Minimum Spanning Trees

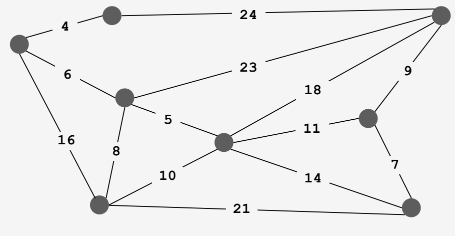
- weighted graph API
- cycles and cuts
- Kruskal's algorithm
- Prim's algorithm
- advanced topics

References:
Algorithms in Java, Chapter 20
`http://www.cs.princeton.edu/algs4/54mst`

MST Origin

Given. Undirected graph G with positive edge weights (connected).

Goal. Find a min weight set of edges that connects all of the vertices.

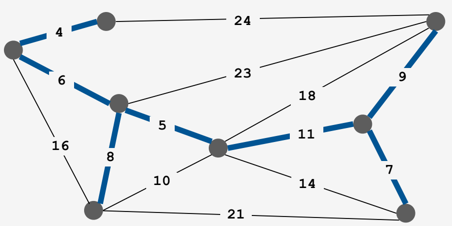


G

MST Origin

Given. Undirected graph G with positive edge weights (connected).

Goal. Find a min weight set of edges that connects all of the vertices.



`weight(T) = 50 = 4 + 6 + 8 + 5 + 11 + 9 + 7`

Brute force. Try all possible spanning trees.

- Problem 1: not so easy to implement.
- Problem 2: far too many of them.

Applications

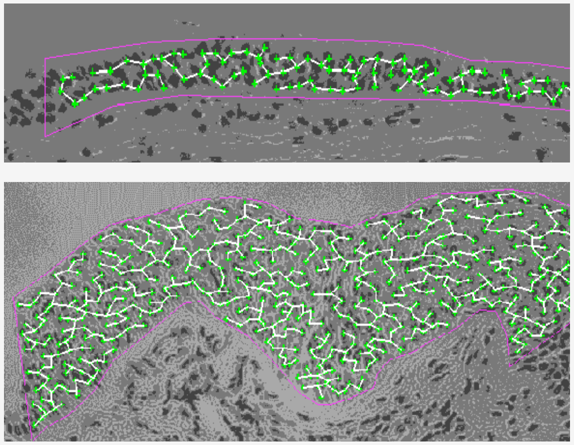
MST is fundamental problem with diverse applications.

- 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.
- Network design (telephone, electrical, hydraulic, cable, computer, road).
- Approximation algorithms for NP-hard problems (e.g., TSP, Steiner tree).

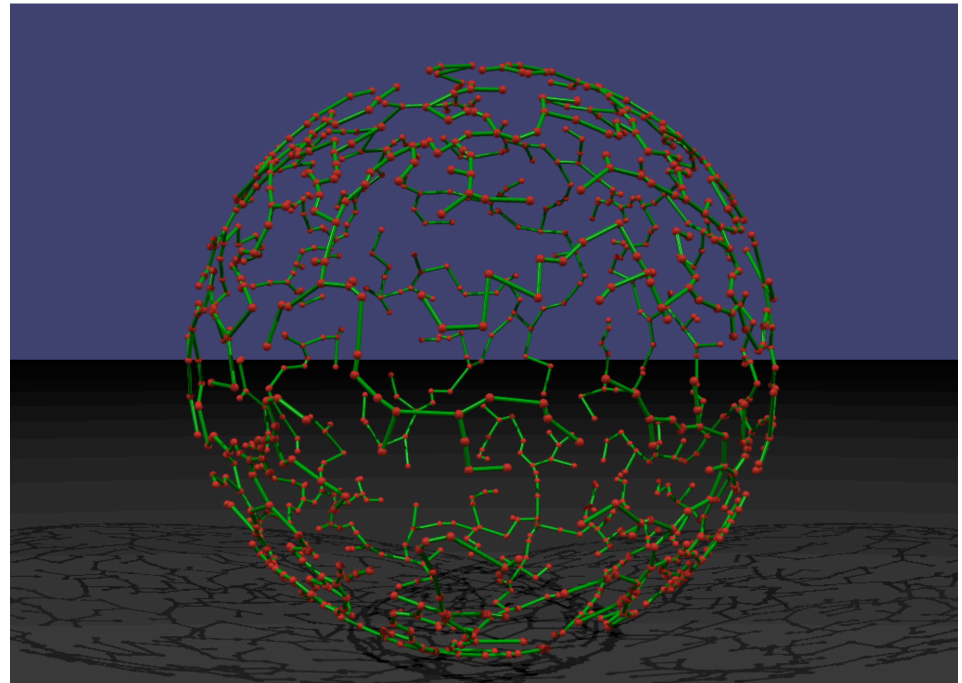
`http://www.ics.uci.edu/~eppstein/gina/mst.html`

Medical image processing

MST describes arrangement of nuclei in the epithelium for cancer research



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



http://ginger.indstate.edu/ge/gfx

Two Greedy Algorithms

Kruskal's algorithm. Consider edges in ascending order of weight. Add to T the next edge unless doing so would create a cycle.

Prim's algorithm. Start with any vertex s and greedily grow a tree T from s. At each step, add to T the edge of min weight that has exactly one endpoint in T.

> *" Greed is good. Greed is right. Greed works. Greed clarifies, cuts through, and captures the essence of the evolutionary spirit. "* — Gordon Gecko

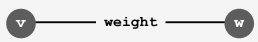
Proposition. Both greedy algorithms compute MST.

- **weighted graph API**
- cycles and cuts
- Kruskal's algorithm
- Prim's algorithm
- advanced topics

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*
double	weight()	*the weight*
String	toString()	*string representation*



Weighted graph API

public class WeightedGraph		*graph data type*
	WeightedGraph(int V)	*create an empty graph with V vertices*
	WeightedGraph(In in)	*create a graph from input stream*
void	insert(Edge e)	*add an edge from v to w*
Iterable<Edge>	adj(int v)	*return an iterator over edges incident to v*
int	V()	*return number of vertices*
String	toString()	*return a string representation*

```
for (int v = 0; v < G.V(); v++)
{
   for (Edge e : G.adj(v))
   {
      int w = e.other(v);
      // process edge v-w
   }
}
```

iterate through all edges (once in each direction)

Weighted graph: adjacency-set implementation

```
public class WeightedGraph
{
   private final int V;
   private final SET<Edge>[] adj;

   public WeightedGraph(int V)
   {
      this.V = V;
      adj = (SET<Edge>[]) new SET[V];
      for (int v = 0; v < V; v++)
         adj[v] = new SET<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 sets of Edges instead of integers

constructor

add edge to both adjacency sets

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 = Math.min(v, w);
      this.w = Math.max(v, w);
      this.weight = weight;
   }

   public int either()
   {  return v;  }

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

   public int weight()
   {  return weight;  }

   // See next slide for compare methods.
}
```

constructor

either endpoint

other endpoint

weight of edge

Weighted edge: Java implementation (cont)

```
public static class ByWeight implements Comparator<Edge>
{
   public int compare(Edge e, Edge f)
   {
      if (e.weight < f.weight) return -1;
      if (e.weight > f.weight) return +1;
      return 0;
   }
}
```

order edges by weight
(for sorting in Kruskal)

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

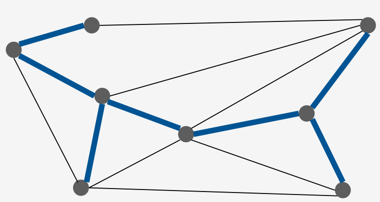
natural order
(for use in a symbol table)

- weighted graph API
- **cycles and cuts**
- Kruskal's algorithm
- Prim's algorithm
- advanced topics

Spanning tree

MST. Given connected graph G with positive edge weights, find a min weight set of edges that connects all of the vertices.

Def. A spanning tree of a graph G is a subgraph T that is connected and acyclic.



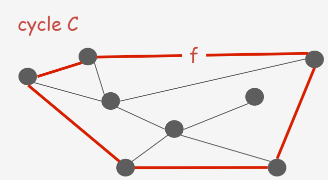
Property. MST of G is always a spanning tree.

Cycle and cut properties

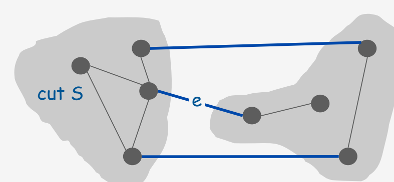
Simplifying assumption. All edge weights w_e are distinct.

Cycle property. Let C be any cycle, and let f be the max weight edge belonging to C. Then the MST does not contain f.

Cut property. Let S be any subset of vertices, and let e be the min weight edge with exactly one endpoint in S. Then the MST contains e.



f is not in the MST



e is in the MST

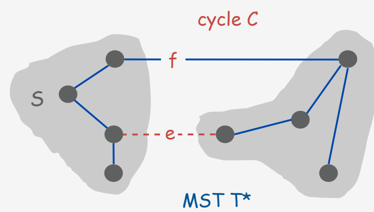
Cycle property: correctness proof

Simplifying assumption. All edge weights w_e are distinct.

Cycle property. Let C be any cycle, and let f be the max weight edge belonging to C. Then the MST T* does not contain f.

Pf. [by contradiction]

- Suppose f belongs to T*. Let's see what happens.
- Deleting f from T* disconnects T*. Let S be one side of the cut.
- Some other edge in C, say e, has exactly one endpoint in S.
- $T = T^* \cup \{ e \} - \{ f \}$ is also a spanning tree.
- Since $w_e < w_f$, weight(T) < weight(T*).
- Contradicts minimality of T*. ▪



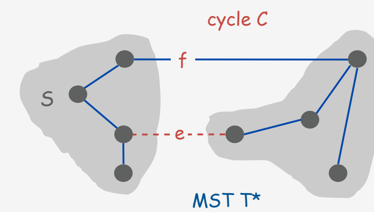
Cut property: correctness proof

Simplifying assumption. All edge weights w_e are distinct.

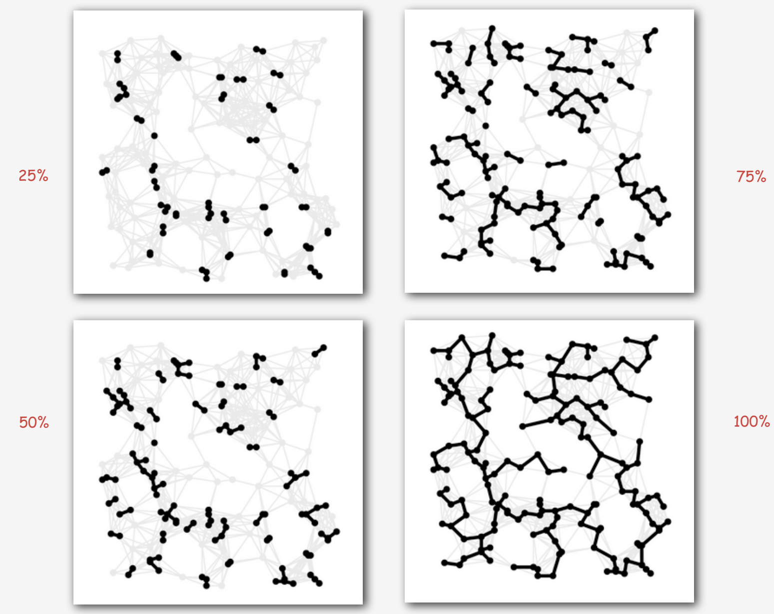
Cut property. Let S be any subset of vertices, and let e be the min weight edge with exactly one endpoint in S. Then the MST T* contains e.

Pf. [by contradiction]

- Suppose e does not belong to T*. Let's see what happens.
- Adding e to T* creates a cycle C in T*.
- Some other edge in C, say f, has exactly one endpoint in S.
- $T = T^* \cup \{ e \} - \{ f \}$ is also a spanning tree.
- Since $w_e < w_f$, weight(T) < weight(T*).
- Contradicts minimality of T*. ▪

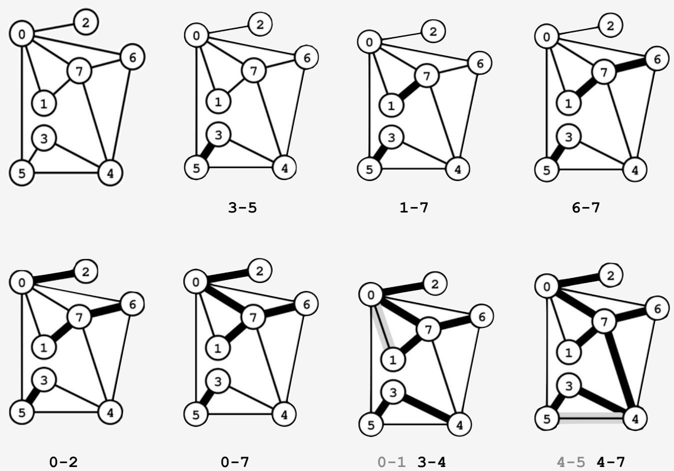


Kruskal's algorithm example



Kruskal's algorithm

Kruskal's algorithm. [Kruskal, 1956] Consider edges in ascending order of weight. Add the next edge to T unless doing so would create a cycle.



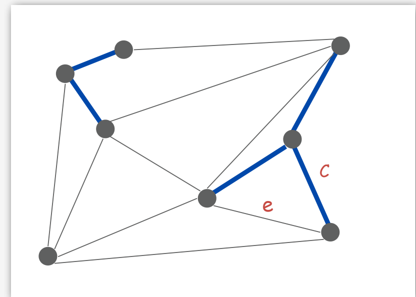
3-5	0.18
1-7	0.21
6-7	0.25
0-2	0.29
0-7	0.31
0-1	0.32
3-4	0.34
4-5	0.40
4-7	0.46
0-6	0.51
4-6	0.51
0-5	0.60

Kruskal's algorithm: correctness proof

Proposition. Kruskal's algorithm computes the MST.

Pf. [Case 1] Suppose that adding e to T creates a cycle C.

- Edge e is the max weight edge in C.
- Edge e is not in the MST (cycle property).

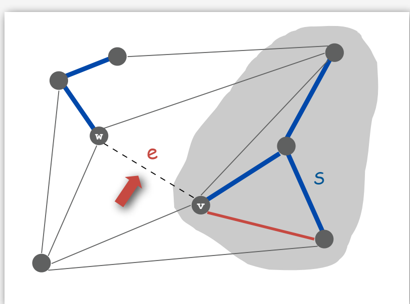


Kruskal's algorithm: correctness proof

Proposition. Kruskal's algorithm computes the MST.

Pf. [Case 2] Suppose that adding e = (v, w) to T does not create a cycle.

- Let S be the vertices in v's connected component.
- Vertex w is not in S.
- Edge e is the min weight edge with exactly one endpoint in S.
- Edge e is in the MST (cut property). ▪

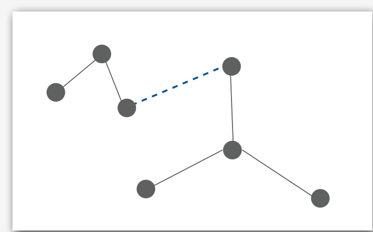


Kruskal implementation challenge

Problem. Check if adding an edge (v, w) to T creates a cycle.

How difficult?

- Intractable.
- O(E + V) time.
- O(V) time. ← run DFS from v, check if w is reachable (T has at most V-1 edges)
- O(log V) time.
- O(log* V) time. ← use the union-find data structure !
- Constant time.

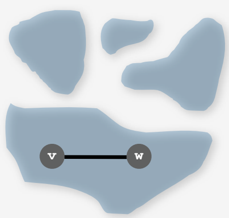


Kruskal's algorithm implementation

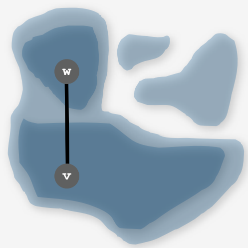
Problem. Check if adding an edge (v, w) to T creates a cycle.

Efficient solution. Use the union-find data structure.

- Maintain a set for each connected component in T.
- If v and w are in same component, then adding v-w creates 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

Kruskal's algorithm: Java implementation

```
public class Kruskal
{
   private SET<Edge> mst = new SET<Edge>();

   public Kruskal(WeightedGraph G)
   {
      Edge[] edges = G.edges();                              // get all edges in graph
      Arrays.sort(edges, new Edge.ByWeight());               // sort edges by weight

      UnionFind uf = new UnionFind(G.V());
      for (Edge e : edges)
      {
         int v = e.either(), w = e.other(v);
         if (!uf.find(v ,w))                                 // greedily add edges to MST
         {
            uf.unite(v, w);
            mst.add(edge);
         }
      }
   }

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

}
```


Kruskal's algorithm running time

Proposition. Kruskal's algorithm computes MST in O(E log V) time.

Pf.

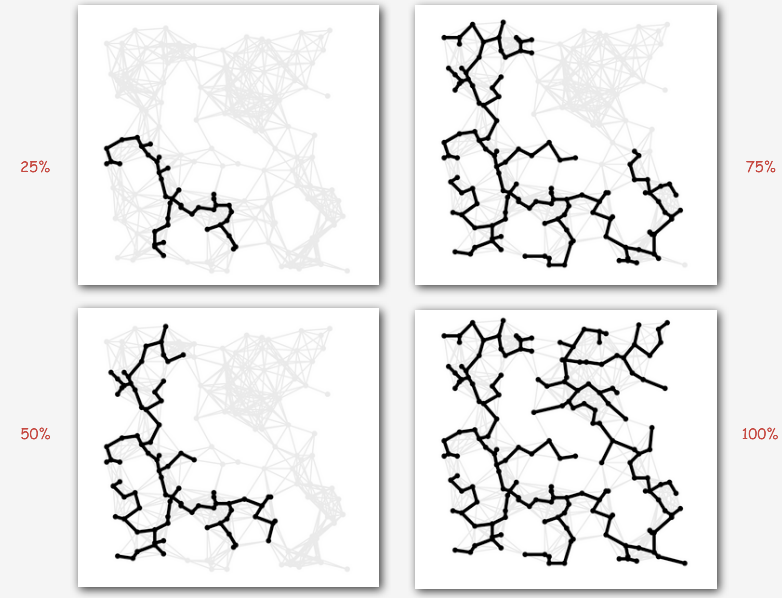
operation	frequency	time per op
sort	1	E log V
union	V	log* V †
find	E	log* V †

† amortized bound using weighted quick union with path compression

Remark. If edges are already sorted, time is proportional to E log* V.

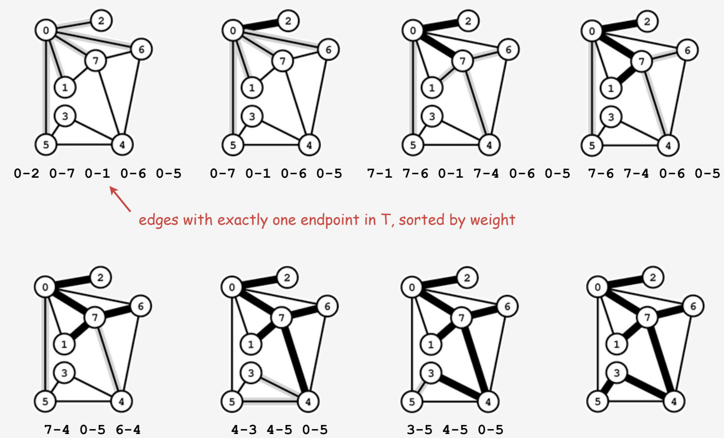
recall: log* V ≤ 5 in this universe

Prim's algorithm example



Prim's algorithm example

Prim's algorithm. [Jarník 1930, Dijkstra 1957, Prim 1959]
Start with vertex 0 and greedily grow tree T. At each step, add edge of min weight that has exactly one endpoint in T.

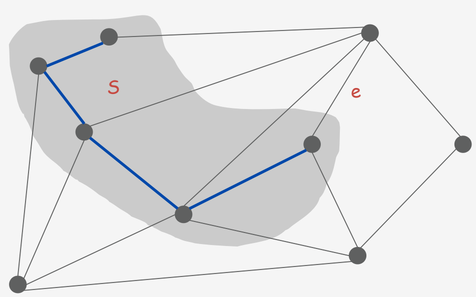


0-1	0.32
0-2	0.29
0-5	0.60
0-6	0.51
0-7	0.31
1-7	0.21
3-4	0.34
3-5	0.18
4-5	0.40
4-6	0.51
4-7	0.46
6-7	0.25

Prim's algorithm correctness proof

Proposition. Prim's algorithm computes the MST.
Pf.

- Let S be the subset of vertices in current tree T.
- Prim adds the min weight edge e with exactly one endpoint in S.
- Edge e is in the MST (cut property). ▪

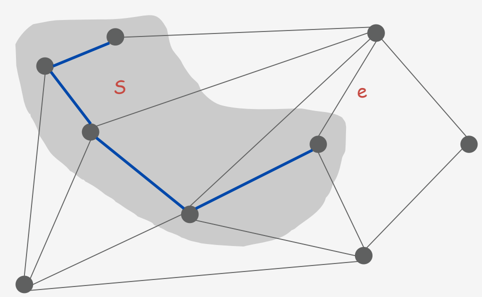


Prim implementation challenge

Problem. Find min weight edge with exactly one endpoint in S.

How difficult?

- Intractable.
- O(E) time. ← try all edges
- O(V) time.
- O(log V) time. ← use a priority queue !
- O(log* V) time.
- Constant time.



Prim's algorithm implementation

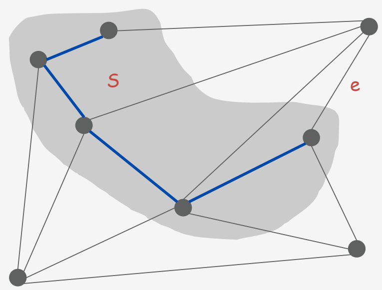
Problem. Find min weight edge with exactly one endpoint in S.

Efficient solution. Maintain a PQ of vertices connected by an edge to S.

- Delete min to determine next vertex v to add to S.
- Disregard v if already in S.
- Add to PQ any vertex brought closer to S by v.

Running time.

- log E steps per edge.
- E log E steps overall.



Key-value priority queue

Associate a value with each key in a priority queue.

```
public class MinPQplus<Key extends Comparable<Key>, Value>

              MinPQplus()                 create key-value priority queue
         void put(Key key, Value val)     put key-value pair into the PQ
        Value delMin()                    return value paired with
                                          minimal key and delete it
      boolean isEmpty()                   is the PQ empty?
```

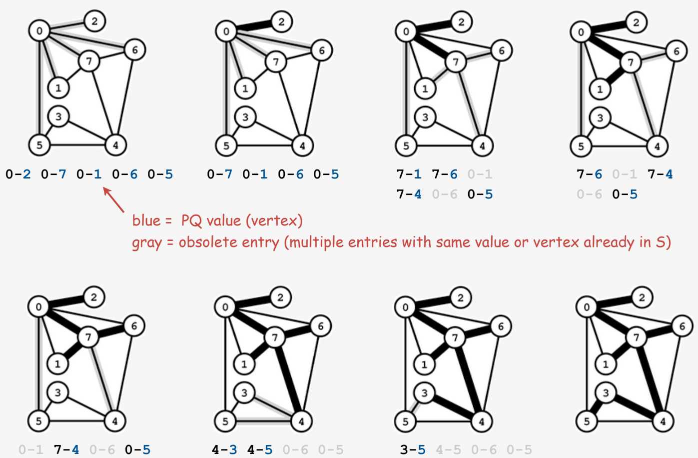
Implementation.

- Start with same code as standard heap-based PQ.
- Use a parallel array `vals[]` (value associated with `keys[i]` is `vals[i]`).
- Modify `exch()` to maintain parallel arrays (do exch in `vals[]`).
- Modify `delMin()` to return `Value`.

Prim's algorithm example: lazy implementation

Use PQ: key = edge weight, value = vertex.

(lazy version leaves some obsolete entries on the PQ)



0-1	0.32
0-2	0.29
0-5	0.60
0-6	0.51
0-7	0.31
1-7	0.21
3-4	0.34
3-5	0.18
4-5	0.40
4-6	0.51
4-7	0.46
6-7	0.25

Lazy implementation of Prim's algorithm

```
public class LazyPrim
{
   private boolean[] marked;  // vertices in MST
   private double[] dist;     // distance to MST
   private Edge[] pred;       // pred[v] is edge attach v to MST

   public LazyPrim(WeightedGraph G)
   {
      marked = new boolean[G.V()];
      pred   = new Edge[G.V()];
      dist   = new double[G.V()];
      for (int v = 0; v < G.V(); v++)
         dist[v] = Double.POSITIVE_INFINITY;
      prim(G, 0);
   }

   // See next slide for prim() implementation.
}
```

Lazy implementation of Prim's algorithm

```
private void prim(WeightedGraph G, int s)
{
   dist[s] = 0.0;
   marked[s] = true;

   MinPQplus<Double, Integer> pq;
   pq = new MinPQplus<Double, Integer>();
   pq.put(dist[s], s);

   while (!pq.isEmpty())
   {
      int v = pq.delMin();
      if (marked[v]) continue;
      marked[v] = true;
      for (Edge e : G.adj(v))
      {
         int w = e.other(v);
         if (!marked[w] && (dist[w] > e.weight()))
         {
            dist[w] = e.weight();
            pred[w] = e;
            pq.insert(dist[w], w);
         }
      }
   }
}
```

key-value PQ

ignore if already in MST

add to PQ any vertices brought closer to S by v

Prim's algorithm running time

Proposition. Prim's algorithm computes MST in O(E log V) time.

operation	frequency	time per op
delmin	V	V log V
insert	E	E log V

Priority queue with decrease-key

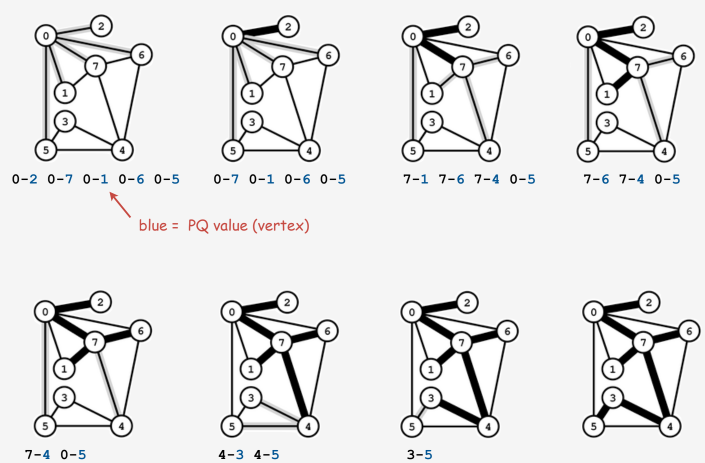
Indexed priority queue.

public class MinIndexPQ<Key extends Comparable<Key>, Integer>		
	`MinIndexPQ()`	*create key-value indexed priority queue*
`void`	`put(Key key, int v)`	*put key-value pair into the PQ*
`int`	`delMin()`	*return value paired with minimal key and delete it*
`boolean`	`isEmpty()`	*is the PQ empty?*
`boolean`	`contains(int v)`	*is there a key associated with value v?*
`void`	`decreaseKey(Key key, int v)`	*decrease the key associated with v to key*

Implementation. More complicated than `MinPQ`, see text.

Prim's algorithm example: eager implementation

Use IndexMinPQ: key = edge weight, value = vertex.
(eager version has at most one PQ entry per vertex)



0-1	0.32
0-2	0.29
0-5	0.60
0-6	0.51
0-7	0.31
1-7	0.21
3-4	0.34
3-5	0.18
4-5	0.40
4-6	0.51
4-7	0.46
6-7	0.25

Eager implementation of Prim's algorithm

Main benefit. Reduces PQ size guarantee from E to V.

- Not important for the huge sparse graphs found in practice.
- PQ size is far smaller in practice.
- Widely used, but practical utility is debatable.

Removing the distinct edge weight assumption

Simplifying assumption. All edge weights we are distinct.

Approach 1. Introduce tie-breaking rule for `compare()`.

```
public int compare(Edge e, Edge f)
{
   if (e.weight < f.weight) return -1;
   if (e.weight > f.weight) return +1;
   if (e.v < f.v) return -1;
   if (e.v > f.v) return +1;
   if (e.w < f.w) return -1;
   if (e.w > f.w) return +1;
   return 0;
}
```

`return e.compareTo(f);`

Approach 2. Prim and Kruskal still find MST if equal weights!
(only our proof of correctness fails)

- weighted graph API
- cycles and cuts
- Kruskal's algorithm
- Prim's algorithm
- **advanced topics**

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 α(V) log α(V)	Chazelle
2000	E α(V)	Chazelle
2002	optimal	Pettie-Ramachandran
20xx	E	???

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