CSE 347 Lecture 3

Greedy Algorithms II – Minimum Spanning Tree

1. Greedy Algorithm for scheduling all requests
2. Problem: Minimum Spanning Tree and Network Design
3. An exchange argument - the cut property

Last lecture, we discussed greedy algorithms for scheduling problems – we are given a set of requests for access to a resource, such as a room, a computer, a work crew, etc. each represented by a (time) interval in which that resource is needed. We introduced the problem of scheduling all intervals – partitioning the set of requests R into sets $I_1, \dots, I_d$ such that (1) no I_j contains a <u>conflict</u>, i.e., a pair of intervals that intersect and (2) d, representing the # of distinct resources we use to serve the requests, is as small as possible. We made the following observation: let's call the <u>depth</u> of R the maximum, over all points in time t, of the number of intervals containing t: $\max_t |\{r \in R : t \in r\}|$

<u>Lemma</u> d must be at least the depth of R.

Why? At each t, all of the intervals r s.t. $t \in R$ conflict,

and must be assigned distinct I_j. Thus $d \geq |\{r \in R : e \in r\}|$ $\forall t$.

We'll show a greedy algorithm that achieves d equal to the depth of R. Hence, by our lemma, this is optimal (it can't possibly be any smaller).

Here is the algorithm: suppose we are given requests $r_1, r_2, \ldots, r_n$ sorted by increasing start time

For $i = 1, 2, \ldots, n$
  For each request $j < i$ such that r_j conflicts with r_i
    Add the index k of the resource allocated to r_j to the list CONFLICTS;
  Allocate r_i to the smallest k not in CONFLICTS;

Theorem The algorithm partitions the requests into $I_1, \ldots I_d$ such that no I_j contains a conflict and d is equal to the depth of the requests.

Proof: For the first part, observe that for any pair of requests r_i and r_j that conflict such that r_i starts after r_j, the algorithm adds the index k of the resource r_j was assigned to to CONFLICTS$_i$, and the algorithm assigns r_i to a resource not in this set, hence distinct from k. Thus, no r_i & r_j conflict.

For the second part, observe that when we assign a request r_i to resource k, there must have been resources that conflict with r_i assigned to $1, 2, \ldots, k-1$,

that furthermore start earlier than r_i. Therefore, all of these k requests include the start time of r_i, so the depth of the set of requests is at least k; equivalently, k is at most the depth. ☒

Minimum Spanning Tree and Network Design

Here is another natural problem: we have a set of locations $V = \{v_1, v_2, \ldots, v_n\}$ that we wish to connect (e.g. by a communication network) as cheaply as possible. Suppose that for a subset of pairs of locations $E \subseteq V \times V$, it is possible to connect v_i and v_j for $(v_i, v_j) \in E$ at a cost $c(v_i, v_j) > 0$. Abstractly the pair (V, E) is called a graph, V is called a set of vertices or nodes and E is a set of edges. (The costs are also sometimes called weights of the edges.) In our network design problem, now, we are seeking a subset of edges T such that (1) the graph (V, T) is connected: for any pair $(v_i, v_j) \in V \times V$, there is a sequence $v_i = u_1, u_2, u_3, \ldots, u_l = v_j$ such that each $(u_k, u_{k+1}) \in T$ ($u_1, u_2, \ldots, u_l$ is called a path from v_i to v_j) and (2) $\sum_{(v_i, v_j) \in T} c(v_i, v_j)$ is as small as possible.

Solutions to the network design problem have a special structure. A cycle is a path from a vertex v back to v that has length at least 1. A tree is a graph with no cycles that is connected.

Lemma: Optimal solutions to the network design problem are trees.

Why? If T contains a cycle $u_1, u_2, \ldots, u_\ell$, we can delete any edge on the cycle, e.g. $(u_{\ell-1}, u_\ell)$ from T. Then any path that would use the edge $(u_{\ell-1}, u_\ell)$ can instead use the path $u_{\ell-1}, u_{\ell-2}, \ldots, u_1 = u_\ell$, so the graph remains connected. Its cost is reduced by $c(u_{\ell-1}, u_\ell) > 0$, so the original T couldn't have been optimal.

So we see that the network design problem is equivalent to the following **minimum spanning tree problem**:
Given a graph (V, E) find $T \subseteq E$ such that (V, T) is a (connected) tree and $\sum_{(v_i, v_j) \in T} c(v_i, v_j)$ is minimized.

Again, there are many possible greedy algorithms for this problem. Unlike our first interval scheduling problem, almost every greedy algorithm for minimum spanning tree returns an optimal tree.
The first is due to Kruskal:

Initialize $T = \emptyset$
For $i = 1, \ldots, n$
  Choose $e_i \in E$ to be the edge of smallest cost such that $T \cup \{e_i\}$ does not contain a cycle
  Put $T \leftarrow T \cup \{e_i\}$

The second is due to Prim

Initialize $V_1 = \{v\}$ for some $v \in V$, $T = \emptyset$

For $i = 1, \dots, n$
  Choose the edge e_i with one vertex in V_i and one vertex not in V_i of smallest cost
  Put $T \leftarrow T \cup \{e_i\}$, $V_{i+1} \leftarrow V_i \cup \{u_i\}$ where $e_i = (v_i, u_i)$ for $v_i \in V_i$, $u_i \notin V_i$

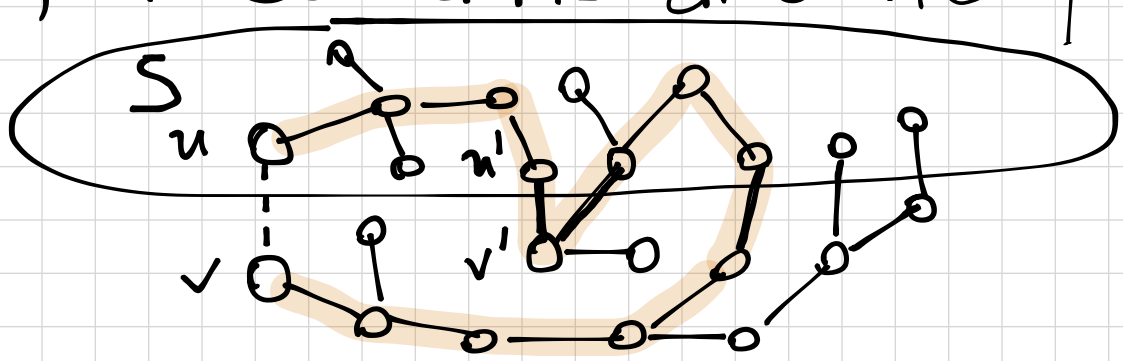
(So, the difference is that Prim guarantees that T is a tree on V_i, and Kruskal does not.)

These algorithms are both optimal for similar reasons

Lemma ("Cut property"): Suppose that all edge costs are distinct, and S is a nonempty set $S \subsetneq V$ (S is a "cut"). Then the edge $(u,v) \in E$ of minimum cost among those such that $v \in S$ and $u \notin S$ must be included in any minimum spanning tree.

Idea: We will show that for any spanning tree T that does not use one of these minimum cost edges e, we can exchange e for some edge of T to obtain a spanning tree of smaller cost, so T is not optimal. This is known as an "exchange argument."

Proof: Let T be a spanning tree such that for some cut S, T does not include our minimum cost edge $e = (u,v)$ with $v \notin S$, $u \in S$. Since T is a spanning tree, T contains another path joining u to v:



If we follow this path from u to v, there must be some first vertex $v' \notin S$. Let u' be the vertex preceding v' on this path (so $u' \in S$). Now, we have by assumption that $c(u,v) < c(u',v')$, and we claim that if we exchange (u,v) for (u',v'), the graph $T \cup \{(u,v)\} \setminus \{(u',v')\}$ is a spanning tree — which therefore has lower cost than T, meaning T was not optimal.

To see that this is a spanning tree, first observe that it is connected, since any path in T that would traverse (u',v') can instead have the path from u' to v' that (i) follows the path from u to u' in reverse, (ii) traverses (u,v) and then (iii) follows the final segment of the path from v' to v in reverse.

We next show that the graph has no cycles: $T \cup \{(u,v)\}$ can only have one cycle, since any cycle must traverse (u,v), and if there were more than one such cycle, $u, u_1, u_2, \ldots, u_\ell, v, u$ and $u, u'_1, \ldots, u'_{\ell'}, v, u$, then $u, u_1, u_2, \ldots, u_\ell, v, u'_{\ell'}, \ldots, u'_2, u'_1, u$ would be a cycle that does not use (u,v), i.e., in T. Thus, since we remove (u',v') from this cycle, $T \cup \{(u,v)\} \setminus \{(u',v')\}$ has no cycles, and is thus a tree. ▨