Announcements

Exam Regrades
• Due by Wednesday’s lecture.

Teaching Experiment: Dynamic Deadlines (WordNet)
• Right now, WordNet is due at 11 PM on April 8th.
• Starting Tuesday at 11 PM:
  – Every submission that passes all Dropbox tests shortens the time limit by 30 minutes.
  – Maximum of 12 hours per day.
  – 3 hour grace period still applies.
• Email will be sent out every night at midnight with new deadline.
• I am lying.

“Dynamic Deadlines for Encouraging Earlier Participation on Assignments,” Garcia, Dan. SIGCSE 2013
http://db.grinnell.edu/sigcse/sigcse2013/Program/viewAcceptedSession.asp?sessionID=7220
4.3 Minimum Spanning Trees

- MST Basics, Kruskal, Prim
- Why Kruskal and Prim work
- Kruskal Implementation
- Prim Implementation
- Harder Problems
4.3 Minimum Spanning Trees

- MST Basics, Kruskal, Prim
- Why Kruskal and Prim work
- Kruskal Implementation
- Prim Implementation
- Harder Problems
Minimum spanning tree and edge weighted graphs

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

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

**Goal.** Find a min weight spanning tree.
**Minimum spanning tree**

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

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

**Goal.** Find a min weight spanning tree.

**Brute force.** Try all spanning trees? There are $\sim V^V$ of them.

![Spanning tree](image)

spanning tree $T$: cost = 50 = 4 + 6 + 8 + 5 + 11 + 9 + 7
Textbook Convention. Edges are drawn with length proportional to weight.

Constraint. This convention constrains the set of possible graphs.

Allowable graph
Can be drawn with length = weight

Allowable graph
Cannot be drawn with length = weight
**Drawing convention**

**Textbook Convention #2.** Edges are straight lines and never cross. **Constraint.** This convention constrains the set of possible graphs.

[Graph Image]


Textbook graphs typically avoid crossings because they’re hard to read.
Textbook Convention #2. Edges are straight lines and never cross.
Constraint. This convention constrains the set of possible graphs.

Q: How hard is it to determine whether a graph can be redrawn in a plane?
Kruskal's algorithm demo

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

![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

*an edge-weighted graph*
Consider edges in ascending order of weight.
  • Add next edge to tree $T$ unless doing so would create a cycle.
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 demo

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

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

Q: Which edge comes next?
A. 4–5 [127350]
B. 4–0 [127809]
C. 2–0 [127963]
Consider edges in ascending order of weight.

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

Q: Which edge comes next?
C. 2–0  [127963]

Kruskal's algorithm demo
Kruskal's algorithm demo

Consider edges in ascending order of weight.

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

in MST

<table>
<thead>
<tr>
<th>Edge</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-7</td>
<td>0.16</td>
</tr>
<tr>
<td>2-3</td>
<td>0.17</td>
</tr>
<tr>
<td>1-7</td>
<td>0.19</td>
</tr>
<tr>
<td>0-2</td>
<td>0.26</td>
</tr>
<tr>
<td>5-7</td>
<td>0.28</td>
</tr>
</tbody>
</table>

does not create a cycle
Consider edges in ascending order of weight.
- Add next edge to tree $T$ unless doing so would create a cycle.

Q: Which edge comes next?
Consider edges in ascending order of weight.

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

<table>
<thead>
<tr>
<th>Edge</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-7</td>
<td>0.16</td>
</tr>
<tr>
<td>2-3</td>
<td>0.17</td>
</tr>
<tr>
<td>1-7</td>
<td>0.19</td>
</tr>
<tr>
<td>0-2</td>
<td>0.26</td>
</tr>
<tr>
<td>5-7</td>
<td>0.28</td>
</tr>
<tr>
<td>1-3</td>
<td>0.29</td>
</tr>
<tr>
<td>1-5</td>
<td>0.32</td>
</tr>
<tr>
<td>2-7</td>
<td>0.34</td>
</tr>
<tr>
<td>4-5</td>
<td>0.35</td>
</tr>
<tr>
<td>1-2</td>
<td>0.36</td>
</tr>
<tr>
<td>4-7</td>
<td>0.37</td>
</tr>
<tr>
<td>0-4</td>
<td>0.38</td>
</tr>
<tr>
<td>6-2</td>
<td>0.40</td>
</tr>
<tr>
<td>3-6</td>
<td>0.52</td>
</tr>
<tr>
<td>6-0</td>
<td>0.58</td>
</tr>
<tr>
<td>6-4</td>
<td>0.93</td>
</tr>
</tbody>
</table>
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.
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.
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
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.

Q: Which edge is added next to the MST?
A. 2–3  [149931]
B. 1–7  [149933]
C. 6–0  [149934]
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.

Q: Which edge is added next to the MST?
B. 1–7
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
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
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

Edges with exactly one endpoint in $T$ (sorted by weight):

- 0–2: 0.26
- 5–7: 0.28
- 1–3: 0.29
- 1–5: 0.32
- 2–7: 0.34
- 1–2: 0.36
- 4–7: 0.37
- 0–4: 0.38
- 6–0: 0.58
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
Q: What is the weight of the MST?
A. 45  [540123]  D. 60  [520105]
B. 50  [540124]  E. 65  [370101]
C. 55  [520104]
Q: What is the weight of the MST?
C. 55
4.3 Minimum Spanning Trees

- What
- Why Kruskal and Prim work
- How - Kruskal’s (data structures)
- How - Prim’s (data structures)
- context
**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. $\blacksquare$
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.

Q: How many distinct cuts are there for the graph above?

A. 7 [229703] D. 16 [229801]
B. 14 [229704] E. 30 [229802]
C. 15 [229705] F. 32 [229803]

Extra: How does the number of distinct cuts grow with V for a general graph?
**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.

---

Q: How many distinct cuts are there for the graph above?  C. 15

Choice of cut is basically a 5 bit binary number: 32 total choices.

Two of these involve an empty set. Total -> 30.

Half are redundant (e.g. 00100 is the same thing as 11011). Total -> 15.
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.

Q: How many distinct cuts are there for the graph above?  
C. 15

Extra: How does the number of distinct cuts grow with $V$ for a general graph?  
$2^{V-1}-1$
226 MST algorithms

Fundamental Idea

- Our algorithms grow an MSSapling until it becomes a full MST.
- The MSSapling starts as $V$ disjoint components.
- Each step of the algorithm connects two MSSapling components.
  - Given 2 cuts, always connect by the smallest connecting edge.
  - This smallest edge belongs to MST by cut property.
  - Each connection reduces number of components by 1.
- Once the MSSapling has 1 component, it is the MST.
Proposition. Once the MSSapling has 1 component, it is the MST.

Pf.

- Any edge in the MSSapling is in the MST (via cut property).
- Fewer than $V - 1$ black edges $\Rightarrow$ There is more than one component.
4.3 Minimum Spanning Trees

- MST Basics, Kruskal, Prim
- Why Kruskal and Prim work
- Kruskal Implementation
- Prim Implementation
- Harder Problems
226 MST algorithms

Fundamental Idea

- Our algorithms grow an MSSapling until it becomes a full MST.
- The MSSapling starts as V disjoint components.
- Each step of the algorithm connects two MSSapling components.
  - Given 2 cuts, always connect by the smallest connecting edge.
  - This smallest edge belongs to MST by cut property.
  - Each connection reduces number of components by 1.
- Once the MSSapling has 1 component, it is the MST.

Kruskal’s and Prim’s

- Specific ways to pick our two MSSapling components.
Kruskal's algorithm demo

Consider edges in ascending order of weight.

- Add next edge to tree $T$ unless doing so would create a cycle (cycle equivalent to having a black crossing edge).

\[
\begin{array}{llllll}
0-7 & 0.16 \\
2-3 & 0.17 \\
1-7 & 0.19 \\
0-2 & 0.26 \\
5-7 & 0.28 \\
1-3 & 0.29 \\
1-5 & 0.32 \\
2-7 & 0.34 \\
4-5 & 0.35 \\
1-2 & 0.36 \\
4-7 & 0.37 \\
0-4 & 0.38 \\
6-2 & 0.40 \\
3-6 & 0.52 \\
6-0 & 0.58 \\
6-4 & 0.93 \\
\end{array}
\]
Consider edges in ascending order of weight.

- Add next edge to tree $T$ unless doing so would create a cycle (cycle equivalent to having a black crossing edge).

In MST

0-7 0.16

does not create a cycle
Kruskal's algorithm demo

Consider edges in ascending order of weight.

- Add next edge to tree $T$ unless doing so would create a cycle (cycle equivalent to having a black crossing edge).

0-7 0.16
2-3 0.17

in MST

does not create a cycle
Kruskal's algorithm demo

Consider edges in ascending order of weight.

- Add next edge to tree $T$ unless doing so would create a cycle (cycle equivalent to having a black crossing edge).

Q: How many components are there?

A. 1 [219103]  
B. 2 [219104]  
C. 3 [602201]  
D. 4 [602202]  
E. 5 [602302]
Consider edges in ascending order of weight.
- Add next edge to tree $T$ unless doing so would create a cycle (cycle equivalent to having a black crossing edge).

Q: How many components are there?
5
Kruskal's algorithm demo

Consider edges in ascending order of weight.
- Add next edge to tree $T$ unless doing so would create a cycle (cycle equivalent to having a black crossing edge).

![Diagram of Kruskal's algorithm](image)

<table>
<thead>
<tr>
<th>Edge</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-7</td>
<td>0.16</td>
</tr>
<tr>
<td>2-3</td>
<td>0.17</td>
</tr>
<tr>
<td>1-7</td>
<td>0.19</td>
</tr>
<tr>
<td>0-2</td>
<td>0.26</td>
</tr>
</tbody>
</table>

This edge does not create a cycle.
4.3 Minimum Spanning Trees

- MST Basics, Kruskal, Prim
- Why Kruskal and Prim work
- Kruskal Implementation
- Prim Implementation
- Harder Problems
Kruskal's algorithm implementation

Kruskal’s algorithm

• Given a collection of all the edges in a graph:
  – Take out the minimum edge.
  – Add this edge to the MST as long as no cycle is created.

Challenges.

• What is the smallest weight edge that has not been considered?
• Would adding edge $v \rightarrow w$ to tree $T$ create a cycle?

In Groups of 3.

• Choose appropriate data structures and algorithms to solve these two subproblems.
• Extra task: How much time does your scheme take to perform each task above? To build the entire MST?

private Queue<Edge> mst;
Debrief - which data structures should we use?

Challenges.

• What is the smallest weight edge that has not been considered?
  - MinPQ<Edge> - compared by weight
  - Edge[] - sorted (comparing by weight)

• Would adding edge $v \rightarrow w$ to tree $T$ create a cycle?
  - [array that tracks connected components], a.k.a. Union find
  - DFS based graph search every time [very slow]
  - DYNAMIC CONNECTIVITY - UF is fast, DFS is slow

• Calls which interact with edges:
  - int $v = e$.either();
  - int $w = e$.other($v$);
  - mst.enqueue($e$);

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

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

        for (Edge e : G.edges())
            pq.insert(e);

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

    public Iterable<Edge> edges()
    {
        return mst;
    }
}
public class KruskalMST
{
    private Queue<Edge> mst = new Queue<Edge>();

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

        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: Java implementation – (book implementation)

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

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

        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;
    }
}
```

<table>
<thead>
<tr>
<th>operation</th>
<th>frequency</th>
<th>time per op</th>
</tr>
</thead>
<tbody>
<tr>
<td>build pq</td>
<td>1</td>
<td>$E \lg E$ could be $E$</td>
</tr>
<tr>
<td>delete-min</td>
<td>$E$</td>
<td>$\lg E$</td>
</tr>
<tr>
<td>union</td>
<td>$V$</td>
<td>$\log^* V$</td>
</tr>
<tr>
<td>connected</td>
<td>$E$</td>
<td>$\log^* V$</td>
</tr>
</tbody>
</table>
Kruskal's algorithm: running time

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

**Pf.**

<table>
<thead>
<tr>
<th>operation</th>
<th>frequency</th>
<th>time per op</th>
</tr>
</thead>
<tbody>
<tr>
<td>build pq</td>
<td>1</td>
<td>$E \log E$</td>
</tr>
<tr>
<td>delete-min</td>
<td>$E$</td>
<td>$\log E$</td>
</tr>
<tr>
<td>union</td>
<td>$V$</td>
<td>$\log^* V$ †</td>
</tr>
<tr>
<td>connected</td>
<td>$E$</td>
<td>$\log^* V$ †</td>
</tr>
</tbody>
</table>

† amortized bound using weighted quick union with path compression

How do we get time $E$?

Construct array of edges and pass to MinPQ constructor.

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

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

- MST Basics, Kruskal, Prim
- Why Kruskal and Prim work
- Kruskal Implementation
- Prim Implementation
- Harder Problems
Prim's algorithm

- Starting with vertex 0.
- Add to $T$ the min weight edge with exactly one endpoint in $T$.
- Repeat until $V - 1$ edges.
Three flavors of Prim’s

Prim’s algorithm

- Intuitive - easy to discover
- Lazy - easy to code version of human
- Eager - optimized version of human
Prim’s algorithm implementation

Prim’s algorithm

• In Kruskal’s, picked MSSaplings by tracking all of the edges in the entire graph and selecting the smallest one.
• In Prim’s, what is the most natural thing to track?
Prim’s algorithm implementation

Prim’s algorithm
- In Kruskal’s, picked MSSaplings by tracking all of the edges in the entire graph and selecting the smallest one.
- In Prim’s, what is the most natural thing to track?
  - All outbound edges from core of the MSSapling.
Prim’s algorithm implementation

Intuitive Prim’s algorithm

- Given a collection $C$ of all edges outbound from core:
  - Add $C$’s minimum edge $v$-$w$ to the MSSapling.
Prim’s algorithm implementation

Intuitive Prim’s algorithm

• Given a collection $C$ of all edges outbound from core:
  – Add $C$’s minimum edge $v-w$ to the MSSapling.
Intuitive Prim’s algorithm

- Given a collection C of all edges outbound from core:
  - Add C’s minimum edge v-w to the MSSapling.
  - Add to C any outward pointing edges from w.
Intuitive Prim’s algorithm

- Given a collection C of all edges outbound from core:
  - Add C’s minimum edge v-w to the MSSapling.
  - Add to C any outward pointing edges from w.
Intuitive Prim’s algorithm

- Given a collection C of all edges outbound from core:
  - Add C’s minimum edge v-w to the MSSapling.
  - Add to C any outward pointing edges from w.
  - Remove from C any edges v-x, where x is also in the core.
Intuitive Prim’s algorithm

- Given a collection $C$ of all edges outbound from core:
  - Add $C$’s minimum edge $v$-$w$ to the MSSapling.
  - Add to $C$ any outward pointing edges from $w$.
  - Remove from $C$ any edges $v$-$x$, where $x$ is also in the core.
Intuitive Prim’s algorithm

- Given a collection $C$ of all edges outbound from core:
  - Add $C$’s minimum edge $v-w$ to the MSSapling.
  - Add to $C$ any outward pointing edges from $w$.
  - Remove from $C$ any edges $v-x$, where $x$ is also in the core.

• Turns out this algorithm is a pain to implement (not in textbook).
Prim’s algorithm implementation

Lazy Prim’s algorithm

- Given a collection $C$ of all edges outbound from core:
  - Add $C$’s minimum edge $v$-$w$ to the MSSapling
    - If it doesn’t create a cycle, otherwise delete $v$-$w$.
  - Add to $C$ any outward pointing edges from $w$.
  - Remove from $C$ any edges $v$-$x$, where $x$ is also in the core.

- Much easier to implement.
Prim's algorithm (lazy) 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
Lazy Prim’s algorithm

- Given a collection $C$ of all edges outbound from core:
  - Add $C$’s minimum edge $v$-$w$ to the MSSapling
    - If it doesn’t create a cycle, otherwise delete $v$-$w$.
  - Add to $C$ any outward pointing edges from $w$.
  - Remove from $C$ any edges $v$-$x$, where $x$ is also in the core.
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.**

<table>
<thead>
<tr>
<th>operation</th>
<th>frequency</th>
<th>binary heap</th>
</tr>
</thead>
<tbody>
<tr>
<td>delete min</td>
<td>$E$</td>
<td>$\log E$</td>
</tr>
<tr>
<td>insert</td>
<td>$E$</td>
<td>$\log E$</td>
</tr>
</tbody>
</table>
Prim's algorithm: lazy implementation

```java
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 v or w to tree
Prim's algorithm: lazy implementation

```java
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
Eager Prim’s algorithm

- Given a collection $C$ of all edges outbound from vertices adjacent to core:
  - Add $C$’s minimum edge $v-w$ to the MSSapling.
  - Remove vertex $w$ that is closest to core, and add edge $?-w$.
  - Add to $C$ any outward pointing edges from $w$.
  - Remove from $C$ any edges $v-x$, where $x$ is also in the core.
  - Update distance to each vertex adjacent to core.
Prim's algorithm (eager) 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.
Prim's algorithm (eager) 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.

IndexMinPQ<Double> pq = new IndexMinPQ<Double>(G.V());
pq.insert(7, 0.16); pq.insert(2, 0.26); ...
Prim's algorithm (eager) 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

- $0-7$  
- $1-7$  
- $0-2$

pq.change(3, 0.17); pq.change(6, 0.4);
Prim's algorithm (eager) 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
Eager Prim's algorithm: which priority queue?

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

<table>
<thead>
<tr>
<th>PQ implementation</th>
<th>insert</th>
<th>delete-min</th>
<th>decrease-key</th>
<th>total</th>
</tr>
</thead>
<tbody>
<tr>
<td>unordered array</td>
<td>1</td>
<td>( V )</td>
<td>1</td>
<td>( V^2 )</td>
</tr>
<tr>
<td>binary heap</td>
<td>( \log V )</td>
<td>( \log V )</td>
<td>( \log V )</td>
<td>( E \log V )</td>
</tr>
<tr>
<td>d–way heap ( (\text{Johnson 1975}) )</td>
<td>( d \log_d V )</td>
<td>( d \log_d V )</td>
<td>( \log_d V )</td>
<td>( E \log_{E/V} V )</td>
</tr>
<tr>
<td>Fibonacci heap ( (\text{Fredman–Tarjan 1984}) )</td>
<td>( 1 \uparrow )</td>
<td>( \log V \uparrow )</td>
<td>( 1 \uparrow )</td>
<td>( E + V \log V )</td>
</tr>
</tbody>
</table>

\( \uparrow \) 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.
4.3 Minimum Spanning Trees

› MST Basics, Kruskal, Prim
› Why Kruskal and Prim work
› Kruskal Implementation
› Prim Implementation
› Harder Problems
B level problems

Suppose the that the MST of the graph below contains the edges with weights x, y, and z.

- True or false: The minimum weight edge from every node must be part of the MST.
- List the weights of the other edges in the MST:
  
  10  ____  ____  ____  ____  ____  ____

- What are the possible values for the weights of x, y, and z?
True or false: The minimum weight edge from every node must be part of the MST - true by cut property!

List the weights of the other edges in the MST:

\[ 10, 30, 50, 20, 40, 100 \]

What are the possible values for the weights of \( x, y, \) and \( z? \)
B level problems

Suppose that the MST of the graph below contains the edges with weights $x$, $y$, and $z$.

- True or false: The minimum weight edge from every node must be part of the MST - true by cut property!
- List the weights of the other edges in the MST:
  
  __10__ __30__ __50__ __20__ __40__ 100

- What are the possible values for the weights of $x$, $y$, and $z$?
  - $x \leq 110$, $y \leq ?$
Suppose the that the MST of the graph below contains the edges with weights $x$, $y$, and $z$.

- True or false: The minimum weight edge from every node must be part of the MST - true by cut property!
- List the weights of the other edges in the MST: 
  
  10  30  50  20  40  100

- What are the possible values for the weights of $x$, $y$, and $z$?
  - $x \leq 110$, $y \leq 60$, 
  - $z \leq 20$,
  - $z \leq 100$,
  - $z \leq 150$,
True or false: The minimum weight edge from every node must be part of the MST - true by cut property!

List the weights of the other edges in the MST:

10 30 50 20 40 100

What are the possible values for the weights of x, y, and z?
- x <= 110, y <= 60, z <= 80
A level problems

- Suppose you know the MST of G. Now a new edge v-w of weight \( c \) is added to G, resulting in a new graph G’. Design a O(V) algorithm to determine if the MST for G is also an MST for G’.

- Bonus: Given a graph G and its MST, if we remove an edge from G that is part of the MST, how do we find the new MST in O(E) time?
A level problems

- Suppose you know the MST of $G$. Now a new edge $v-w$ of weight $c$ is added to $G$, resulting in a new graph $G'$. Design a $O(V)$ algorithm to determine if the MST for $G$ is also an MST for $G'$.

- Bonus: Given a graph $G$ and its MST, if we remove an edge from $G$ that is part of the MST, how do we find the new MST in $O(E)$ time?

Hint: Consider the blue path.
A level problems

• Suppose you know the MST of G. Now a new edge v-w of weight c is added to G, resulting in a new graph G’. Design a O(V) algorithm to determine if the MST for G is also an MST for G’.

• If any edge on the blue path is longer than c:
  – Replace that edge with c - you get a new MST with shorter distance.
• If every edge on the blue path is shorter than c:
  – Then we know original MST was the best.
• Finding the blue path: Run DFS from one of c’s vertices to the other, only taking steps along the MST.
A level problems

• Given a graph $G$ and its MST, if we remove an edge from $G$ that is part of the MST, how do we find the new MST in $O(E)$ time?