CS2ME3 Lecture Notes

Michael Soltys

2006 Winter Term

1 Greedy Programs

1.1 Minimum Cost Spanning Trees

We start by giving several definitions.

Definition 1.1 An *undirected graph* G is a pair (V, E) where V is a set (of vertices or nodes), and $E \subseteq V \times V$ and $(u, v) \in E$ iff $(v, u) \in E$, and $(u, u) \notin E$. The *degree* of a vertex v is the number of edges touching v. A *path* in G between v_1 and v_k is a sequence $v_1, v_2, \ldots, v_k$ such that each $(v_i, v_{i+1}) \in E$. G is *connected* if between every pair of distinct nodes there is a path. A *cycle* is a simply closed path $v_1, \ldots, v_k, v_1$ with $v_1, \ldots, v_k$ all distinct, and $k \geq 3$. A graph is *acyclic* if it has no cycles. A *tree* is a connected acyclic graph. A *spanning tree* of a connected graph G is a subset $T \subseteq E$ of the edges such that (V, T) is a tree. (In other words, the edges in T must connect all nodes of G and contain no cycles.)

If G has a cycle, then there is more than one spanning tree for G, and in general G may have many spanning trees, but each spanning tree has the same number of edges.

Lemma 1.1 Every tree with n nodes has exactly $(n-1)$ edges.

Exercise 1.1 Prove this lemma. (**Hint:** first show that every finite tree has a *leaf*, i.e. a node of degree one. Then show the lemma by induction on n.).

Lemma 1.2 A graph with n nodes and more than $(n-1)$ edges must contain at least one cycle.

Exercise 1.2 Prove this lemma.

We are interested in finding a minimum cost spanning tree for G, assuming that each edge e is assigned a cost $c(e)$. (Assume for now that the cost $c(e)$ is a nonnegative real number.) In this case, the cost $c(T)$ is the sum of the costs of the edges in T.

Definition 1.2 We say that T is a *minimum cost spanning tree* for G if T is a spanning tree for G and given any spanning tree T' for G, $c(T) \leq c(T')$.

Given a graph $G = (V, E)$, where $c(e_i)$ = "cost of edge e_i", we want to find a minimum cost spanning tree. It turns out (fortunately) that in this case, an obvious greedy algorithm (Kruskal's algorithm) always works. Kruskal's algorithm is the following: sort the edges in non-decreasing order of costs, so that $c(e_1) \leq c(e_2) \leq \ldots \leq c(e_m)$, add an edge one at a time, but throw the edge out if it forms a cycle.

1.1.1 Kruskal's Algorithm

Sort the edges: $c(e_1) \leq c(e_2) \leq \ldots \leq c(e_m)$
$T \longleftarrow \emptyset$
for $i : 1..m$
$(*)$ if $T \cup \{e_i\}$ has no cycle then
$T \longleftarrow T \cup \{e_i\}$
end if
end for

But how do we test for a cycle (i.e. execute $(*)$)? After each execution of the loop, the set T of edges divides the vertices V into a collection $V_1, \ldots, V_k$ of *connected components*. Thus V is the disjoint union of $V_1, \ldots, V_k$, each V_i forms a connected graph using edges from T, and no edge in T connects V_i and V_j, if $i \neq j$. A simple way to keep track of $V_1, \ldots, V_k$ is to use an array $D[i]$ where $D[i] = j$ if vertex $i \in V_j$. Initialize D as follows:

for $i : 1..n$
$D[i] \longleftarrow i$
end for

To check whether $e_i = (r, s)$ forms a cycle with T, check whether $D[r] = D[s]$. If e_i does not form a cycle with T, we update: $T \longleftarrow T \cup \{(r, s)\}$, and we merge the component $D[r]$ with $D[s]$ as follows:

$k \longleftarrow D[r]$
$l \longleftarrow D[s]$
for $j : 1..n = \#$ of vertices
if $D[j] = l$ then $D[j] \longleftarrow k$
end if
end for

Exercise 1.3 Given that the edges can be ordered in m^2 steps (using insertion sort, or in $O(m \log m)$ using something more fancy like heapsort, *and* assuming again that the cost of a comparison is that of a single operation), what is the running time of this algorithm?

1.1.2 Correctness of Kruskal's Algorithm

It is not immediately clear that Kruskal's algorithm yields a spanning tree at all, let alone a minimum cost spanning tree.

To see that the resulting collection T of edges is a spanning tree for G (assuming that G is connected), we must show that (V, T) is connected and acyclic. It is obvious that T is acyclic, because we never add an edge that results in a cycle. To show that (V, T) is connected, we reason as follows. Let u and v be two distinct nodes in V. Since G is connected, there is a path p connecting u and v in G. The algorithm considers each edge e_i of G in turn, and puts e_i in T *unless* $T \cup \{e_i\}$ forms a cycle. But in the latter case, there must be a path in T connecting the end points of e_i, so deleting e_i does not disconnect the graph.

This argument can be formalized by showing that the following statement is an invariant of the loop in Kruskal's algorithm:

$$\textit{The edge set } T \cup \{e_{i+1}, \ldots, e_m\} \textit{ connects all nodes in } V. \tag{1}$$

Exercise 1.4 Prove, by induction, that (1) is a loop invariant. First show that (1) holds before before the main loop of Kruskal's Algorithm executes (the 0-th iteration; this is the Basis Case—remember the assumption (*precondition*) that $G = (V, E)$ is connected). Then show that if (1) holds after the i-th execution of the loop, then $T \cup \{e_{i+2}, \ldots, e_m\}$ connects all nodes of V after the $(i + 1)$-st execution of the loop. Conclude by induction that (1) holds for all i. Finally, show how to use this loop invariant to prove that T is connected.

Exercise 1.5 Suppose that $G = (V, E)$ is *not* connected. Show that in this case, when G is given to Kruskal's Algorithm as input, KA computes a *spanning forest* of G. Define first precisely what is a spanning forest (first define the notion of a *connected component*). Then give a formal proof using the idea of a loop invariant, as in exercise 1.4.

To show that the spanning tree resulting from the algorithm is in fact a minimum cost spanning tree, we reason that after each execution of the loop, the set T of edges can be expanded to a minimum cost spanning tree using edges that have not yet been considered. Hence after termination, all edges have been considered, so T must itself be a minimum cost spanning tree. We can formalize this reasoning as follows:

Definition 1.3 A set T of edges of G is *promising* if T can be expanded to a minimum cost spanning tree for G.

Lemma 1.3 The following is a loop invariant for Kruskal's algorithm: T *is promising.*

PROOF: The proof is by induction on the number of iterations of the main loop of Kruskal's Algorithm. **Basis Case:** (at this stage the algorithm has gone through the loop zero (0) times): Initially T is the empty set, which is obviously promising.

Induction Step: We assume that T is promising, and show that T continues to be promising after one more execution of the loop.

Notice that the edges used to expand T to a spanning tree must come from those not yet considered, because the edges that have been considered are either in T already, or have been rejected because they form a cycle. We examine by cases what happens after edge e_i has been considered:

case(I): e_i is rejected. T remains unchanged, and so it is still promising.