

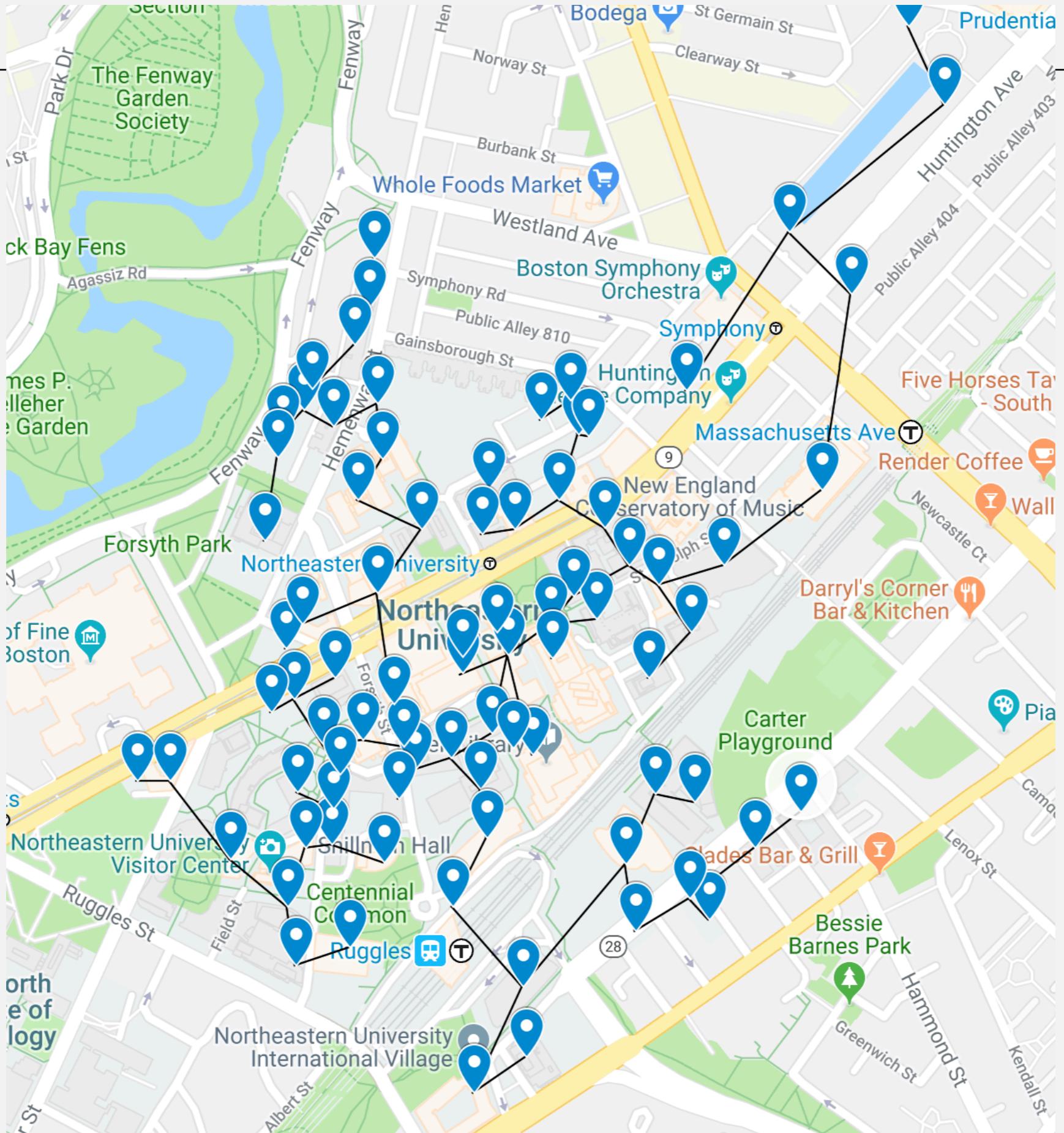
ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

4.3 MINIMUM SPANNING TREES

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

Tunnels Map



Graphs and Trees

Remember how we constructed a tree from a graph to answer the question

- is s connected to v ?

We did this for depth-first-search and breadth-first-search.

We did it for undirected graphs and directed graphs.

In each case we knew the starting vertex s .

Can we build a more general tree that connects all the vertices of an undirected graph without having to specify the starting vertex s ?

- Yes, this is called a spanning tree.

What if the edges have weights and we can choose a particular spanning tree, the total weight of all edges being a minimum?

- Yes, this is called a minimum spanning tree.

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

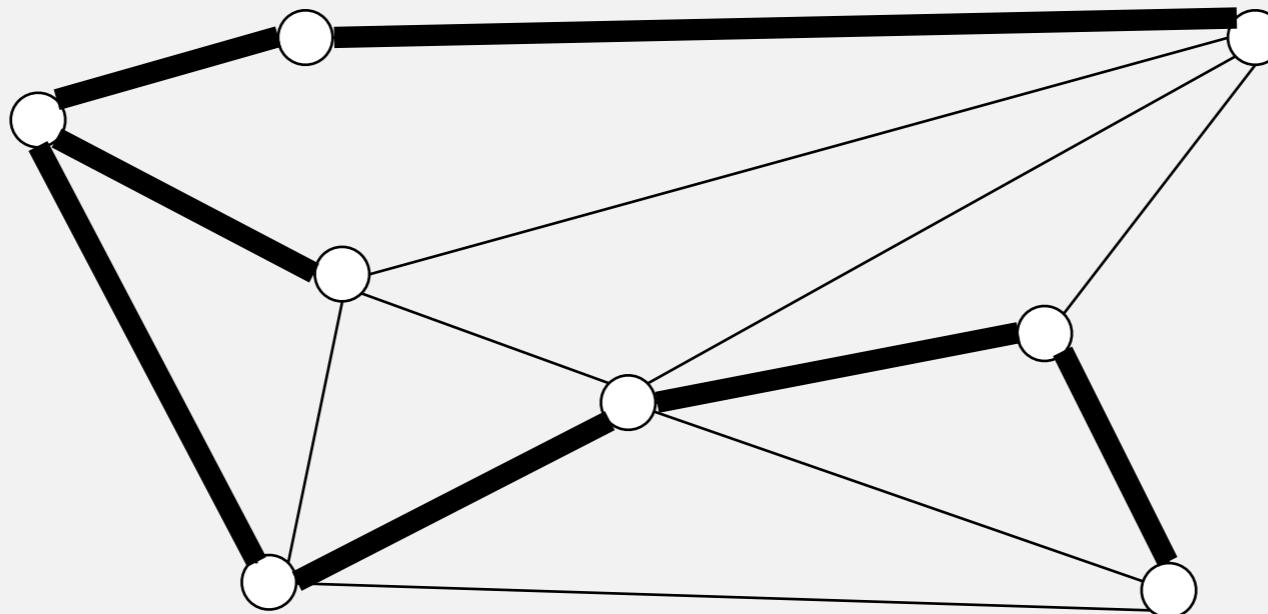
4.3 MINIMUM SPANNING

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

Minimum spanning tree

Def. A **spanning tree** of G is a subgraph T that is:

- Connected.
- Acyclic.
- Includes all of the vertices.

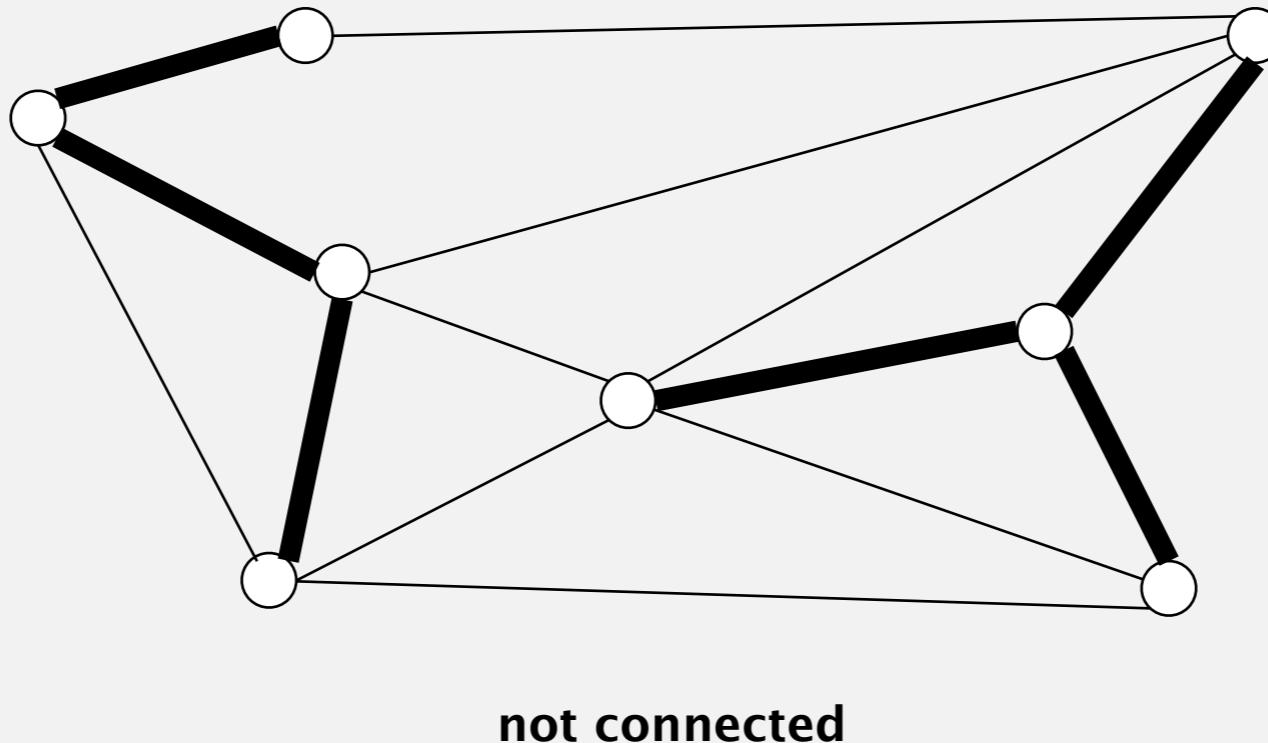


graph G

Minimum spanning tree

Def. A **spanning tree** of G is a subgraph T that is:

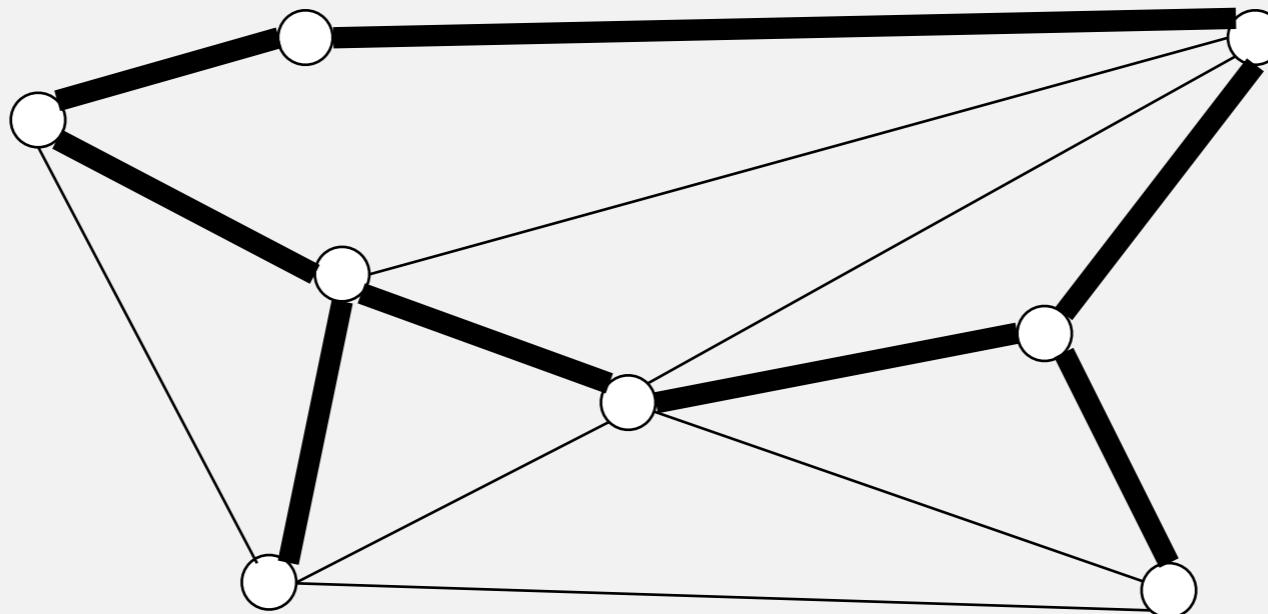
- Connected.
- Acyclic.
- Includes all of the vertices.



Minimum spanning tree

Def. A **spanning tree** of G is a subgraph T that is:

- Connected.
- Acyclic.
- Includes all of the vertices.

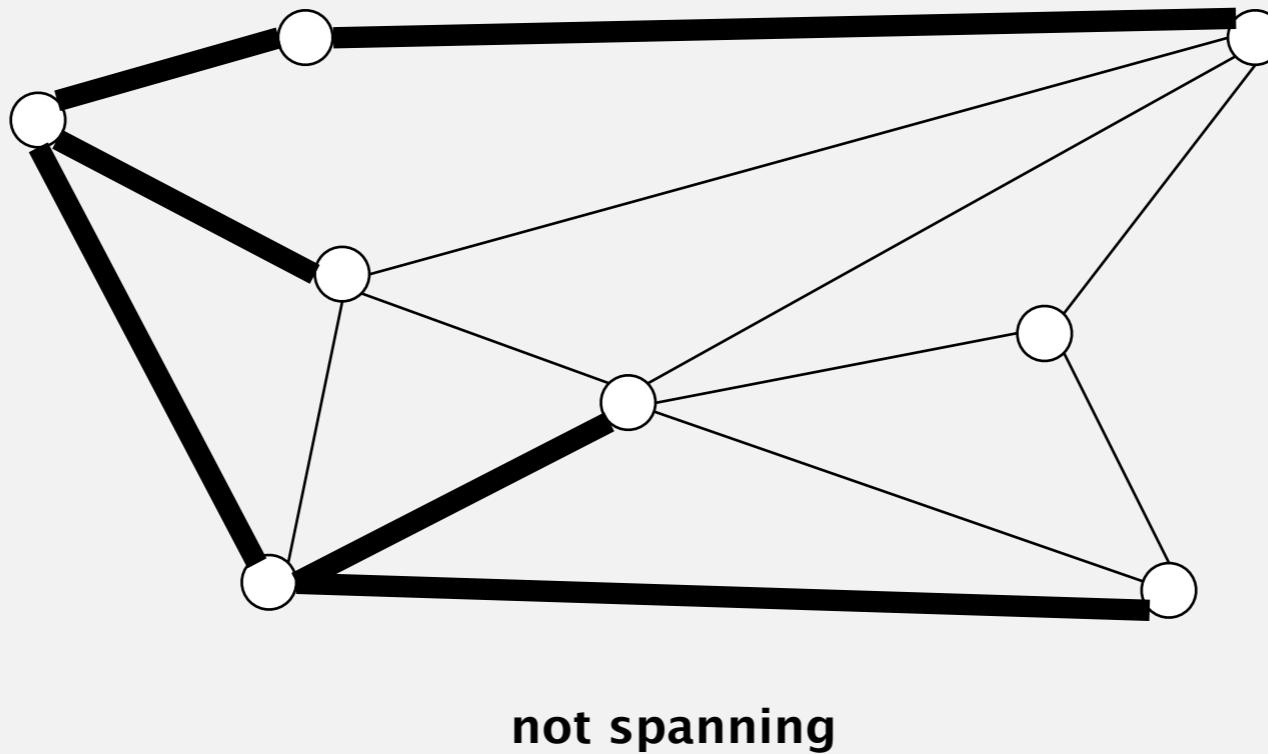


not acyclic

Minimum spanning tree

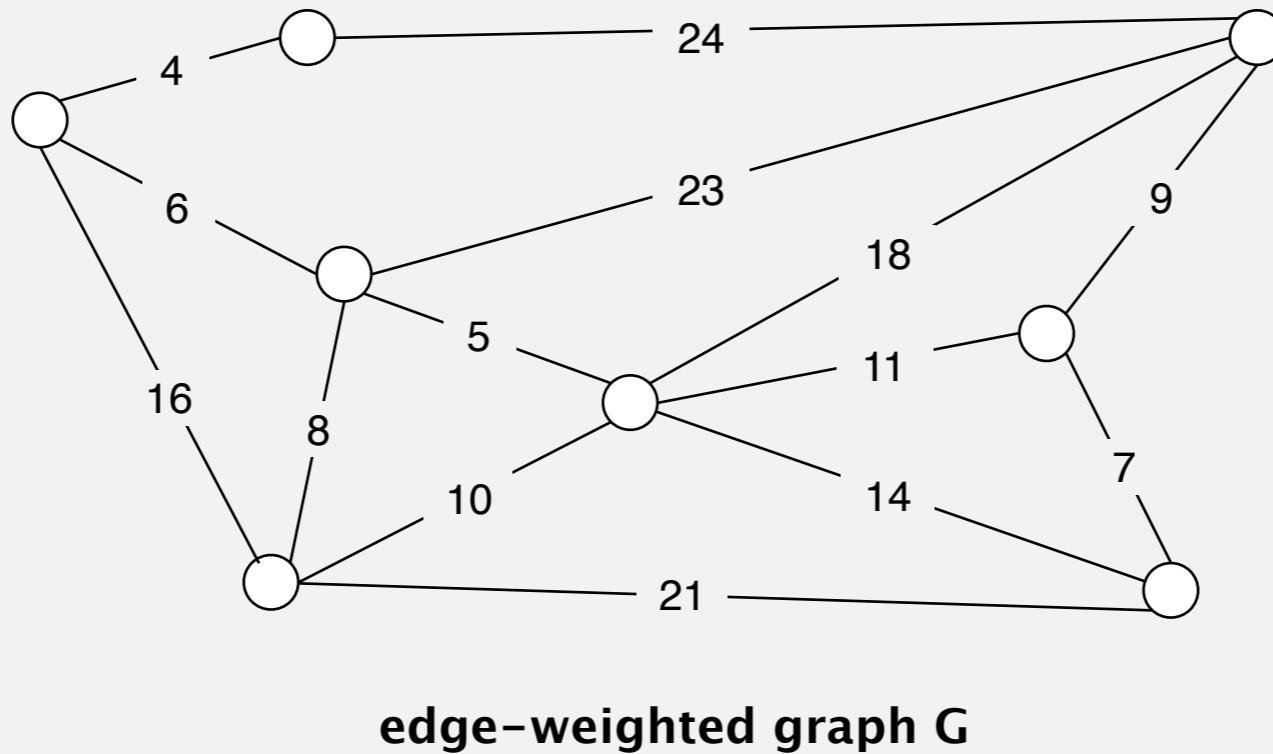
Def. A **spanning tree** of G is a subgraph T that is:

- Connected.
- Acyclic.
- Includes all of the vertices.



Minimum spanning tree problem

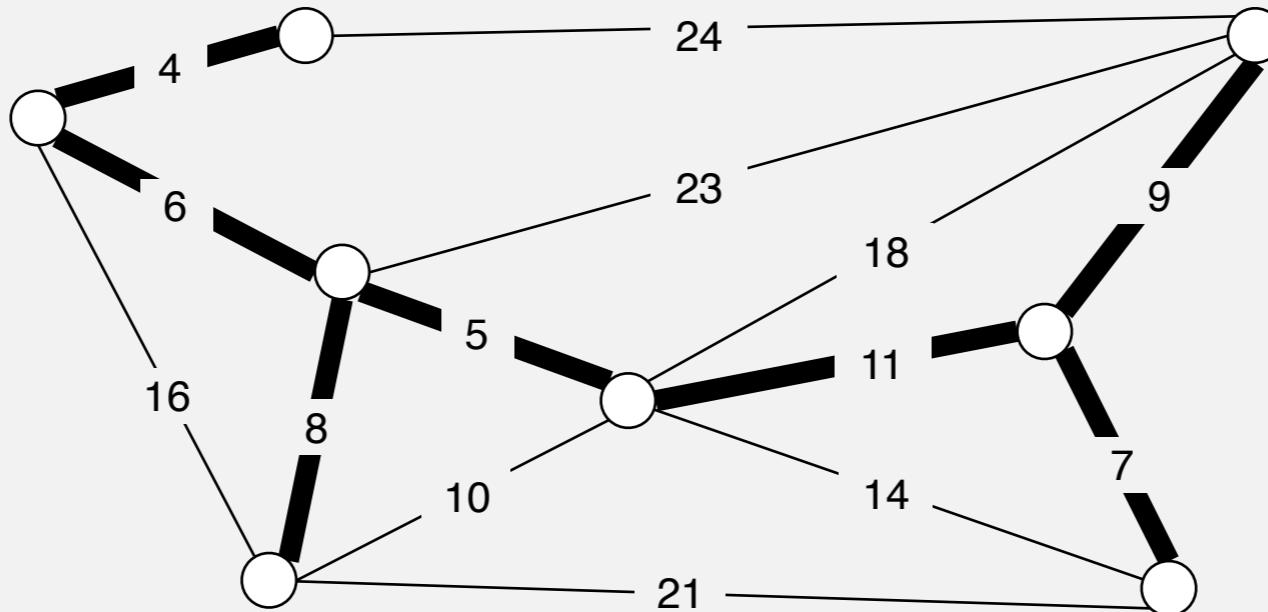
Input. Connected, undirected graph G with positive edge weights.



Minimum spanning tree problem

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

Output. A min weight spanning tree.

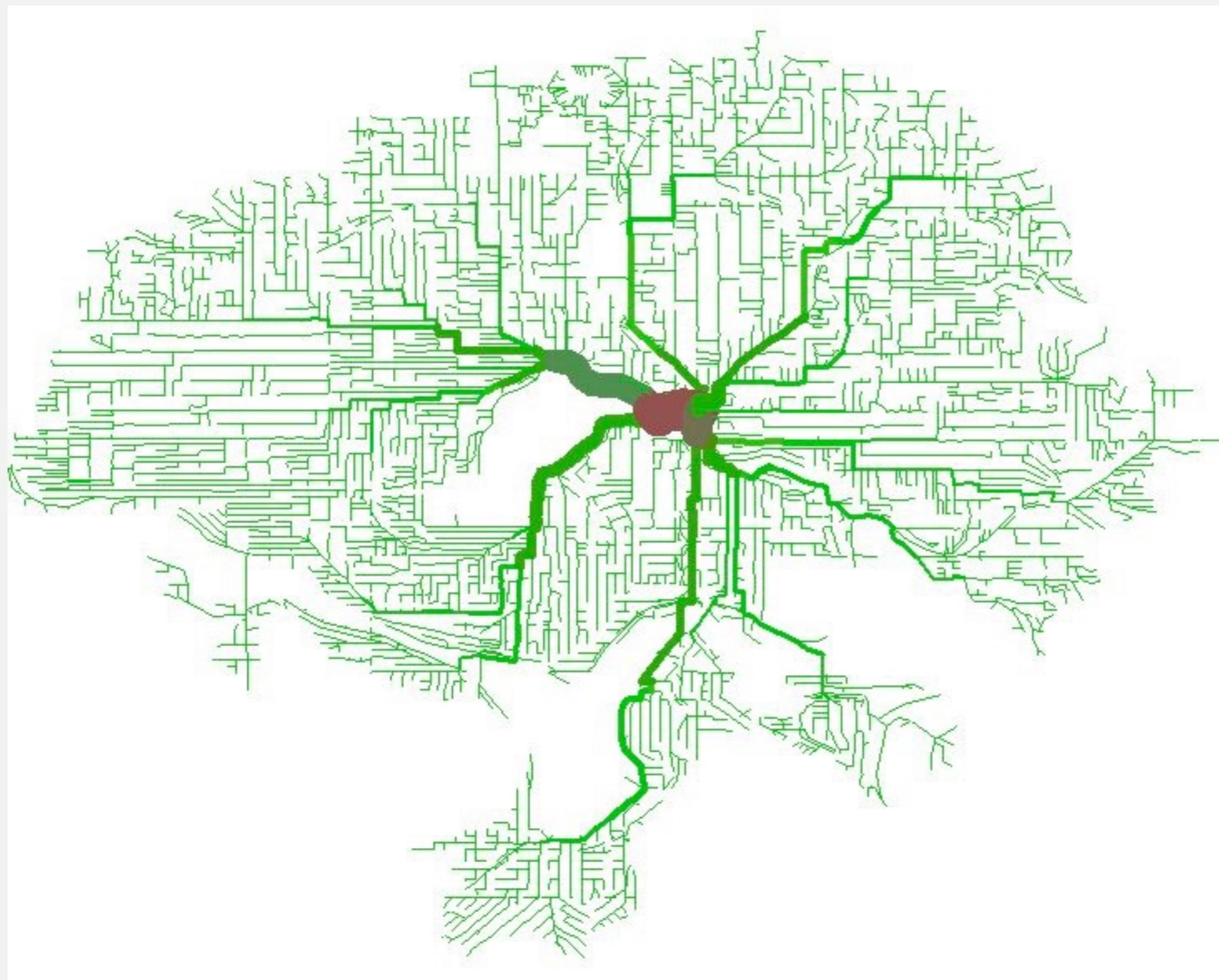


minimum spanning tree T
(weight = $50 = 4 + 6 + 8 + 5 + 11 + 9 + 7$)

Brute force. Try all spanning trees?

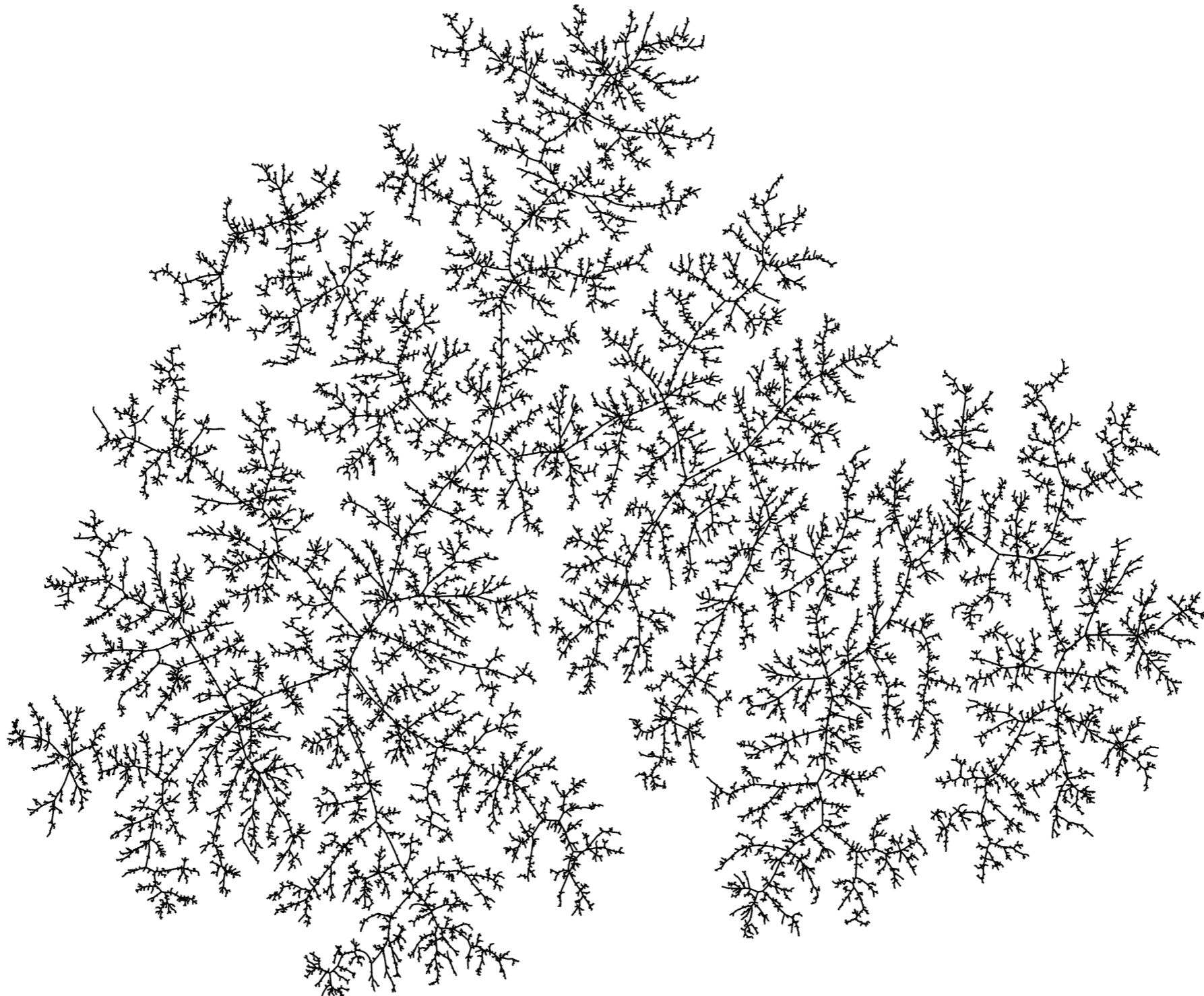
Network design

MST of bicycle routes in North Seattle



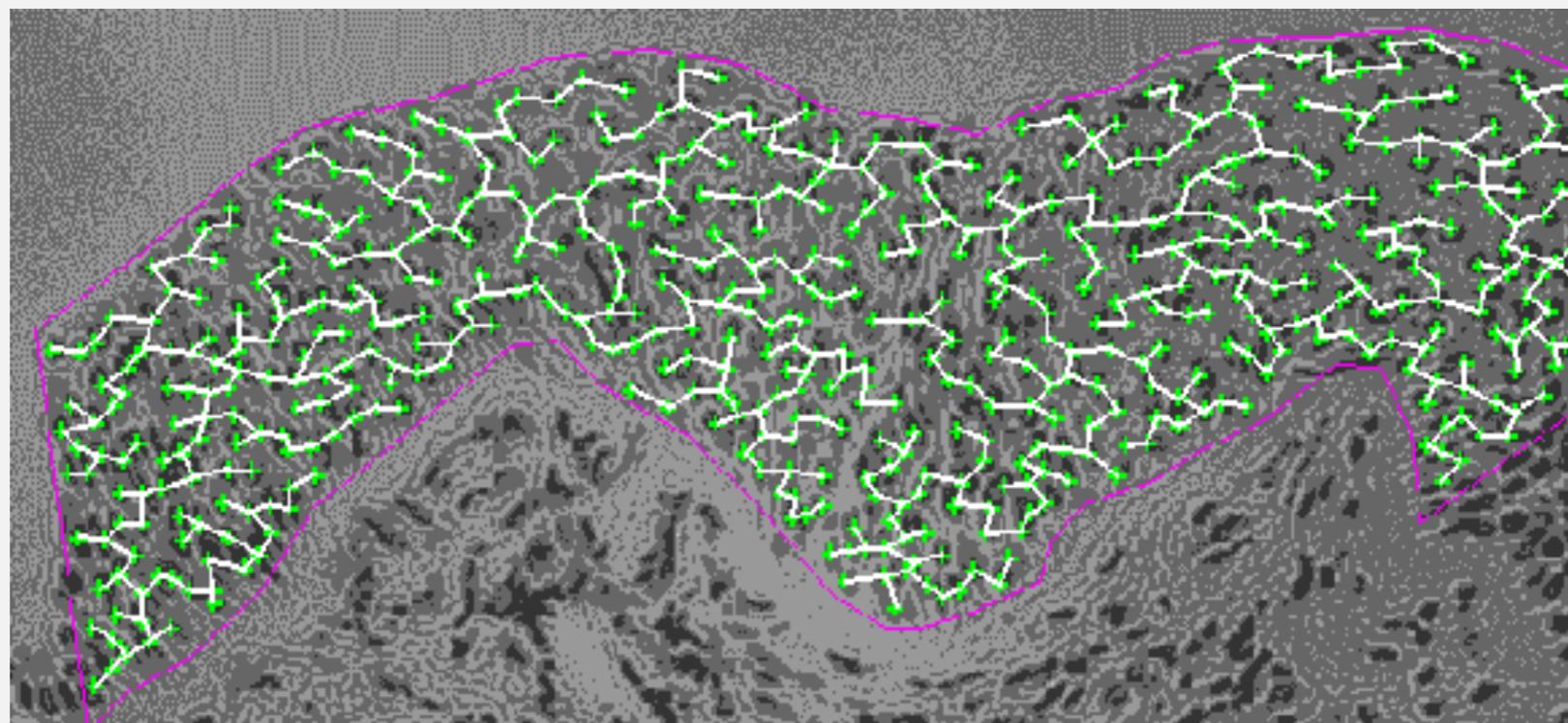
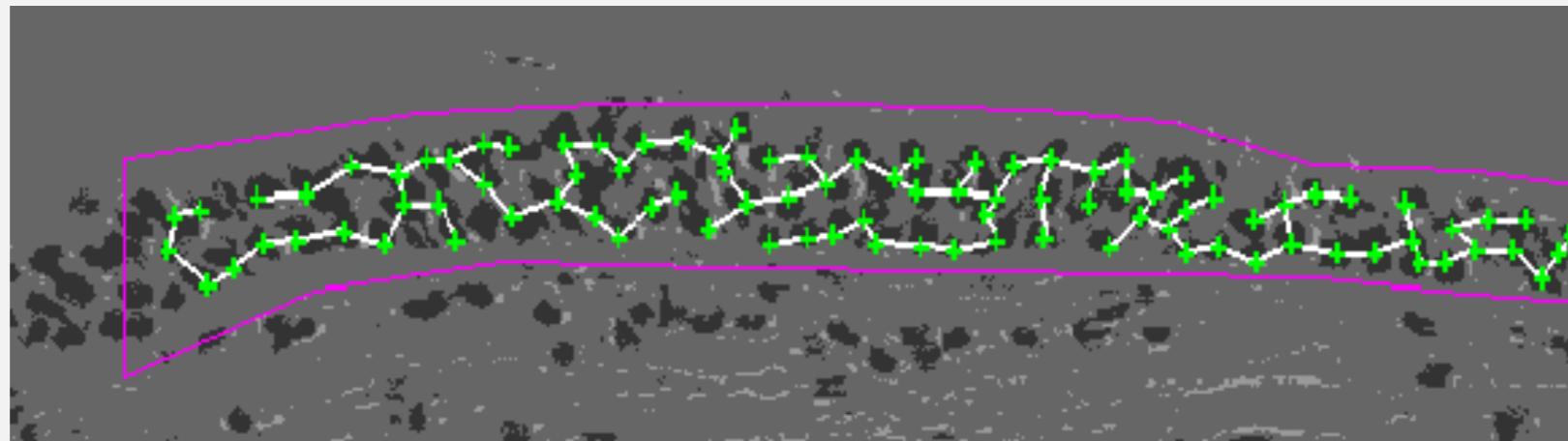
Models of nature

MST of random graph



Medical image processing

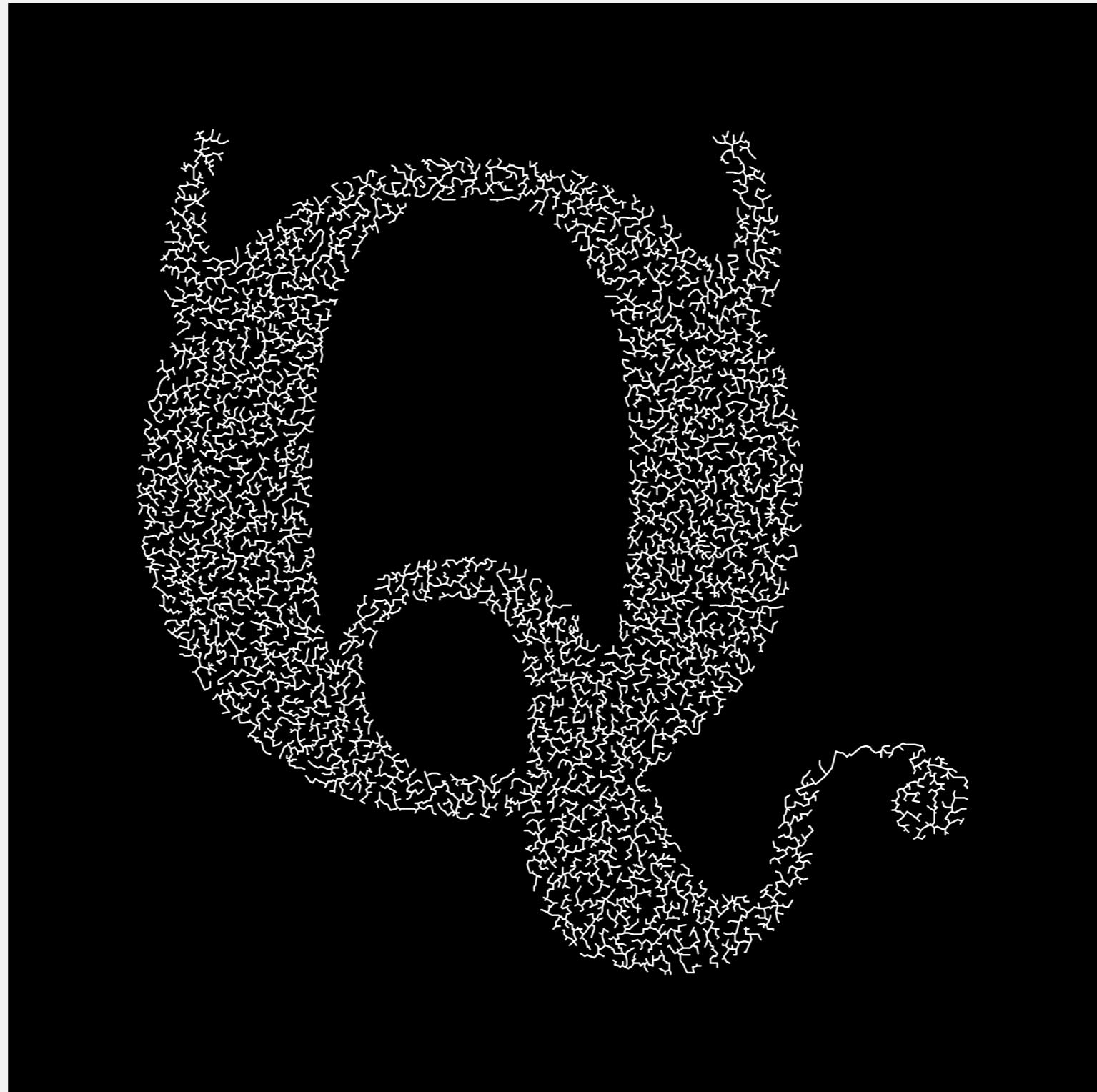
MST describes arrangement of nuclei in the epithelium for cancer research



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

Medical 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

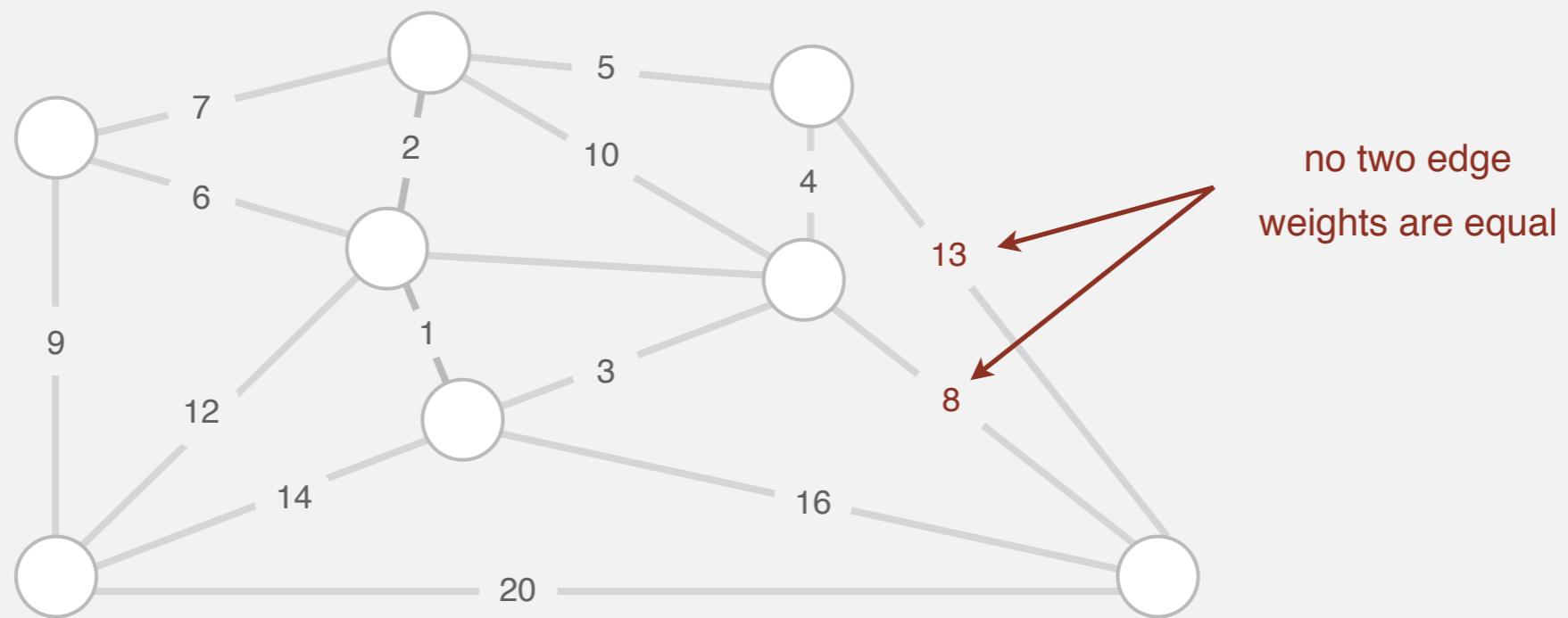
- ▶ introduction
- ▶ greedy algorithm
- ▶ edge-weighted graph API
- ▶ Kruskal's algorithm
- ▶ Prim's algorithm
- ▶ context

Simplifying assumptions

- Graph is connected.
- Edge weights are distinct.

Consequence. MST exists and is unique.

Pf. Exercise for the bored.

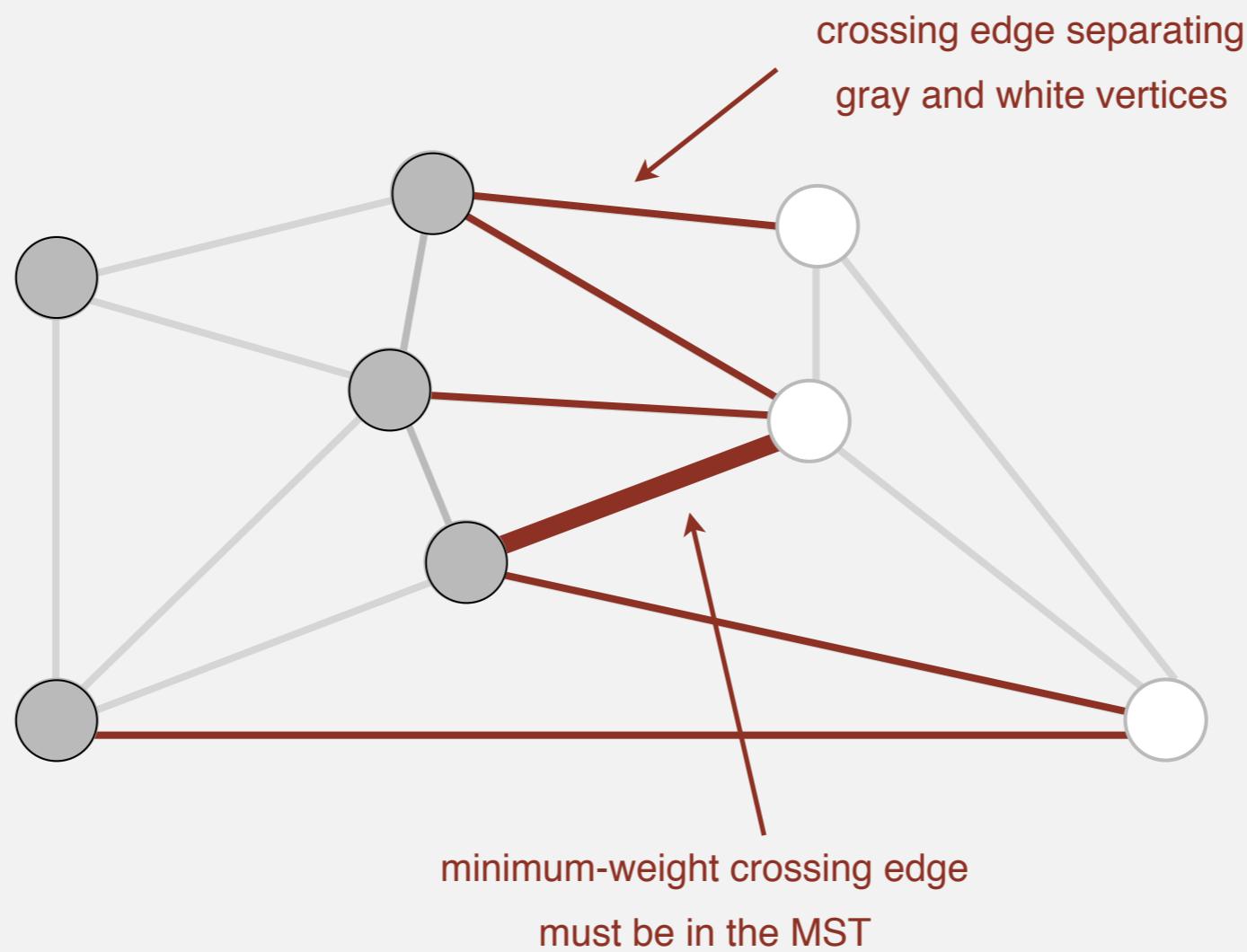


Cut property

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.



Cut property: correctness proof

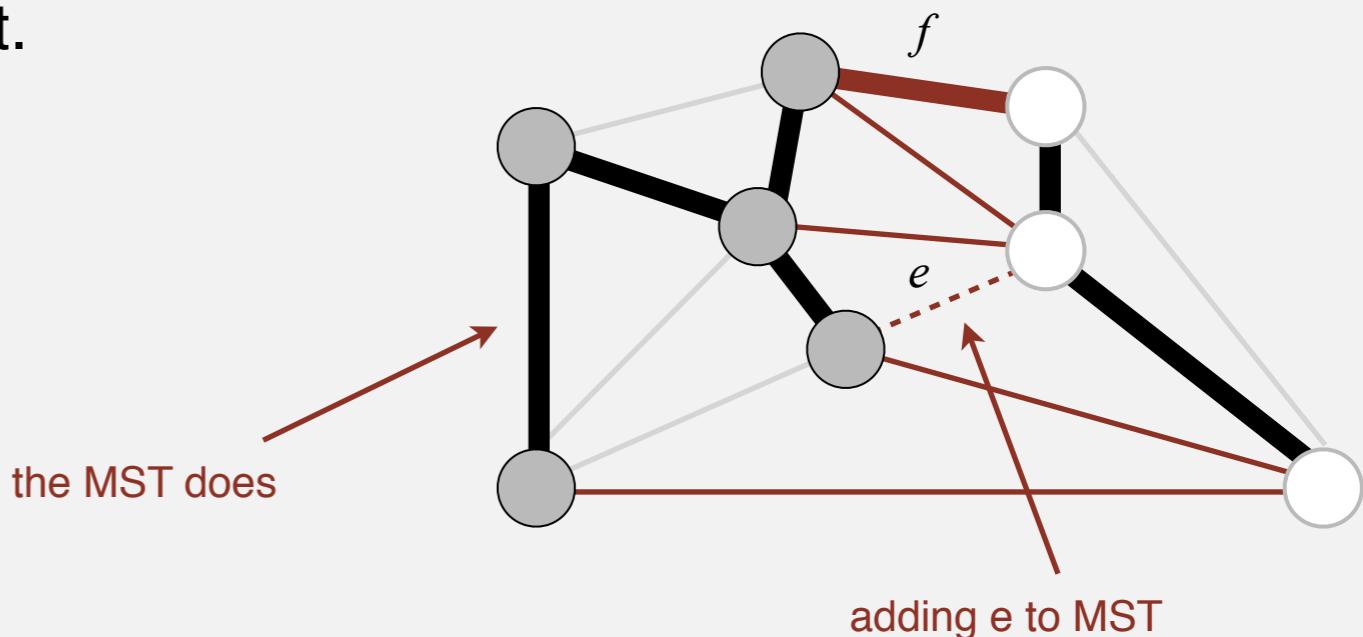
Def. A **cut** in a graph is a partition of its vertices into two (nonempty) sets.

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.

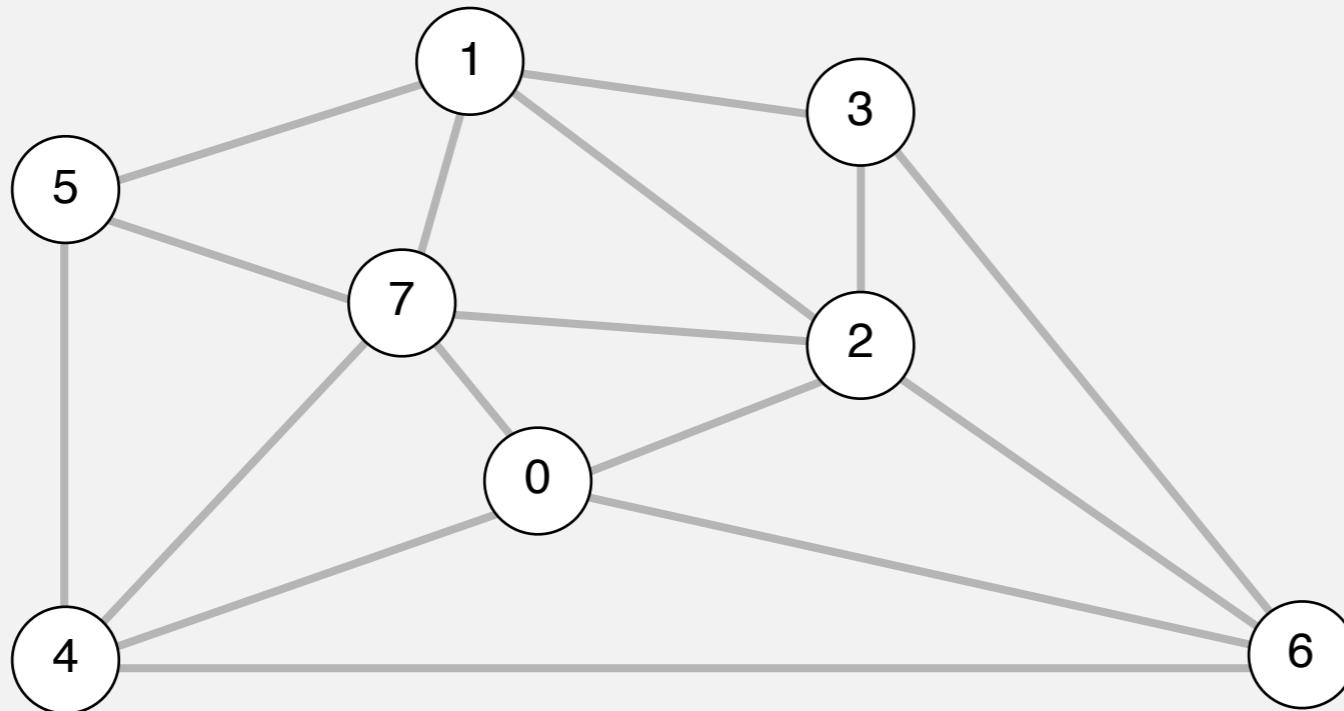
Pf. Suppose min-weight crossing edge e is not in the MST.

- 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 is lower weight.
- Contradiction. ▀



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.

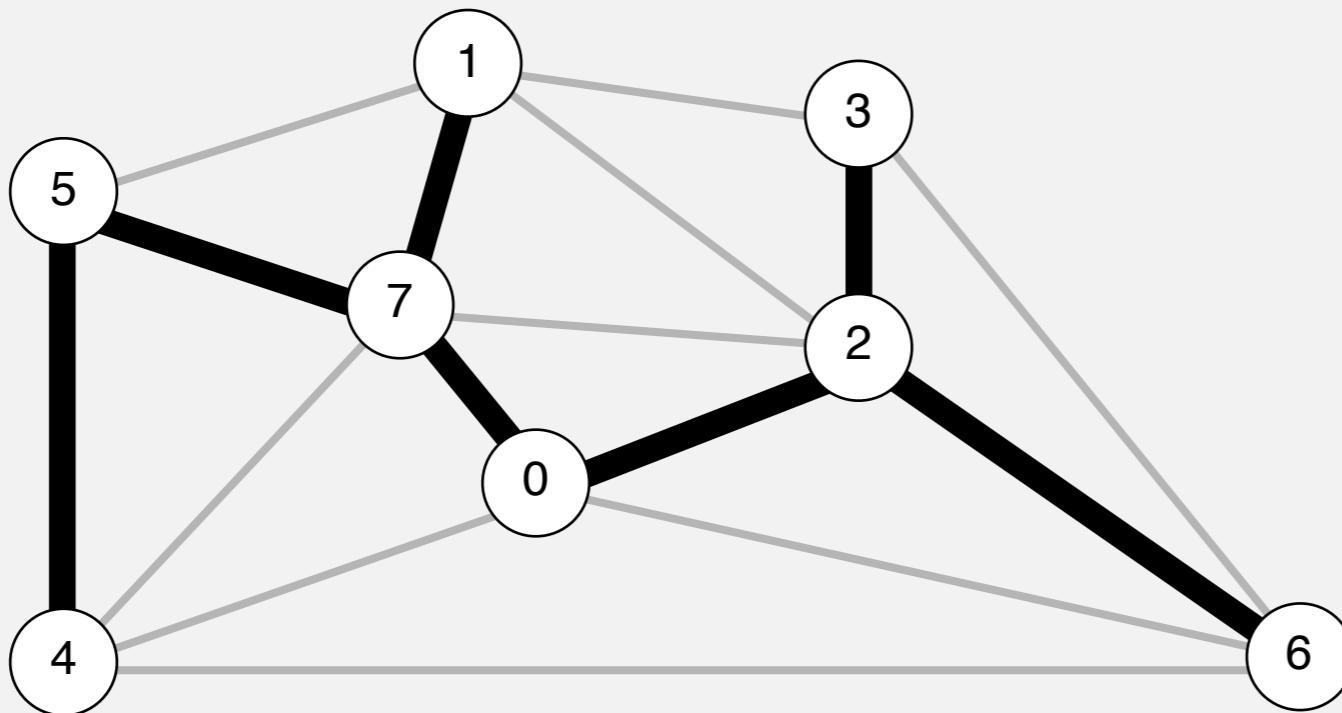


an edge-weighted graph

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

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.



MST edges

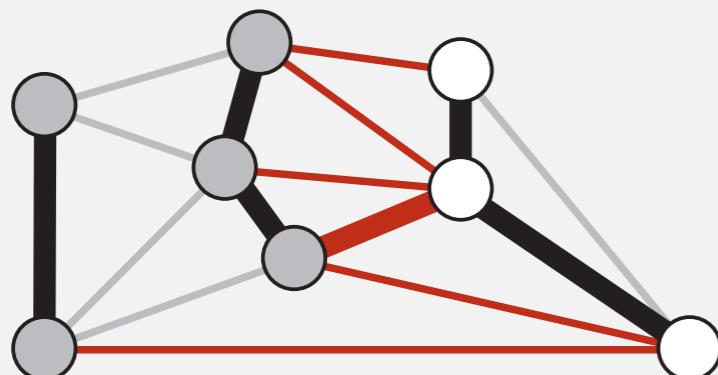
0-2 5-7 6-2 0-7 2-3 1-7 4-5

Greedy MST algorithm: correctness proof

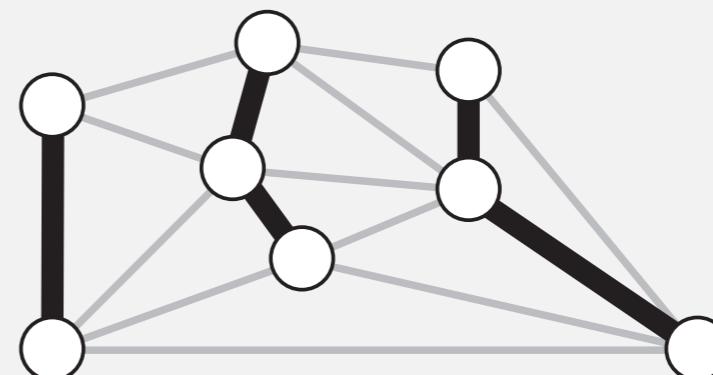
Proposition. The greedy algorithm computes the MST.

Pf.

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



a cut with no black crossing edges



fewer than $V-1$ edges colored black

Greedy MST algorithm: efficient implementations

Proposition. The greedy algorithm computes the MST.

Efficient implementations. Choose cut? Find min-weight edge?

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

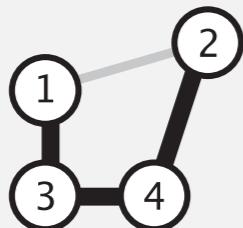
Ex 2. Prim's algorithm. [stay tuned]

Ex 3. Borüvka's algorithm.

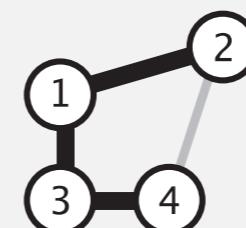
Removing two simplifying assumptions

Q. What if edge weights are not all distinct?

A. Greedy MST algorithm still correct if equal weights are present!
(our correctness proof fails, but that can be fixed)



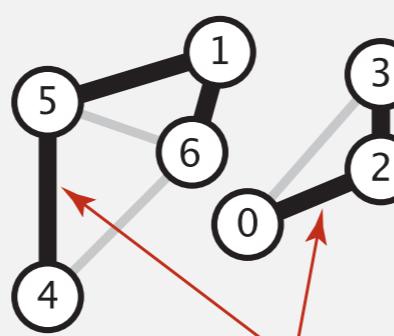
1	2	1.00
1	3	0.50
2	4	1.00
3	4	0.50



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 = MST of each 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

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

4.3 MINIMUM SPANNING

- ▶ 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)

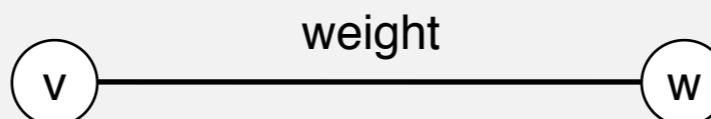
compare this edge to that edge

 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;
}
```

constructor

```
public int either()
{ return v; }
```

either endpoint

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

other endpoint

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

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

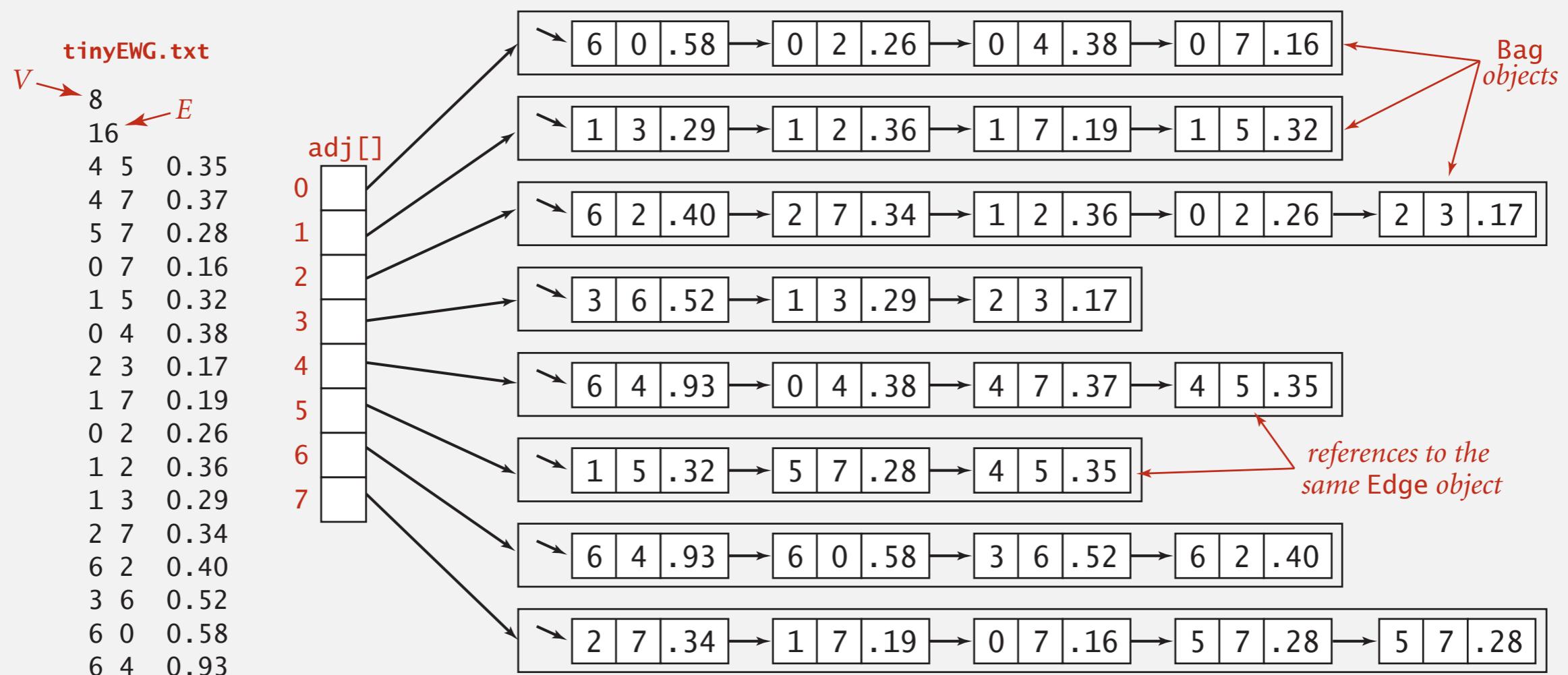
```
    String toString()
```

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;
```

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

```
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>();
```

```
}
```

constructor

```
public void addEdge(Edge e)
```

```
{
```

```
    int v = e.either(), w = e.other(v);
```

```
    adj[v].add(e);
```

```
    adj[w].add(e);
```

```
}
```

add edge to both
adjacency lists

```
public Iterable<Edge> adj(int v)
```

```
{ return adj[v]; }
```

```
}
```

Minimum spanning tree API

Q. How to represent the MST?

```
public class MST
```

```
MST(EdgeWeightedGraph G)
```

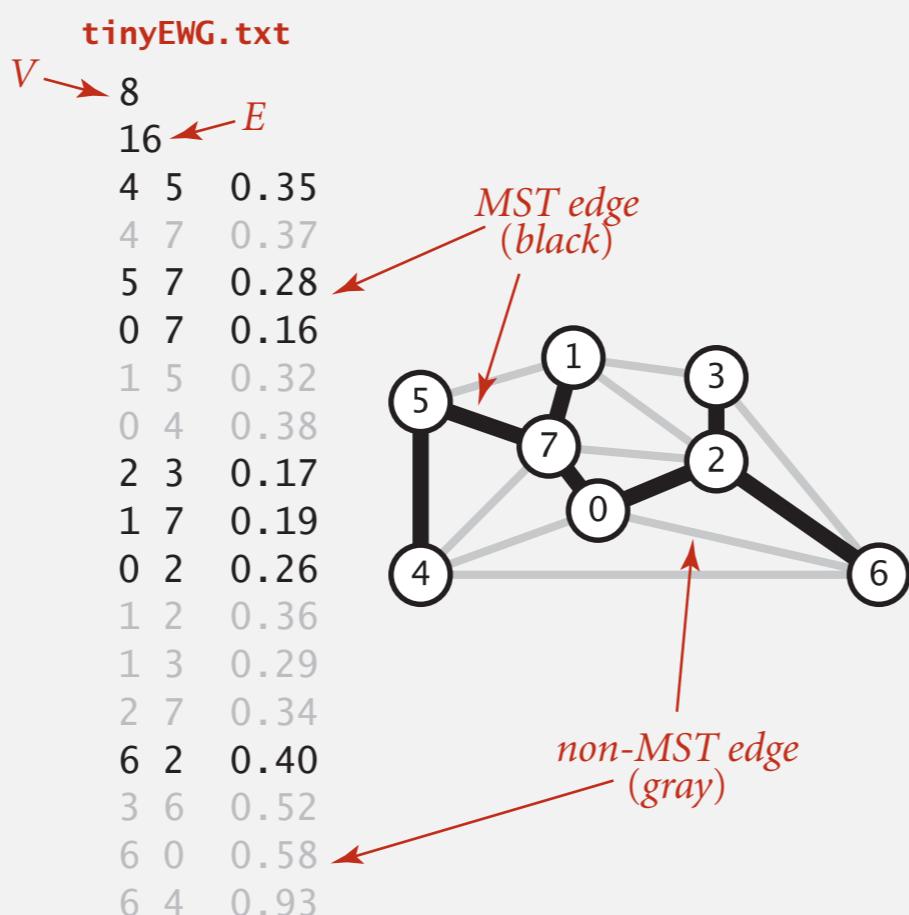
constructor

```
Iterable<Edge> edges()
```

edges in MST

```
double weight()
```

weight of MST



```
% java MST tinyEWG.txt
```

0-7 0.16

1-7 0.19

0-2 0.26

2-3 0.17

5-7 0.28

4-5 0.35

6-2 0.40

1.81

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

```
public static void main(String[] args)
{
    In in = new In(args[0]);
    EdgeWeightedGraph G = new EdgeWeightedGraph(in);
    MST mst = new MST(G);
    for (Edge e : mst.edges())
        StdOut.println(e);
    StdOut.printf("%.2f\n", mst.weight());
}
```

```
% java MST tinyEWG.txt
0-7 0.16
1-7 0.19
0-2 0.26
2-3 0.17
5-7 0.28
4-5 0.35
6-2 0.40
1.81
```

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

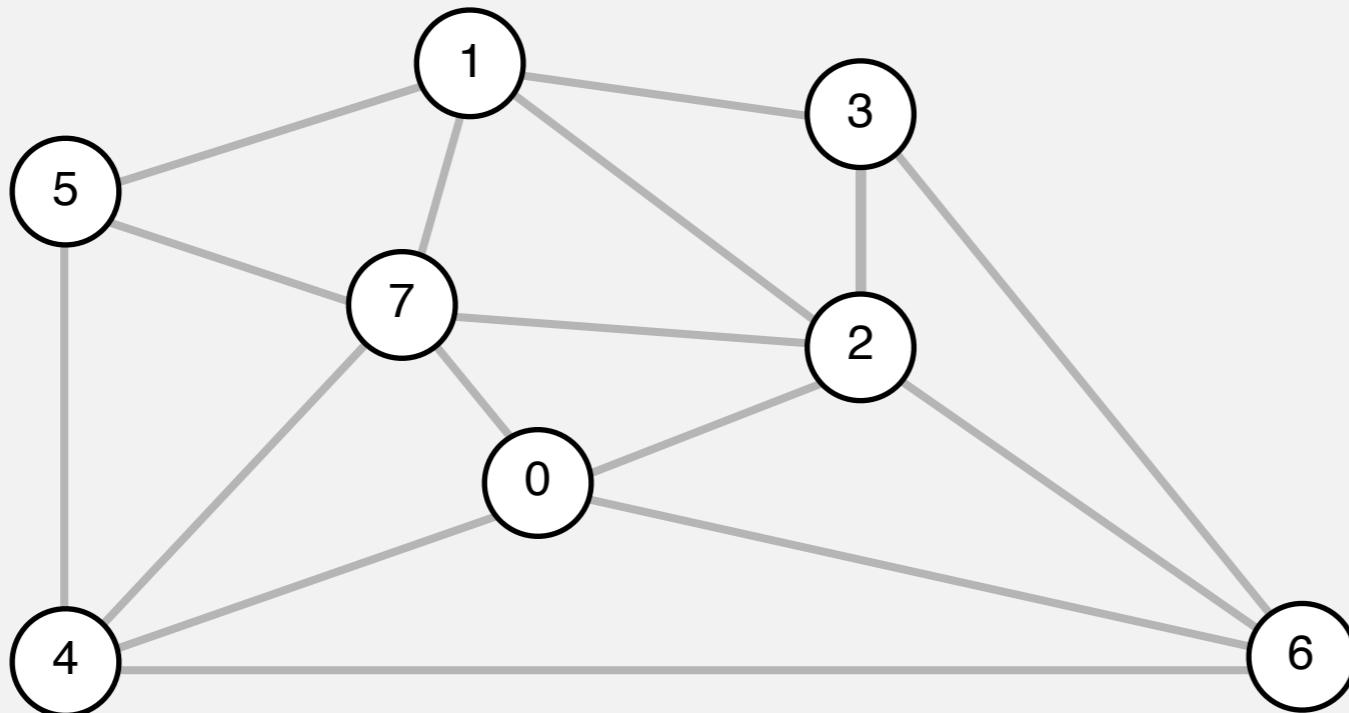
4.3 MINIMUM SPANNING

- ▶ 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.



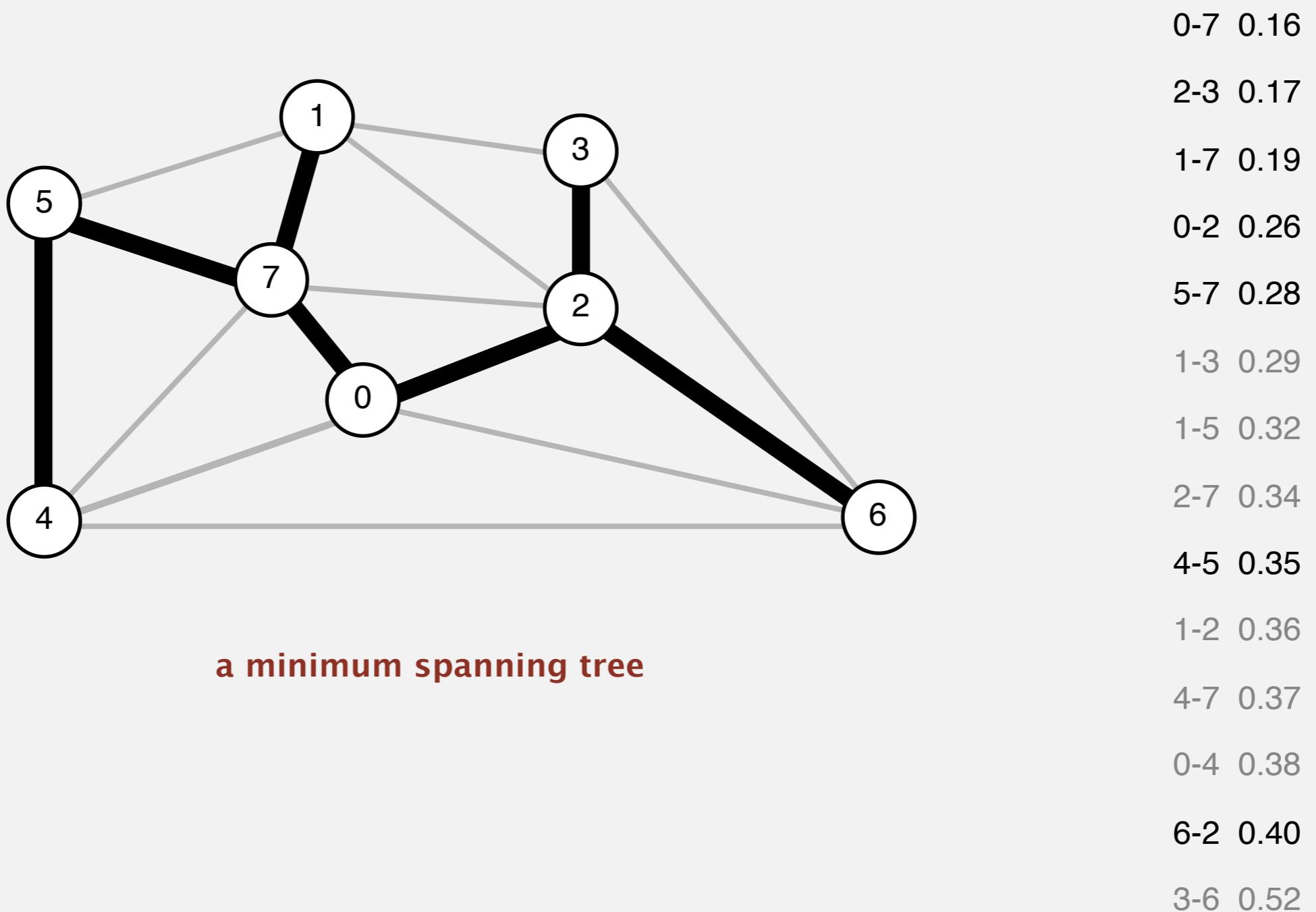
an edge-weighted graph

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	

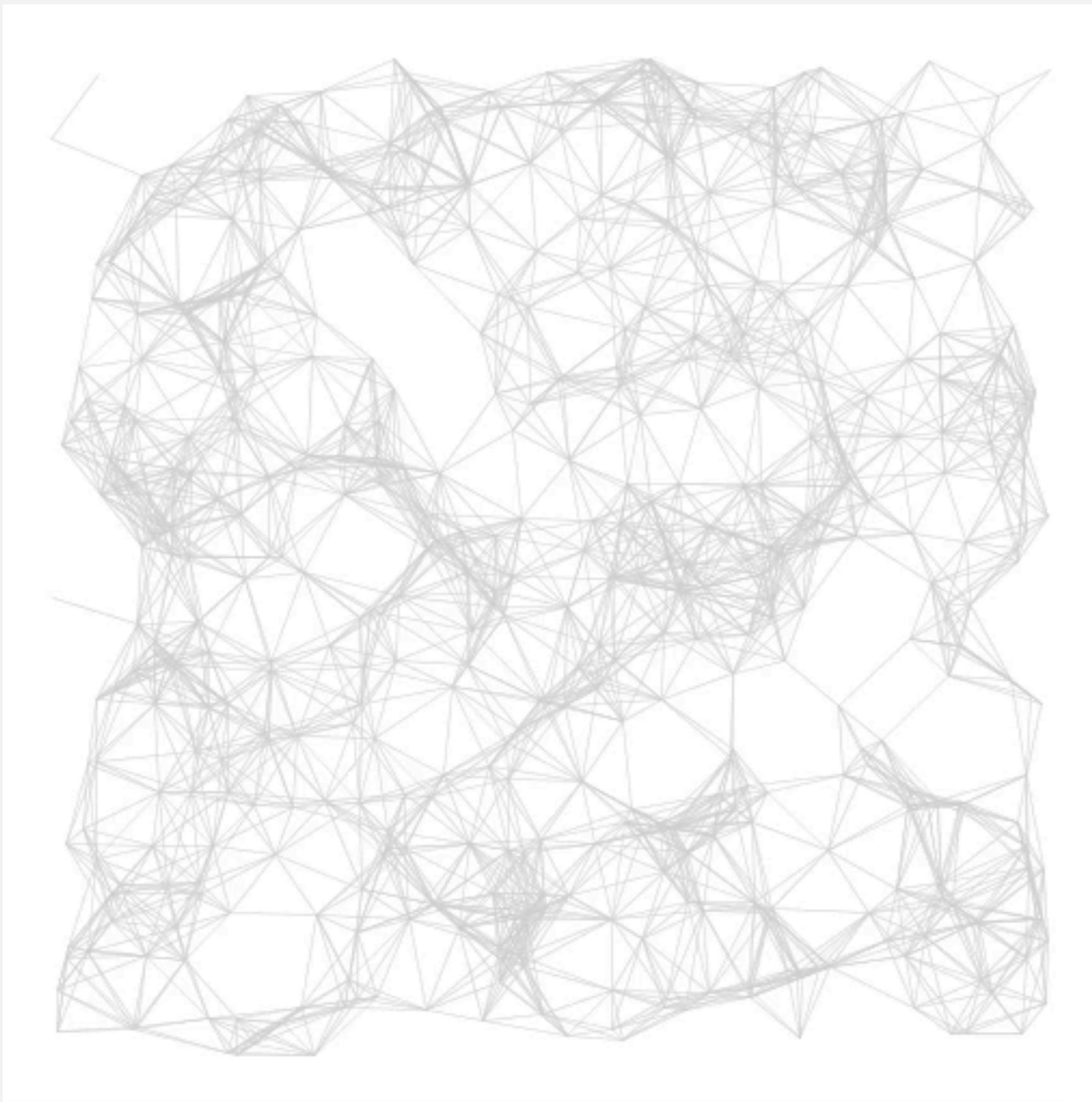
Kruskal's algorithm demo

Consider edges in ascending order of weight.

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



Kruskal's algorithm: visualization

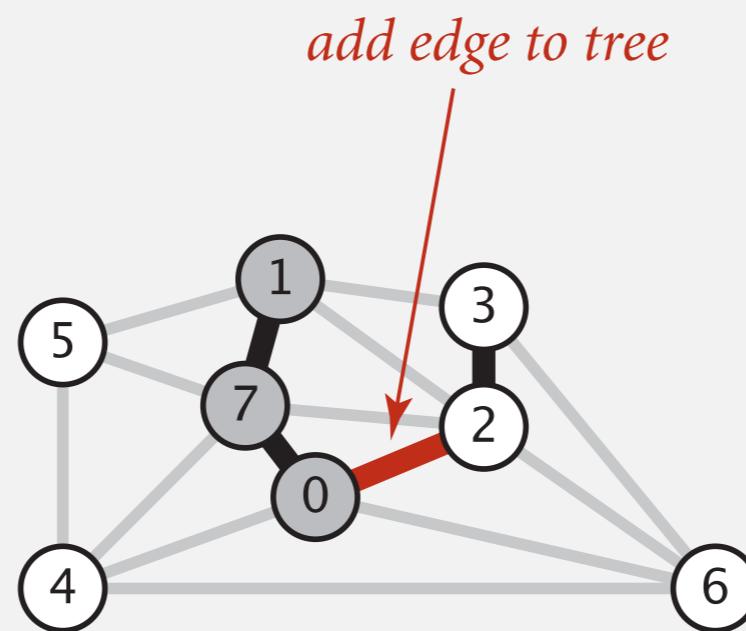


Kruskal's algorithm: correctness proof

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.
- Therefore the cut is between:
 - set of vertices connected to v in tree T ;
 - set of vertices connected to w in tree T .
- No crossing edge is black (because we check for cycles).
- No crossing edge has lower weight (because we use cheapest unused edge).

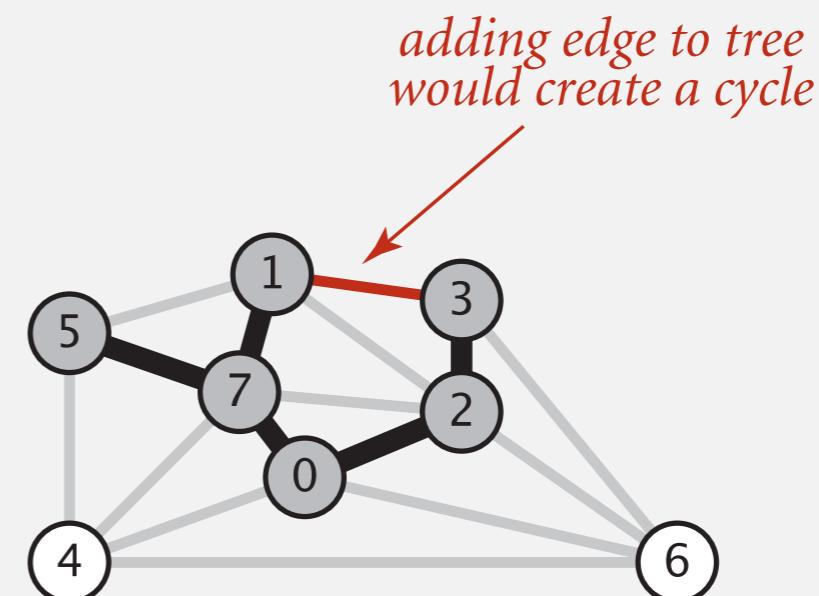
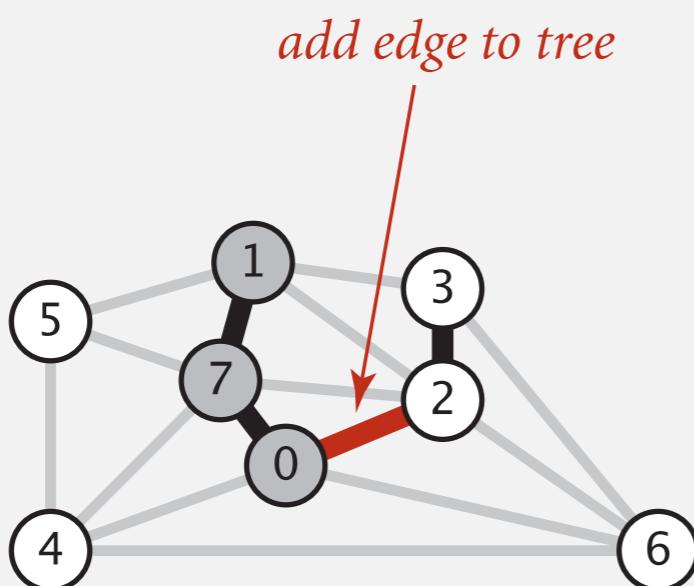


Kruskal's algorithm: implementation challenge

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

How difficult?

- $E + V$
- $V \leftarrow$ run DFS from v , check if w is reachable
(T has at most $V - 1$ edges)
- $\log V$
- $\log^* V \leftarrow$ use the union-find data structure !
- 1

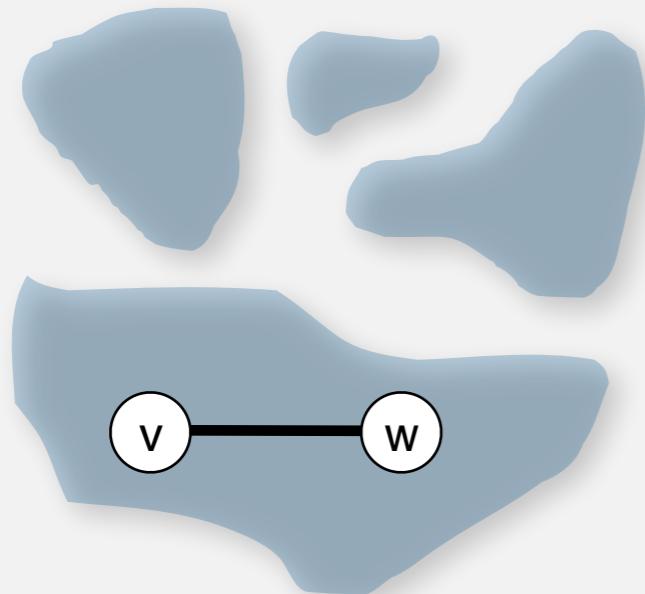


Kruskal's algorithm: implementation challenge

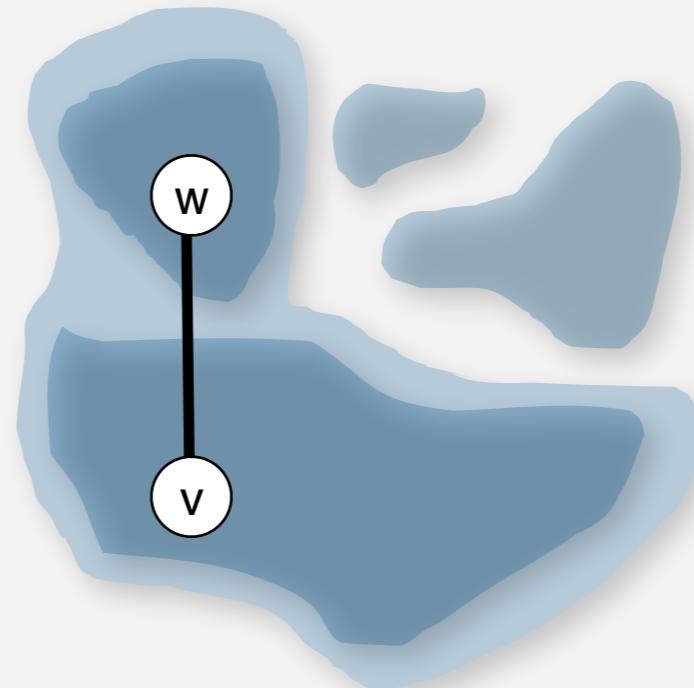
Challenge. Would adding edge $v-w$ to tree T create a cycle? If not, add it.

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 sets

add edge 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	$V-1 \dots E$	$\log E$
	union	$V-1$	$\log^* V^\dagger$
	connected	$V-1 \dots E$	$\log^* V^\dagger$

\dagger amortized bound using weighted quick union with path compression

recall: $\log^* V \leq 5$ in this universe



Remark. If edges are already sorted, order of growth is $E \log^* V$.

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

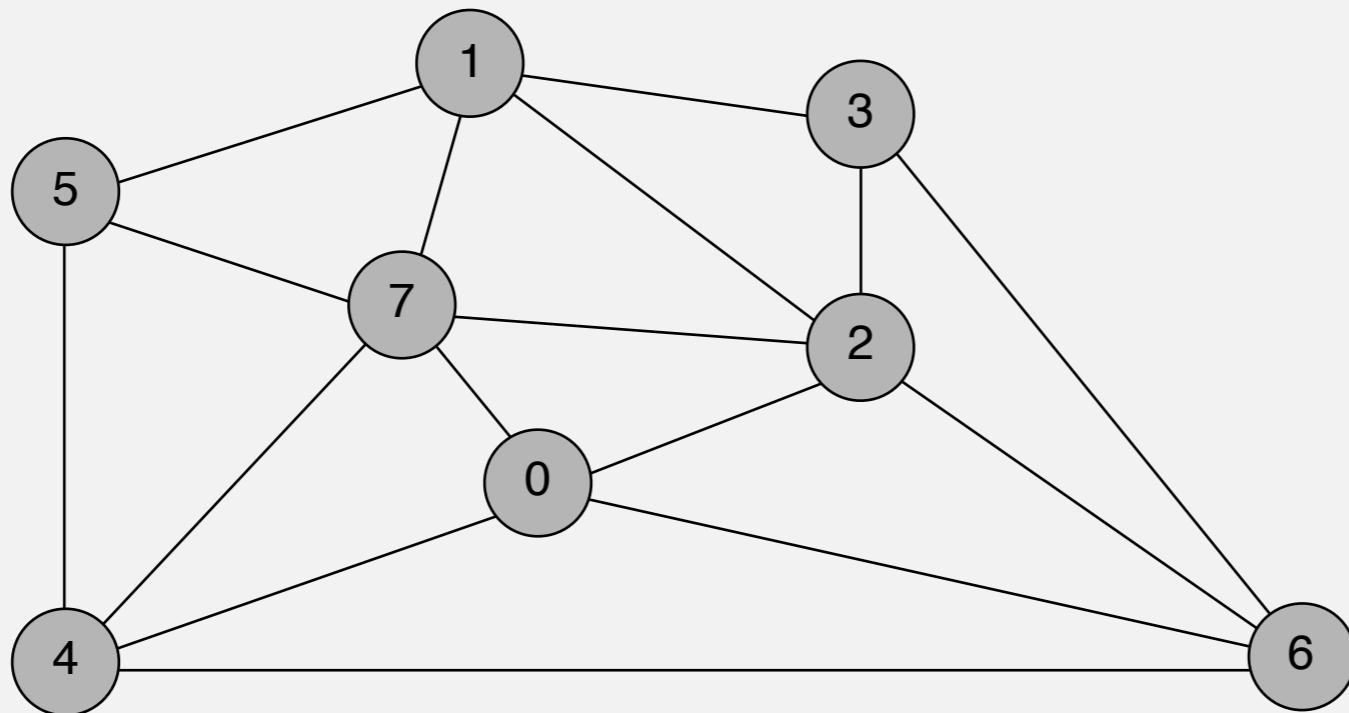
<http://algs4.cs.princeton.edu>

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.

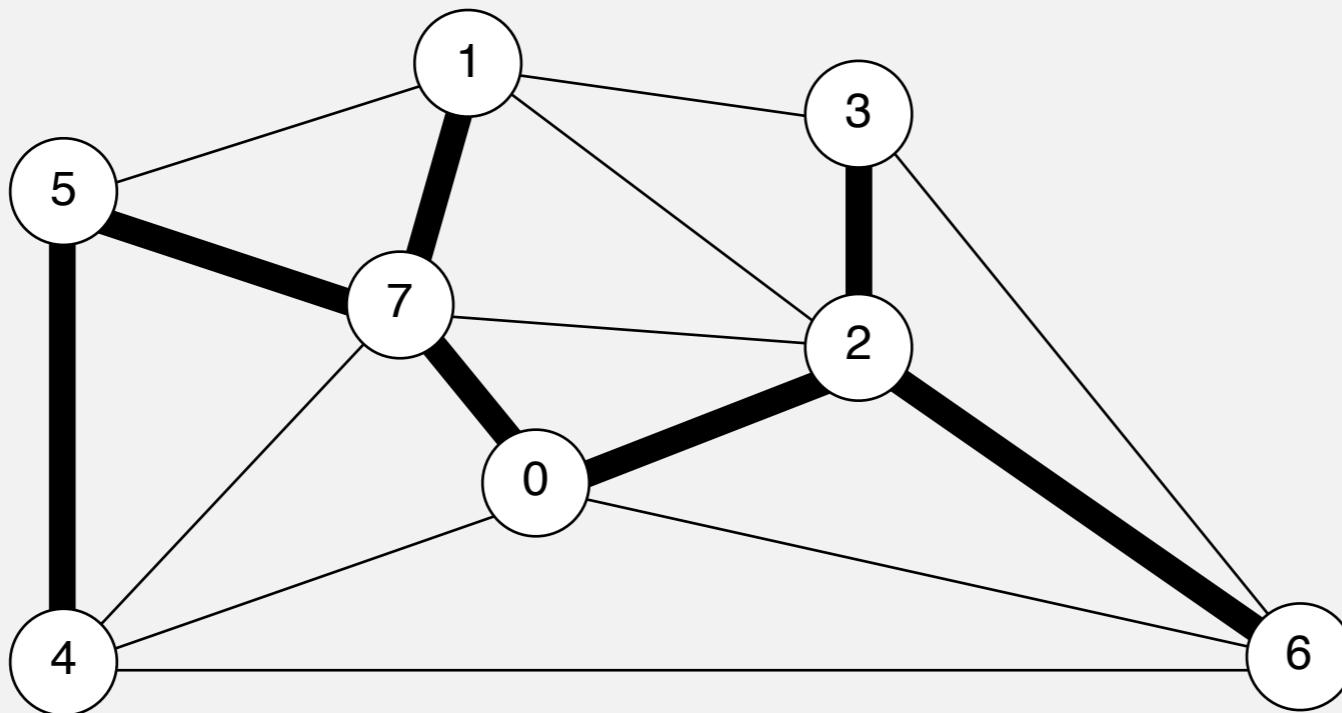


an edge-weighted graph

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
6-1	0.92

Prim's algorithm demo

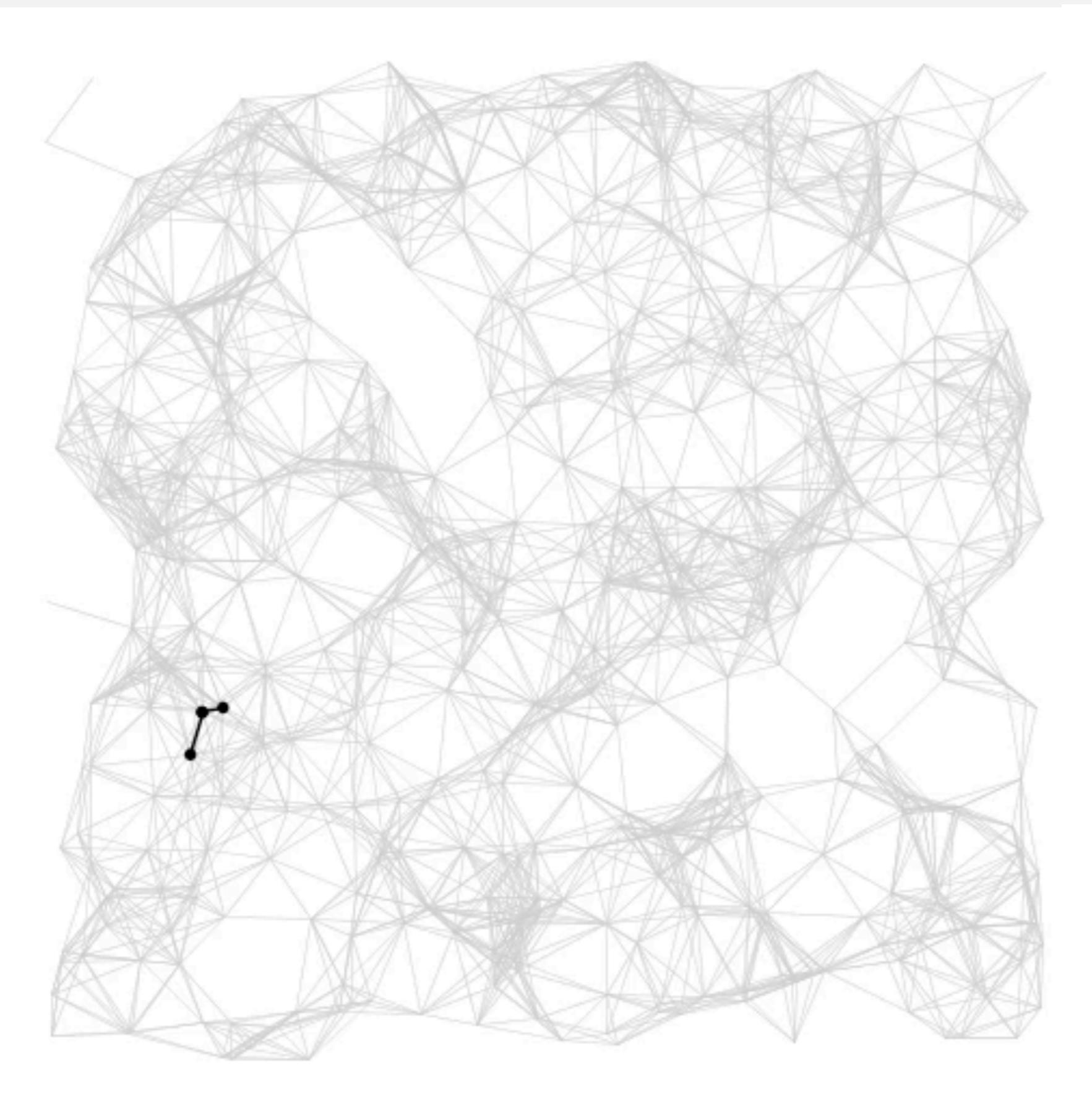
- Start with vertex 0 and greedily grow tree T .
- Add to T the min weight edge with exactly one endpoint in T .
- Repeat until $V - 1$ edges.



MST edges

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

Prim's algorithm: visualization



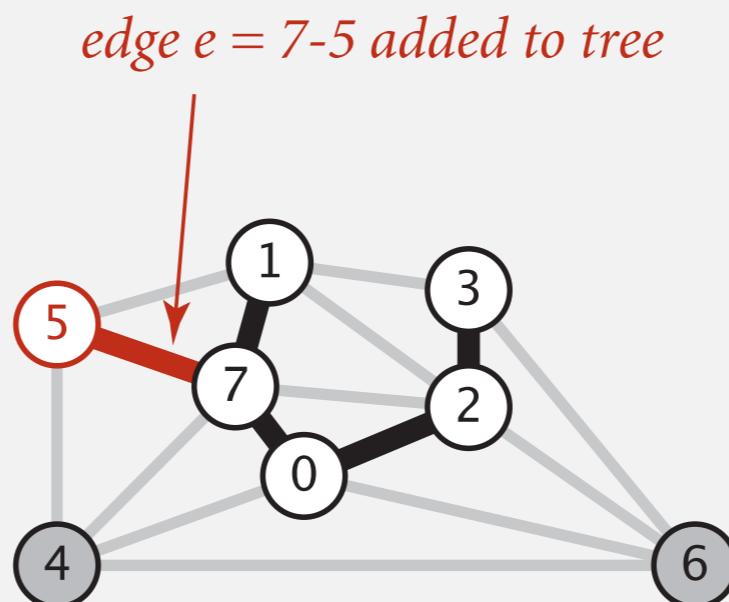
Prim's algorithm: proof of correctness

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.



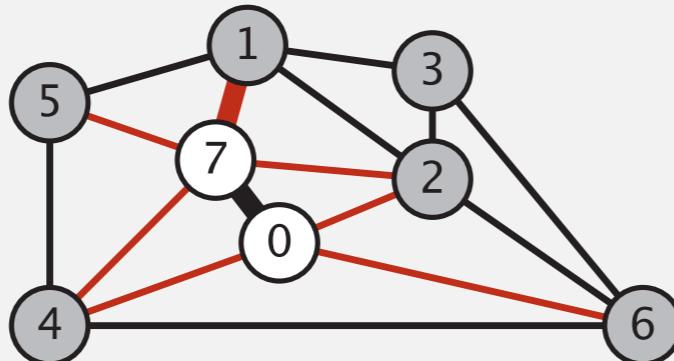
Prim's algorithm: implementation challenge

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

How difficult?

- E ← try all edges
- V
- $\log E$ ← use a priority queue!
- $\log^* E$
- 1

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



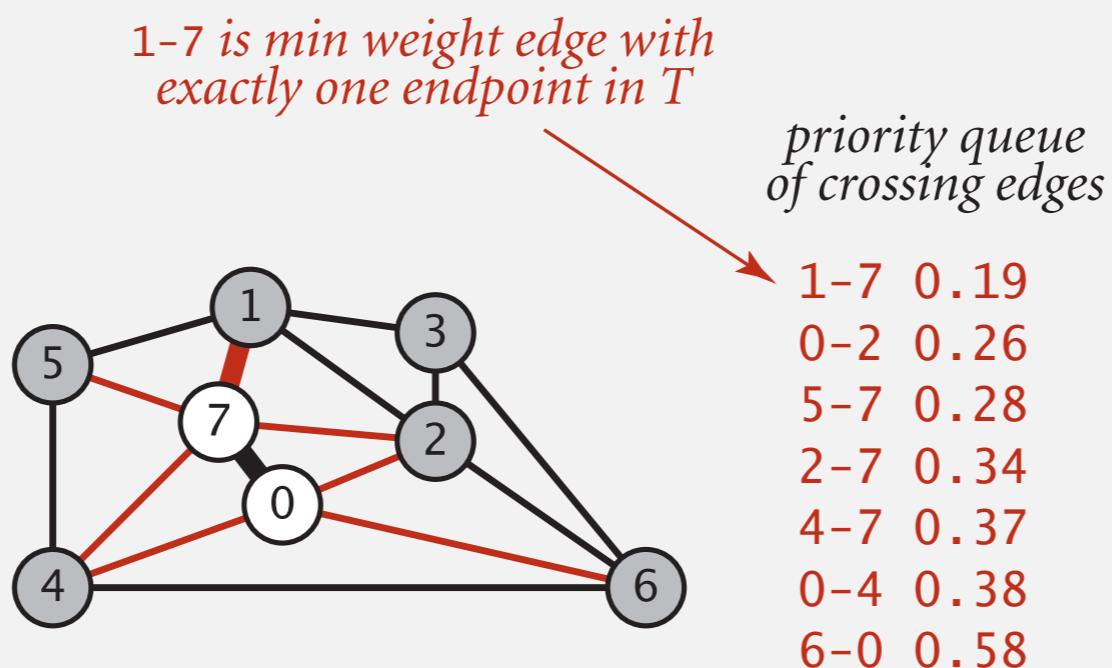
1-7	0.19
0-2	0.26
5-7	0.28
2-7	0.34
4-7	0.37
0-4	0.38
6-0	0.58

Prim's algorithm: lazy implementation

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

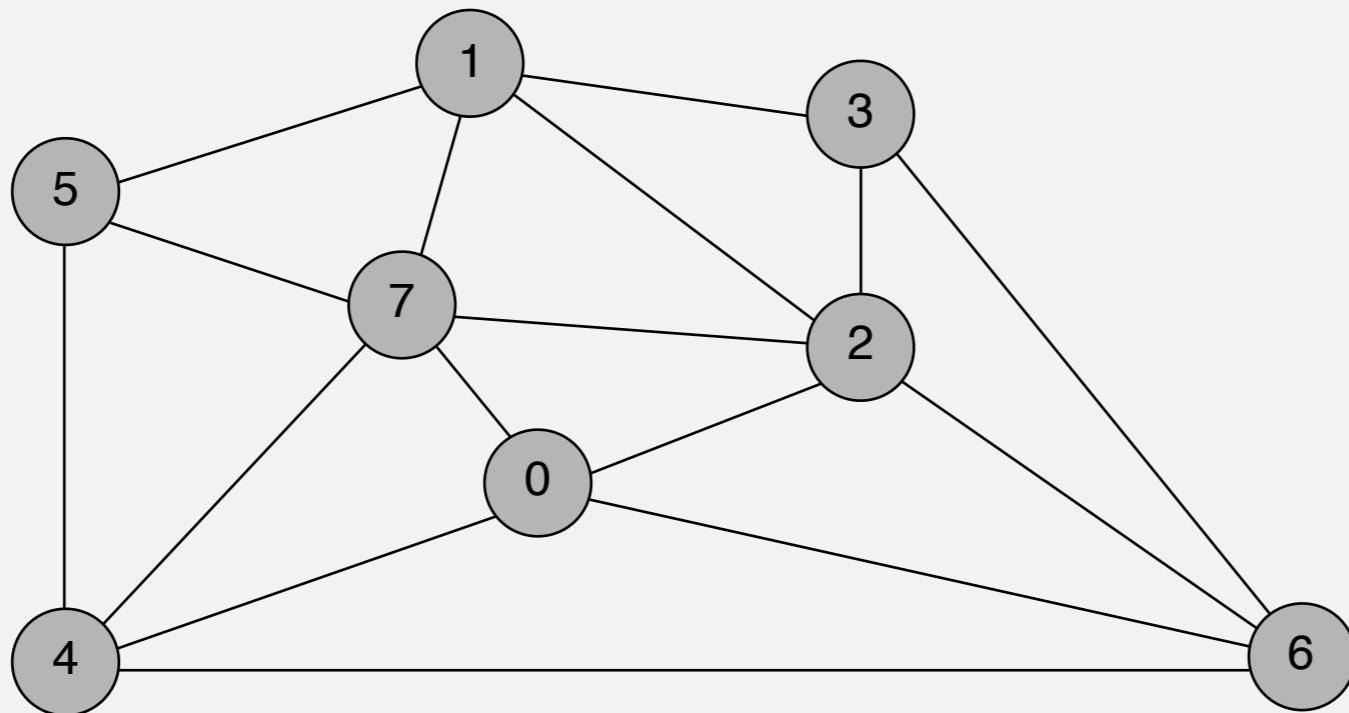
Lazy solution. Maintain a PQ of edges with (at least) one endpoint in T .

- 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 to PQ any edge incident to w (assuming other endpoint not in T)
 - add e to T and mark w



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.

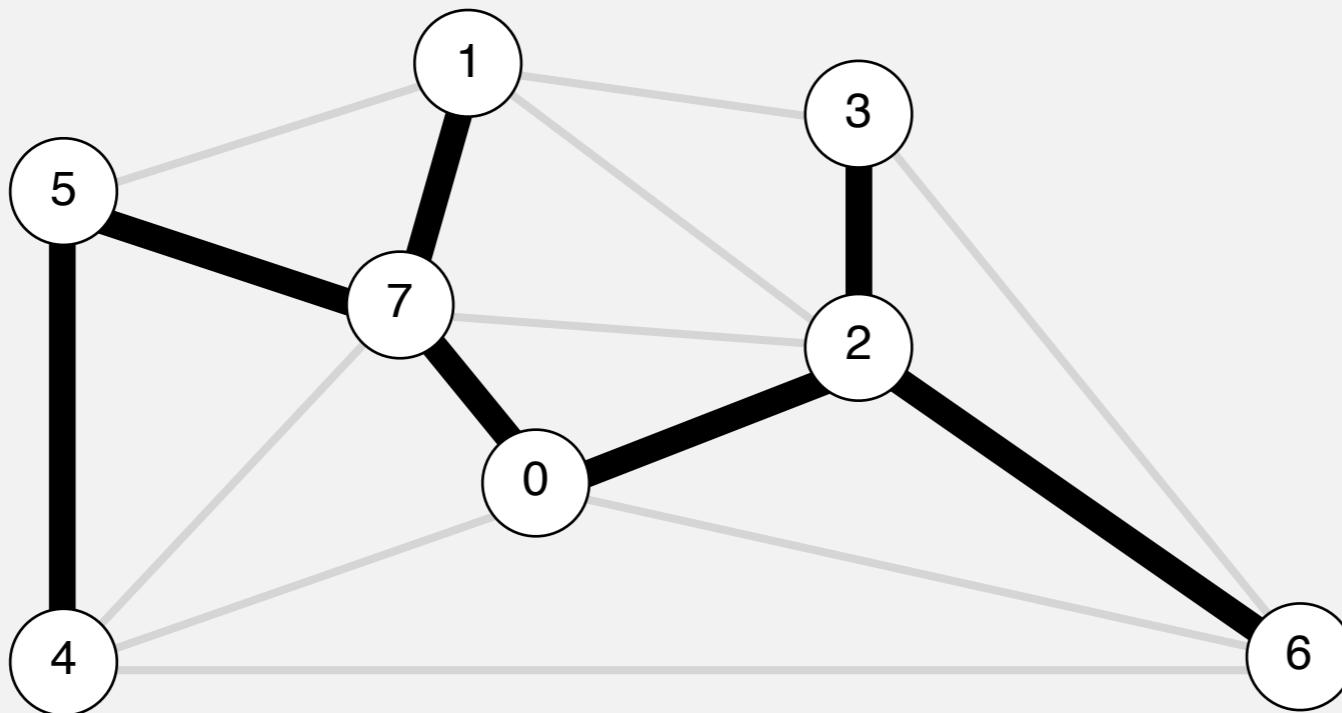


an edge-weighted graph

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

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.



MST edges

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

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);
        }
    }

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

- ← 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 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).

Pf.

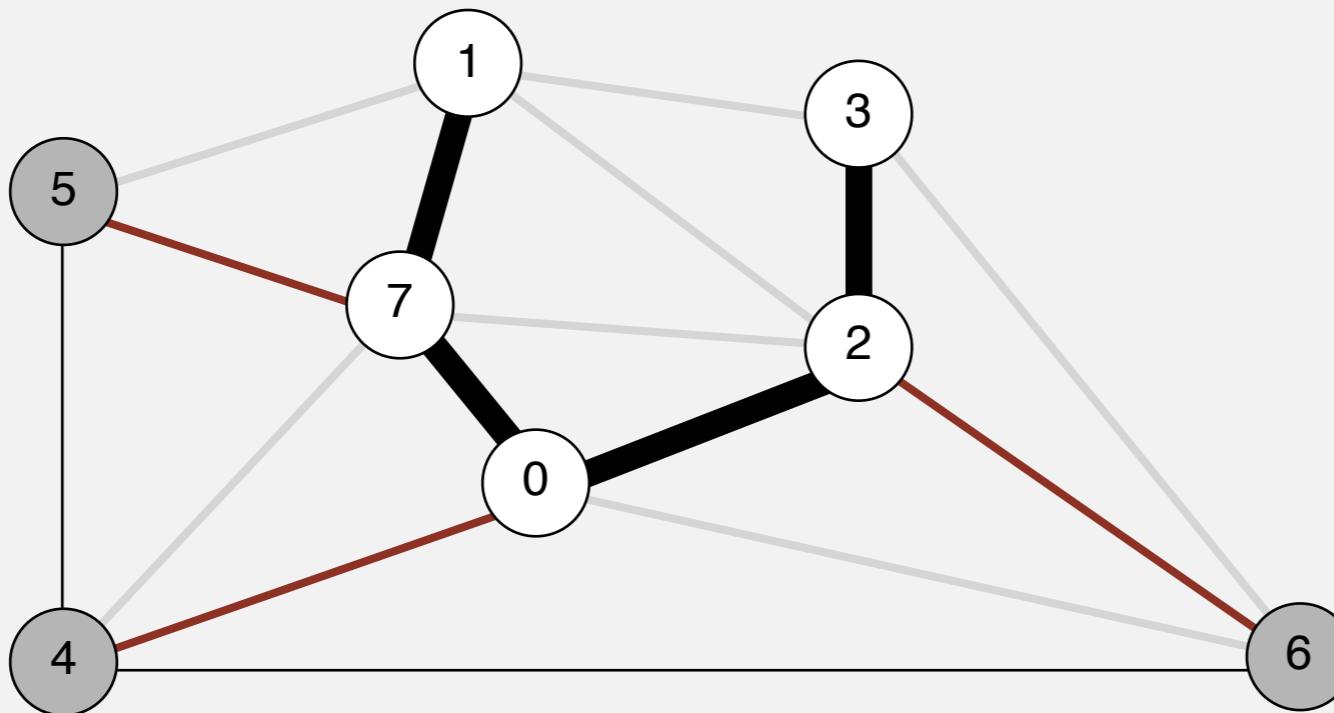
operation	frequency	binary heap
delete min	E	$\log E$
insert	E	$\log E$

Prim's algorithm: eager implementation

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

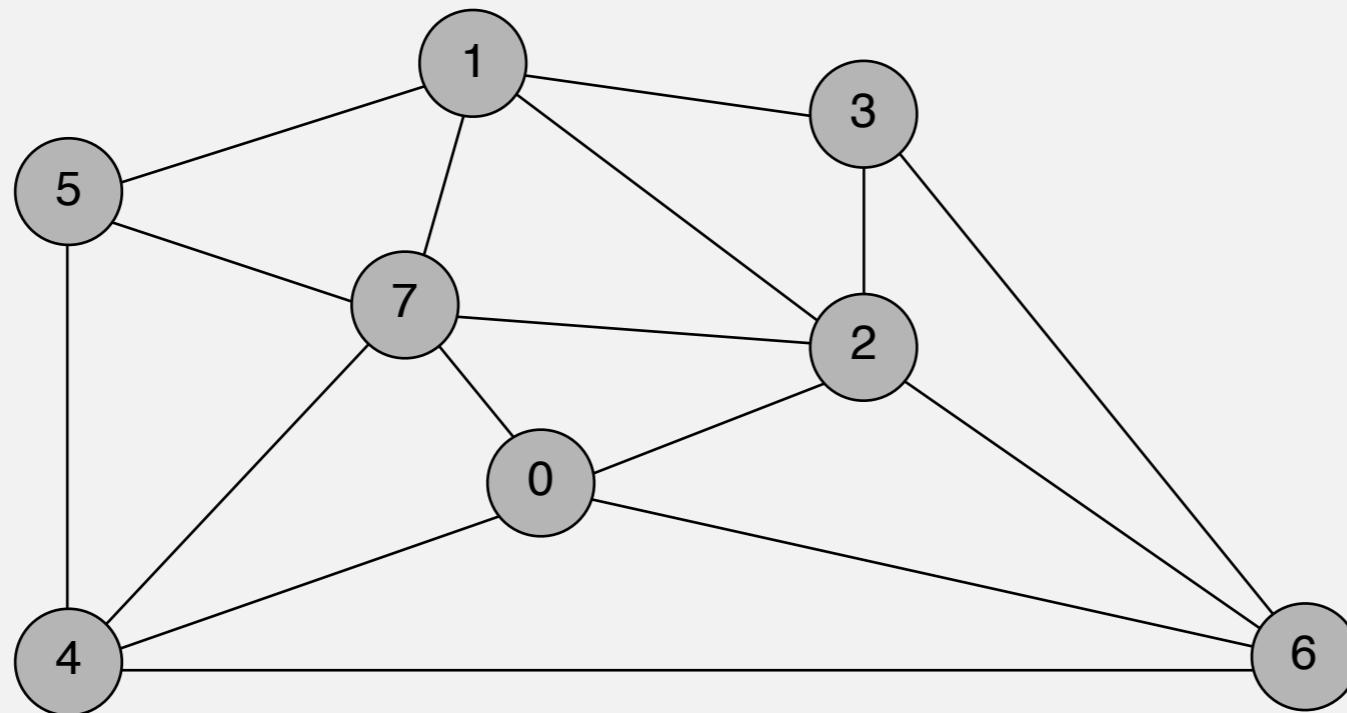
Observation. For each vertex v , need only **shortest** edge connecting v to T .

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



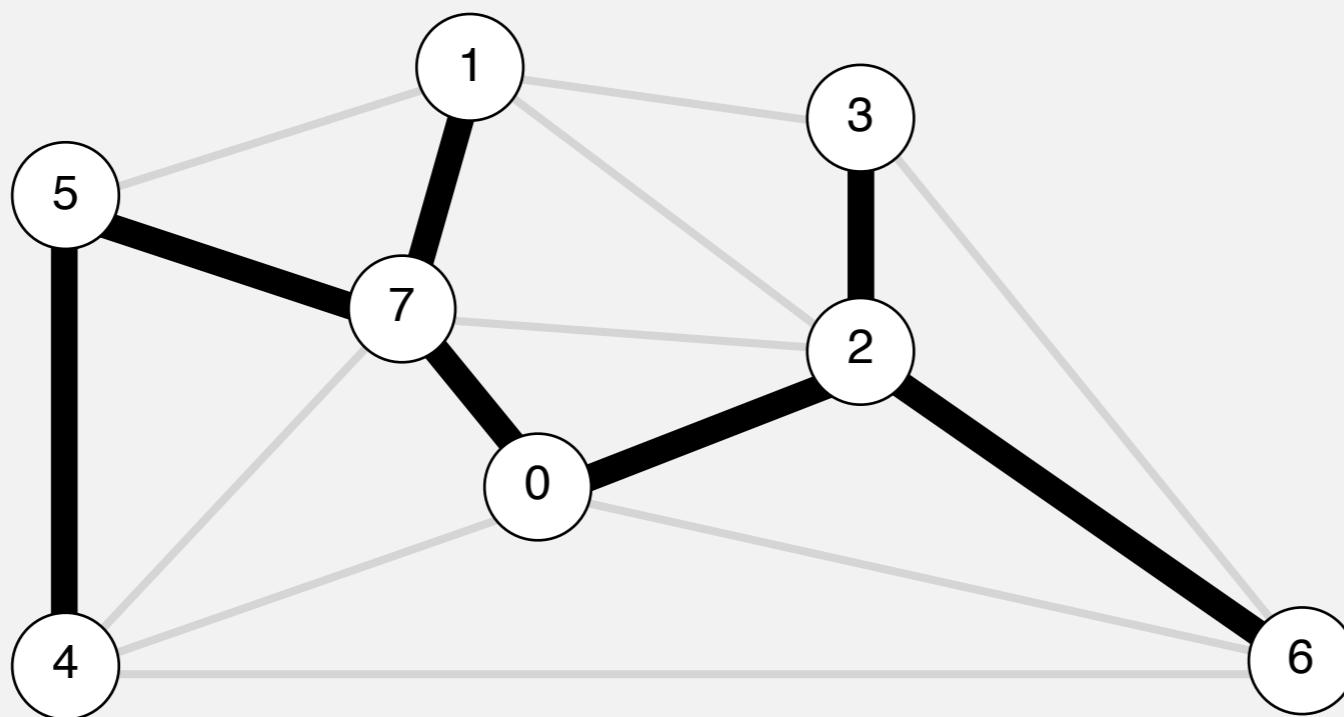
an edge-weighted graph

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

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

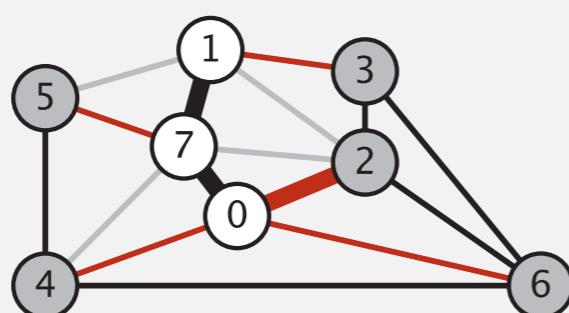
0-7 1-7 0-2 2-3 5-7 4-5 6-2

Prim's algorithm: eager implementation

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

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



black: on MST

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

Indexed priority queue

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

- Supports **insert** and **delete-the-minimum**.
- Supports **decrease-key** given the index of the key.

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

```
IndexMinPQ(int N)
```

create indexed priority queue

with indices 0, 1, ..., $N - 1$

associate key with index i

```
void insert(int i, Key key)
```

decrease the key associated with index i

```
void decreaseKey(int i, Key key)
```

```
boolean contains(int i)
```

is i an index on the priority queue?

```
int delMin()
```

*remove a minimal key and return its
associated index*

```
boolean isEmpty()
```

is the priority queue empty?

```
int size()
```

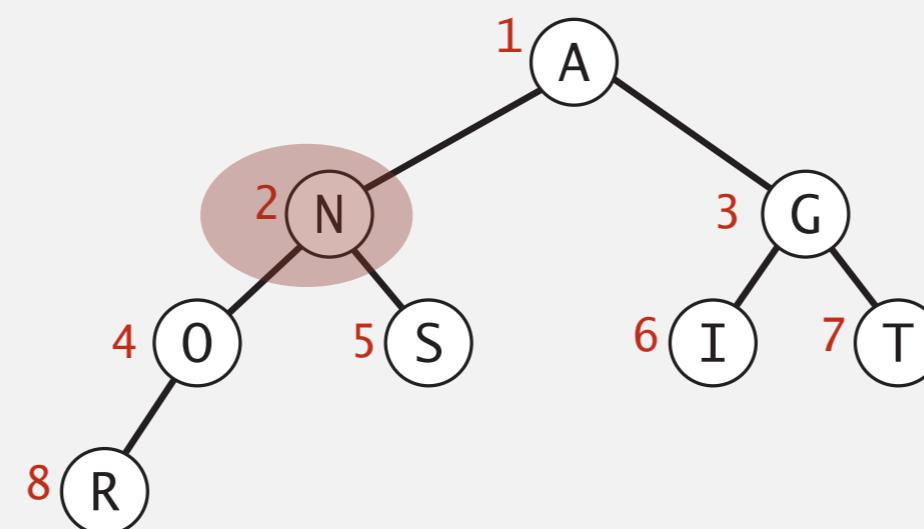
number of keys in the priority queue

Indexed priority queue implementation

Binary heap implementation. [see Section 2.4 of textbook]

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

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



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^\dagger	$\log V^\dagger$	1^\dagger	$E + V \log V$

\dagger 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

- ▶ 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

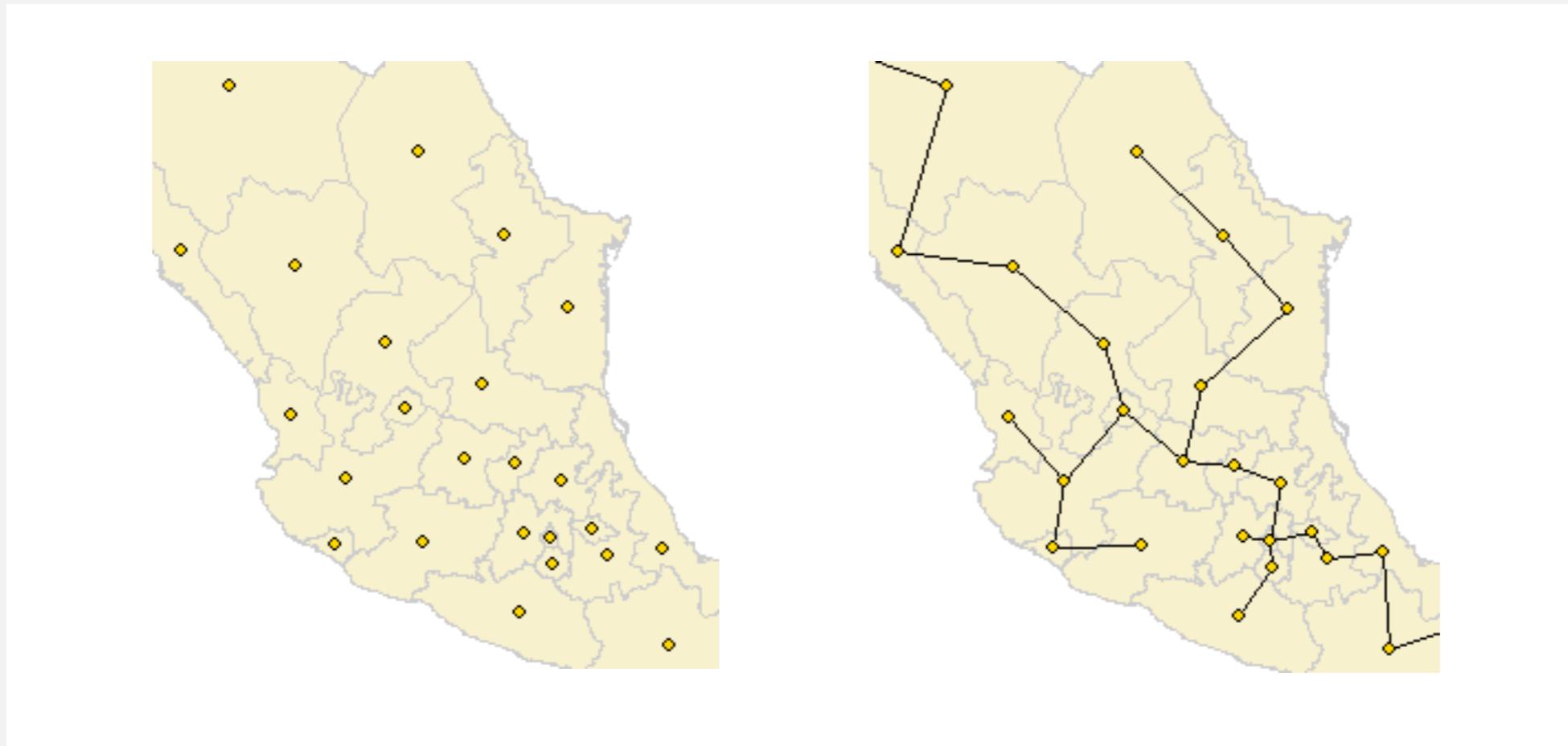
year	worst case	discovered by
1975	$E \log \log V$	Yao
1976	$E \log \log V$	Cheriton-Tarjan
1984	$E \log^* V, E + V \log V$	Fredman-Tarjan
1986	$E \log (\log^* V)$	Gabow-Galil-Spencer-Tarjan
1997	$E \alpha(V) \log \alpha(V)$	Chazelle
2000	$E \alpha(V)$	Chazelle
2002	<i>optimal</i>	Pettie-Ramachandran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



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.

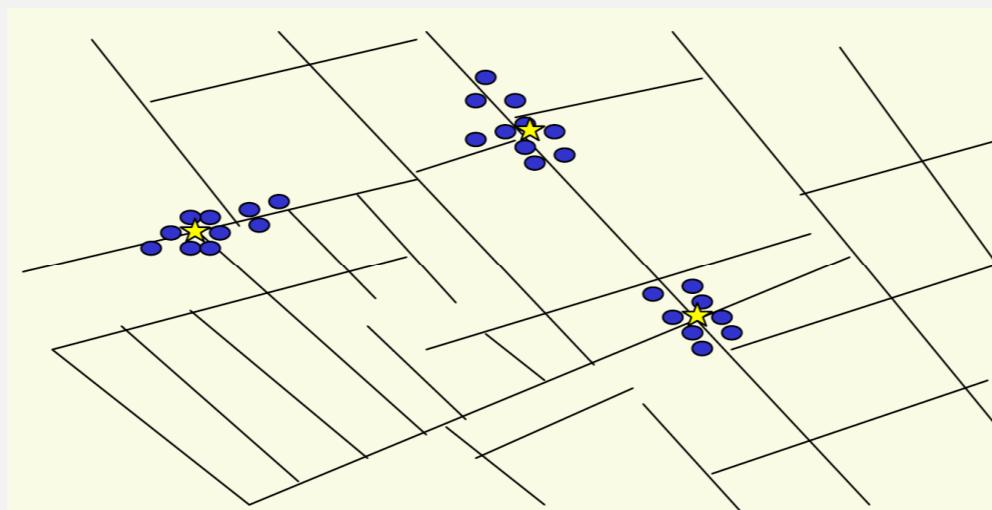
Ingenuity. Exploit geometry and do it in $\sim c N \log N$.

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.

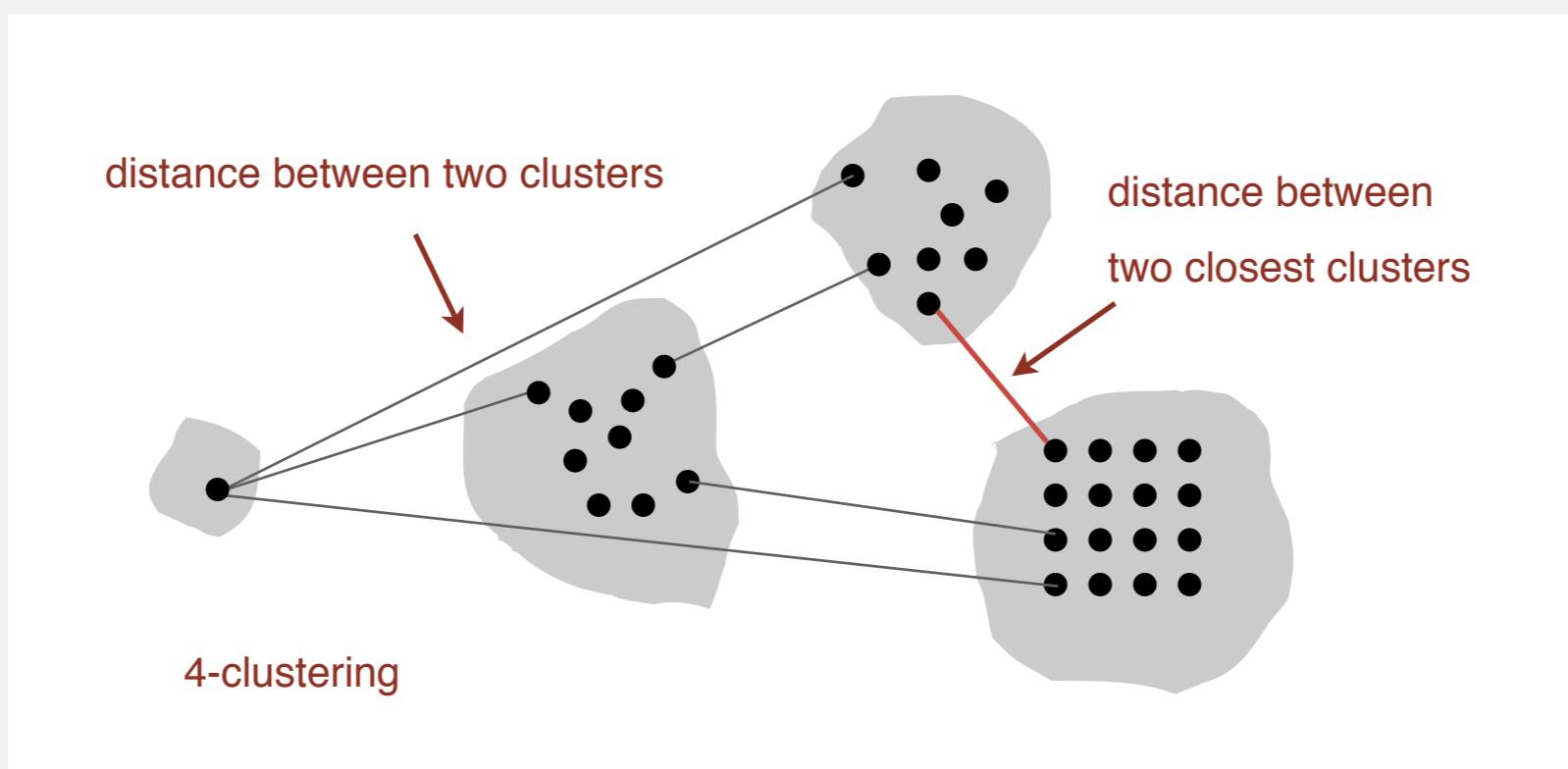
Single-link clustering

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.

