Problem can also be defined for directed graphs...

Minimum Spanning Tree (MST)

- A tree (connected acyclic graph) that includes every node, and minimizes the total sum of edge weights

Problem can also be defined for directed graphs...

Application: Internet Backbone Planning

- Want to connect n cities with internet backbone links
  - Direct links possible between each pair of cities
  - Each link has a certain dollar cost (excavation, materials, distance & time, legal costs...)
  - Want to minimize total cost

Application: Image Segmentation

Images are easier for a machine learning algorithm to understand.

Break edges into large, highly similar segments, and assign the dominant color to each segment

APPLICATION: CURVILINEAR FEATURE EXTRACTION

Want some way to recognize this object

Edge detection algorithm

Find result

APPLICATION: INTERNET BACKBONE PLANNING

Want to connect n cities with internet backbone links

- Direct links possible between each pair of cities
- Each link has a certain dollar cost (excavation, materials, distance & time, legal costs...)
- Want to minimize total cost

APPLICATION: IMAGE SEGMENTATION

- Images are easier for a machine learning algorithm to understand.
- Break edges into large, highly similar segments, and assign the dominant color to each segment

APPLICATION: CURVILINEAR FEATURE EXTRACTION

Want some way to recognize this object

Edge detection algorithm

Find result

APPLICATION: INTERNET BACKBONE PLANNING

Want to connect n cities with internet backbone links

- Direct links possible between each pair of cities
- Each link has a certain dollar cost (excavation, materials, distance & time, legal costs...)
- Want to minimize total cost
If you add an edge $e$ to a tree and this creates a cycle $C$, then removing any other edge $e' \in C$ will break the cycle and produce a tree.

Suppose we have an oracle $\text{CreateCycle}(e_j)$.

Increasing edge weights: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20

8 would create a cycle: a, c, b, d, a
11 would create a cycle: d, e, b, d
14 would create a cycle: c, f, e, b, d, a, c
15 would create a cycle: g, f, h, j, l, g
16 would create a cycle…
17 would create a cycle…
18 would create a cycle…
19 would create a cycle…

Done!

To avoid strange/long names, keep one of the original set names.

Both endpoints already in same set! Do not add.
Suppose $K$ is not an MST for contradiction. Let $O$ be an optimal MST. Note $O \neq K$.

Let $\omega(\mathcal{O}) = \min\{\omega(f_1), \omega(f_2), \ldots, \omega(f_n)\}$. Since $O$ is optimal, for any distinct $\omega(f_i), \omega(f_j)$, we have $\omega(f_i) > \omega(f_j)$.

Let $f_n - 1$ be the first edge in $O \setminus K$. Adding $f_n - 1$ to $O$ would create a cycle $C$.

Let $e' = \min\{\omega(e) \in C \setminus K\}$. Let $\mathcal{O}'$ be the same as $\mathcal{O}$ but with $e'$ and $f_n - 1$ swapped.

Note $\omega(\mathcal{O}') = \omega(\mathcal{O}) + \omega(f_n - 1) - \omega(e') \geq \omega(\mathcal{O})$ since $O$ is optimal. So $\omega(f_n - 1) - \omega(e') \geq 0$, so $\omega(f_n - 1) > \omega(e')$.

Kruskal considers $e'$ before $f_n - 1$, and rejects $e'$ despite taking $f_1, \ldots, f_{n-1}$. So, $f_1, \ldots, f_{n-1}, e'$ contains a cycle $C'$.

But $f_1, \ldots, f_{n-1}, e' \in \mathcal{O}$, contradiction!

Need to know runtime for union find...

**Problem:** Slow find()

**Union-Find with Union by Rank**

- Keep track of heights of trees
- Make root with greater height be the parent
- Union of two trees with height $h$ has height $h + 1$
- Union of tree with height $h$ and tree with height $< h$ has height $h$
- Runtime with union by rank?

**Runtime of Union by Rank**

- Can prove the following lemma by induction:
  - Each tree of height $h$ contains at least $2^h$ nodes

Case 1: trees of different height

By I.H., left tree already has $\geq 2^h$ nodes. So result has height $h$ and $\geq 2^h$ nodes.
**RUNTIME OF UNION BY RANK**

- Can prove the following lemma by induction:
  - Each tree of height \( h \) contains at least \( 2^h \) nodes

**Case 2: trees of same height**

By IH, each tree has \( \geq 2^h \) nodes. Result has height \( h+1 \) and \( \geq 2^h + 2^h \) nodes. And \( 2^h + 2^h = 2^{h+1} \). QED

**TIME COMPLEXITY USING UNION BY RANK**

```
Kruskal(V, E, F): // decreasing order by weight
uf = new UnionFind data structure
uf = ufUF(V, E, F)[0]
for i = 1 to |E|
    set a = uf.find(i); source = uf.parent(a)
    set b = uf.find(i); target = uf.parent(b)
    if set a = set b
        err()
    else
        add E[i] to set a
        uf.union(set a, set b)
return uf
```

Total \( \Theta(n \log n) \) in time

\( \log n \leq \log \log n \leq 2 \log n \leq \log n + 1 \).

So runtime is in \( \Theta(n \log n) \).

**MAKING THIS EVEN FASTER**

- In addition to union by rank, union-find can be implemented with path compression

```
Path compression
```

Using both union by rank and path compression, we get a total runtime of \( \Theta(n \log n) \) for Kruskal's algorithm of \( \Theta(n \log n) \) for Kruskal's algorithm of \( (\log n) \).

**Efficient Union-Find**

```
def UnionFind():
    uf = [0] * len(V)
    rank = [0] * len(V)
    return uf
```

**OTHER NOTABLE MST ALGORITHMS**

- Prim's algorithm
  - Incrementally extend a tree \( T \) into an MST, by:
  - Initializing \( T \) to contain an arbitrary node in \( G \)
  - Repeatedly selecting the smallest weight edge from any node in \( T \) to any node outside of \( T \)
  - Visualization: [https://www.cs.utexas.edu/~galles/visualization/Prim.html](https://www.cs.utexas.edu/~galles/visualization/Prim.html)

- Borůvka's algorithm
  - Like Kruskal (merging components), but with phases
  - In each phase, select an outgoing edge for every component, and add all edges found in the phase
A FUN APPLICATION: MAZE BUILDING

- Create grid graph with edges up/down/left/right
- Randomize edge weights then run Kruskal's

VISUALIZING KRUSKAL'S (WITHOUT PATH COMPRESSION)

- [https://www.cs.usfca.edu/~galles/visualization/Kruskal.html](https://www.cs.usfca.edu/~galles/visualization/Kruskal.html)