MA2C03 - DISCRETE MATHEMATICS - TUTORIAL NOTES

Brian Tyrrell

23/02/2017

Summary: This week we spoke about Kruskal's algorithm and Prim's algorithm for constructing minimal spanning trees. Any questions concerning the upcoming homework and countability were also answered.

1 Kruskal's Algorithm

Suppose you're given a connected graph with some *cost* associated to every edge - something like a 'shipping cost' to go from one vertex to another. You want to find a spanning tree T in this graph that is *minimal*, in the sense that if S is another spanning tree of the graph, the total cost of S (the sum of the costs of all the edges) is greater than or equal to the total cost of T.

Kruskal's algorithm allows to to build a minimal spanning tree from a connected graph (V, E), with a cost function $c : E \to \mathbb{R}$ on its edges.

So, what's the algorithm?

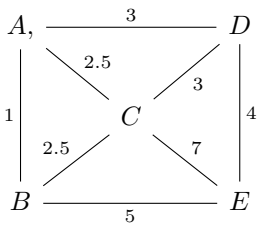
(1) Start with $(V, \emptyset)$; the subgraph consisting of the vertices and no edges.

(2) List the edges in a *queue* from smallest cost to biggest cost. More technically, in a *non-decreasing* order - some edges may have the same cost, and it doesn't matter in which order in the queue we place them. What matters is we are consistent in ordering smallest to biggest.

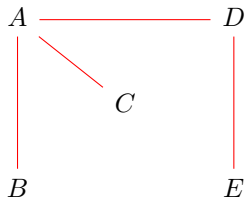
(3) Take an edge from the front of the queue, and decide whether or not adding it to the current subgraph will result in a circuit.
IF adding this edge creates a circuit, DISCARD the edge from the queue.
ELSE, ADD the edge to the subgraph.

(4) Repeat until you run out of edges to add (the queue is emptied).

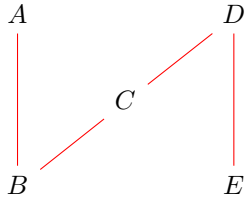
Example 1.1. Consider the following example, worked out in the tutorial. The following graph



has a minimal spanning tree



This minimal spanning tree isn't unique; this is also an acceptable solution:



Sidenote: these two graphs are **not** isomorphic. For example, $\deg A = 3$ in the top graph, and no vertex in the bottom graph has such a degree.

Theorem 1.1. *If (V, E) is a connected graph with a cost function $c : E \to \mathbb{R}$, then Kruskal's algorithm yields a* minimal *spanning tree of (V, E).*

Moral of the proof:
We need to show two things; first, Kruskal's algorithm yields *a* spanning tree, and second, it yields a *minimal* spanning tree.

For the first claim, note that at every stage of the algorithm we are careful to ensure we're not creating any circuits. Thus for the end result to be a spanning tree, we just need to check the output of the algorithm is a connected graph. If the resulting spanning tree (V, E') isn't connected, we look at its connected components.

Note that as the full graph is connected, it is possible to connect these disjoint, connected components. If we connect two components by a single edge e, however, we couldn't have possibly made a circuit. This is a contradiction to Kruskal's algorithm, which tells us to add in an edge if it doesn't result in a circuit. Therefore there cannot be two distinct connected components, meaning (V, E') is connected, as required.

To show the algorithm gives a *minimal* spanning tree, consider what we did. We can assume the spanning tree is a proper subgraph of (V, E) (if not, then it is minimal immediately). In the algorithm, we create a queue of the edges such that, if the edges in this queue are labelled $e_1, \ldots, e_m$,

$$c(e_1) \leq c(e_2) \leq \cdots \leq c(e_{m-1}) \leq c(e_m).$$

The algorithm then chooses $\#(V)-1$ edges[1] that are the *lowest cost*. If (V, E'') is any other spanning tree, then $c(E') \leq c(E'')$, as required. ■

2 Prim's Algorithm

This is similar to the **constructive approach** of last week, and has the benefit that at every step the subgraph is a tree.

How does the algorithm go?

(1) Start with some vertex $v \in V$. The corresponding subgraph is $(\{v\}, \emptyset)$.

(2) List all the edges from smallest to biggest cost, as last time.

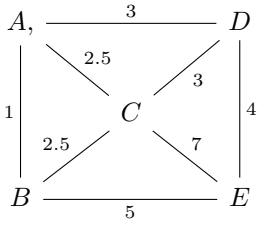
(3) We identify the front-most edge in the queue that has one vertex in the current subgraph and one vertex *not* in the current subgraph.

(4) Add this edge and vertex to the graph.

(5) Repeat (3) and (4) until the queue is emptied or it is not longer possible to select edges as in (3).

By its very nature this algorithm yields the minimal spanning tree of a graph; at each stage, it generates a tree, the end result contains all the vertices[2] (making it a spanning tree), and it is minimal by the use of the queue.

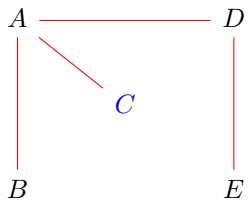
Example 2.1. Consider the following example, worked out in the tutorial.



Starting with vertex C, we obtain the minimal spanning tree:

[1] From the first half of the proof the algorithm creates a tree & in a tree $\#(E) = \#(V) - 1$

[2] The graph is connected, so no isolated vertices!



Sidenote: the minimal spanning tree given by Kruskal or Prim is unique only if no two edges share the same cost. This is not the case for *Examples 1.1/2.1*.