Network Design Problem

- **Given**: an undirected graph $G = (V, E)$ with edge costs (weights) $c_e > 0$. Assume for now that all edge weights are distinct.
- **Find**: subset of edges $T \subseteq E$ such that $(V, T)$ is connected and the total cost of edges in $T$ is as small as possible.
- **Examples on board. Discuss applications.**
- **Call** $T \subseteq E$ a spanning tree if $(V, T)$ is a tree (connected, no cycles).
- **Claim**: in a minimum-cost solution, $T$ is a spanning tree.
- Therefore, we call this the **minimum spanning tree (MST)** problem.

Cuts

- A key to understanding MSTs is a concept called a cut.
- **Definition**: A cut in $G$ is a partition of the nodes into two nonempty subsets $(S, V - S)$.
- **Definition**: Edge $e = (v, w)$ crosses cut $(S, V - S)$ if $v \in S$ and $w \in V - S$.

Proof of Cut Property

- Suppose $T$ is a spanning tree that doesn’t include $e$. We’ll construct a different spanning tree $T'$ such that $w(T') < w(T)$ and hence $T$ can’t be the MST.
- Since $T$ is a spanning tree, there’s a $u \rightarrow v$ path $P$ in $T$. Since the path starts in $S$ and ends up outside $S$, there must be an edge $e' = (u', v')$ on this path where $u' \in S, v' \notin S$.
- Let $T' = T - \{e'\} + \{e\}$. This is still connected, since any path in $T$ that needed $e'$ can be routed via $e$ instead, and it has no cycles, so it is a spanning tree.
- But since $e$ was the lightest edge between $S$ and $V \setminus S$, $w(T') = w(T) - w(e') + w(e) \leq w(T) - w(e') + w(e') = w(T)$

Kruskal’s algorithm

- Armed with the cut property, how can we find a MST?
- Starting with an empty set of edges, which edge do you want to add first? How can you prove it is safe to add?
- What edge do you want to add next? How can you prove it is safe?
- Next?
- Where do you get stuck? How can you fix it?
- **Kruskal’s algorithm**: add edges in order of increasing weight, as long as they don’t cause a cycle.
**Kruskal’s algorithm**

Assume edges are numbered $e = 1, \ldots, m$
Sort edges by weight so $c_1 \leq c_2 \leq \ldots \leq c_m$
Initialize $T = \{\}$
for $e = 1$ to $m$
do
if adding $e$ to $T$ does not form a cycle then
$T = T \cup \{e\}$
end if
end for

Exercise: argue correctness (use cut property)

**Kruskal’s algorithm proof**

- Consider the partial spanning tree $T$ just before edge $e = (u, v)$
- Let $S$ be the connected component containing $u$
- Then $e$ crosses the cut $(S, V - S)$, otherwise it would create a cycle when added to $T$
- No other edge crossing $(S, V - S)$ has been considered yet; it could have been added without creating a cycle, and would have connected $S$ to $V - S$
- Therefore, $e$ is the cheapest edge across $(S, V - S)$, so it belongs to every MST
- So, every edge added belongs to the MST
- The final output $T$ is a spanning tree, because the algorithm will not stop until the graph is connected, and by design it creates no cycles
- Therefore, the output is a MST

**Prim's Algorithm**

- What if we want to grow a tree as a single connected component starting from some vertex $s$?
- Which edge should we add first? How can you prove it is safe?
- Which edge should we add next? How can you prove it is safe?
- Prim’s algorithm: Let $S$ be the connected component containing $s$. Add the cheapest edge from $S$ to $V \setminus S$.

**Prim's Algorithm proof**

- Consider the partial spanning tree $T$ just before edge $e = (u, v)$ is added
- Let $S$ be the connected component containing $s$
- By construction, $e$ is the cheapest edge across the cut $(S, V - S)$
- Therefore, $e$ belongs to every MST
- So, every edge added belongs to the MST
- The algorithm creates no cycles and does not stop until the graph is connected, therefore, the final output is a spanning tree
- The final output is a minimum-spanning tree

**Remove Distinctness Assumption?**

- Hack: break ties in weights by perturbing each edge weight by a tiny unique amount.
- Implementation: break ties in an arbitrary but consistent way (e.g., lexicographic order)
- This is correct. There is a slightly more principled way that requires a stronger cut property.
Implementation of Prim’s algorithm

Initialize $T = \{\}$
Initialize $S = \{s\}$

while $T$ is not a spanning tree do
Let $e = (u, v)$ be the minimum-cost edge from $S$ to $V - S$
$T = T \cup \{e\}$
$S = S \cup \{s\}$
end while

What does this remind you of?

Prim Implementation

Set $A = V$. Unattached nodes
Set $a(v) = \infty$ for all nodes. Attachment cost
Set $a(s) = 0$ Set edgeTo($s$) = null. Attachment edge

while $A$ not empty do
Nodes left to attach
Extract node $v \in A$ with smallest $a(v)$ value
Set $T = T \cup$ edgeTo($v$)
for all edges $(v, w)$ where $w \in A$ do
if $c(v, w) < a(w)$ then
$a(w) = c(v, w)$
edgeTo($w$) = $(v, w)$
end if
end for
end while

Nearly identical to Dijkstra. Priority queue for $A \rightarrow O(m \log n)$

Kruskal Implementation?

Sort edges by weight so $c_1 \leq c_2 \leq \ldots \leq c_m$
Initialize $T = \{\}$
for $e = 1$ to $m$ do
if adding $e = (u, v)$ to $T$ does not form a cycle then
$T = T \cup \{e\}$
end if
end for

Ideas?
BFS to check if $u$ and $v$ in same connected component: $O(mn)$.
(Each BFS is $O(n)$: why?)
Can we do better?

Kruskal Implementation: Union-Find

Idea: use clever data structure to maintain connected components of growing spanning tree. Should support:

- find($v$): return name of set containing $v$
- Union($A, B$): merge two sets

for $e = 1$ to $m$ do
Let $u$ and $v$ be endpoints of $e$
if find($u$) != find($v$) then
Not in same component?
union($u, v$) Merge components
end if
end for

Goal: union = $O(1)$, find = $O(\log n) \Rightarrow O(m \log n)$ overall

Union-Find Data Structure

Board work

Conclusion:
- Union is $O(1)$: update one pointer
- Find is $O(\log n)$: follow at most $\log_2(n)$ pointers to find representative of set