CS1570 Lecture Notes

clairemmathieu

Fall 2026

1 September 10: Greedy Algorithms and Minimum Spanning Trees (MST)

1.1 Overview of Greedy Algorithms

Greedy algorithms make locally optimal choices at each step with the hope of finding a global optimum. There is no backtracking: decisions are final. Two classic examples on graphs are:

- **Minimum Spanning Tree (MST):** Solved using **Kruskal's Algorithm** or **Prim's Algorithm**.
- **Single-Source Shortest Paths:** Solved using **Dijkstra's Algorithm**.

1.2 Minimum Spanning Tree (MST) Problem Definition

One thing I did *not* do in class was to provide motivation for this problem. Here are some applications: building a physical infrastructure (a network with fiver optics, say) minimizing cost, where the cost is proportional to the total length; circuit design; and it's also used as a subroutine towards solving some other problems such as the traveling salesman path. Plus, it's a basic, core problem of network algorithms.

Input

An undirected, connected, edge-weighted graph $G = (V, E)$, where:

- $V = \{u_1, u_2, \ldots, u_n\}$ is the set of n vertices.
- $E = \{e_1, e_2, \ldots, e_m\}$ is the set of m edges.
- Each edge has a non-negative weight: $e \mapsto w(e) \geq 0$.

Output

A tree $T \subseteq E$ such that:

1. T spans all vertices in V.
2. The total weight $w(T) = \sum_{e \in T} w(e)$ is minimum.

1.3 MST Approaches: Prim's vs. Kruskal's

Following the observation that a spanning tree always has $n - 1$ edges and that the two shortest edges belong to the minimum spanning tree, we wondered whether taking the $n - 1$ shortest edges of E might provide a MST. That is not true: a counterexample is immediate (exercise). We then turned to classic MST algorithms.

1.3.1 Prim's Algorithm

- **Core Idea:** Builds a **connected** set of edges.
- **Algorithm:** Start by taking the shortest edge, spanning its two endpoints. Repeatedly add the shortest edge that connects a vertex already spanned to a vertex not yet spanned.

1.3.2 Kruskal's Algorithm

- **Core Idea:** Builds an **acyclic** set of edges (a forest that eventually becomes connected).
- **Algorithm:** Start with the empty set of edges. Repeatedly add the shortest edge that does not create a cycle.

Pseudocode for Kruskal's Algorithm

```
1: T ← ∅                                                    ▷ Initialize the tree as empty
2: Sort all edges in non-decreasing order of weight: 0 ≤ w(e_1) ≤ w(e_2) ≤ ··· ≤ w(e_m)
3: for i = 1 to m do
4:     if T ∪ {e_i} is acyclic then
5:         T ← T ∪ {e_i}
6:     end if
7: end for
```

1.4 Proof of Correctness for Kruskal's Algorithm

Theorem 1. *Kruskal's algorithm successfully constructs a Minimum Spanning Tree (MST).*

Proof. To prove the theorem, we must demonstrate two properties of the output T:

1. T is a spanning tree (it is acyclic and connected).
2. T has the minimum possible weight among all spanning trees.

Part 1: Proving T is a Spanning Tree

Acyclicity Property: Holds by construction.

Connectivity Property: Proof by Contradiction. Assume T is not connected. This means T consists of more than one connected components, denoted S_1, S_2, S_3, etc.

Since the underlying graph G is connected, there must exist at least one edge in G that leaves component S_1. Consider a lightest edge in G coming out of S_1 (in case of ties, consider the first such edge that is considered by the algorithm). Because this edge connects two separate components, adding it to T would not create a cycle. Therefore, when it was considered in Kruskal's algorithm, it would have been added to T, a contradiction.

Part 2: Proving T has Minimum Weight (Optimality)

The proof is by induction. What statement can be proved by induction?

[A detour: The first attempt was that for all i, the forest obtained after the first i additions of edges to T by Kruskal's algorithm have minimum weight among all acyclic sets of i edges. That is clearly true for $i = 0$. Assume that it is true for $i - 1$. Then it is not clear how to use that assumption to prove that the statement still holds for i. In the interest of time, in class we stopped exploring that venue without resolving the question of whether that statement could be proved by induction.]

Here is a standard statement to analyze Kruskal's algorithm by induction.

Claim 1. *Let T_i be the forest obtained after the i-th edge addition by Kruskal's algorithm. For all $i \geq 0$, there exists a minimum spanning tree T^* of G that contains all edges of T_i (i.e., $T_i \subseteq T^*$).*

Proof of Claim. **Base Case ($i = 0$):** $T_0 = \emptyset$. The claim holds.

Inductive Step: Assume the claim holds for $i - 1$. This means there exists an MST T^* such that $T_{i-1} \subseteq T^*$. Now, consider the next edge e added by the algorithm to form $T_i = T_{i-1} \cup \{e\}$.

If $e \in T^*$, then $T_i \subseteq T^*$, and the inductive step holds immediately. So for the rest of this proof, let's assume that $e \notin T^*$.

Consider the endpoints u and v of e. Since T^* is a spanning tree, it has a unique path p between u and v, and so, adding e to T^* creates a cycle C. To prove the claim, we will find an edge $e' \in C$ such that $T^{**} = T^* \setminus \{e'\} \cup \{e\}$ is a MST containing T_i.

How do we define e'? I went kind of quickly over the rest of this proof in class because I was short of time, so it was only sketched, mostly orally. Please go over the rest of this proof carefully to make sure that you understand it. Use paper and pencil to draw and visualize the objects used in the proof.

When I tried to ask an AI to fill out the details of the proof for me, it wrote the following: *"Since e connects two components in T_{i-1} and T^* is a spanning tree, there must be another edge e' on the cycle C that also crosses between these same components."* That is wrong (Exercise for the reader). Here is a correct definition of e': since $T_{i-1} \cup \{e\}$ is acyclic, edge e connects u, in some connected component S_u of T_{i-1}, to v, in some other connected component S_v of T_{i-1}. Follow path p starting from vertex u, until you encounter an edge that crosses from S_u to some other connected component of T_{i-1} (it must exist since p ends at $S_v \neq S_u$): that edge is e'.

Now we must prove that $T^{**} = T^* \setminus \{e'\} \cup \{e\}$ is a MST containing T_i.

First, T^{**} is a tree by construction, since e' is on the cycle created by the addition of e to T^{**}. It has the same number of edges, so it is also spanning the graph vertices, just like T^*.

Second, T^{**} contains T_i, because it contains e, and it contains T_{i-1} just like T^*: the only edge removed from T^*, e', does not belong to T_{i-1}. (Can you see why?)

What can we say about the weight of T^{**}? $w(T^{**}) = w(T^*) - w(e') + w(e)$.

How can we compare $w(e)$ to $w(e')$? Well, we know that when e is added by Kruskal's algorithm to define T_i, the algorithm, that looks at edges by order of non-decreasing weight, has already looked at all edges that are strictly lighter than $w(e')$. So, if $w(e')$ was strictly less than $w(e)$, then edge e' would have been considered by the algorithm before e. And $T_{i-1} \cup \{e'\}$ is acyclic, since e' crosses out of component S_u of T_{i-1}. So e' would have been added by the algorithm: a contradiction. This proves that $w(e') \geq w(e)$, and therefore $w(T^{**}) \leq w(T^*)$. Because T^* is an MST, T^{**} must also be an MST, completing the proof by induction.

Remark: At this point the reader may be a little bit confused and ask: how can it be that tree T^{**} is lighter than T^*, when T^* is already a minimum spanning tree? The answer is: you're right. T^{**} cannot be lighter than T^*. The only way is that they must have the same weight: $w(e) = w(e')$. □

For $i = n - 1$, the claim states that tree T_{n-1}, that contains $n - 1$ edges, is a subset of an MST, meaning T_{n-1} is itself that MST. □

1.5 Next Steps / TODO

- Complete the precise **runtime analysis** of Kruskal's algorithm (evaluating the edge sorting step and the cycle-testing).

This depends on the precise choice of data structure used to implement the high-level algorithm. We defer that until later.