MATH 409 LECTURE 8
KRUSKAL'S ALGORITHM FOR MINIMUM SPANNING TREES

REKHA THOMAS

Kruskal's Algorithm

Input: A connected graph G with edge costs $c_e \in \mathbb{R}$, $\forall\ e \in E(G)$.
Output: A MST T of G.

(1) Sort the edges such that $c_{e_1} \leq c_{e_2} \leq \cdots \leq c_{e_m}$.
(2) Set $T = (V(G), \emptyset)$
(3) For i from 1 to m do
    If $T + e_i$ contains no circuit then set $T = T + e_i$.

(Note that $T + e_i$ will not have circuits if and only if the end points of e_i are in different components of T.)

Exercise 1. Find an MST for the graph in Exercise 3 of Lecture 7 using Kruskal's algorithm.

Theorem 2. *The graph T found by Kruskal's algorithm is an MST of G.*

Proof. Let T be the output of Kruskal's algorithm. We need to first show that T is a spanning tree of G.
(i) T is a spanning subgraph of G since all intermediate graphs in Kruskal's algorithm are spanning subgraphs of G.
(ii) By construction, T is acyclic since we only add edges during the algorithm if they do not create cycles.
(iii) Suppose T is not connected. Then T has at least two non-empty components. Let C be such a component and consider $\delta(V(C))$. Since G is connected, there is at least one edge $e = xy \in \delta(V(C))$. When this edge was checked by Kruskal's algorithm, it was not added to the current forest T since it created a circuit D in T. Note that This means that both end points x and y of e were in the same component of T which means they are both in C. This contradicts that $e \in \delta(V(C))$. Therefore, T is a spanning tree in G.

Date: April 14, 2010.

We now argue that T satisfies property (2) of Theorem 7 from Lecture 6: *for all $e = xy \in E(G)\backslash E(T)$ no edge on the $(x-y)$-path in T has higher cost than e* which will prove that it is an MST. Take an edge $e = xy \in E(G)\backslash E(T)$. Then this edge was checked at some step of the algorithm and it was not added since it would have created a cycle in the current T. This means that there was an $(x-y)$-path in the current tree T. Since all the edges in this path were chosen before e, none of them are more expensive than e as we are checking edges in increasing order of their costs. □

Theorem 3. *Kruskal's algorithm has a running time of $O(m \log n)$ where $n = |V(G)|$ and $m = |E(G)|$.*

Proof. • In Step (1) we need to sort m numbers. This can be done in $O(m\, log_2\, m)$ time. However, in our case, $O(m\, log_2\, m) = O(m\, log_2\, n^2) = O(m\, 2log_2\, n) = O(m\, log_2\, n)$. (Recall that $m = |E(G)| = O(n^2)$ when $n = |V(G)|$.)
• Step (2) takes a constant amount of time, so we ignore it.
• Now we analyze Step (3) which consists of m iterations of the same task. In iteration i, we check whether the endpoints of $e_i = vw$ are in the same connected component of T. If not, set $T = T + e_i$.

We use a special data structure. Maintain each connected component of T as a tree with a unique root and at most one entering edge at each vertex. The union of all these directed graphs is called a *branching*. Call it B.

To check whether adding $e_i = vw$ creates a cycle, find the root r_v of the component containing v and the root r_w of the component containing w. The time needed to do this is roughly the sum of the lengths of the (r_v, v)-path in B and the (r_w, w)-path in B. If $r_v \neq r_w$, insert e_i into T and coalesce the components of v and w by adding an edge connecting them. We need to do this cleverly, so that what results is again a branching.

Let $h(r)$ = max length of a path from r in B. If $h(r_v) \geq h(r_w)$ then add (r_v, r_w) to B directed from r_v to r_w. Else, add (r_w, r_v) to B directed from r_w to r_v. Why? By this rule, if $h(r_v) = h(r_w)$ then r_v is the root of the coalesced component and **new** $h(r_v)$ **= old** $h(r_v) + 1$. If $h(r_v) > h(r_w)$ then again r_v is the root of the coalesced component and **new** $h(r_v)$ **= old** $h(r_v)$. If $h(r_v) < h(r_w)$ then we use this last

argument again but with the roles of v and w reversed. In all cases, you see that the h-values can be easily updated by this procedure. Initially, $B = (V(G), \emptyset)$ – i.e., each vertex is its own connected component and there are no edges in any of the components. This means that $h(v) = 0$ for all $v \in V(G)$.

Claim: A connected component of B with root r contains at least $2^{h(r)}$ vertices.

Before we prove the claim, we argue that the claim will give us the result we are looking for. If the claim is true then $n = |V(G)| \geq 2^{h(r)}$ for any root vertex r in B and hence $h(r) \leq log_2 n$ for any r. This means that the sum of the lengths of the (r_v, v)-path in B and the (r_w, w)-path in B is at most $2\, log_2 n$. Since there are m iterations in Step (3), we have a total of $O(m(2\, log_2 n))$ work in Step (3). Adding this to the $O(m\, log_2 n)$ work in Step (1) we see that Kruskal's algorithm runs in $O(m\, log_2 n)$ time.

Proof of the claim: At the start, $B = (V(G), \emptyset)$ which means that there are n connected components, $h(v) = 0$ for all $v \in V(G)$, and each connected component has at least $2^0 = 1$ vertex.

Suppose the property of the claim holds up to a certain stage of the algorithm and in this stage we add the directed edge xy, directed from x to y, to B to coalesce two components. We saw above that either $h(x)$ remains the same with this addition or it increases by one. If $h(x)$ does not change, the new coalesced component has at least $2^{h(x)}$ vertices since the old component with root x already had at least that many vertices by our assumption. Otherwise, $h(x) = h(y)$ before adding xy and each component had at least $2^{h(x)}$ vertices by assumption. This implies that the new coalesced component has at least $2 \cdot 2^{h(x)} = 2^{h(x)+1}$ vertices. However, the h-value of the new root is also $h(x) + 1$ which proves the claim. □

Exercise 4. [1, Exc 2.16] Suppose that instead of the sum of the costs of edges of a spanning tree, we wish to minimize the maximum cost of an edge of a spanning tree. That is, we want the most expensive edge of the tree to be as cheap as possible. This is called the *minmax spanning tree problem.* Prove that every MST is actually a minmax spanning tree.

Recall that a **cut** in a graph G is a collection of edges defined as follows. Let V_1 be a subset of $V(G)$ and let G_1 be the graph induced by V_1 in G. Then $\delta(G_1)$, the set of edges in G with exactly one end

point in V_1, is called a cut in G. Removing the edges in $\delta(G_1)$ would disconnect G.

Exercise 5. [2, Exc 6, pp 132] Consider the following coloring algorithm for a connected graph G with edge costs c_e for all $e \in E(G)$. Initially all edges are uncolored. Then apply the following rules in arbitrary order until all the edges are colored:
Blue rule: Select a cut containing no blue edge. Among the uncolored edges in the cut, select one of minimum cost and color it blue.
Red rule: Select a circuit containing no red edge. Among the uncolored edges in the circuit, select one of maximum cost and color it red.
Show that one of the rules is always applicable as long as there are uncolored edges left. Then show that there always exists a MST containing all blue edges but no red edge. Can you see Kruskal's and Prim's algorithms as special cases of this coloring algorithm?

References

[1] W. Cook, W. Cunningham, W. Pulleyblank, and A. Schrijver. *Combinatorial Optimization*. Wiley-Interscience Series in Discrete Mathematics, 1998.
[2] B. Korte and J. Vygen. *Combinatorial Optimization*. Springer, Berlin, 2000.