Minimum Spanning Tree

Problem: Given a connected graph $G = (V, E)$ with weights $w : E \rightarrow \mathbb{R}$ on the edges, find an edge subset of size $n - 1$ that connects all the vertices and has minimum weight.

E.g.

Recall: Any connected graph on $n$ vertices and $n - 1$ edges is a tree.
The edge subset is called a \text{minimum spanning tree}.

Greedy algorithms will find minimum spanning trees.

In fact, there are several possible correct greedy approaches, with different implementation challenges. E.g.

- add cheapest edge first, never build a cycle
  Kruskal’s algorithm
- grow connected graph from one vertex
  Prim’s algorithm
- throw away expensive edges, never disconnect
Kruskal’s Algorithm
Order edges by weight $e_1 \cdots e_m$
\[ w(e_i) \leq w(e_{i+1}) \]
\[ T := \emptyset \]
for $i$ from 1 to $m$ do
  if $e_i$ does not make a cycle with $T$ then
    $T := T \cup \{e_i\}$
  fi
od

$e$ makes a cycle with $T$ iff $e$ joins vertices in same connected component

e.g. edge $e$ makes a cycle $\rightarrow$ throw it out
  edge $f$ does not $\rightarrow$ add $f$ to $T$

Correctness — an exchange proof. Let $T$ have edges $t_1 \cdots t_{n-1}$.
Prove by induction on $i$ that there is a MST matching $T$ on the first $i$ edges.

Base case: $i = 0$. (Trivially true.)
Assume by induction that there is a MST $M$ matching $T$ on the first $i - 1$ edges.

\begin{align*}
\text{alg.} & \quad T \quad t_1 \quad \cdots \quad t_{i-1} \quad t_i \quad \cdots \quad t_n \\
\text{mst} & \quad M \quad m_1 \quad \cdots \quad m_{i-1} \quad m_i \quad \cdots \quad m_n
\end{align*}

Let $t_i = e = (a, b)$ and let $C$ be the connected component of $T$ containing $a$.

Note: When the algorithm considers $t_i$, all edges of weight $< w(t_i)$ have been considered, and none of them go from $C$ to $V - C$.

Look at red path in $M$ from $a$ to $b$. It must cross from $C$ to $V - C$, say on edge $e'$. Then $w(e) \leq w(e')$ by note above, so $e'$ is later in ordering.

Exchange: Let $M' = (M - \{e'\}) \cup \{e\}$.

Claim $M'$ is a MST. (Then we’re done, since $M'$ matches $T$ on $i$ edges.)

Proof 1. $M'$ is a spanning tree because it connects all vertices (replace $e'$ by blue path from $a'$ to $a$ in $M$, edge $e$, from $b$ to $b'$ in $M$) and has same number of edges.

2. $w(M') = w(M) - w(e') + w(e) \leq w(M)$ so $M'$ is a minimum spanning tree.

[Use fact: Any connected graph on $n$ vertices and $n - 1$ edges is a tree.]
Implementing and analyzing Kruskal’s Algorithm

Graph $G = (V, e)$  $|V| = n$  $|E| = m$

$O(m \log m)$ to sort edges = $O(m \log n)$ because $m \leq n^2$ so $\log m$ is $O(\log n)$.

Then we need to maintain connected components as we add edges. Also test

- if $(a, b)$ has $a, b$ in same component (don’t add edge), or
- if $(a, b)$ have different components (do add edge).

Union-Find Problem

Maintain a collection of disjoint sets. Operations:

- Find($x$) — which set contains element $x$?
- Union — unite two sets

In our case the elements are vertices and the sets are connected components of $T$, the tree so far. This Abstract Data Type has a very simple implementation that gives $O(m \log n)$ for Kruskal.

Aside: There is a fancier implementation — CS 466. Algorithm is pretty simple, analysis is hard and true run time involves the very slowly growing inverse Ackerman’s function.
Simple implementation of Union-Find.

Keep array \( S[1 \cdots n] \), \( S[i] = \) components of element \( i \) and keep linked list of elements in each set.

\[
\begin{align*}
C_1 : & \quad 1, 3, 5, 6 \\
\text{e.g. } C_2 : & \quad 2, 4 \\
C_3 : & \quad 7 \\
\end{align*}
\]

\[
\begin{array}{cccccccc}
S & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\hline
1 & 2 & 1 & 2 & 1 & 1 & 3 \\
\end{array}
\]

Find is \( O(1) \). Union — must join 2 linked lists \( O(1) \) and must rename one of the two sets so \( O(n) \) in worst case.

But renaming smaller set does better!
e.g., to unite \( C_1 \) and \( C_2 \) do \( C_1 \leftarrow C_1 \cup C_2 \) and must update \( S(2) = 1 \) and \( S(4) = 1 \)

If an element’s set number changes, then its set (more than) doubles.
This happens \( \leq \log n \) times. Therefore total renaming work is \( O(n \log n) \).

Total run time:
\[
\underbrace{O(m \log n)}_{\text{sort}} + \underbrace{O(m)}_{\text{finds}} + \underbrace{O(n \log n)}_{\text{unions}}
\]
so \( O(m \log n) \) assuming \( G \) is connected so \( m \geq n - 1 \).