Announcements

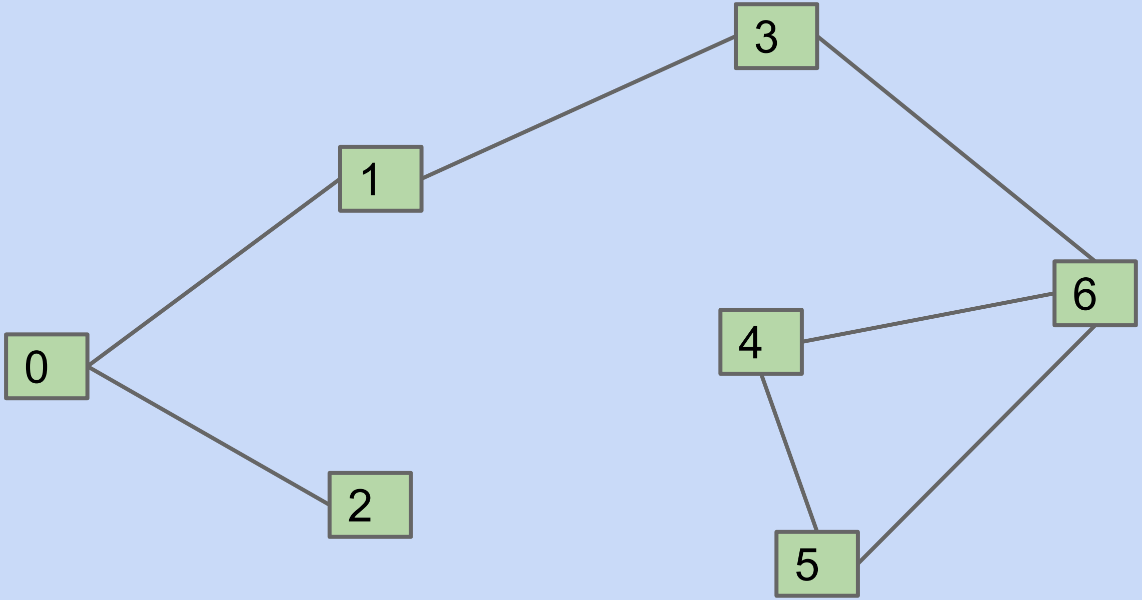
Project 3 released by Saturday (but maybe a day or two sooner).

- Significant project so don't put off starting until the last minute!
  - At the very least, make sure you can compile and run the starter code (MapServer.java) by the end of the weekend!
- On par with project 2 in scale, but it's a **solo project**.
- This project will be the closest to what you might do at an internship.
  - Using datasets.
  - Working with and improving upon an existing code base.
  - Reading javadocs and documentation.
  - Use of many tools at once (webbrowser, Maven, IntelliJ).

Warm-up Problem

Given a undirected graph, determine if it contains any cycles.

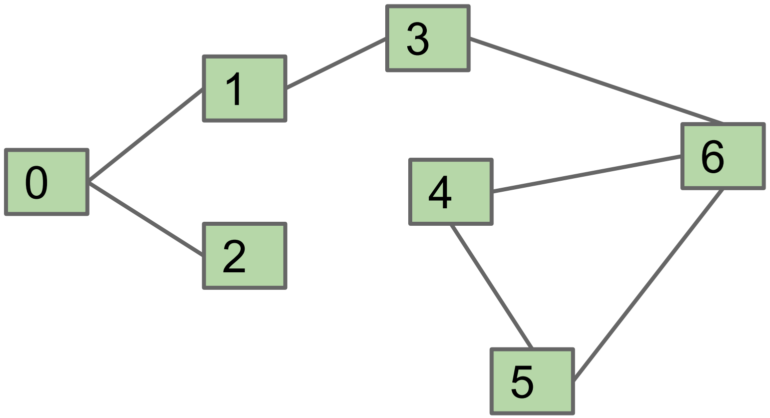
- May use any data structure or algorithm from the course so far.



Warm-up Problem

Given a undirected graph, determine if it contains any cycles.

- May use any data structure or algorithm from the course so far.



Approach 1: Do DFS from 0 (arbitrary vertex).

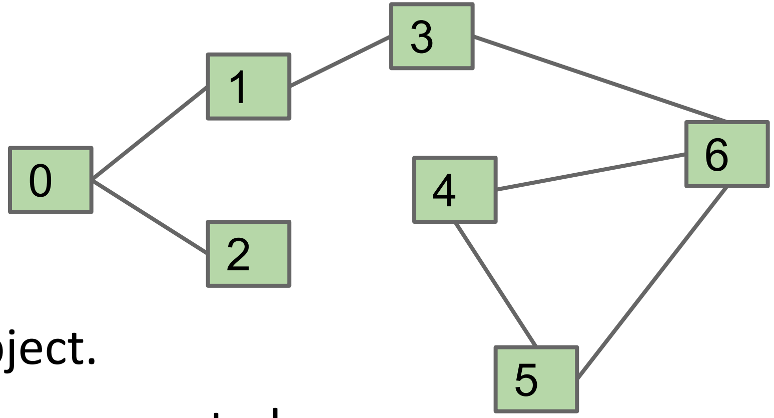
- Keep going until you see a marked vertex.
- Potential danger:
  - 1 looks back at 0 and sees marked.
  - Solution: Just don’t count the node you came from.

Worst case runtime: Θ(V + E) -- do study guide problems to reinforce this.

Warm-up Problem

Given a undirected graph, determine if it contains any cycles.

- May use any data structure or algorithm from the course so far.



Approach 2: Use a WeightedQuickUnionUF object.

- For each edge, check if the two vertices are connected.
  - If not, union them.
  - If so, there is a cycle.

Worst case runtime: O(V + E log* V) if we have path compression.

CS61B

Lecture 30: Graphs IV: Minimum Spanning Trees

- MST, Cut Property, Generic MST Algorithm
- Prim's Algorithm
- Kruskal's Algorithm

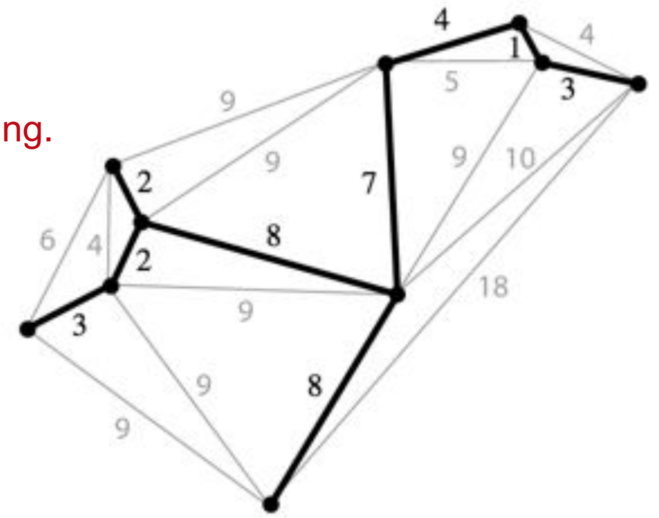
Spanning Trees

Given an **undirected** graph, a ***spanning tree*** T is a subgraph of G, where T:

- Is connected.
- Is acyclic.
- Includes all of the vertices.

These two properties make it a tree.

This makes it spanning.



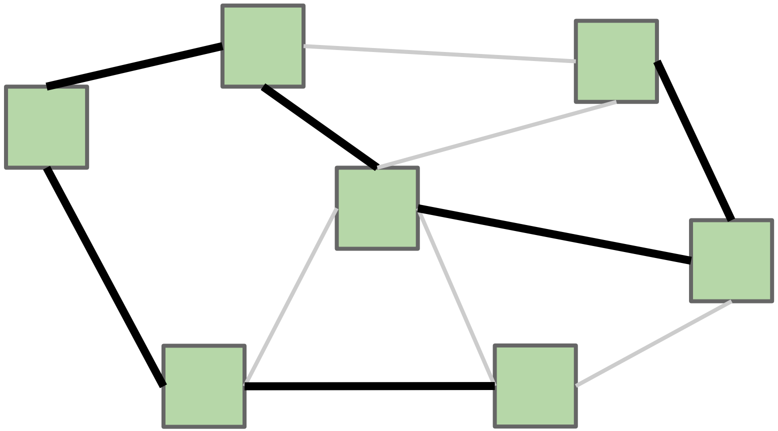
Example:

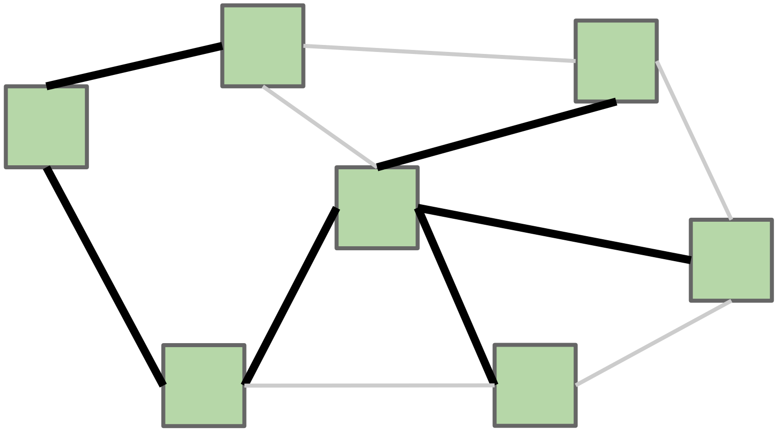
- Spanning tree is the black edges and vertices.

A ***minimum spanning tree*** is a spanning tree of minimum total weight.

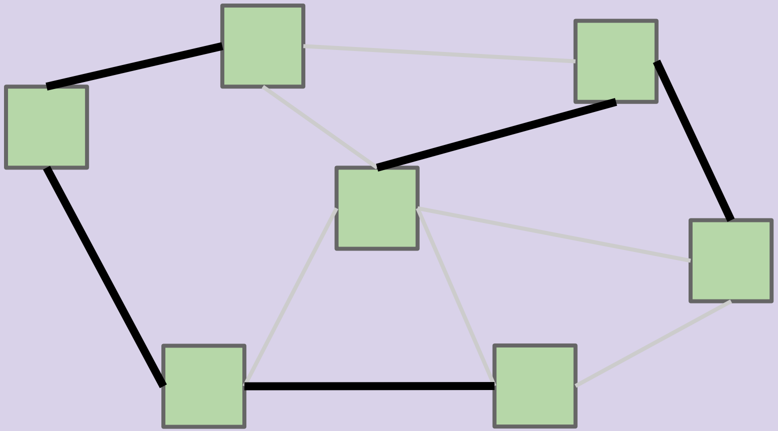
- Example: Directly connecting buildings by power lines.

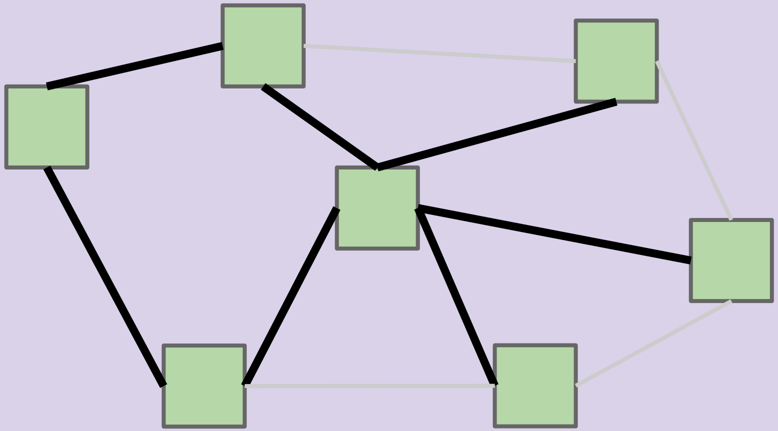
Spanning Trees

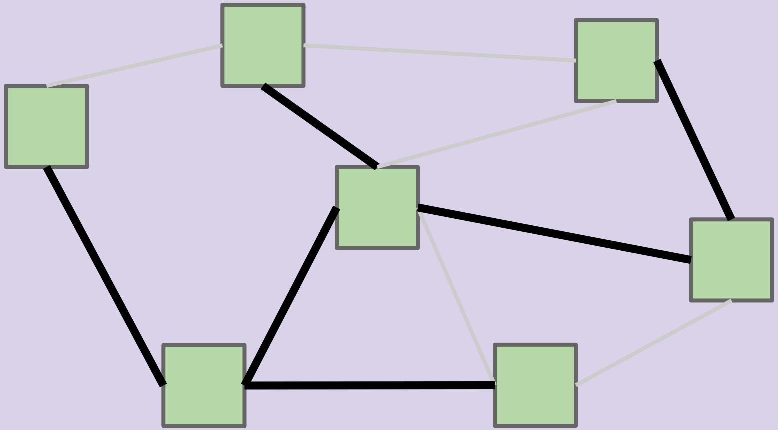




How Many are Spanning Trees? http://yellkey.com/fall







MST Applications

Old school handwriting recognition (left ([link](#)))

Medical imaging (e.g. arrangement of nuclei in cancer cells (right))

For more, see: http://www.ics.uci.edu/~eppstein/gina/mst.html

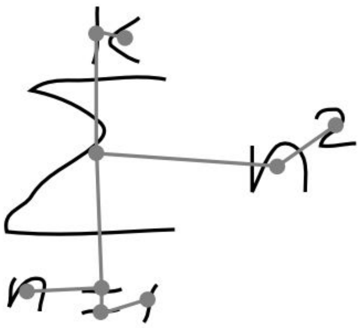
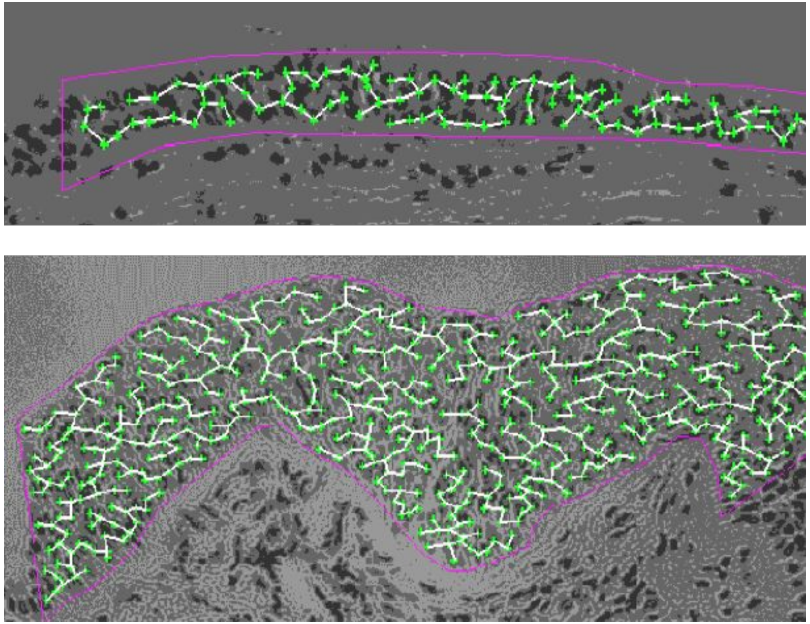
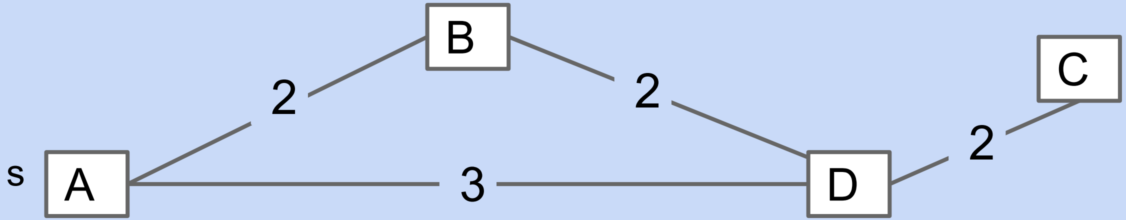


Figure 4-3: A typical minimum spanning tree



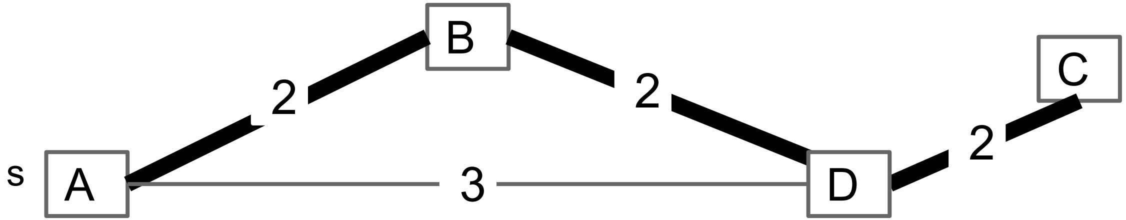
MST

Find the MST for the graph.



MST

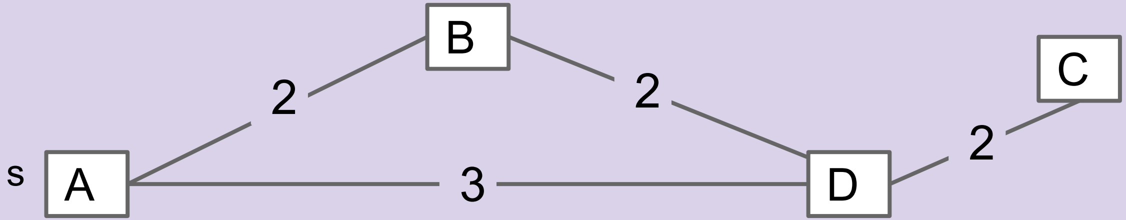
Find the MST for the graph.



MST vs. SPT, http://yellkey.com/approach

Is the MST for this graph also a shortest paths tree? If so, using which node as the starting node for this SPT?

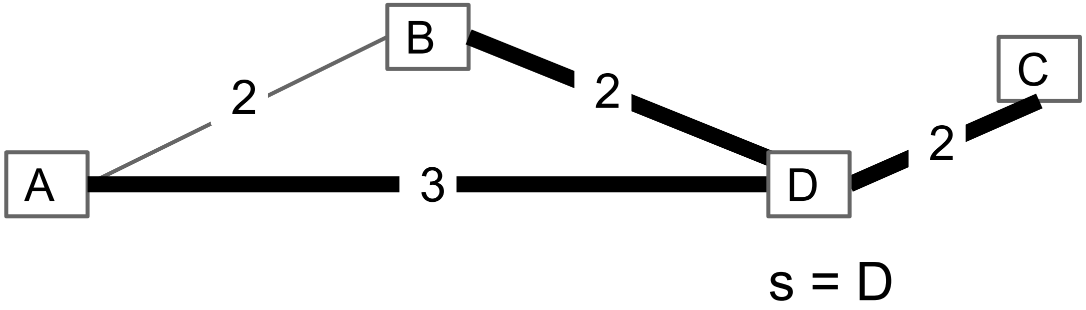
A. A
B. B
C. C
D. D
E. No SPT is an MST.



MST vs. SPT, http://yellkey.com/approach

Is the MST for this graph also a shortest paths tree? If so, using which node as the starting node for this SPT?

A. A
B. B
C. C
D. D
E. No SPT is an MST.



Doesn't work for s = D!

MST vs. SPT, http://yellkey.com/approach

Is the MST for this graph also a shortest paths tree? If so, using which node as the starting node for this SPT?

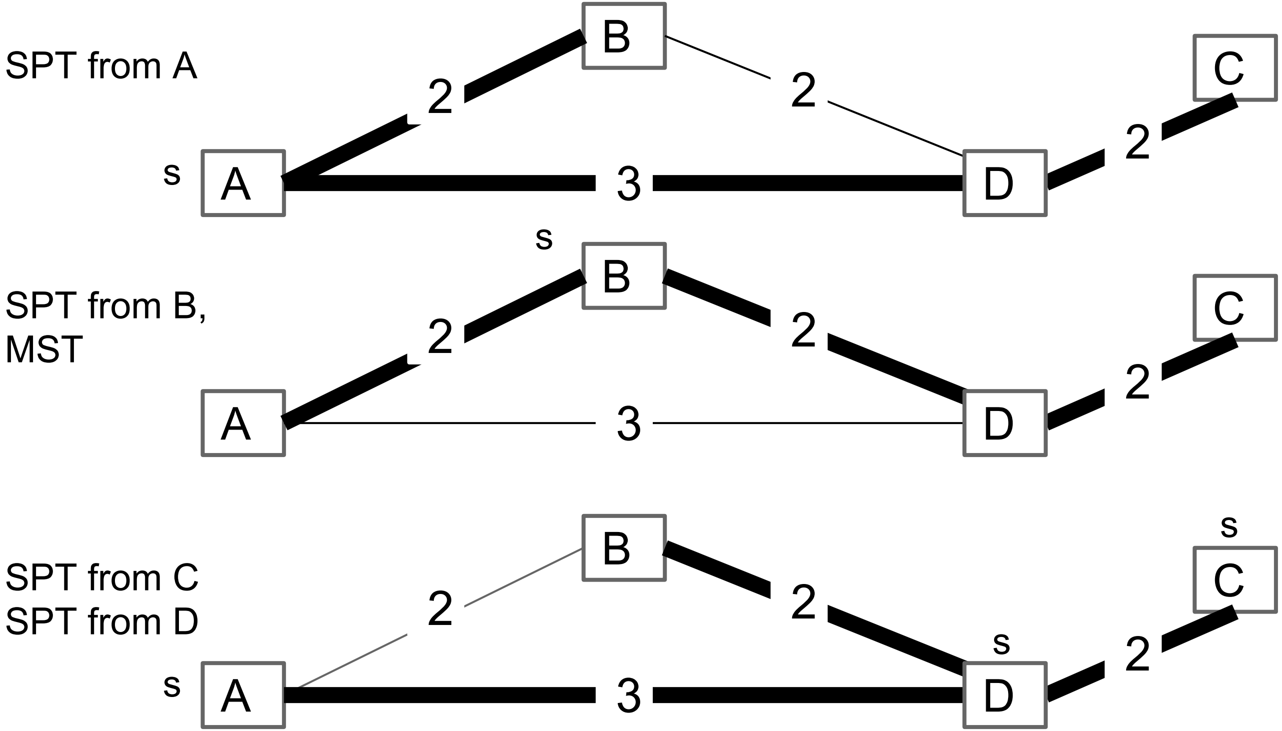
A. A

B. B

C. C

D. D

E. No SPT is an MST.



MST vs. SPT, http://yellkey.com/approach

A shortest paths tree depends on the start vertex:

- Because it tells you how to get from a source to EVERYTHING.

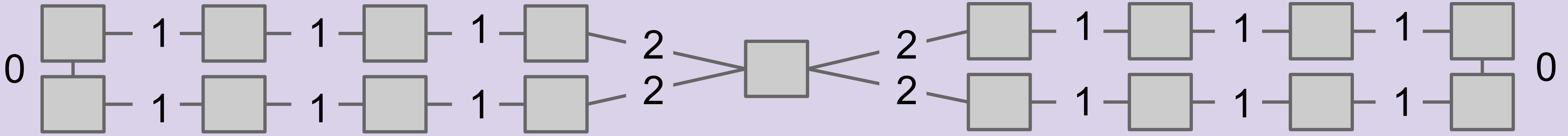
There is no source for a MST.

Nonetheless, the MST sometimes happens to be an SPT for a specific vertex.

Spanning Tree, http://yellkey.com/maintain

Give a valid MST for the graph below.

- Hard B level question: Is there a node whose SPT is also the MST?



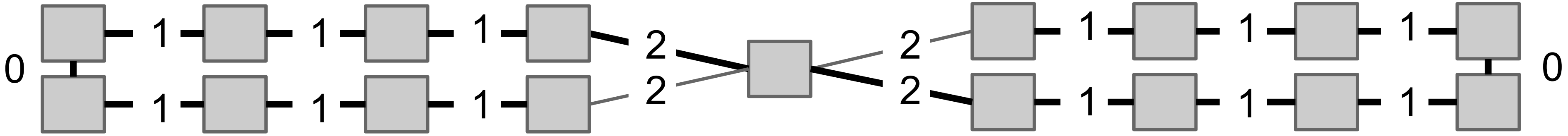
A. Yes

B. No

Spanning Tree

Give a valid MST for the graph below.

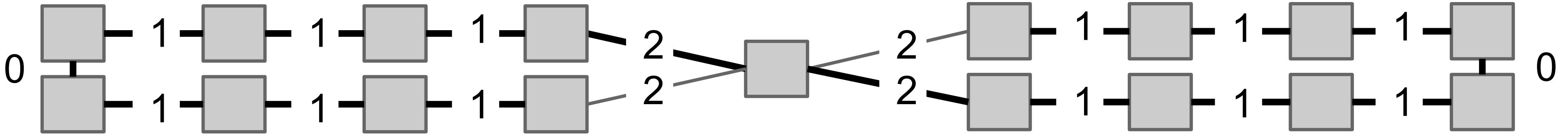
- Is there a node whose SPT is also the MST? [see next slide]



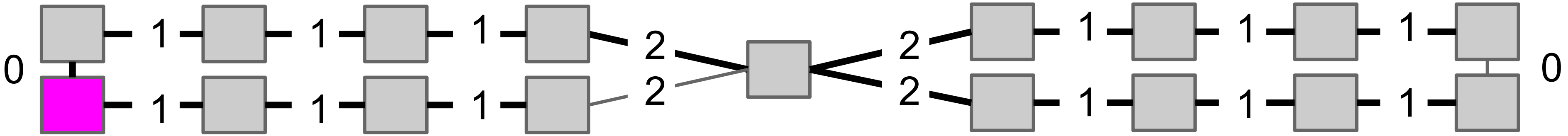
Spanning Tree

Give a valid MST for the graph below.

- Is there a node whose SPT is also the MST?
- **No!** Minimum spanning tree must include only 2 of the 2 weight edges, but the SPT always includes at least 3 of the 2 weight edges.

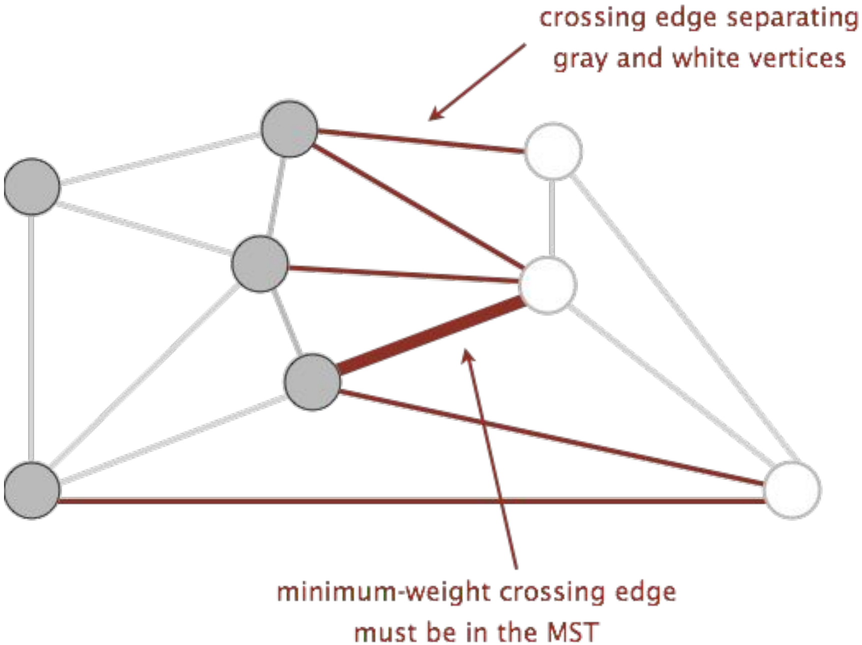


Example SPT from bottom left vertex:



A Useful Tool for Finding the MST: Cut Property

- A ***cut*** is an assignment of a graph's nodes to two non-empty sets.
- A ***crossing edge*** is an edge which connects a node from one set to a node from the other set.



Cut property: Given any cut, minimum weight crossing edge is in the MST.

- For rest of today, we'll assume edge weights are unique.

Prim's Runtime

Exactly like Dijkstra's runtime:

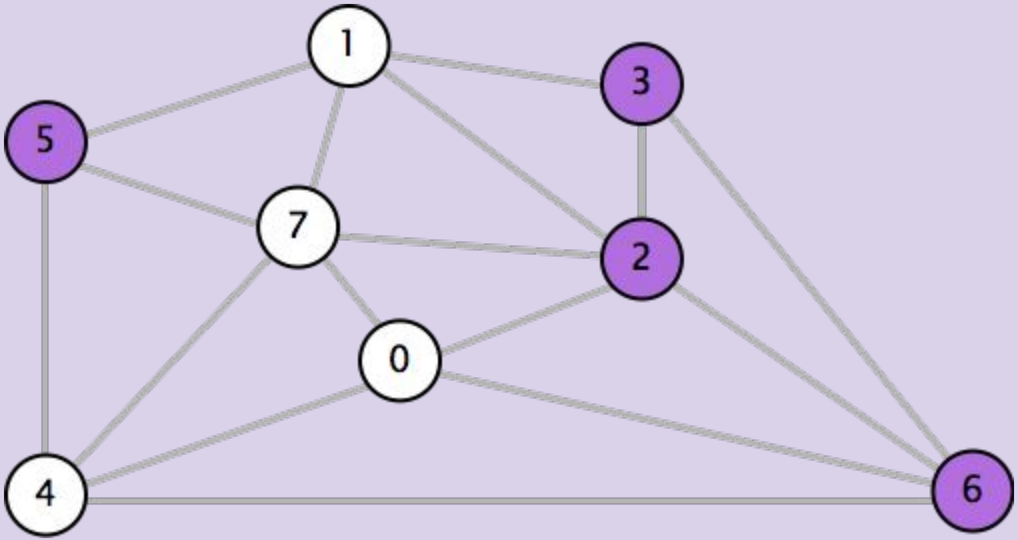
- Insertions: V, each costing O(log V) time.
- Delete-min: V, each costing O(log V) time.
- DecreasePriority: E, each costing O(log V) time.
    - Data structure not discussed in class.

Overall runtime, assuming E > V, we have O(E log V) runtime.

Operation	Number of Times	Time per Operation	Total Time
Insert	V	O(log V)	O(V log V)
Delete minimum	V	O(log V)	O(V log V)
Decrease priority	E	O(log V)	O(E log V)

Cut Property in Action: http://yellkey.com/each

Which edge is the minimum weight edge crossing the cut {2, 3, 5, 6}?

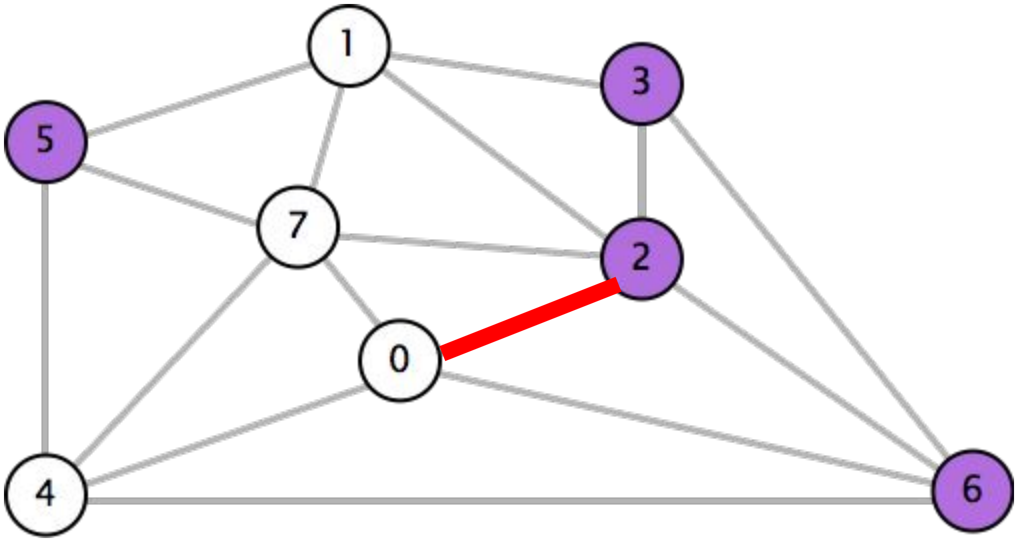


0-7	0.16
2-3	0.17
1-7	0.19
0-2	0.26
5-7	0.28
1-3	0.29
1-5	0.32
2-7	0.34
4-5	0.35
1-2	0.36
4-7	0.37
0-4	0.38
6-2	0.40
3-6	0.52
6-0	0.58
6-4	0.93

Cut Property in Action

Which edge is the minimum weight edge crossing the cut {2, 3, 5, 6}?

- 0-2. Must be part of the MST!

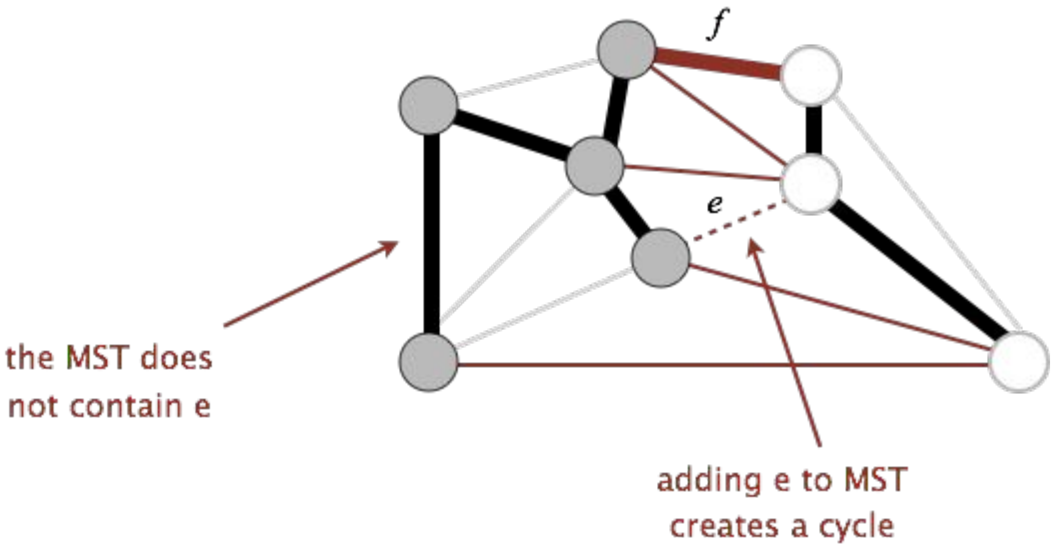


0-7	0.16
2-3	0.17
1-7	0.19
0-2	**0.26**
5-7	0.28
1-3	0.29
1-5	0.32
2-7	0.34
4-5	0.35
1-2	0.36
4-7	0.37
0-4	0.38
6-2	0.40
3-6	0.52
6-0	0.58
6-4	0.93

Cut Property Proof

Suppose that the minimum crossing edge *e* were not in the MST.

- Adding *e* to the MST creates a cycle.
- Some other edge *f* must also be a crossing edge.
- Removing f and adding e is a lower weight spanning tree.
- Contradiction!



Generic MST Finding Algorithm

Start with no edges in the MST.

- Find a cut that has no crossing edges in the MST.
- Add smallest crossing edge to the MST.
- Repeat until V-1 edges.

This should work, but we need some way of finding a cut with no crossing edges!

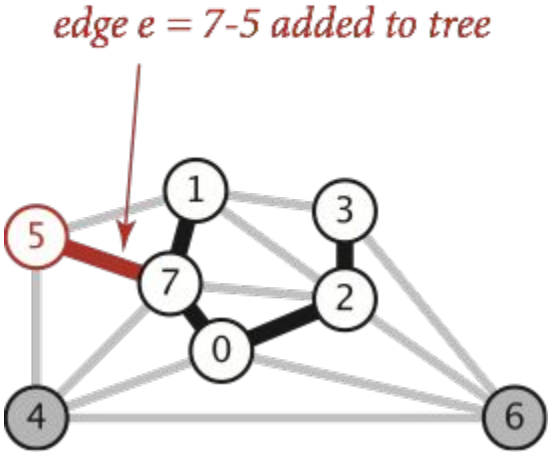
- Random isn't a very good idea.

Prim’s Algorithm

Prim's Algorithm

Start from some arbitrary start node.

- Repeatedly add shortest edge (mark black) that has one node inside the MST under construction.
- Repeat until V-1 edges.

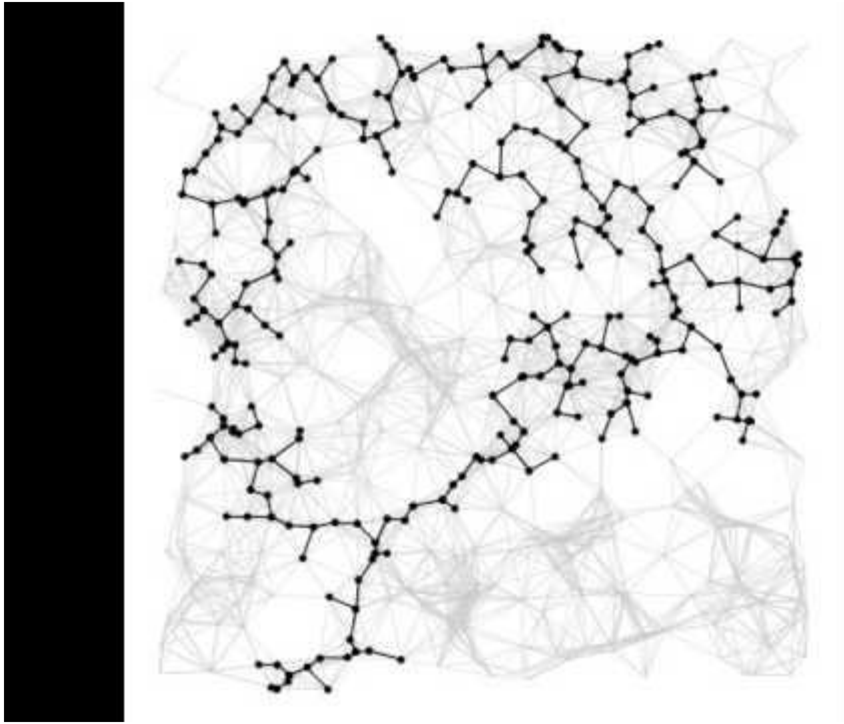


Conceptual Prim's Algorithm Demo (Link)

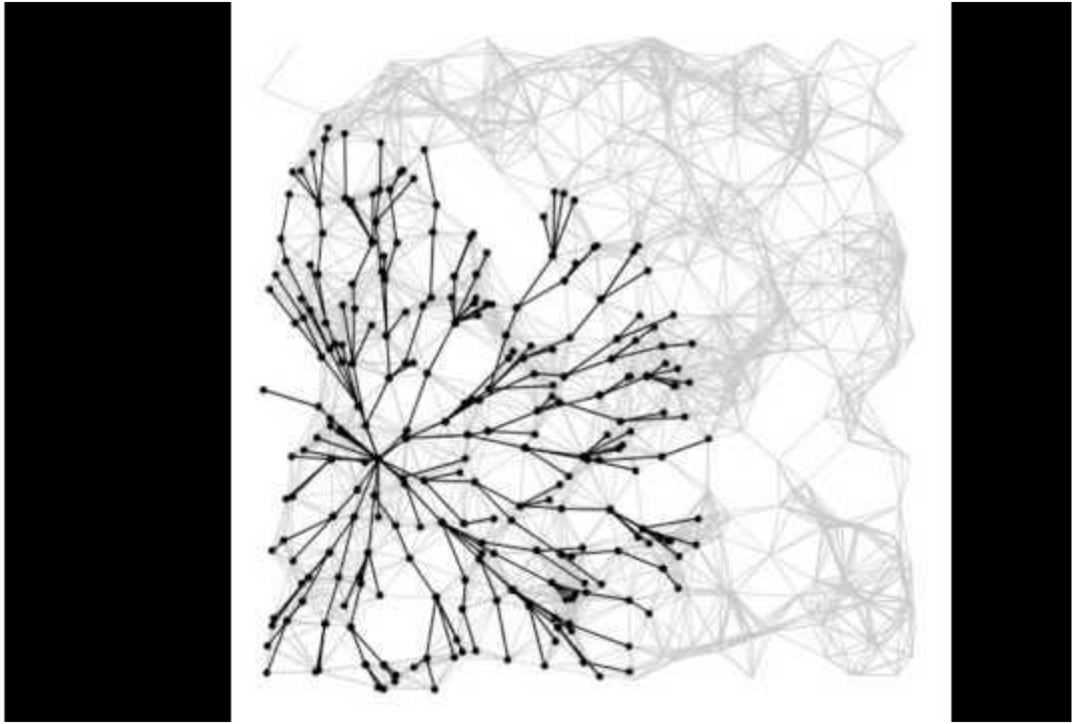
Why does Prim's work? Special case of generic algorithm.

- Suppose we add edge e = v->w.
- Side 1 of cut is all vertices connected to start, side 2 is all the others.
- No crossing edge is black (all connected edges on side 1).
- No crossing edge has lower weight (consider in increasing order).

Prim's vs. Dijkstra's (visual)



Prim's Algorithm



Dijkstra's Algorithm

Demos courtesy of Kevin Wayne, Princeton University

Prim’s Algorithm Implementation

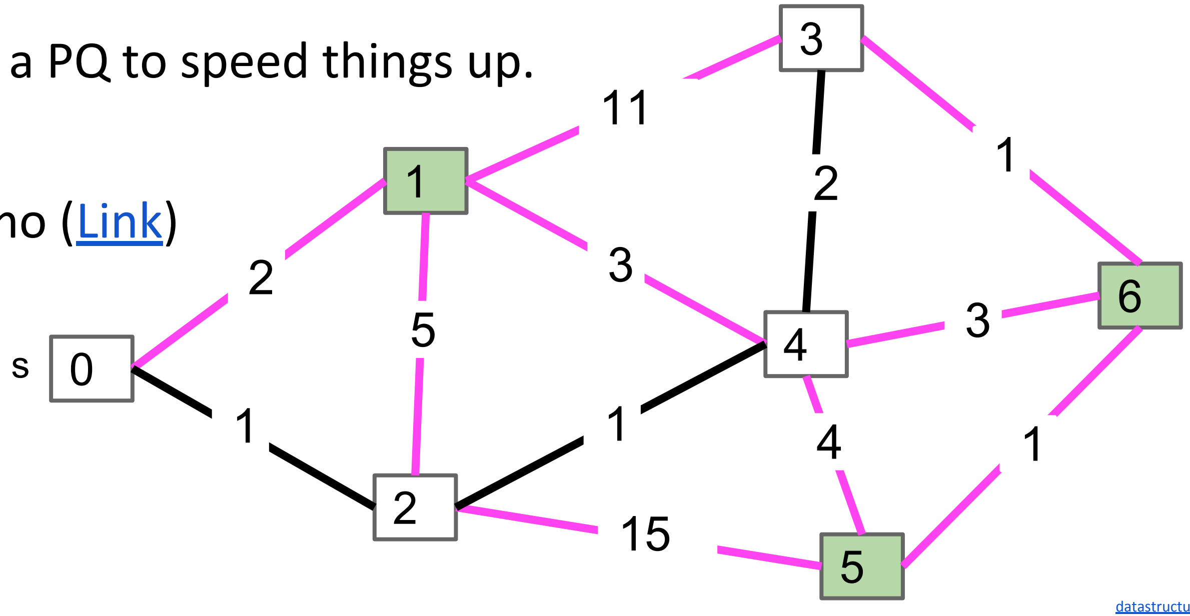
The natural implementation of the conceptual version of Prim’s algorithm is highly inefficient.

- Example: Iterating over purple edges shown is unnecessary and slow.

Can use some cleverness and a PQ to speed things up.

Realistic Implementation Demo (Link)

- Very similar to Dijkstra’s!



Prim’s vs. Dijkstra’s

Prim’s and Dijkstra’s algorithms are exactly the same, except Dijkstra’s considers “distance from the source”, and Prim’s considers “distance from the tree.”

Visit order:

- Dijkstra’s algorithm visits vertices in order of distance from the source.
- Prim’s algorithm visits vertices in order of distance from the MST under construction.

Relaxation:

- Relaxation in Dijkstra’s considers an edge better based on distance to source.
- Relaxation in Prim’s considers an edge better based on distance to tree.

Prim's Implementation (Pseudocode, 1/2)

```
public class PrimMST {
  public PrimMST(EdgeWeightedGraph G) {
    edgeTo = new Edge[G.V()];
    distTo = new double[G.V()];
    marked = new boolean[G.V()];
    fringe = new SpecialPQ<Double>(G.V());

    distTo[s] = 0.0;
    setDistancesToInfinityExceptS(s);
    insertAllVertices(fringe);

    /* Get vertices in order of distance from tree. */
    while (!fringe.isEmpty()) {
      int v = fringe.delMin();
      scan(G, v);
    }
  }
  ...
```

Fringe is ordered by distTo tree. Must be a specialPQ like Dijkstra's.

Get vertex closest to tree that is unvisited.

Scan means to consider all of a vertices outgoing edges.

Prim's Implementation (Pseudocode, 2/2)

```
while (!fringe.isEmpty()) {
  int v = fringe.delMin();
  scan(G, v);
}
```

Important invariant, fringe must be ordered by current best known distance from tree.

```
private void scan(EdgeWeightedGraph G, int v) {
  marked[v] = true;
  for (Edge e : G.adj(v)) {
    int w = e.other(v);
    if (marked[w]) { continue; }
    if (e.weight() < distTo[w]) {
      distTo[w] = e.weight();
      edgeTo[w] = e;
      pq.decreasePriority(w, distTo[w]);
    }
  }
}
```

Vertex is closest, so add to MST.

Already in MST, so go to next edge.

Better path to a particular vertex found, so update current best known for that vertex.

Prim's Runtime

```
while (!fringe.isEmpty()) {
  int v = fringe.delMin();
  scan(G, v);
}
```

```
private void scan(EdgeWeightedGraph G, int v) {
  marked[v] = true;
  for (Edge e : G.adj(v)) {
    int w = e.other(v);
    if (marked[w]) { continue; }
    if (e.weight() < distTo[w]) {
      distTo[w] = e.weight();
      edgeTo[w] = e;
      pq.decreasePriority(w, distTo[w]);
    }
  }
}
```

What is the runtime of Prim's algorithm?

- Assume all PQ operations take O(log(V)) time.
- Give your answer in Big O notation.

Prim's Algorithm Runtime

Priority Queue operation count, assuming binary heap based PQ:

- Insertion: V, each costing O(log V) time.
- Delete-min: V, each costing O(log V) time.
- Decrease priority: O(E), each costing O(log V) time.
  - Operation not discussed in lecture, but it was in lab 10.

Overall runtime: O(V*log(V) + V*log(V) + E*logV).

- Assuming E > V, this is just O(E log V).

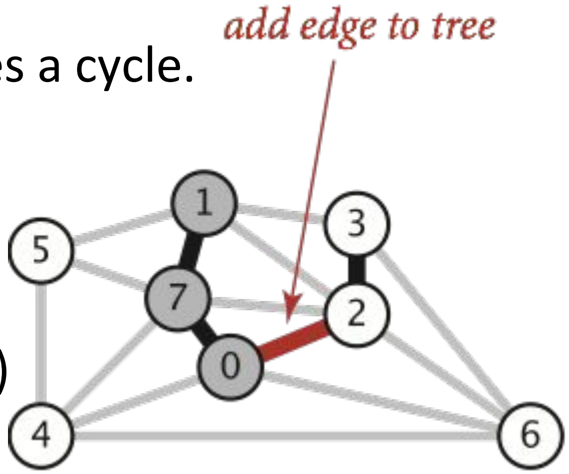
	# Operations	Cost per operation	Total cost
PQ add	V	O(log V)	O(V log V)
PQ delMin	V	O(log V)	O(V log V)
PQ decreasePriority	O(E)	O(log V)	O(E log V)

Kruskal’s Algorithm

Kruskal's Algorithm

Initially mark all edges gray.

- Consider edges in increasing order of weight.
- Add edge to MST (mark black) unless doing so creates a cycle.
- Repeat until V-1 edges.



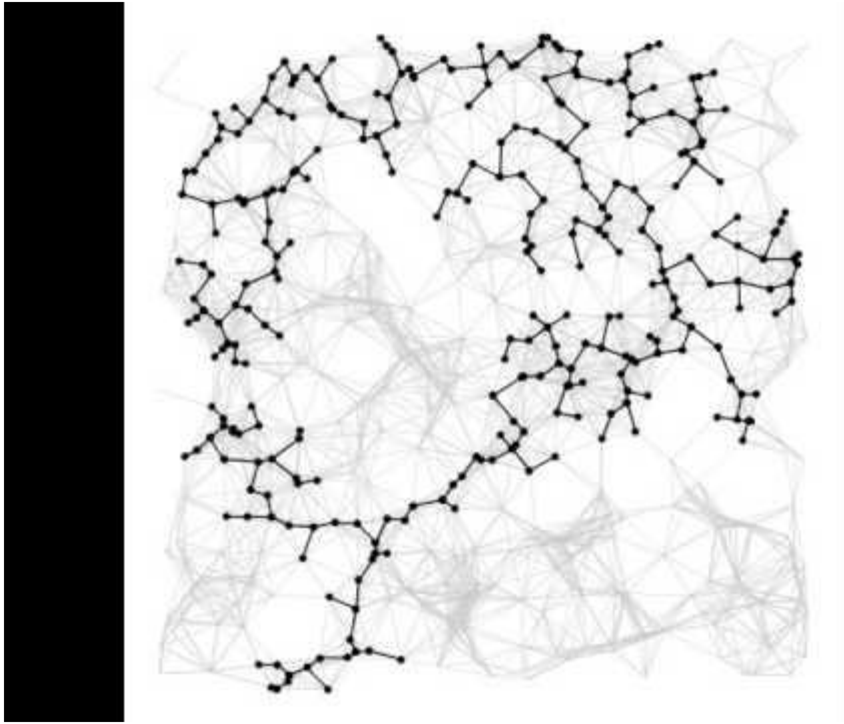
Conceptual Kruskal's Algorithm Demo (Link)

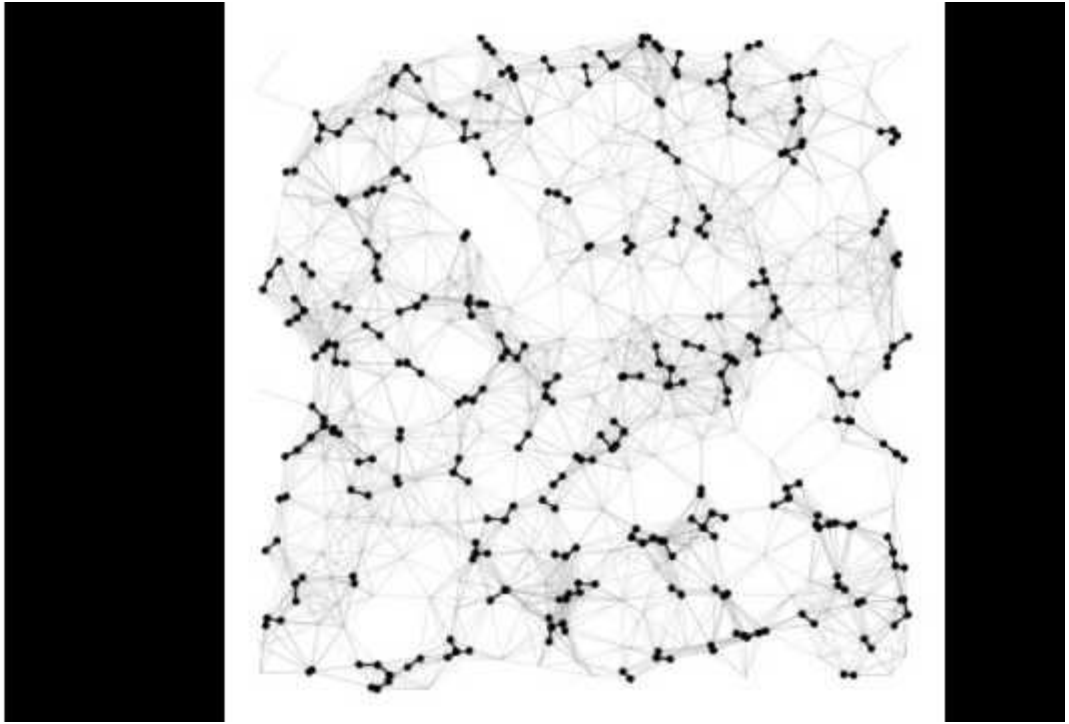
Realistic Kruskal's Algorithm Implementation Demo (Link)

Why does Kruskal's work? Special case of generic MST algorithm.

- Suppose we add edge e = v->w.
- Side 1 of cut is all vertices connected to v, side 2 is everything else.
- No crossing edge is black (since we don't allow cycles).
- No crossing edge has lower weight (consider in increasing order).

Prim's vs. Kruskal's





Prim's Algorithm

Kruskal's Algorithm

Demos courtesy of Kevin Wayne, Princeton University

Kruskal’s Implementation (Pseudocode)

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

  public KruskalMST(EdgeWeightedGraph G) {
    MinPQ<Edge> pq = new MinPQ<Edge>();
    for (Edge e : G.edges()) {
      pq.insert(e);
    }
    WeightedQuickUnionPC uf =
             new WeightedQuickUnionPC(G.V());
    while (!pq.isEmpty() && mst.size() < G.V() - 1) {
      Edge e = pq.delMin();
      int v = e.from();
      int w = e.to();
      if (!uf.connected(v, w)) {
        uf.union(v, w);
        mst.add(e);
} } } }
```

What is the runtime of Kruskal’s algorithm?

- Assume all PQ operations take O(log(V)) time.
- Assume all WQU operations take O(log* V) time.
- Give your answer in Big O notation.

Kruskal's Runtime

Kruskal's algorithm on previous slide is O(E log E).

Fun fact: In HeapSort lecture, we discuss how do this step in O(E) time using "bottom-up heapification".

Operation	Number of Times	Time per Operation	Total Time
Insert	E	O(log E)	O(E log E)
Delete minimum	O(E)	O(log E)	O(E log E)
union	O(V)	O(log* V)	O(V log* V)
isConnected	O(E)	O(log* V)	O(E log*V)

Note: If we use a pre-sorted list of edges (instead of a PQ), then we can simply iterate through the list in O(E) time, so overall runtime is O(E + V log*V + E log* V) = O(E log*V).

Shortest Paths and MST Algorithms Summary

Problem	Algorithm	Runtime (if E > V)	Notes
Shortest Paths	Dijkstra’s	O(E log V)	Fails for negative weight edges.
MST	Prim’s	O(E log V)	Analogous to Dijkstra’s.
MST	Kruskal’s	O(E log E)	Uses WQUPC.
MST	Kruskal’s with pre-sorted edges	O(E log* V)	Uses WQUPC.

Question: Can we do better than O(E log* V)?

170 Spoiler: State of the Art Compare-Based MST Algorithms

year	worst case	discovered by
1975	*E log log V*	Yao
1984	*E log* V*	Fredman-Tarjan
1986	*E log (log* V)*	Gabow-Galil-Spencer-Tarjan
1997	*E α(V) log α(V)*	Chazelle
2000	*E α(V)*	Chazelle
2002	*optimal (link)*	Pettie-Ramachandran
???	*E ???*	???

(Slide Courtesy of Kevin Wayne, Princeton University)

Citations

Tree fire: http://www.miamidade.gov/fire/library/hotlines/2011-december_files/tree-fire.jpg

Bicycle routes in Seattle: https://www.flickr.com/photos/ewedistrict/21980840

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