CS 344
LECTURE 9
MINIMUM SPANNING TREES
DIJKSTRA'S ALGORITHM

Given a graph $G$ and a starting vertex $s$, find shortest paths to all reachable vertices
procedure \texttt{dijkstra}(G,l,s) \\
\textbf{Input:} \hspace{1em} \text{Graph } G = (V,E), \text{ directed or undirected;} \\
\hspace{2em} \text{positive edge lengths } \{l_e : e \in E\}; \text{ vertex } s \in V \\
\textbf{Output:} \hspace{1em} \text{For all vertices } u \text{ reachable from } s, \text{ dist}(u) \text{ is set} \\
\hspace{3em} \text{to the distance from } s \text{ to } u. \\
\\
\text{for all } u \in V: \\
\hspace{2em} \text{dist}(u) = \infty \\
\hspace{2em} \text{prev}(u) = \text{nil} \\
\text{dist}(s) = 0 \\
\\
H = \text{makequeue}(V) \hspace{2em} \text{(using dist-values as keys)} \\
\text{while } H \text{ is not empty:} \\
\hspace{2em} u = \text{deleteMin}(H) \\
\hspace{2em} \text{for all edges } (u,v) \in E: \\
\hspace{3em} \text{if } \text{dist}(v) > \text{dist}(u) + l(u,v): \\
\hspace{4em} \text{dist}(v) = \text{dist}(u) + l(u,v) \\
\hspace{4em} \text{prev}(v) = u \\
\hspace{4em} \text{decreaseKey}(H,v)
Finally, we get this tree for paths from $A$:
Dijkstra's algorithm with negative edges
Bellman-Ford algorithm:

```plaintext
procedure shortest-paths(G,l,s)
Input: Directed graph G = (V,E);
    edge lengths \{l_e : e \in E\} with no negative cycles;
    vertex s \in V
Output: For all vertices u reachable from s, \text{dist}(u) is set
to the distance from s to u.

for all u \in V:
    \text{dist}(u) = \infty
    \text{prev}(u) = \text{nil}
\text{dist}(s) = 0
repeat |V| - 1 times:
    for all e \in E:
        update(e)
```

Bellman-Ford algorithm:

A | 0  | 0  | 0  | 0  | 0  
---|----|----|----|----|----
B | ∞  | 1  | -3 | -3 |  
C | ∞  | 4  | 3  | -1 |  
D | ∞  | 5  | 5  | 5  |  

Diagram:

- A to B: 1
- B to C: 2
- B to D: 4
- C to D: -8
GREEDY ALGORITHM

• Builds up a solution piece by piece
• Picks the best option at each stage
MINIMUM SPANNING TREE

- We want to connect all vertices in a graph with minimum cost
Note that removing a cycle edge cannot disconnect a graph.
MINIMUM SPANNING TREE

- Tree (no cycles)
- Connects all vertices (spanning)
- Minimum cost
MINIMUM SPANNING TREE

Given an undirected graph $G = (V, E)$ with edge weights $w_e$,
find a tree $T = (V, E')$, where $E' \subseteq E$,
that minimizes

$$\text{weight}(T) = \sum_{e \in E'} w_e$$
KRUSKAL'S ALGORITHM

Start with the empty set

Repeatedly add the next lightest edge that doesn't create a cycle
ASIDE: TREES

- undirected graph
- connected
- acyclic
A tree on $n$ nodes has $n - 1$ edges
NICE TREE PROPERTIES

Any connected, undirected graph with $|E| = |V| - 1$ is a tree.
NICE TREE PROPERTIES

An undirected graph is a tree iff there is a unique path between any pair of nodes.
Is Kruskal's algorithm correct?
Cut property: Suppose edges $X$ are part of a MST of $G$. Pick any subset of nodes $S$, where none of the edges in $X$ cross between $S$ and $V - S$. Let $e$ be the lightest edge across this cut. Then $X \cup \{e\}$ is part of some MST.
Then

\[ \text{weight}(T') = \text{weight}(T) + w(e) - w(e') \]

But \( w(e) \leq w(e') \), so \( \text{weight}(T') \leq \text{weight}(T) \).

Since \( T \) is an MST, the weights must be equal, and \( T' \) is also an MST.
Edges $X$: 

MST $T$: 

![Image of a graph with labeled edges and nodes, showing a tree structure with edges A-B, A-C, and B-D, and a different graph with edges C-D and E-F.]
The cut:

\[ S = \{A, B, C, D\} \quad \text{and} \quad V - S = \{E, F\} \]

\[ e = (E, F) \]

MST \( T' \):

\[ T' = \{B, D, E, F\} \]
So Kruskal's algorithm starts with $n$ trees (single vertices)

At each stage it adds an edge to connect two trees $T_1$ and $T_2$

Let the cut be $T_1$ and $V - T_1$.

Since $e$ is the lightest edge in the graph (that doesn't create a cycle), it must be the lightest edge in this cut.
- makeset(x) -- create a singleton set
- find(x) -- find which set x belongs to
- union(x, y) -- merge sets containing x and y
procedure \texttt{kruskal}(G, w)

Input: A connected undirected graph $G = (V, E)$ with edge weights $w_e$

Output: A minimum spanning tree defined by the edges $X$

for all $u \in V$:
\hspace{1em} \text{makeSet}(u)

$X = \{\}$

Sort the edges $E$ by weight

for all edges $\{u, v\} \in E$, in increasing order of weight:
\hspace{1em} if $\text{find}(u) \neq \text{find}(v)$:
\hspace{2em} add edge $\{u, v\}$ to $X$
\hspace{2em} union($u, v$)
UNION BY RANK

- Use directed trees
- Each node has a parent pointer $\pi$
- Each node has a rank (height of its subtree)
Directed tree representation of sets \( \{B, E\} \) and \( \{A, C, D, F, G, H\} \):
procedure makeset$(x)$
\[ \pi(x) = x \]
\[ \text{rank}(x) = 0 \]

function find$(x)$
while $x \neq \pi(x)$:
\[ x = \pi(x) \]
return $x$
procedure union(x, y)

r_x = find(x)

r_y = find(y)

if r_x = r_y: return

if rank(r_x) > rank(r_y):
    \( \pi(r_y) = r_x \)

else:
    \( \pi(r_x) = r_y \)

    if rank(r_x) = rank(r_y):
        rank(r_y) = rank(r_y) + 1
After makeset on $A$ through $G$: 

$A^0$ $B^0$ $C^0$ $D^0$ $E^0$ $F^0$ $G^0$
After union(A, D), union(B, E), union(C, F):
After union(C, G), union(E, A):
After union(B, G):
\[ \text{rank}(x) < \text{rank}(\pi(x)) \]
A root node of rank $k$ is created by merging two trees with rank $k - 1$, so a root node of rank $k$ has at least $2^k$ nodes in its tree (internal nodes too!)
If there are $n$ elements, there can be at most $n/2^k$ nodes of rank $k$.

So the maximum rank is $\log n$
procedure \texttt{kruskal} (G, w)

\textbf{Input:} A connected undirected graph \(G = (V, E)\) with edge weights \(w_e\)

\textbf{Output:} A minimum spanning tree defined by the edges \(X\)

for all \(u \in V\):
\[
\text{makeset}(u)
\]

\(X = \{\}\)

Sort the edges \(E\) by weight

for all edges \(\{u, v\} \in E\), in increasing order of weight:
\[
\text{if } \text{find}(u) \neq \text{find}(v):
\]
\[
\text{add edge } \{u, v\} \text{ to } X
\]
\[
\text{union}(u, v)
\]
Time for Kruskal's

- $|V|$ makeset operations
- $2|E|$ find operations
- $|V| - 1$ union operations
Time for Kruskal's

• $O(|E| \log |V|)$ for sorting edges
• $O(|E| \log |V|)$ for union and find
Can we do better?
function find(x)
if \( x \not= \pi(x) \):
    \( \pi(x) = \text{find}(\pi(x)) \)
return \( \pi(x) \)
After find(I)
After find(K)
• find operations look at the inside of the tree
• union operations look at the root
• path compression doesn't affect union or change ranks
• rank properties still hold
Ranks can vary between 0 and \( \lg n \)

- \( \{1\} \)
- \( \{2\} \)
- \( \{3, 4\} \)
- \( \{5, 6, \ldots, 16\} \)
- \( \{17, 18, 2^{16} = 65536\} \)
- \( \{65537, 65538, \ldots, 2^{65536}\} \)
- \( \ldots \)

That is, \( \{k + 1, k + 2, \ldots, 2^k\} \)
Let $\lg^* n$ be the number of lgs to bring it down to 1:

$$\lg^* 1000 = 4$$
\[
\begin{align*}
lg 1000 & \approx 9.97 \\
lg 9.97 & \approx 3.32 \\
lg 3.32 & \approx 1.73 \\
lg 1.73 & \approx 0.79
\end{align*}
\]
Some find operations may take longer than others. Give each node some cash, so the total amount is \( \leq n \lg^* n \) dollars.

Then find takes \( O(\lg^* n) \) steps plus a bit (covered by the cash)
Node gets its cash when it is no longer a root (then rank is fixed)

If rank is in \( \{k + 1, k + 2, \ldots, 2^k\} \), it gets \( 2^k \) dollars.
Number of nodes with rank $> k$ is bounded by

$$\frac{n}{2^{k+1}} + \frac{n}{2^{k+2}} + \cdots \leq \frac{n}{2^k}$$

So total amount given to nodes in this interval is at most $n$
Total amount given to nodes in this interval is at most $n$

There are $\lg^* n$ intervals

Total amount given out is $\leq n \lg^* n$
Time for a specific find operation -- number of pointers followed

Nodes $x$ on the chain to the root are in one of two categories:

- rank of $\pi(x)$ is in a higher interval than rank of $x$
- rank of $\pi(x)$ is in the same interval
• rank of $\pi(x)$ is in a higher interval than rank of $x$
• rank of $\pi(x)$ is in the same interval

At most $\lg^* n$ nodes of the first type: $O(\lg^* n)$ work

Remaining nodes pay a dollar for their processing
We require that each node has enough to cover this.

- Each time \( x \) pays a dollar, parent's rank increases
- If \( x \)'s rank is in \( \{ k + 1, k + 2, \ldots, 2^k \} \), then it pays at most \( 2^k \) dollars

Then \( m \) find operations take at most \( O(m \lg^* n) \) plus at most \( O(n \lg^* n) \)
In general, this form of greedy scheme will work:

\[ X = \{ \} \text{ (edges picked so far)} \]
repeat until \( |X| = |V| - 1 \):
    pick a set \( S \subseteq V \) for which \( X \) has no edges between \( S \) and \( V - S \)
    let \( e \in E \) be the minimum-weight edge between \( S \) and \( V - S \)
    \[ X = X \cup \{e\} \]
PRIM'S ALGORITHM

- Edge set $X$ is always a subtree of $G$
- $X$ grows by one (lightest) edge each time

(or think of $S$ as growing to include the cheapest next vertex)
procedure prim\((G, w)\)

Input: A connected undirected graph \(G = (V, E)\) with edge weights \(w_e\)

Output: A minimum spanning tree defined by the array prev

for all \(u \in V:\)
    cost\((u)\) = \(\infty\)
    prev\((u)\) = nil

Pick any initial node \(u_0\)

\(\text{cost}(u_0) = 0\)

\(H = \text{makequeue}(V)\)  \((\text{priority queue, using cost-values as keys})\)

while \(H\) is not empty:
    \(v = \text{deletemin}(H)\)
    for each \(\{v, z\} \in E:\)
        if \(\text{cost}(z) > w(v, z)\):
            \(\text{cost}(z) = w(v, z)\)
            \(\text{prev}(z) = v\)
            \(\text{decreasekey}(H, z)\)
<table>
<thead>
<tr>
<th>Set $S$</th>
<th>$A$</th>
<th>$B$</th>
<th>$C$</th>
<th>$D$</th>
<th>$E$</th>
<th>$F$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\emptyset$</td>
<td>0/nil</td>
<td>$\infty$/nil</td>
<td>$\infty$/nil</td>
<td>$\infty$/nil</td>
<td>$\infty$/nil</td>
<td>$\infty$/nil</td>
</tr>
<tr>
<td>$A$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$A, D$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$A, D, B$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$A, D, B, C$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$A, D, B, C, F$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>