Admin: Reminder - Quiz in class 10/19

Today: Greedy algorithms & MST
(minimum spanning trees)

- Defs
- Greedy choice theorem for MST
- Prim's alg
- Kruskal's alg
  (Union Find data structure)
Review:

**Undirected graph** $G = (V, E)$
- $V =$ vertices
- $E =$ edges (unordered pairs of vertices)

Assume adjacency-list representation:
- Give, for each vertex $u$, list $\text{Adj}[u]$ of $u$'s neighbors $= \{ v \mid (u, v) \in E \}$

![Graph Example](image)

$\text{Adj}[c] = \{ a, d \}$

**Weighted undirected graph:** $G = (V, E)$
- with weight function $w: E \to \mathbb{R}$

**Tree:** graph with no cycles & is connected

**Spanning Tree:** (of graph $G = (V, E)$)
- a subset $T \subseteq E$ that forms a tree
- that spans graph (touches all vertices)

**Fact:** Spanning tree has $|V| - 1$ edges.
Minimum Spanning Tree (MST) Problem:

Given: undirected connected graph \( G = (V, E) \)
Find: a spanning tree \( T \subseteq E \)
of minimum weight \( w(T) = \sum_{e \in T} w(e) \)

Example:

![Graph representing the minimum spanning tree](image)

Minimum = MST

Many applications:

For example, connecting cities with minimum amount of fiber-optic cable.

Fact: If there are edge-weight ties, MST may not be unique.

(But no ties \( \Rightarrow \) MST is unique)
Trying to find

best subset of a given set

that is legal

(min weight) (T) (CE) (connected & acyclic)
i.e. spanning tree

"Greedy approach" may work: pick elements of $T$ one after another according to some local "greedy" (myopic) method.

This works if:

(a) you can identify easily (in "greedy" manner) some edge that is in an MST
(b) after committing to that edge, remaining problem has same form.

($\equiv$ "optimal substructure")
In our example, suppose we had reason to believe that $e = (u, v)$ was in some optimal MST $T^*$.

We can then "contract" $e$ to make a new graph $G' = G/e = (V', E')$ with "supernode" $uv$

$$w((g, uv)) = \min(w((g, u)), w((g, v)))$$

$$= \min(14, 8) = 8$$

(Node inside supernode already connected internally, so connecting to $u$ as good as connecting to $v$; take cheapest. But keep track that $(g, uv)$ "comes from" $(g, v)$.)
Let $T'$ be MST for $G'=(V',E') = G/e$

Claim: $T = T' \cup \{e\}$ is MST for $G$

given that $e$ is in some MST $T^*$ for $G$
and where edges in $T'$ interpreted in
"pre-contracted" form.

Proof:

$T^*/e$ is spanning tree of $G'$.

$\Rightarrow w(T') \leq w(T^*/e)$

$\Rightarrow w(T) = w(T') + w(e)$

$\leq w(T^*/e) + w(e)$

$= w(T^*)$

$\Rightarrow T$ is MST
Thus, if we have procedure for picking an edge $e$ we know to be in some MST, we can commit to it, & solve remaining problem on contracted graph $G' = G/e$.

Grow $T$ edge by edge, Supernodes correspond to connected components so far.
Def. A cut of $G = (V, E)$ is a partition of $V$ into two nonempty subsets:

\[ V = S \cup T \]

\[ S \neq \emptyset \]
\[ T \neq \emptyset \]
\[ S \cap T = \emptyset \]

What is our "trick" for being able to make greedy choice (pick an edge $e$ that is in some MST $T^*$)?

**Theorem:** Let $(S, T)$ be any cut of $G = (V, E)$.

Let $e$ be any edge crossing cut (i.e., $e=(u,v)$ where $u \in S$ & $v \in T$) of minimum weight.

Then $(\exists$ MST $T^*)$ $e \in T^*$.

(Illustrate on our example. Note that $S$ could contain a single vertex, but need not.)
Proof:
- Let $T$ be an MST for $G$.
- If $e \in T$ we are done (take $T^* = T$).
  
  So, suppose $e \notin T$.
- Let $e = (u, v)$ ($u \in S$, $v \in T$)
- There is path from $u$ to $v$ in $T$
- Let $e' = (u', v')$ be first edge on path crossing cut.
  (path must cross cut, since $u \in S$, $v \in T$)
- Let $T^* = T \setminus \{e'\} \cup \{e\}$
- $T^*$ is a spanning tree of $G$  
  (Any path that used $e' \in T$
  can be restructured to use
  $e$ instead.)
- $w(e) \leq w(e')$
- $w(T^*) = w(T) - w(e') + w(e) \leq w(T)$

$\Rightarrow T^*$ is MST too.
Prim's Algorithm:

Grow single supernode until it includes all of V:

\[ S = \{v_0\} \quad \text{arbitrary seed point} \]
\[ T = \emptyset \quad \text{empty tree} \]

while \( S \neq V \):

\[ \text{Let } e \text{ be cheapest edge crossing from } S \text{ to } V - S \]
\[ e = (u, v) \]
\[ \text{Add } v \text{ to } S \]
\[ \text{Add } e \text{ to } T \]

stop: \( T \) is MST

Correctness: Follows from previous theorem

(By induction, \( T \) is always subset of some MST \( T^* \))
Efficient Implementation:

- Keep priority queue \( Q \) on \( V - S \),
  where \( v.\text{key} = \min(w(e)) \) where \( e = (u, v) u \in S \)
  (or \( \infty \) if there are no such edges)
  \( Q \) supports Extract-Min & Decrease-Key

- Initialize \( Q \) with \( V \)
  initialize \( v.\text{key} = \infty \) (except \( v_0.\text{key} = 0 \))

- while \( Q \neq \emptyset \):
  \( v = \text{Extract-Min}(Q) \)
  for \( x \in \text{Adj}[v] \):
    if \( x \in Q \land w(v, x) < x.\text{key} \):
      \( x.\text{key} = w(v, x) \)
      Decrease-Key \((Q, x)\)
      \( x.\text{parent} = v \)

return \( \{(v, v.\text{parent}) : v \in V - \{v_0\}\} \) as MST \( T \)
Example:
\[ \text{Time:} \]
\[ = \Theta(v) \cdot T_{\text{Extract-Min}} \]
\[ + \Theta(E) \cdot T_{\text{Decrease-key}} \]

<table>
<thead>
<tr>
<th>Priority queue</th>
<th>( T_{\text{Extract-Min}} )</th>
<th>( T_{\text{Decr.-key}} )</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>array</td>
<td>( O(v) )</td>
<td>( O(1) )</td>
<td>( O(v^2) )</td>
</tr>
<tr>
<td>binary heap</td>
<td>( O(\log v) )</td>
<td>( O(\log v) )</td>
<td>( O(E \log v) )</td>
</tr>
<tr>
<td>Fibonacci heap</td>
<td>( O(\log v) ) \text{ amortized}</td>
<td>( O(1) )</td>
<td>( O(E + V \log v) )</td>
</tr>
</tbody>
</table>
Kruskal's MST Algorithm:

- grows forest instead of single tree, until trees get connected into single tree
- $T = \emptyset$
  - sort $E$ into increasing order by weight
  - for each edge $e = (u, v)$ in turn:
    - if $u$ & $v$ in different components of $T$
      - add $e$ to $T$
      - (merge their components)

- Correctness: By Theorem cheapest edge out of any vertex is in some MST. Cheapest edge out of any component is also in some MST.
Union-Find problem (Chap 21):

- maintain a collection of disjoint sets

- Operations:
  - Make-Set(x) — create set \( \{x\} \)
  - Find-Set(x) — return set containing \( x \)
  - Union(x, y) — merge sets containing \( x, y \)
    (or destroy old sets)

- Best alg:
  - Make-Set(x)
  - \( x \)
  - Find-Set(x)
    \[ \Rightarrow \varepsilon \]
    (path compression)
  - \( \Rightarrow \)
    add shortcut
    to replace upward links
    from \( x \) (\& its ancestors) to \( \varepsilon \)
  - Union(x, y):
    \[ x \rightarrow y \]
    (or reverse, depending
    on "ranks" of \( x, y \))
Running Time: (Union-Find)

$\Theta(\alpha(n))$ amortized cost/opn

where $\alpha(n)$ is extremely slowly growing

(inverse of Ackerman's function) $\alpha(n) = \Theta(1)$ "almost".

Running Time (Kruskal):

$T_{\text{Sort}}(E) + \Theta(V) \cdot T_{\text{make set}} + \Theta(E) \cdot T_{\text{find set}} + \Theta(V) \cdot T_{\text{union}}$

if weights are integers & small, can sort in time $\Theta(E)$ & beat Prim

But in general $\Theta(E \log E)$

Best MST algorithm:

$O(V+E)$ expected time (randomized alg)

[Karger, Klein, Tarjan 1993]