 1-3 0.29
4 0-4 0.38
5 5-7 0.28
6 6-0 0.58
7 0-7 0.16
```

red: on PQ

black: on MST

Indexed priority queue

Associate an index between 0 and $N - 1$ with each key in a priority queue.

- Insert a key associated with a given index.
- Delete a minimum key and return associated index.
- Decrease the key associated with a given index.

for Prim's algorithm,
N = V and index = vertex.

`public class IndexMinPQ<Key extends Comparable<Key>>`

	`IndexMinPQ(int N)`	*create indexed priority queue with indices $0, 1, \ldots, N - 1$*
`void`	`insert(int i, Key key)`	*associate key with index i*
`int`	`delMin()`	*remove a minimal key and return its associated index*
`void`	`decreaseKey(int i, Key key)`	*decrease the key associated with index i*
`boolean`	`contains(int i)`	*is i an index on the priority queue?*
`boolean`	`isEmpty()`	*is the priority queue empty?*
`int`	`size()`	*number of keys in the priority queue*

Indexed priority queue: implementation

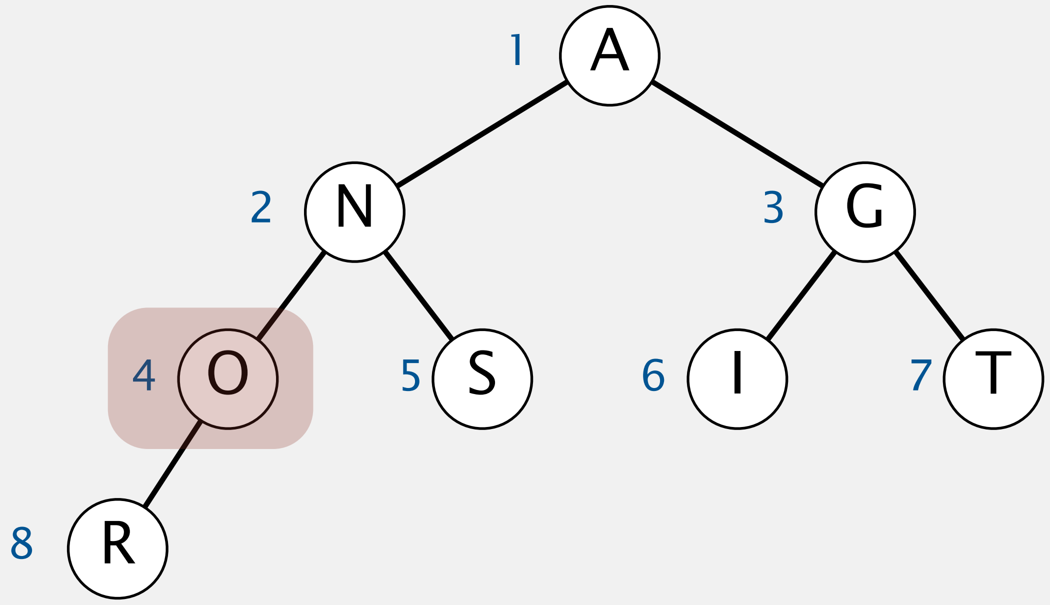
Binary heap implementation. [see Section 2.4 of textbook]

- Start with same code as `MinPQ`.
- Maintain parallel arrays so that:
  - `keys[i]` is the priority of vertex `i`
  - `qp[i]` is the heap position of vertex `i`
  - `pq[i]` is the index of the key in heap position `i`
- Use `swim(qp[i])` to implement `decreaseKey(i, key)`.

i	0	1	2	3	4	5	6	7	8
keys[i]	A	S	O	R	T	I	N	G	-
qp[i]	1	5	4	8	7	6	2	3	-
pq[i]	-	0	6	7	2	1	5	4	3

vertex 2 is at heap index 4

decrease key of vertex 2 to C



Prim's algorithm: which priority queue?

Depends on PQ implementation: V insert, V delete-min, E decrease-key.

PQ implementation	insert	delete-min	decrease-key	total
unordered array	1	V	1	V^2
binary heap	$\log V$	$\log V$	$\log V$	$E \log V$
d-way heap	$\log_d V$	$d \log_d V$	$\log_d V$	$E \log_{E/V} V$
Fibonacci heap	1 †	$\log V$ †	1 †	$E + V \log V$

† amortized

Bottom line.

- Array implementation optimal for dense graphs.
- Binary heap much faster for sparse graphs.
- 4-way heap worth the trouble in performance-critical situations.
- Fibonacci heap best in theory, but not worth implementing.


Algorithms

Robert Sedgewick | Kevin Wayne

http://algs4.cs.princeton.edu

4.3 MINIMUM SPANNING TREES

- *introduction*
- *greedy algorithm*
- *edge-weighted graph API*
- *Kruskal's algorithm*
- *Prim's algorithm*
- ***context***

Does a linear-time MST algorithm exist?

deterministic compare-based MST algorithms

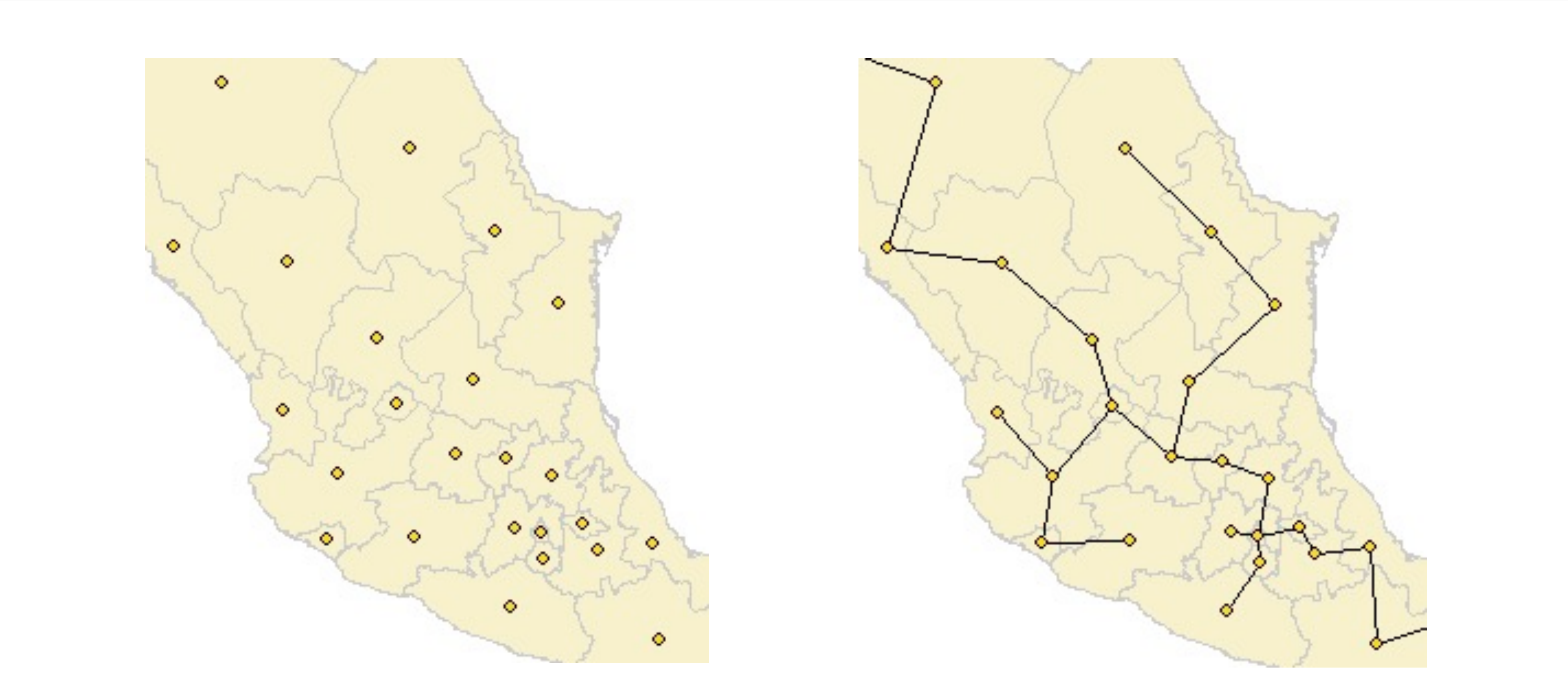
year	worst case	discovered by
1975	$E \log \log V$	Yao
1976	$E \log \log V$	Cheriton-Tarjan
1984	$E \log^* V,\ E + V \log V$	Fredman-Tarjan
1986	$E \log (\log^* V)$	Gabow-Galil-Spencer-Tarjan
1997	$E\, \alpha(V) \log \alpha(V)$	Chazelle
2000	$E\, \alpha(V)$	Chazelle
2002	*optimal*	Pettie-Ramachandran
20xx	E	???

PRINCETON UNIVERSITY

Remark. Linear-time randomized MST algorithm (Karger-Klein-Tarjan 1995).

Euclidean MST

Given N points in the plane, find MST connecting them, where the distances between point pairs are their Euclidean distances.



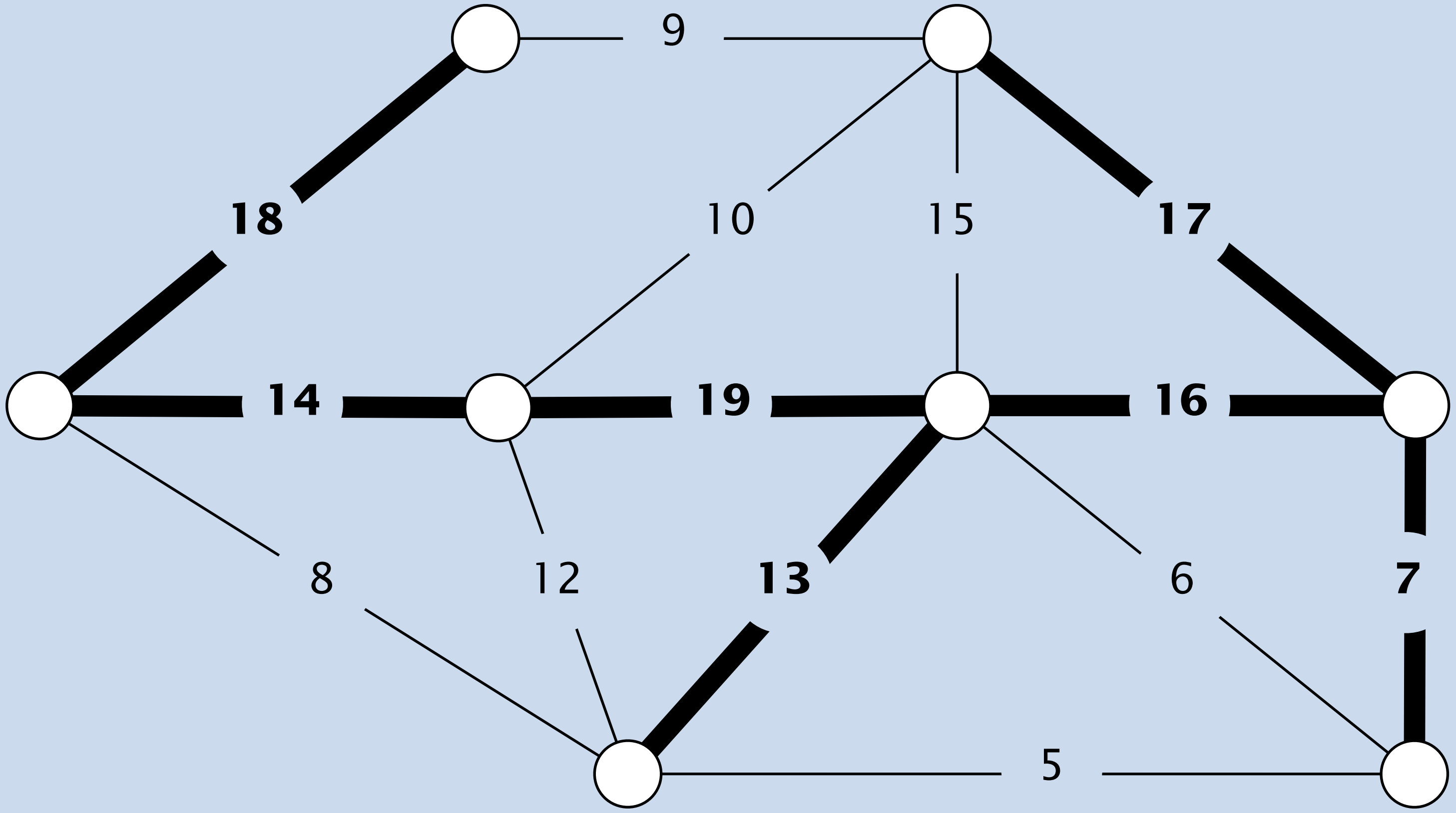
Brute force. Compute $\sim N^2 / 2$ distances and run Prim's algorithm.

Ingenuity. Exploit geometry and do it in $N \log N$ time.

MAXIMUM SPANNING TREE

Problem. Given an edge-weighted graph G, find a spanning tree that maximizes the sum of the edge weights.

Running time. $E \log E$ (or better).

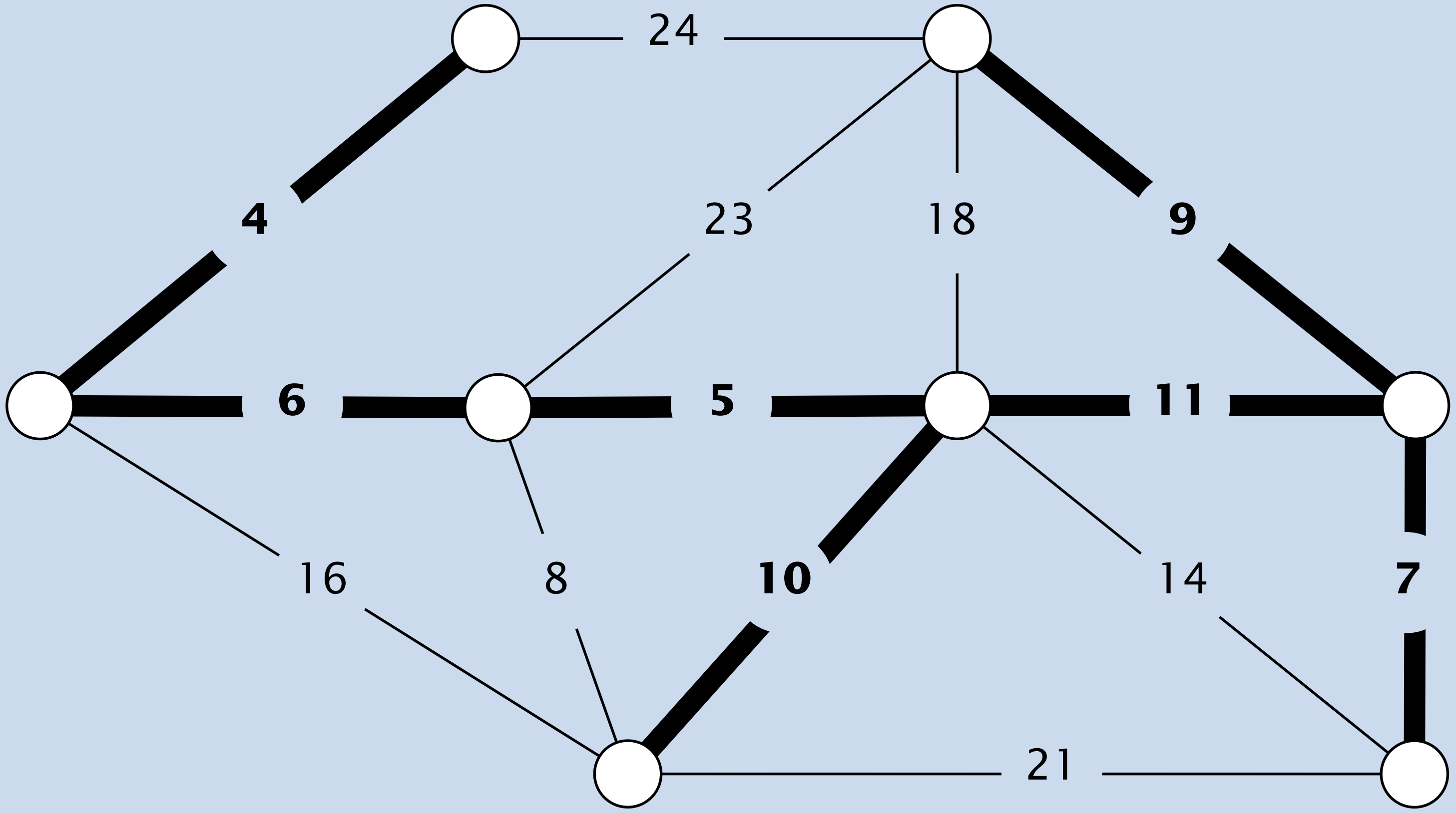


maximum spanning tree T (weight = 104)

MINIMUM SUM-OF-SQUARES SPANNING TREE

Problem. Given an edge-weighted graph G, find a spanning tree that minimizes the sum of the squares of its edge weights.

Running time. $E \log E$ (or better).

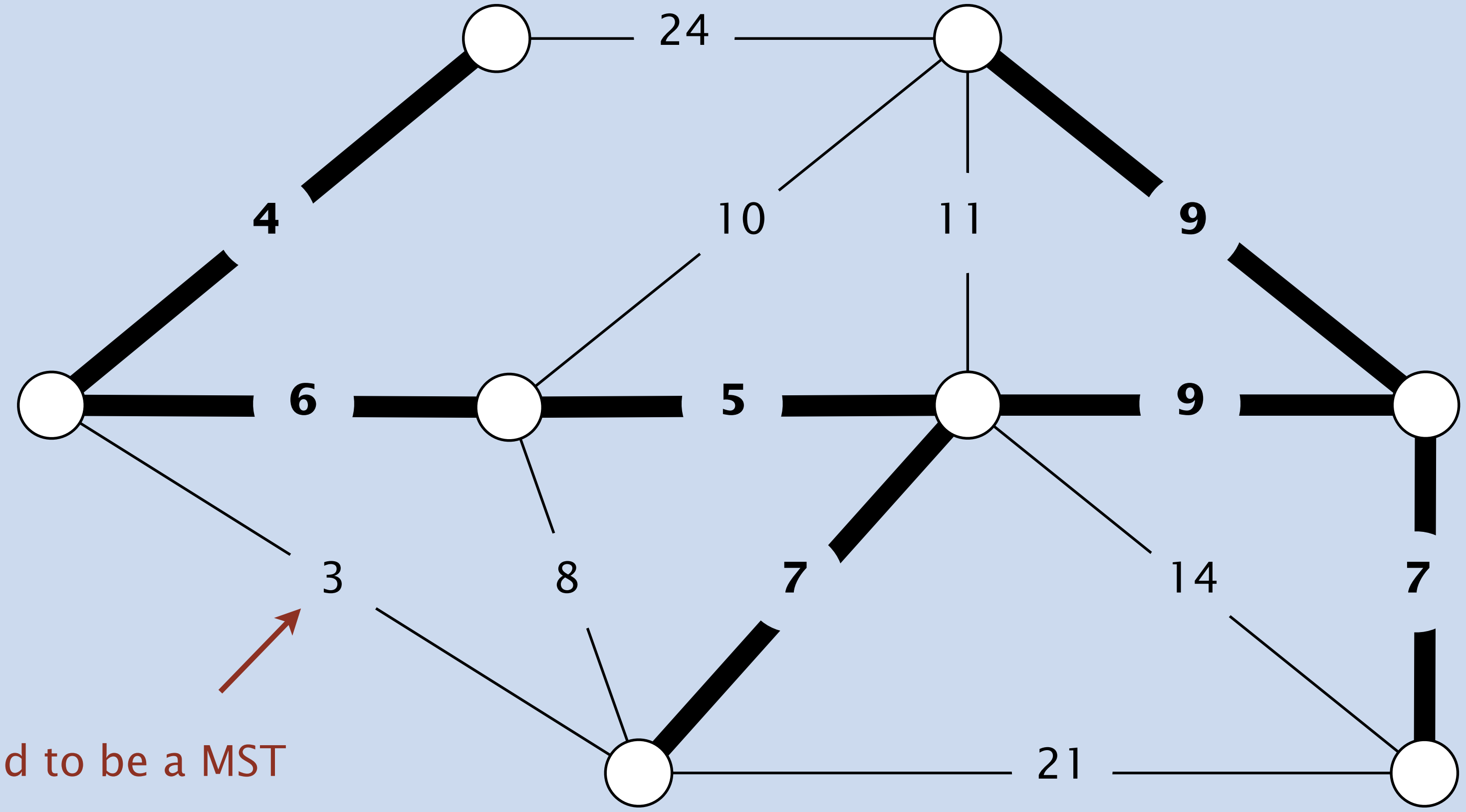


maximum spanning tree T (weight = $4^2 + 6^2 + \ldots + 7^2$)

MINIMUM BOTTLENECK SPANNING TREE

Problem. Given an edge-weighted graph G, find a spanning tree that minimizes the maximum weight of its edges.

Running time. $E \log E$ (or better).



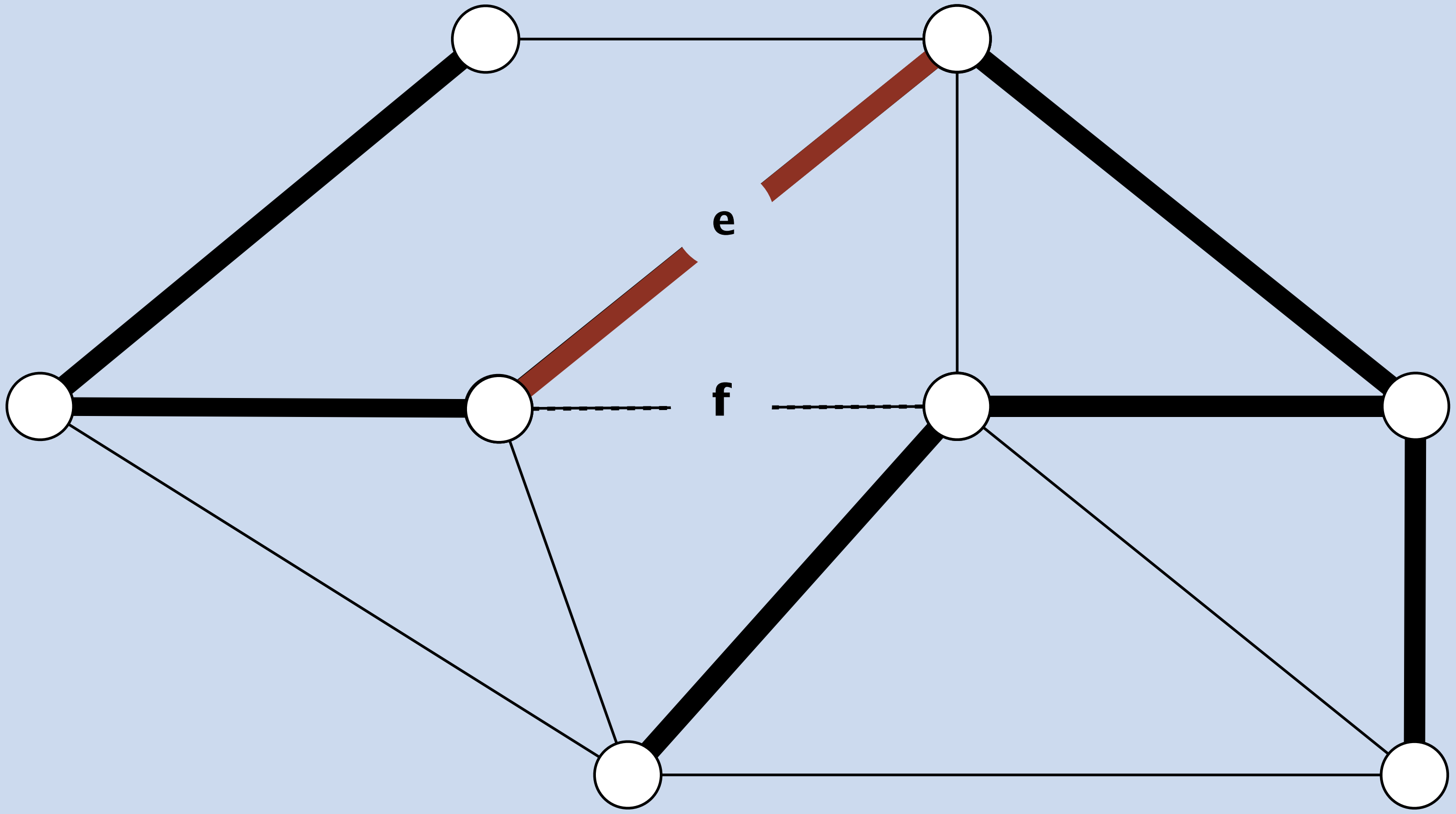
minimum bottleneck spanning tree T (bottleneck = 9)

MINIMUM BOTTLENECK SPANNING TREE

Solution. Compute a MST; it is a MBST.

Pf. Suppose MST is not a MBST.

- Let e = edge in MST with weight strictly larger than bottleneck weight.
- Consider cut formed by deleting e from MST.
- MBST contains at least one edge f crossing cut.
- Adding f to MST and deleting e yields better MST.



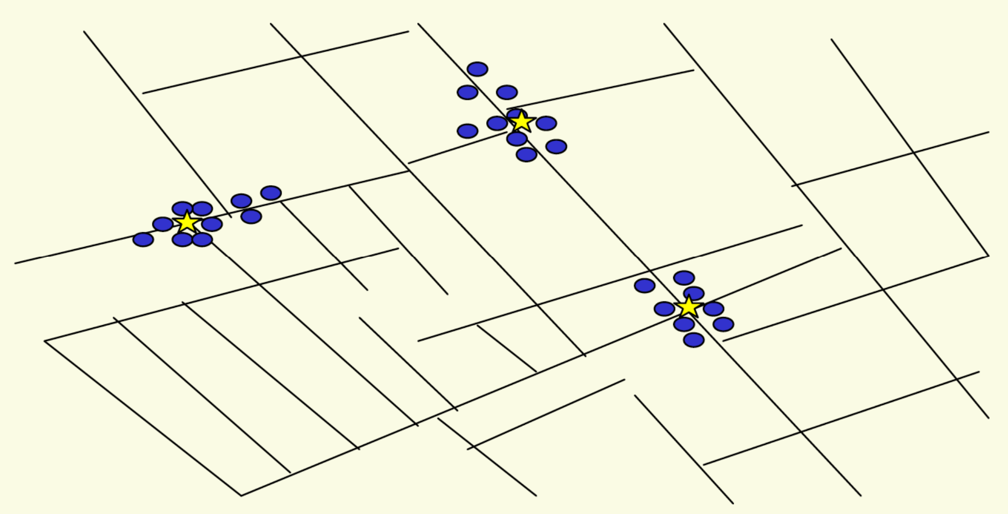
minimum spanning tree

Scientific application: 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.



outbreak of cholera deaths in London in 1850s (Nina Mishra)

Applications.

- Routing in mobile ad hoc networks.
- Document categorization for web search.
- Similarity searching in medical image databases.
- Skycat: cluster 10^9 sky objects into stars, quasars, galaxies.

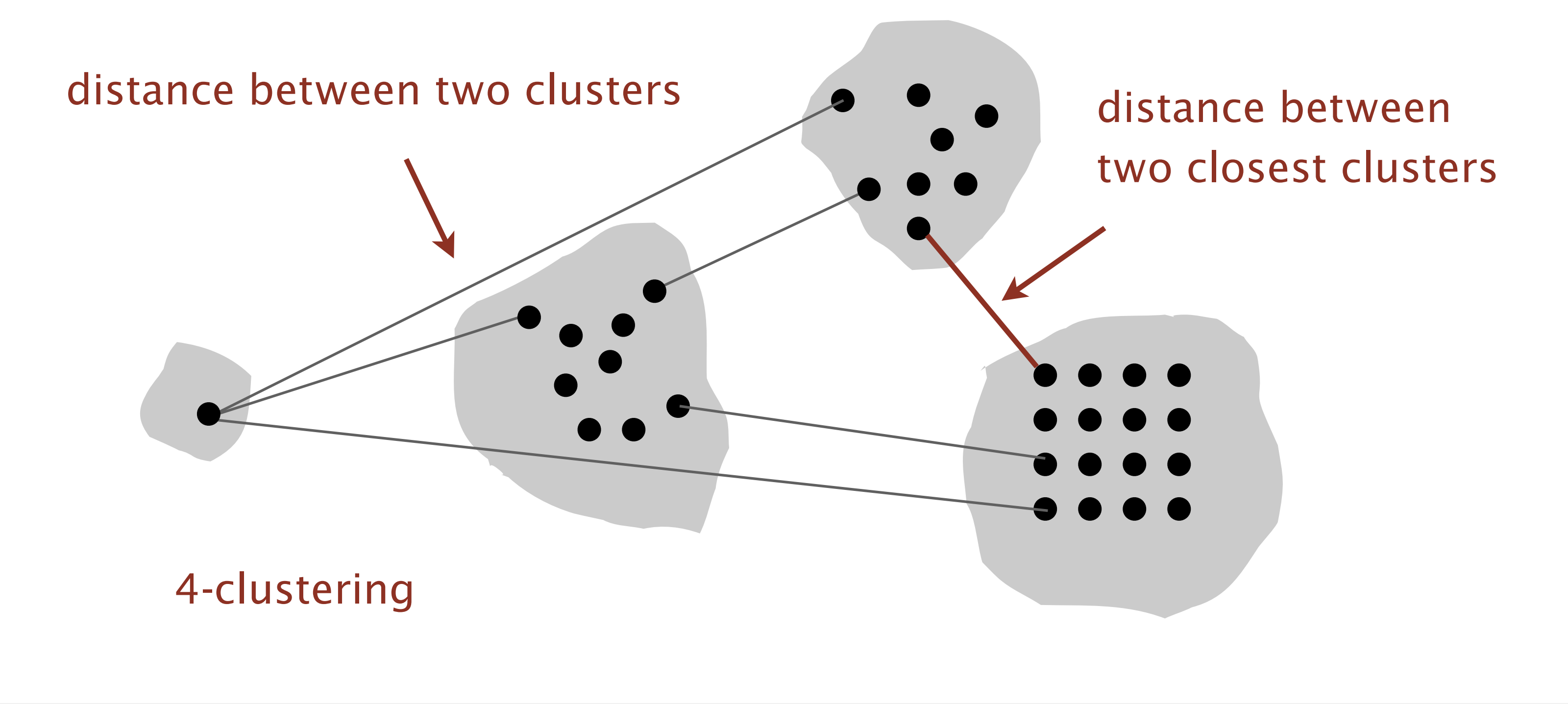
Single-link clustering

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

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

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

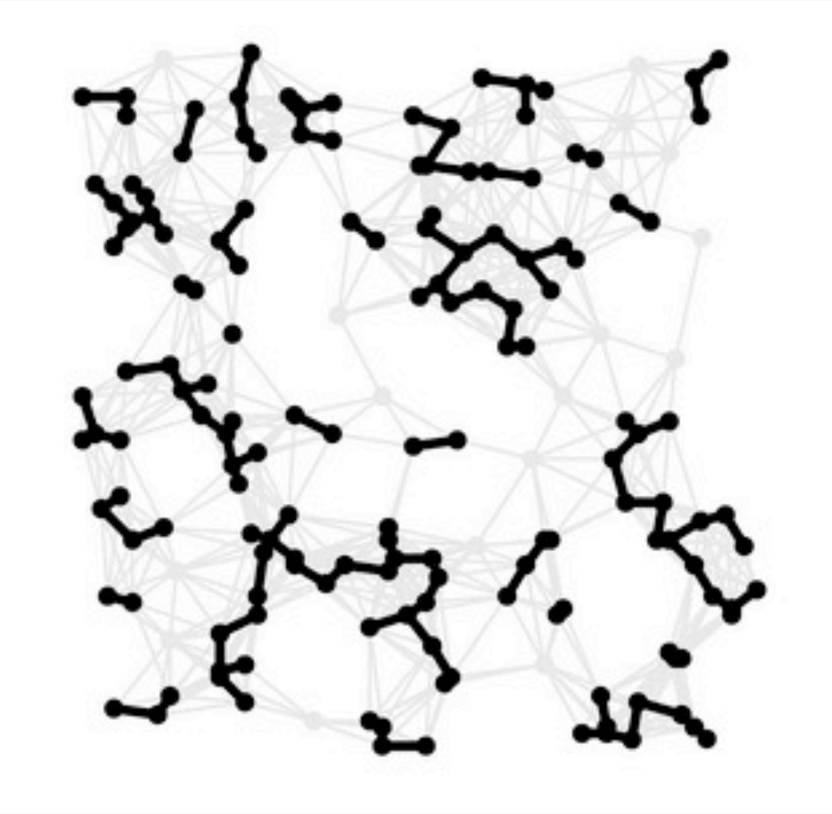
Single-link clustering. Given an integer k, find a k-clustering that maximizes the distance between two closest clusters.



Single-link clustering algorithm

"Well-known" algorithm in science literature for single-link 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.

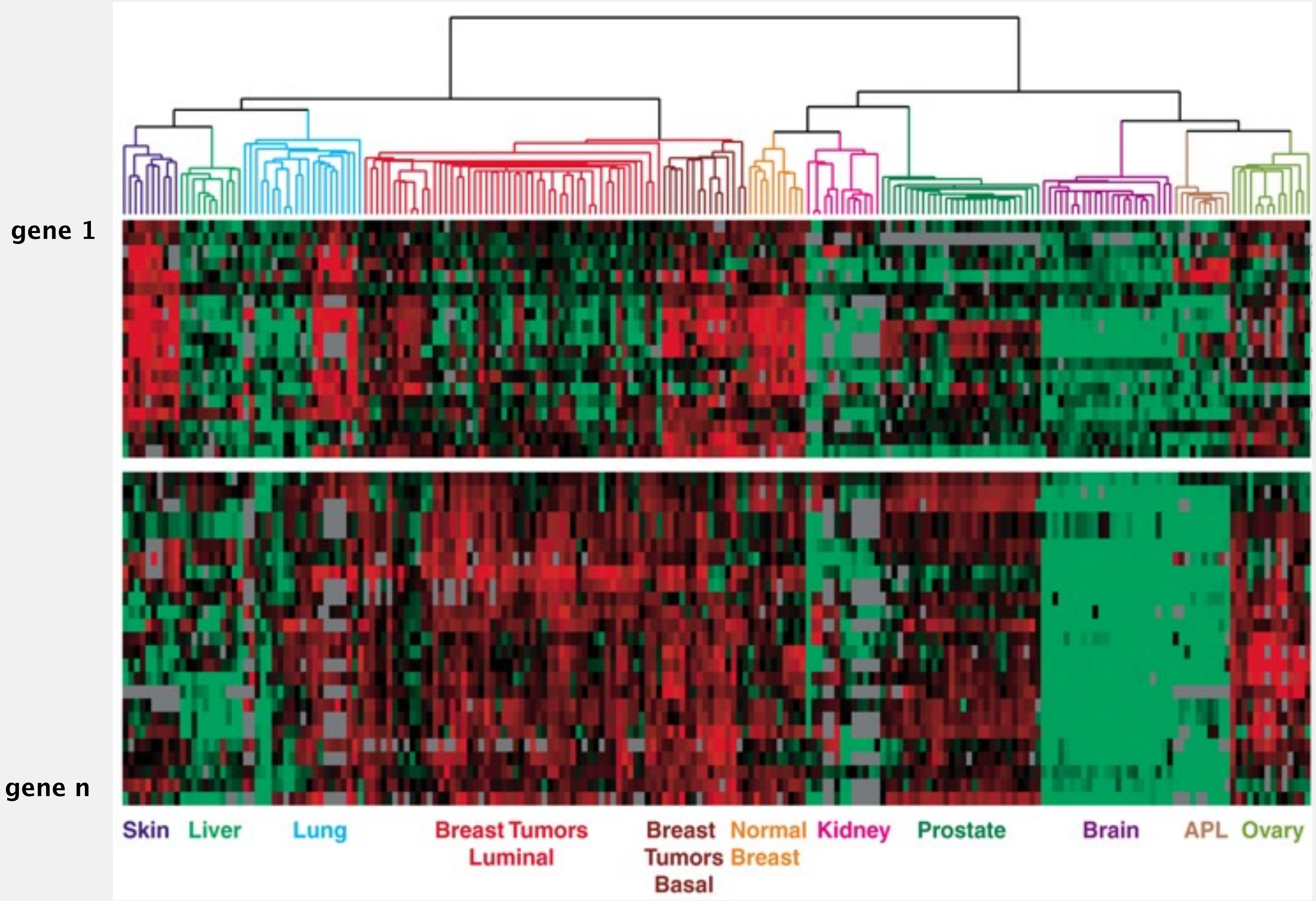


Observation. This is Kruskal's algorithm.
(stopping when k connected components)

Alternate solution. Run Prim; then delete $k-1$ max weight edges.

Dendrogram of cancers in human

Tumors in similar tissues cluster together.



Reference: Botstein & Brown group

🟥 **gene expressed**

🟩 **gene not expressed**