lecture28: Minimum Spanning Trees

Sean Massung

Largely based on slides by Cinda Heeren
CS 225 UIUC

29th July, 2013
Outline

1. Announcements
2. MSTs
3. Kruskal’s
4. Prim’s
Announcements

- mp7 due tonight!
- lab_graphs out tomorrow, due Thursday, 8/1
- final exam this Friday (8/2), 10:30am-12:30pm in this room
Outline

1. Announcements
2. MSTs
3. Kruskal's
4. Prim's
MST definition

- A **spanning tree** $T$ of a graph $G$ connects all vertices together with no cycles (hence a tree).
- If we define the weight of a spanning tree to be the sum of its edge weights, a *minimal* spanning tree has a minimal weight for a given graph.
- We’ll investigate two algorithms to find the MST of a graph: Kruskal’s algorithm and Prim’s algorithm.
- We’ll assume both these algorithms run on weighted, undirected graphs (it’s easy enough to modify them to work on directed graphs).
Example graph
Outline

1. Announcements
2. MSTs
3. Kruskal’s
4. Prim’s
Kruskal’s Pseudocode

1. Initialize our output graph, \( T = (V', E') \), \( V' = V \), \( E' = \{\} \)
2. Initialize a disjoint sets structure \( S \), where each vertex represents a set (all in own set to begin with)
3. Initialize a priority queue, \( P \), holding all the edges in the original graph
4. \( e = P.removeMin() \)
   - If \( e \) connects two vertices from different sets, add \( e \) to \( E' \), and union the two vertices from \( e \) in \( S \)
   - If \( e \) connects two vertices from the same set, do nothing (this would create a cycle)
5. Repeat step 4 until \(|E'| = n - 1\)
Running Kruskal’s
Kruskal’s Analysis

Here’s the outline of Kruskal’s:

- Initialize disjoint sets
- Initialize priority queue
- Call \texttt{removeMin} \( n - 1 \) times:
  - Call \texttt{union} if necessary

<table>
<thead>
<tr>
<th></th>
<th>Binary Heap</th>
<th>Sorted Array</th>
</tr>
</thead>
<tbody>
<tr>
<td>build each \texttt{removeMin}</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Total time using a binary heap:

Total time using a sorted array:
Outline

1. Announcements
2. MSTs
3. Kruskal's
4. Prim's
Prim’s Pseudocode

1. Initialize a priority queue $P$, to hold vertices based on a “cost” (initially all $\infty$); change a start vertex’s cost to 0
2. Initialize an empty dictionary $D$: vertex $\rightarrow$ parent
3. $v = P.removeMin()$, label $v$ as visited
   $\forall w \in adjacent(v)$:
   - if $w$ is unvisited and $cost(v, w) < P[w]$
     - $P.decreaseKey(w, cost(v, w))$
     - $D[w] = v$
4. Repeat step 3 for $n$ times (until $P$ is empty), then create $T$ by using the parents in $D$
DecreaseKey

- decreaseKey is a possible priority queue ADT function
- It assumes we have direct access to objects inside the PQ
- How would you write decreaseKey for a binary heap, given an index?
- This means the running time for decreaseKey in a binary heap is...
Running Prim’s
Prim’s Analysis

Here’s an outline of Prim’s (assuming \( D \) is a hash table):

- Initialize priority queue
- Call removeMin \( n \) times:
  - Call getAdjacent
  - Call decreaseKey if necessary

<table>
<thead>
<tr>
<th></th>
<th>Binary Heap</th>
<th>Fibonacci Heap</th>
</tr>
</thead>
<tbody>
<tr>
<td>build</td>
<td></td>
<td>( O(n) )</td>
</tr>
<tr>
<td>each removeMin</td>
<td></td>
<td>( O(\log n)^* )</td>
</tr>
<tr>
<td>each decreaseKey</td>
<td></td>
<td>( O(1)^* )</td>
</tr>
</tbody>
</table>

But how many times do we call decreaseKey?

\[
\sum_{v \in V} \deg(v) = 2m = O(m)
\]
Prim’s Analysis II

We call build once, removeMin $n$ times, getAdjacent $n$ times, and decreaseKey $2m$ times.

Using all this information, we can build the final running time for Prim’s based on the following:

$$O(build + n \cdot (removeMin + getAdjacent) + m \cdot decreaseKey)$$

You should be able to plug in running times (and simplify) using any PQ structure and any graph implementation! Also remember for connected graphs:

$$n - 1 \leq m \leq n^2$$
Example running times

Find the running time for Prim’s using...

- adjacency list and binary heap
- adjacency matrix and binary heap
- adjacency matrix and Fibonacci heap
- adjacency list and sorted array

What’s the best running time for Prim’s you can build?