CMPSCI611: **Greedy Algorithms and Matroids** Lecture 4

Our next algorithmic paradigm is **greedy algorithms**. A greedy algorithm tries to solve an **optimization problem** by always choosing a next step that is **locally optimal**. This will generally lead to a **locally optimal** solution, but not necessarily to a **globally optimal** one.

When the goal of our optimization is to **maximize** some quantity, we call a locally optimal solution **maximal** and a globally optimal one **maximum**. If we are **minimizing** the quantity, we call these **minimal** and **minimum** respectively.

We begin with a minimization problem that can be solved with a greedy algorithm.

Suppose we have a **weighted undirected graph** G that is **connected**. A **spanning tree** is a subset of the edges that is connected and is a **tree** (that is, it has **no cycles**). Any connected graph has a spanning tree, and it may have many of them. A **minimum spanning tree (MST)** is a spanning tree that has a total weight (sum of the weights of its edges) that is no greater than that of any other spanning tree. There may be more than one MST in case of ties.

Note that edge weights are always positive. (We may sometimes use edge weights of zero, but these could if necessary be thought of as just very small positive numbers.)

Kruskal's Algorithm is a greedy solution to the minimum spanning tree problem.

- Sort the edges by weight, cheapest first.
- Set F to be the empty set.
- For each edge e in order, add e to F unless this would create a cycle in F.
- Return F.

We need a way to determine whether an edge e creates a cycle in F. To do this, we keep track of the **connected components** of F – the sets of vertices of G such that two vertices are in the same component iff there is a path from one to the other in F. Let $e = (x, y)$ where x and y are vertices. If x and y are in the same component, adding e to F would create a cycle. If not, adding e merges the two connected components.

The simplest way to merge the components is to scan a table mapping vertices to components, moving each vertex of the smaller component into the larger one.

We now need to prove that the Kruskal algorithm actually produces an MST. First, though, a warmup problem to get us used to working with graphs and induction:

One-Question Final Exam for CMPSCI 250:

Let F be a forest (a acyclic undirected graph) with n vertices, k edges, and c connected components. Prove that $c = n - k$.

Proof by induction on e – we show that the desired fact is an **invariant**:

"As it was in the beginning, is now, and ever shall be" (Anglican Book of Common Prayer)

Base Case: No edges, n components, $n = n - 0$.

Inductive Step: Add an edge – it merges two connected components into one, since if its endpoints were already connected it would create a cycle. By induction $c = n - k$ before, now $c - 1 = n - (k + 1)$.

Conclusion: If we ever reach $k = n - 1$, then $c = 1$ and we have a tree.

Correctness of Kruskal:

Given a forest F, let $S(F)$ be the set of spanning trees that contain all of F's edges.

As we add edges, the following invariant stays true: "$S(F)$ contains an MST".

Base Case: Since G is connected, an MST exists, and it contains F which is $\emptyset$.

Inductive Step: We assume that $S(F)$ has an MST, and we add the cheapest available edge, called $e = (x, y)$, to F. We must show that $S(F \cup \{e\})$ contains an MST.

We'll show that for any tree T in $S(F)$ that *doesn't* include e, there is a tree that *does* include e and has the same or lower total weight.

Since T is a tree, it contains a path between x and y, the endpoints of e. Since there was no such path in F, the path contains some edge e' that is not in F. Our cheaper tree is going to be $T' = (T \setminus \{e'\}) \cup \{e\}$. Since e was the cheapest available edge, this set has total weight equal or smaller than that of T. But how do we know that T' is a tree?

$$T' = (T \setminus \{e'\}) \cup \{e\}$$

In T, $e' = (u, v)$ was part of a path from x to y. In T', we can still get from u to v by going from u back along the path to x, over e to y, and backward on the path to v. So the only edge in T that is not in T' can be successfully bypassed in T', so every path in T can be simulated in T', and $T'$is connected.

Why doesn't T' have a cycle? A cycle would have to connect x to y by another route. But in the tree T there was a unique path from x to y, and we broke it by removing e'.

Conclusion: At any time in the algorithm, if F is not a tree, there must be edges in G that join separate connected components of F. These edges must be at least as expensive as any edges in F, since otherwise we would have looked at them already and added them to F because they join components of F.

So the algorithm can stop only when F is a tree. At that point $S(F) = \{F\}$, so F itself must be an MST.

Timing of Kruskal:

The timing will depend on both n, the number of vertices, and k, the number of edges. (Note that $k \geq n - 1$ for a connected graph, and $k \leq \binom{n}{2} = O(n^2)$ for any graph.)

Sorting the list of edges takes $O(k \log k)$ time.

Setting up our table of components for each vertex takes $O(n)$ time. By the simple method we have described, updating the table takes one pass over the table, or $O(n)$ time. We will make $n - 1$ such sweeps for $O(n^2)$ total time, because we will add an edge to F exactly $n - 1$ times. We also might spend up to $O(k)$ time checking edges that we don't add to F. Thus:

$$\begin{aligned} T(n,k) &= O(k \log k) + O(n) + O(n^2) + O(k) \\ &= O(k \log k + n^2) \end{aligned}$$

In a few lectures we'll see how to reduce this to $O(k \log k)$ by using a better **union-find data structure**.

When do greedy algorithms work? It turns out that we can give an answer to this question for a wide class of optimization problems.

A **subset system** is a set E together with a *set of subsets* of E, called I, such that I is **closed under inclusion**. This means that if $X \subseteq Y$ and $Y \in I$, then $X \in I$.

The **optimization problem** for a subset system has as input a positive weight for each element of E. Its output is a set $X \in I$ such that X has at least as much total weight as any other set in I.

We call I the "independent sets" of the subset system, because in general I will be defined so it will include exactly those sets that don't have a particular kind of "dependence" among their elements. Let's see some examples.

Examples of Subset Systems

Example 0: Let E be any set of vectors in some vector space, and let I be the set of sets of **linearly independent** vectors. Clearly this I is closed under inclusion. This is the origin of the name "independent sets".

Example 1:

$$
\begin{aligned}
E &= \{e_1, e_2, e_3\} \\
I &= \{\emptyset, \{e_1\}, \{e_2\}, \{e_3\}, \{e_1, e_2\}, \{e_2, e_3\}\}
\end{aligned}
$$

The closure under inclusion can be checked directly. I can also be described as "all sets that don't contain both e_1 and e_3".

Example 2: Let E be the edges of an undirected graph, and I be the set of all *acyclic* sets of edges.

Example 3: Let E be the edges of an undirected graph, and I be the set of sets of edges that don't have two or edges sharing a vertex. (These sets of edges are often called "independent sets" even outside this context, and are also known as "matchings").

The Generic Greedy Algorithm

Given any *finite* subset system (E, I), we find a set in I as follows:

- Set X to $\emptyset$.
- Sort the elements of E by weight, heaviest first.
- For each element of E in this order, add it to X iff the result is in I.
- Return X.

This certainly gives us a set in I (unless I is itself empty, since $\emptyset$ must be in I if any set is). It is a *maximal* set, meaning that no element of E can be added to it without bringing X outside of I. But a solution to the optimization must be a *maximum* set, with weight greater than or equal to that of any other set in I.

Our main result is that there is a property of set systems that determines whether this greedy algorithm is guaranteed to give a maximum set for all possible weighting functions.

Greedy Algorithm Examples:

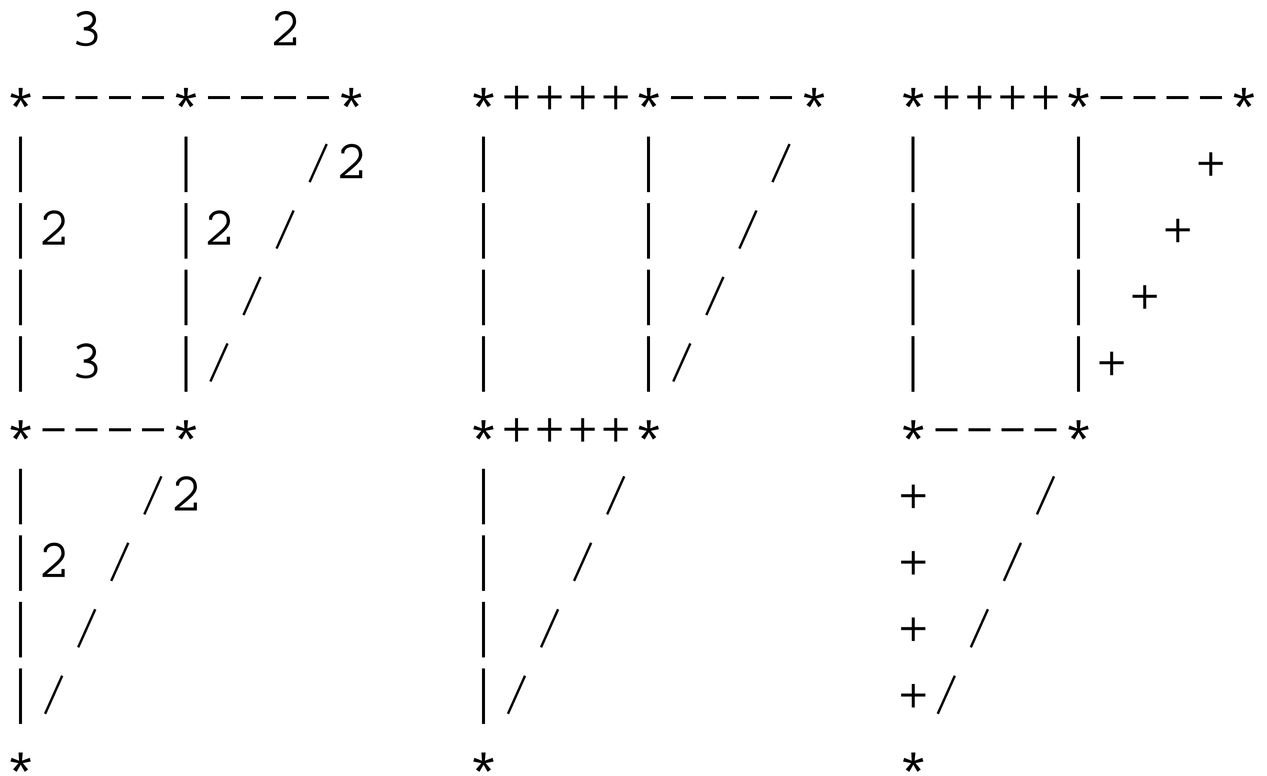
Subset System 1: (not both e_1 and e_3) We will get e_2 and the larger of e_1 and e_3, which must be a maximum set in I.

Subset System 2: (acyclic sets) This is the **Maximum Weight Forest (MWF)** problem, which is the same as the MST problem except that (a) the input graph need not be connected, and (b) we want to maximize instead of minimizing.

But we can convert the MST problem into an equivalent MWF problem, and vice versa, as follows. Let m be the maximum weight of any edge in the MWF problem, and set the weight of each edge e in the MST problem to be $m - w(e) + \epsilon$.

The greedy algorithms produce exactly the same set for the two weighting functions, and (since the number of edges in the MST must be $n - 1$), a maximum set for one function is a minimum for the other.

Subset System 3: (no edges sharing a vertex) Here is an example where the greedy algorithm gets a maximal set that is not maximum:



(This is Figure 3.2 of the text.) The greedy algorithm takes both weight-3 edges first, and gets the set in the center, with a total weight of 6. But there is a different independent set that has weight 7, shown on the right.

A subset system is a **matroid** if it satisfies the **exchange property**: If i and i' are sets in I and i has fewer elements than i', then there exists an element $e \in i' \setminus i$ such that $i \cup \{e\} \in I$.

Subset System 1: This *is* a matroid, as we can check the exchange property by inspection. For example, if $i = \{e_1\}$, $i' = \{e_2, e_3\}$, we can let $e = e_2$.

Subset System 2: Next lecture we'll show that this *is* a matroid.

Subset System 3: This is *not* a matroid – let i be the greedy matching and i' be the maximum matching in our example. There is no edge e *at all*, much less in i', that can be added to i while keeping it independent.

Next time we'll prove:

Theorem: For any subset system (E, I), the greedy algorithm solves the optimization problem for (E, I) if and only if (E, I) is a matroid.

This is good mathematics! We've found a property of set systems that characterizes the behavior of the greedy algorithm, but doesn't have anything obvious to do with the algorithm. Furthermore, this property can be expressed in more than one way. We'll also prove next time:

Cardinality Theorem: A set system (E, I) is a matroid iff for any set $A \subseteq E$, any two maximal independent subsets of A have the same number of elements.

A property with ostensibly different characterizations is more likely to be mathematically interesting. Examples of such properties in CMPSCI 601 include the regular languages and the Turing-decidable languages.