Minimum Spanning Tree

Graph $G(V, E)$. Undirected.

$V = \{1, 2, 3, 4\}$.

$E = \{(1, 2), (2, 3), (3, 4), (2, 4)\}$

- $n$ vertices. Number of edges $\leq \binom{n}{2}$.
- Path. "Walk" if vertices allowed to repeat.
- (Simple) cycle.
- Connected.

Graph that is not connected.

(conn) components
Def. A tree is a connected graph with no cycles.

\[ n = 4 \quad \begin{array}{c}
\text{Graph 1}
\end{array} \]

\[ n = 5 \quad \begin{array}{c}
\text{Graph 2}
\end{array} \]

Note A tree with \( n \) vertices has \( n-1 \) edges.

Def. A spanning tree of a connected graph \( G(V, E) \) is a graph
- \( T(V, E') \) such that
- \( E' \subseteq E \) and \( T \) is a tree.

Minimum Spanning Tree Problem
- Given a graph \( G(V, E) \) and
- for each edge \( (u, v) = e \), cost \( c_e \geq 0 \),
- find a spanning tree \( T \) with min cost:

\[
\text{cost}(T) = \sum_{e \in T} c_e
\]
Idea: Start with empty graph. Add edges one by one, starting with minimum cost edge, without introducing cycles.

[Drawing of a graph with numbers on the edges and vertices.]

[Skip 7, 10.]

Done!

[Ignore 20, 25.]

[Skip 14, 16.]
Algorithm
Let \( m = \# \text{edges} \).
- Sort edges according to their cost.
  \[ e_1, e_2, e_3, \ldots, e_m \]
  \[ c_1 \leq c_2 \leq c_3 \leq \ldots \leq c_m \]
- Start with graph \( H \) with no edges.
- For \( i = 1, 2, 3, \ldots, m \),
  - Add \( e_i \) to \( H \) if it does not introduce a cycle.

Theorem
The algo. produces a M.S.T.

Proof idea
We show that for every \( i \), decisions made up to \( i^{th} \) step are correct in the sense that there exists a (hypothetical) M.S.T. \( T \) that among the edges \( \{e_1, e_2, \ldots, e_i\} \) includes precisely those edges that are selected by the algorithm.
Warm-up \( i = 1 \). Since algo. selects \( e_1 \), we need to show that \( \exists \text{ M.S.T. } T \) such that \( e_1 \in T \).

**Proof** Exchange argument.

- Let \( T' \) be hypothetical M.S.T.
- If \( e_1 \in T' \), done.
- So assume \( e_1 \notin T' \).
- \( \therefore T' \cup \{e_1\} \) contains a cycle. Let \( e_j \), \( j > 1 \), be any other edge on the cycle.
- Note \( \text{cost}(e_1) \leq \text{cost}(e_j) \).
- Let \( T = T' \cup \{e_1\} \setminus \{e_j\} \).
- \( \text{cost}(T) \leq \text{cost}(T') \).
- \( \therefore T \) is optimal and contains \( e_1 \). Done!

(M.S.T.)
Formal claim: Let $A_i = \{e_1, e_2, \ldots, e_i\}$.

Let $S_i =$ Set of edges selected by algo. from $A_i$ (i.e., after examining $e_1, \ldots, e_i$).

Then there exists (hypothetical) M.S.T. $T$ such that $T \cap A_i = S_i$.

This holds for $0 \leq i \leq m$.

Note: Setting $i = m$, one concludes that the algo. outputs a M.S.T. $S_m = T \cap A_m = T$.

Proof:

For $i = 0$, $T \cap \emptyset = \emptyset$. Nothing to prove.

Inductive step: Suppose that for $i \leq m-1$, there is M.S.T. $T'$ such that $T' \cap \{e_1, e_2, \ldots, e_i\} = S_i$.
Consider the edge $e_{i+1}$.

**Case 1.** $e_{i+1}$ introduces an edge in $H(V, S_i)$.
- $\therefore$ Alg. skips $e_{i+1}$, $S_{i+1} = S_i$
- $S_i \subseteq T'$, $T'$ is a tree, $\therefore e_{i+1} \notin T'$

$\therefore e_{i+1}$ is neither in $S_{i+1}$ nor in $T'$.

$\therefore T' \cap \underline{\{e_1, \ldots, e_{i+1}\}} = S_{i+1}$.
(A) same M.S.T., $T'$ works.

**Case 2.** $e_{i+1}$ does not introduce a cycle in $H(V_i, S_i)$.
- $\therefore$ Alg. selects $e_{i+1}$, $S_{i+1} = S_i \cup \{e_{i+1}\}$.

**Sub-case (I)** If $e_{i+1}$ appears in $T'$ as well, then same $T'$ works,

$T' \cap \underline{\{e_1, \ldots, e_{i+1}\}} = S_{i+1}$.
Sub-case (2) $e_{i+1}$ does not appear in $T'$.

- Let $C$ be the cycle in $T' \cup \{e_{i+1}\}$.
- Since $C$ is not contained in $S_i \cup \{e_{i+1}\}$ otherwise the algorithm would not select $e_{i+1}$.
- $C$ must contain an edge $e_j$ s.t. $j > i+1$.
  and hence $\text{cost}(e_j) \geq \text{cost}(e_{i+1})$.

$T = T' \cup \{e_{i+1}\} \setminus \{e_j\}$

is M.S.O. with

$T \cap \underbrace{\{e_i, \ldots, e_{i+1}\}}_{A_{i+1}} = S_{i+1}$. Done. $\Box$
Implementation (Kruskal's Algorithm)

- Keep connected components as sets.
- (Union-Find data structure)
- Maintain sets as trees.
- Merge \((T_1, T_2)\) := If \(|T_1| \geq |T_2|\) then make root of \(T_2\) a child of root of \(T_1\).

- Exercise Prove that height (of all trees) remains \(\leq O(\log n)\).

- Runtime \(O(m \cdot \log n)\).
Huffman Codes

\[ \begin{align*}
    a & \rightarrow 10 \\
    b & \rightarrow 111 \\
    c & \rightarrow 100 \\
    d & \rightarrow 001 \\
    e & \rightarrow 01 \\
\end{align*} \]

abcd \rightarrow 10111100001

Ambiguity.

\[
\begin{array}{c}
\text{10001} \\
\hline
\text{ce} \\
\text{ad}
\end{array}
\]

**Def** Given an alphabet \( \Sigma = \{x_1, x_2, \ldots, x_n\} \),
a code is a map \( C : \Sigma \rightarrow \{0,1\}^* \).

**Note** Code must be unambiguous.

**One solution** - Let \( k = \lceil \log n \rceil \).
- \( C : \Sigma \rightarrow \{0,1\}^k \), all encodings of same length.

However a,e occur more frequently than b,c,d. Can we be more efficient?

Alternate \( C \) is prefix-free, i.e. for \( i \neq j \)
\( C(x_i) \) is not a prefix of \( C(x_j) \).
\[ \Rightarrow \text{unambiguous} \]
Problem: Given $\Sigma = \{x_1, x_2, \ldots, x_n\}$ and frequencies $f_1, f_2, \ldots, f_n$, find a prefix-free code $C: \Sigma \to \{0, 1\}^*$ so as to minimize $\sum_{i=1}^{n} |C(x_i)| \cdot f_i$.

Prefix-free codes $\equiv$ Binary trees

Binary tree $\Rightarrow$ Prefix-free code

Prefix-free code $\Rightarrow$ Binary tree

11, 011, 010, 001, 000

Could have shrunk to
- W.l.o.g. we restrict to full binary tree, i.e., every node is either a leaf or has two children.

Problem Find a Full Binary Tree with
- \( n \) leaves \( \{x_1, \ldots, x_n\} \)
- A bijection from \( \Sigma \) to set of leaves.
- Minimize \( \sum_{i=1}^{n} f_i \cdot \text{depth (leaf of } x_i) \).

Question If the tree is given, what is optimal assignment of characters to leaves?

Clear Arrange leaves in decreasing order of depth
"characters in increasing " frequency

Observation
- There are two leaves that are siblings \& at largest depth.
- W.l.o.g., they can be assigned characters with lowest frequency.
Let \( I = \{(x_1, f_1), (x_2, f_2), \ldots, (x_n, f_n)\} \) be an instance of P.F.C. problem with \( f_1 \leq f_2 \leq \ldots \leq f_n \). Let

\[ J = \{(y, f_1+f_2)\} \cup \{(x_3, f_3), \ldots, (x_n, f_n)\} \]

be a new instance. Then

\[ \text{OPT}(I) = \text{OPT}(J) + f_1 + f_2. \]

Proof: Exercise!

Algorithm: Exercise!