Greedy Algorithms (continued)

Last Time
- The student’s Problem.
- MST
- The Cut Property
- Kruskal’s Algorithm

Today
- Finish Kruskal.
- Prim’s Algorithm.
- Compression - Huffman Coding.
- Set Cover.

Recap:
MST: Given an undirected graph $G = (V, E)$ with weights $w: E \rightarrow \mathbb{N}$, find a tree $T$ that connects all vertices in $V$ and minimizes $\sum_{e \in E(T)} w(e)$.

"The Cut Property":
- The cheapest edge crossing a cut $(S, V \setminus S)$ appears in some MST.
- More refined: Let $X \subseteq E$ and an MST $T$ such that $X \subseteq E(T)$. Suppose that $X$ doesn’t cross a cut $(S, V \setminus S)$. If $e$ is the cheapest edge crossing $(S, V \setminus S)$ then $X \cup \{e\}$ is contained in some MST $T'$.
Meta Algorithm

- $X \leftarrow \emptyset$
- For $i = 1, \ldots, |V|-1$:
  - Find a cut $(S, V \setminus S)$ s.t. $X$ doesn't cross it.
  - Add cheapest edge in the cut to $X$.

Kruskal(G, w):

1. $X \leftarrow \emptyset$,
2. Sort edges by their weight.
   a. For every $v \in V$, make set $(\{v\})$.
3. For all edges $e \in E$, ordered by weight:
   - If adding $e$ to $X$ creates a cycle, then skip
   - Else, $X \leftarrow X \cup e$.

Last Time: We saw the PoC for Kruskal, based on the Cut Property.

Implementation / Runtime?

Naive Implementation

Runtime: $n$ makesets $m$ finds $n-1$ unions $O(i)$ $O(\log v)$ $O(\log v)$ $O(E \log E + E \log |V|)$.

$\Rightarrow$ Total runtime: $O(E \log E)$. 
**Meta Algorithm**

- \( X \leftarrow \emptyset \)
- For \( i = 1, \ldots, |V| - 1 \):
  - Find a cut \((S, V \setminus S)\) such that \( X \) doesn’t cross it.
  - Add the cheapest edge in the cut to \( X \).

---

**Prim’s Algorithm**

**Idea:** At each point in time, \( X \) would be a subtree.

- At each step, pick the minimal edge between
  \( S = \{ v : X \text{ touches } v \} \)
  and its complement.

**How to implement?**

Similar to Dijkstra’s Alg.
- \( \text{prev}(v) = \arg\min_{u \in S} w(u, v) \)
- \( \text{cost}(v) = \min_{u \in S} w(u, v) \)

Maintain \( A \forall v \in S \)

Pick \( v \in S \) with smallest cost.
Add \((v, \text{prev}(v))\) to \( X \).
Data Compression

We have an alphabet of 32 characters & frequencies \( f_1, \ldots, f_{32} \). Say we have a text with \( N \) characters and we want to encode it as efficiently as possible in binary.

**Naive:** Every character \( \rightarrow 5 \) bits  
**Overall:** \( 5N \) bits

---

**Example:**

<table>
<thead>
<tr>
<th>Freq</th>
<th>Encoding 1</th>
<th>Cost 1</th>
<th>Encoding 2</th>
<th>Cost 2</th>
<th>Encoding 3</th>
<th>Cost 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.4</td>
<td>00</td>
<td>0.4N \cdot 2</td>
<td>0</td>
<td>0.4N \cdot 1</td>
<td>0</td>
<td>0.4N \cdot 1</td>
</tr>
<tr>
<td>0.3</td>
<td>01</td>
<td>0.3N \cdot 2</td>
<td>1</td>
<td>0.3N \cdot 2</td>
<td>10</td>
<td>0.3N \cdot 2</td>
</tr>
<tr>
<td>0.2</td>
<td>10</td>
<td>0.2N \cdot 2</td>
<td>00</td>
<td>0.2N \cdot 3</td>
<td>110</td>
<td>0.2N \cdot 3</td>
</tr>
<tr>
<td>0.1</td>
<td>11</td>
<td>0.1N \cdot 2</td>
<td>01</td>
<td>0.1N \cdot 3</td>
<td>111</td>
<td>0.1N \cdot 3</td>
</tr>
</tbody>
</table>

\( 2N \cdot 2 + 0.9N \) bits
Prefix Free Codes & Trees

Any prefix-free code corresponds to a binary tree & vice versa.

Example:

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>C</th>
<th>G</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Prefix-free encoding:

Given symbol frequencies \( f_1, \ldots, f_n \), find the best prefix-free code.

\[
\text{cost}(\text{tree}) = \sum_{i=1}^{n} f_i \cdot \text{(depth of symbol } i \text{ in the tree)}
\]

Cost: Associate with every internal node \( v \), \( f_v = \sum_{u \text{ descendants of } v} f_u \)

Cost of prefix tree:

\[
\sum_{\text{internal node } v} f_v
\]
Greedy: which nodes to merge first.

Example: (A, 0.4) (C, 0.1) (T, 0.2) (G, 0.3)

(A, 0.4)

(C, 0.1)

(T, 0.2)

(G, 0.3)

6.3 (S, 0.3)

0.6 (S, 0.6)

Huffman: There's an optimal tree where the two lowest freq are siblings.

Proof: toLoc an optimal tree is a full-binary tree.

\( f_1 \leq f_2 \leq f_3 \ldots \leq f_n \)

swap(1, u) \( \not\equiv \) either improves value or maintains the same value.

swap(2, v)
Claim: Huffman's strategy gives an optimal prefix-free tree.

Proof: By induction on $n$.

Base case: $n=1$, $n=2$  

\[ \text{Induction step: Let } T \text{ be an optimal tree s.t. } \]
\[ f_i \text{ and } f_2 \text{ are siblings. (Assuming } t_1 \leq f_2 \leq f_3 \text{ -- )} \]

Huffman merge $f_i$ & $f_2$ and then solve recursively on $f_3, f_4, \ldots, f_n$.  

\[ \Rightarrow \text{ Huffman on } n \text{ freq.} \]
Set Cover

**Input:** Universe $U = \{1, 2, 3, \ldots, n\}$
Collection of subsets $S_1, S_2, S_3, \ldots, S_m \in U$

(s.t. $S_1 \cup S_2 \cup \ldots \cup S_m = U$)

**Output:** Minimal subcollection that covers $U$.

Minimal Size $J \subseteq [m]$\text{ s.t. } \bigcup_{j \in J} S_j = U$

For Example:

- $S_1 = \{1, 2, 3\}$
- $S_2 = \{3, 4\}$
- $S_3 = \{2, 3, 4\}$

$$S_1 \cup S_2 \cup S_3 = \{1, 2, 3, 4\}$$

- $J = \{1, 3\}$
- $S_1 \cup S_3 = \{1, 2, 3, 4\}$

Greedy strategy? Pick at any time the set that covers most new points.
Algorithm:

1. $J \leftarrow \emptyset$.

2. While $S_J \neq U$:

   Pick $i \in J$ with largest $|S_i \setminus S_J|$ (covers the most new points)

   Add $i \in J$.

3. Output $J$.

Is it correct? No.

Counterexample:

Greedy strategy:

Greedy: will pick all sets $\Rightarrow |J| = 6$.

Optimal solution: 5.
Claim: “Greedy solution is not too bad”:
If optimal solution uses $k$ sets, then greedy finds $J$ with $|J| \leq k \cdot \ln(n) + 1$. 