Today’s announcements:

MP7 available. Due 12/9, 11:59p.
Code challenge #2, 12/10, 9p, Siebel 0224. (next week)
Final exam: 12/15, 7-10p, locations TBA

email c-heeren@illinois.edu asap w conflict in subject line

Minimum Spanning Tree Algorithms:

• Input: connected, undirected graph G with unconstrained edge weights
• Output: a graph G’ with the following characteristics -
  • G’ is a spanning subgraph of G
  • G’ is connected and acyclic (a tree)
  • G’ has minimal total weight among all such spanning trees -

```graph

__________________________
Kruskal’s Algorithm

(a,d)  
(e,h)  
(f,g)  
(a,b)  
(b,d)  
(g,e)  
(g,h)  
(e,c)  
(c,h)  
(e,f)  
(f,c)  
(d,e)  
(b,c)  
(c,d)  
(a,f)  
(d,f)
1. Initialize graph $T$ whose purpose is to be our output. Let it consist of all $n$ vertices and no edges.

2. Initialize a disjoint sets structure where each vertex is represented by a set.

3. RemoveMin from $PQ$. If that edge connects 2 vertices from different sets, add the edge to $T$ and take Union of the vertices' two sets, otherwise do nothing. Repeat until ______ edges are added to $T$. 
Algorithm KruskalMST\( (G) \)

\[
\text{disjointSets forest;}
\]
\[
\text{for each vertex } v \text{ in } V \text{ do}
\]
\[
\quad \text{forest.makeSet}(v);
\]

\[
\text{priorityQueue } Q;\]

Insert edges into \( Q \), keyed by weights

\[
\text{graph } T = (V,E) \text{ with } E = \emptyset;
\]

\[
\text{while } T \text{ has fewer than } n-1 \text{ edges do}
\]
\[
\quad \text{edge } e = Q.\text{removeMin}()
\]
Let \( u, v \) be the endpoints of \( e \)
\[
\quad \text{if forest.find}(v) \neq \text{forest.find}(u) \text{ then}
\]
Add edge \( e \) to \( E \)
\[
\quad \quad \text{forest.smartUnion} (\text{forest.find}(v), \text{forest.find}(u))
\]

\[
\text{return } T
\]

<table>
<thead>
<tr>
<th>Priority Queue:</th>
<th>Heap</th>
<th>Sorted Array</th>
</tr>
</thead>
<tbody>
<tr>
<td>To build</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Each removeMin</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Algorithm \textit{KruskalMST}(G)

\begin{itemize}
  \item \texttt{disjointSets forest;}
  \item \texttt{for each vertex }\texttt{v} \texttt{in }\texttt{V} \texttt{do}
  \item \hspace{2em} \texttt{forest.makeSet(v);} \\
  \item \texttt{priorityQueue Q;}
  \item \hspace{2em} \texttt{Insert edges into }Q, \texttt{keyed by weights}
  \item \texttt{graph }T = (V,E) \texttt{with }E = \emptyset; \\
  \item \texttt{while }T \texttt{has fewer than }n-1 \texttt{edges do}
  \item \hspace{2em} \texttt{edge }e = Q.\texttt{removeMin()}
  \item \hspace{2em} \texttt{Let }u, v \texttt{be the endpoints of }e
  \item \hspace{2em} \texttt{if }\texttt{forest.find(v)} \neq \texttt{forest.find(u)} \texttt{then}
  \item \hspace{3em} \texttt{Add edge }e \texttt{to }E
  \item \hspace{3em} \texttt{forest.smartUnion}
  \item \hspace{3em} \hspace{2em} \texttt{(forest.find(v),forest.find(u))}
  \item \hspace{2em} \texttt{return }T
\end{itemize}

<table>
<thead>
<tr>
<th>Priority Queue:</th>
<th>Total Running time:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heap</td>
<td></td>
</tr>
<tr>
<td>Sorted Array</td>
<td></td>
</tr>
</tbody>
</table>
Prim’s algorithms (1957) is based on the Partition Property:

Consider a partition of the vertices of $G$ into subsets $U$ and $V$.

Let $e$ be an edge of minimum weight across the partition.

Then $e$ is part of some minimum spanning tree.

Proof: See cs374
MST - minimum total weight spanning tree

Theorem suggests an algorithm...
Example of Prim’s algorithm -

Initialize structure:
1. For all $v$, $d[v] = \text{“infinity”}$, $p[v] = \text{null}$
2. Initialize source: $d[s] = 0$
3. Initialize priority (min) queue
4. Initialize set of labeled vertices to $\emptyset$. 
Example of Prim’s algorithm -

Initialize structure:
1. For all v, d[v] = “infinity”, p[v] = null
2. Initialize source: d[s] = 0
3. Initialize priority (min) queue
4. Initialize set of labeled vertices to ∅.

Repeat these steps n times:
• Find & remove minimum d[] unlabelled vertex: v
• Label vertex v
• For all unlabelled neighbors w of v, If cost(v,w) < d[w]
  d[w] = cost(v,w)
  p[w] = v
Prim’s Algorithm (undirected graph with unconstrained edge weights):

Initialize structure:
1. For all v, d[v] = “infinity”, p[v] = null
2. Initialize source: d[s] = 0
3. Initialize priority (min) queue
4. Initialize set of labeled vertices to Ø.

Repeat these steps n times:
• Remove minimum d[] unlabeled vertex: v
• Label vertex v (set a flag)
• For all unlabeled neighbors w of v,
  If cost(v,w) < d[w]
    d[w] = cost(v,w)
    p[w] = v
Prim’s Algorithm (undirected graph with unconstrained edge weights):

Initialize structure:
1. For all v, d[v] = “infinity”, p[v] = null
2. Initialize source: d[s] = 0
3. Initialize priority (min) queue
4. Initialize set of labeled vertices to ∅.

Repeat these steps n times:
• Remove minimum d[] unlabeled vertex: v
• Label vertex v (set a flag)
• For all unlabeled neighbors w of v,
  If cost(v,w) < d[w]
    d[w] = cost(v,w)
    p[w] = v

Which is best?

Depends on density of the graph:

Sparse
Dense