CS 344: Design and Analysis of Computer Algorithms **Rutgers: Fall 2019**

Lecture 17

November 4, 2019

Instructor: Sepehr Assadi

1 Minimum Spanning Trees

We switch gear now and consider *minimum spanning trees* of *weighted* graphs. Let $G(V, E)$ be an undirected *connected* graph with *positive* weights $w_{e_1}, \ldots, w_{e_m}$ on edges $e_1, \ldots, e_m$ of E (namely, each edge e has a weight w_e). Recall that a spanning tree of a connected subgraph G is any subgraph of G which is a tree – a tree itself is a connected subgraph which has no cycle[1]. We have,

Problem 1 (Minimum Spanning Tree (MST)). The minimum spanning tree problem (MST for short) is defined as follows:

- **Input:** An undirected connected graph $G(V, E)$ with weights over the edges.
- **Output:** A spanning tree of G with minimum total weights of edges in the tree.

Before getting to design an algorithm for this problem, we first need to take a detour and study an important graph-theoretic notion, namely, *graph cuts*.

1.1 Quick Detour: Graph Cuts

For an undirected graph $G(V, E)$, a *cut* is simply any partitioning of vertices into two sets $(S, V - S)$ which are both non-empty. In other words, any set S of vertices define a cut $(S, V - S)$ in G. For a cut $(S, V - S)$, we define the set of *cut edges* as all edges $e = (u, v) \in E$ where $u \in S$, $v \in V - S$ or vice versa. For a cut defined by set of vertices S, (namely, the cut $(S, V - S)$), we use $\delta(S)$ to denote the set of cut edges.

Let us prove the following claim about cuts as an exercise.

Claim 1. *Any undirected graph $G(V, E)$ is connected if and only if every cut $(S, V - S)$ in G has at least one cut edge, namely, $|\delta(S)| > 0$.*

Proof. We prove each part separately:

- If G is connected then every cut in G has at least one cut edge. We prove this by contradiction. Suppose G is connected but there exists a cut $(S, V - S)$ with $|\delta(S)| = 0$. Pick any vertex u in S and any vertex v in $V - S$. Since G is connected, there should be path from u to v in G. Let us say the path is $P = u, w_1, w_2, \ldots, w_k, v$. Find the *first* index i such that $w_i \in S$ and $w_{i+1} \in V - S$; such an index should exists because $u \in S$ and $v \in V - S$ and so along this path we should eventually move from S-part to $(V - S)$-part. But now the edge (w_i, w_{i+1}) is a cut edge of S, a contradiction.
- If G is not connected then there exists a cut in G with no cut edge. Since G is not connected, it has more than one connected component. Let S be any connected component of G and consider the cut $(S, V - S)$. By definition, there is no edge going "out of" S and hence $|\delta(S)| = 0$.

□

[1] Any connected graph has at least one spanning tree; simply greedily remove any edge e from G which is part of some cycle: this 'breaks' the cycle without making G disconnected; continue until G has no cycle and hence is a tree.

1.2 A Generic "Meta-Algorithm" for MST

In the following, we propose a general approach for solving MST problem. We call this a "meta-algorithm" (a rather made up name with no formal definition) since, as it will become clear, it is not detailed enough at a level of a true algorithm.

1. Let $F = \emptyset$ be an empty *forest* initially (a forest is any subgraph of a tree on all vertices – alternatively, a forest is any graph with no cycle (but it may not necessarily be connected as a tree)).
2. For $i = 1$ to $n - 1$ steps:
   (a) Find a *safe* edge $e \in G - F$ (we will define safe edges shortly).
   (b) Add e to F, namely, $F \leftarrow F \cup \{e\}$.
3. Return F as an MST of G.

It should be clear that what we have written above, at this stage, is indeed *not* an algorithm: the step which asks for finding a safe edge is quite unspecified yet (and we have not even defined safe edges!). We now make this part more specific by defining safe edges. For that, we will also need another definition.

- **MST-good forest:** We say that a forest F is *MST-good*, if there exists *some* MST T of G such that every edge of F belongs to T, namely, $F \subseteq T$ (note that MSTs are not unique in general).
- **Safe edge:** An edge e is called *safe* with respect to a forest F if the following condition holds: if F is MST-good, then $F \cup \{e\}$ is also MST-good.

Let us now see how does this definition help us in the above meta-algorithm. Clearly, at the first step of the meta-algorithm, F is MST-good (since it has no edges). The meta-algorithm finds a safe edge e_1 in the first step, so by definition of safe edges, the forest consisting of only the single edge $\{e_1\}$ is also MST-good. Now inductively, at any iteration i, F is already MST-good and we add another edge e_i to it which still keep $F \cup \{e_i\}$ MST-good. This means that after the $(n-1)$-th iteration we ended up with having F as a tree: this is because F has $n - 1$ edges and is still subgraph of a tree and since any tree has exactly $n - 1$ edges, F itself is also a tree. Moreover, even at the every last step, F is MST-good, and thus we find a tree which is a subgraph of a MST, thus this tree itself should also be MST.

By the above meta-algorithm, we find a general way of finding an MST even though it is still very unspecified – we have not yet give any instructions on how to find a safe edge. This the content of next section.

1.3 One Approach to Finding Safe Edges

In the following, we show a set of edges that are all safe with respect to an MST-good forest F (but we emphasize that there might be other safe edges also; however, we do not need to worry about finding *all* safe edges but rather just one suffice for our purpose in each iteration of the algorithm).

Theorem 1. *Let $G(V, E)$ be any undirected connected graph and F be any MST-good forest of G which is not a tree. Suppose $(S, V - S)$ is any cut in F with no cut edges*[2]*. Then the edge e with* minimum weight *among the cut edges of* $(S, V - S)$ in G *is a safe edge with respect to F*[3].

Proof. The proof is similar to an exchange argument. Since F is MST-good, we know that there exists a MST T such that $F \subseteq T$. Now, if $e \in T$, we will be done since we will have $F \cup \{e\} \subseteq T$ and thus $F \cup \{e\}$ is also MST-good, implying that e is safe. We thus assume in the following that $e \notin T$.

[2]Such a cut always exists by Claim 1 since F is not connected yet

[3]Such an edge always exists by Claim 1 since G is connected.

In this case, we are going to find another MST T' such that $F \cup \{e\} \subseteq T'$; since T' is still an MST, this implies that $F \cup \{e\}$ is MST-good, and allows us to finalize the proof.

Suppose the endpoints of e are u, v, i.e., $e = \{u, v\}$. Recall that $u \in S$ and $v \in V - S$ as e is a cut edge of $(S, V - S)$. Since T is connected, we know that there exists a path from u to v in T. Moreover, as we already show in Claim 1, at least one edge of this path, say edge f, should also belong to the cut edges of $(S, V - S)$.

Now consider the subgraph $T' = T \cup \{e\} - \{f\}$. We claim that T' is an MST which finalizes the proof since $F \subseteq T'$ (because $F \subseteq T$ and the edge f we removed from T to get T' cannot belong to F since F had no cut edge in $(S, V - S)$), and $e \in T'$ (since we explicitly added e to T').

Firstly, by definition of e, we have $w_e \leq w_f$. Thus, the total weight of edges in T' is at most equal to that of T. We only need to prove that T' is a tree now. Moreover, notice that T' has exactly $n - 1$ edges since T had $n - 1$ edges (because it was a tree). So, to prove that T' is a tree, we only need to show it is connected (any connected graph with $n - 1$ edges is a tree as it cannot have a cycle). But T' is connected because $T \cup \{e\}$ was definitely connected (T itself was connected), and we only removed an edge f from a *cycle* of $T \cup \{e\}$, (the cycle created by going from u to v by the path that goes through f and coming from v to u by edge e). Since removing an edge from a cycle does not make the graph disconnected, we have that T' is also connected. This concludes the proof. □

2 Algorithms for MST

The generic meta-algorithm from previous section together with Theorem 1 gives us a way of finding MST. We now only need to find a way of finding a minimum weight edge in some cut that is still "empty" in F. Both the algorithms we study in this course for MST, namely, Kruskal's and Prim's algorithms, exactly do this although via different approaches.

2.1 Kruskal's Algorithm

Kruskal's algorithm works as follows:

1. Sort the edges of G in increasing (non-decreasing) order of their weights.
2. Let $F = \emptyset$.
3. For $i = 1$ to m (in this ordering of edges):
    - If adding e_i to F does *not* create a cycle, let $F \leftarrow F \cup \{e_i\}$.
4. Return F.

In the next lecture, we will prove the correctness of this algorithm and see how to implement the step of this algorithm that needs to check whether e_i creates a cycle or not in F efficiently, using the *union-find* data structure (see Chapter 21 of CLRS for more details about this wonderful data structure).