CS 341: ALGORITHMS
Lecture 15: graph algorithms IV – minimum spanning trees
Readings: see website
Trevor Brown
https://student.cs.uwaterloo.ca/~cs341
trevor.brown@uwaterloo.ca

Consider an undirected graph in which each edge has a weight (or cost)

We can also define the problem for directed graphs.

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

Problem can also be defined for minimum spanning forests.

Applications:
- Internet Backbone Planning
- Image Segmentation
- Curvilinear Feature Extraction

Segments are easier for a machine learning algorithm to understand.
USEFUL TREE FACTS

- A tree on $n$ vertices has $n - 1$ edges.
- There is a unique path between any two vertices in a tree.
- If $T$ is a tree and an edge $e \not\in T$ is added to $T$, then the resulting graph contains a unique cycle $C$.
- If $e' \in C$ then $T \cup \{e'\} \setminus \{e\}$ is a tree.

EXAMPLE EXECUTION

7 would create a cycle... 11 would create a cycle... 14 would create a cycle... 15 would create a cycle...
8 would create a cycle: $a, c, b, d, a$
9 would create a cycle: $d, e, b, d$
10 would create a cycle: $c, f, e, b, d, a, c$
16 would create a cycle...

UNION FIND

Represents a partition of set $S = \{e_1, e_2, ..., e_n\}$ into disjoint subsets
- Initially $n$ disjoint subsets $S_i = \{e_i\}$
- Operations
  - Union($S_i, S_j$) replaces $S_i$ and $S_j$ by their union $S_i \cup S_j$.
  - Find($e_i$) returns the label of the set containing $e_i$.

PSEUDOCODE FOR KRUSKAL’S USING UNION-FIND

Kruskal($G[1..n], E[1..m]$)
  1. sort $E[1..m]$ in increasing order by weight
  2. u = new UnionFind data structure
  3. set $S$ = new List
  4. for $j = 1$ to $m$
  5.   set $a = uf$.find($E[j]$.source)
  6.   set $b = uf$.find($E[j]$.target)
  7.   if Set $a$ = set $b$
  8.     $uf$.add($E[j]$)
  9.   else
  10.     $uf$.merge($set_a$, set_b)
  11. return $m$
CORRECTNESS
Suppose $K$ is not an MST, for contradiction. Let $O$ be an (optimal) MST. Note $O = K$.

Let $e$ be a smallest edge in $C$. Let $f$ be a shortest edge in $K$. There is an edge $e'$ in $C$ with $e' = e$.

Adding $f$ to $O$ would create a cycle $C$.

Kruskal considers $e'$ before $f$, and rejects $e'$ despite looking $(e', e)$.

So, $e = e'$ contains a cycle $C$.

Contradiction!

UNION FIND IMPLEMENTATION
Suppose we are partitioning set $\{1, \ldots, n\}$ into subsets $S_1, \ldots, S_m$.

- Represent the partition as a forest of trees
  - Initially one single-node tree per subset
  - Each node has a parent pointer

- $\text{Find}(i)$ returns the root of the tree containing element $i$
- $\text{Union}(i, j)$ makes one root the parent of the other

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?

TIME COMPLEXITY?

$\text{Kruskal}(V \cup E, E[\cdot,\cdot])$
- Sort $E[\cdot,\cdot]$ in increasing order by weights
- $uf =$ new UnionFInd data structure
- $set =$ new List
- for $i = 1 \ldots \text{Length}(E)$:
  - $set.a =$ uf.find($E[i].\text{source}$)
  - $set.b =$ uf.find($E[i].\text{target}$)
  - if $set.a \neq set.b$:
    - $uf.$add($K[i])$
    - uf.merge($set.a, set.b$)

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 I.H., each tree has \( \geq 2^h \) nodes. Result has height \( h+1 \) and \( \geq 2^{h+1} \) nodes

And \( 2^h + 2^h = 2^{h+1} \). QED

RUNTIME OF UNION BY RANK

- How does the lemma help?
  - Each tree of height \( h \) contains at least \( 2^h \) nodes
  - There are only \( n \) nodes in the graph
  - So height is at most \( \log n \)
    (Lemma: a tree of height \( \log n \) contains at least \( 2^{\log n} \) nodes and \( 2^{\log n} = n \))

So the longest path in the union-find forest is \( \log n \)

So all union-find operations run in \( \Theta(\log n) \) time!

TIME COMPLEXITY USING UNION BY RANK

\[
\text{Total } \Omega(m \log n) = \log n \log(n + 2) = \log^2 n \Rightarrow \text{So runtime is in } \mathcal{O}(\log^2 n)
\]

MAKING THIS EVEN FASTER

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

EFFICIENT UNION-FIND

OTHER NOTABLE MST ALGORITHMS

- Prim’s algorithm
  - Incrementally extend a tree \( T \) into an MST, by:
    - Initializing \( T \) to contain any 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.unlv.edu/~gates/visualization/Prim.html](https://www.cs.unlv.edu/~gates/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

Efficient Union-Find

- Use priority queue to store outgoing edges from \( T \) (and repeatedly extract the minimum weight one)

There is also a fast parallel hybrid of Prim and Borůvka
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