Greedy Algorithms / Minimum Spanning Tree

Yin Tat Lee
Spanning Tree

Given a connected undirected graph $G = (V, E)$. We call $T$ is a spanning tree of $G$ if

- All edges in $T$ are from $E$.
- $T$ includes all of the vertices of $G$. 
Why spanning tree?

Many problems is easy for tree.

General framework:
• Approximate the graph by a tree.
• Solve the problem on a tree.

We have covered different tree:
• BFS tree / Dijkstra tree
  • Remember all shortest paths from $s$.
• DFS tree
  • Compute lowest common ancestor.

There are many other different trees depending on applications.
Minimum Spanning Tree (MST)

Given a connected undirected graph $G$ with weights $c_e \geq 0$.

An MST $T$ is a spanning tree whose sum of edge weights is minimized.

$G = (V, E)$

$c(T) = \sum_{e \in T} c_e = 50$

Applications: Network design, …
Kruskal’s Algorithm [1956]

Kruskal(G, c) {
    Sort edges weights so that \( c_1 \leq c_2 \leq \ldots \leq c_m \).
    \( T \leftarrow \emptyset \)

    for each \( (u \in V) \) make a set containing singleton \( \{u\} \)

    for \( i = 1 \) to \( m \)
        Let \( (u, v) = e_i \)
        if \( (u \text{ and } v \text{ are in different sets}) \) {
            \( T \leftarrow T \cup \{e_i\} \)
            merge the sets containing \( u \) and \( v \)
        }
    return \( T \)
}

Question:
How to find maximum spanning tree?
Cuts

In a graph $G = (V, E)$, a cut is a **bipartition** of $V$ into disjoint sets $S, V - S$ for some $S \subseteq V$. We write it as $(S, V - S)$.

An edge $e = \{u, v\}$ is in the cut $(S, V - S)$ if exactly one of $u, v$ is in $S$. 
Properties of the OPT

Simplifying assumption: All edge costs $c_e$ are distinct.

Cut property: Let $S$ be any subset of nodes (called a cut), and let $e$ be the min cost edge with exactly one endpoint in $S$. Then every MST contains $e$.

Cycle property. Let $C$ be any cycle, and let $f$ be the max cost edge belonging to $C$. Then no MST contains $f$.

red edge is in the MST

Green edge is not in the MST
Cycles and Cuts

Claim. A cycle crosses a cut (from $S$ to $V - S$) an even number of times.

Proof. (by picture)

Every time the cycle crosses a cut, it goes from $S$ to $V - S$ or from $V - S$ to $S$. 
Simplifying assumption: All edge costs $c_e$ are distinct.

Cut property. Let $S$ be any subset of nodes, and let $e$ be the min cost edge with exactly one endpoint in $S$. Then the $T^*$ contains $e$.

Proof. By contradiction

Suppose $e = \{u, v\}$ does not belong to $T^*$.

Adding $e$ to $T^*$ creates a cycle $C$ in $T^*$. (coz all tree has $n - 1$ edges)

There is a path from $u$ to $v$ in $T^*$ $\Rightarrow$ there exists another edge, say $f$, that leaves $S$.

$T = T^* \cup \{e\} - \{f\}$ is also a spanning tree.

Since $c_e < c_f$, $c(T) < c(T^*)$.

This is a contradiction.
Cycle Property: Proof

Simplifying assumption: All edge costs $c_e$ are distinct.

Cycle property: Let $C$ be any cycle in $G$, and let $f$ be the max cost edge belonging to $C$. Then the MST $T^*$ does not contain $f$.

Proof. By contradiction
Suppose $f$ belongs to $T^*$.
Deleting $f$ from $T^*$ cuts $T^*$ into two connected components. There exists another edge, say $e$, that is in the cycle and connects the components.

$T = T^* \cup \{e\} - \{f\}$ is also a spanning tree.
Since $c_e < c_f$, $c(T) < c(T^*)$.
This is a contradiction.

Every connected graph has a spanning tree. Hence it has at least $n-1$ edges.
Proof of Correctness (Kruskal)

Consider edges in ascending order of weight.

**Case 1:** adding $e$ to $T$ creates a cycle, $e$ is the maximum weight edge in that cycle. The cycle property shows $e$ is not in any minimum spanning tree.

**Case 2:** $e = (u, v)$ is the minimum weight edge in the cut $S$ where $S$ is the set of nodes in $u$’s connected component. So, $e$ is in all minimum spanning trees.

This proves MST is unique if weights are distinct.
Implementation: Kruskal’s Algorithm

Implementation. Use the union-find data structure.

- Build set $T$ of edges in the MST.
- Maintain a set for each connected component.
- $O(m \log n)$ for sorting and $O(m \log n)$ for union-find

```plaintext
Kruskal(G, c) {
    Sort edges weights so that $c_1 \leq c_2 \leq \cdots \leq c_m$.
    $T \leftarrow \emptyset$

    foreach $(u \in V)$ make a set containing singleton \{u\}

    for $i = 1$ to $m$
        Let $(u, v) = e_i$
        if $(u$ and $v$ are in different sets) {
            $T \leftarrow T \cup \{e_i\}$
            merge the sets containing $u$ and $v$
        }
    return $T$
}
```
Union Find Data Structure

The goal is to have a data structure to tracks a set of elements partitioned into disjoint subsets.

Initially, we have \{\{1\},\{2\},\{3\},\{4\},\ldots\}.

We need two operations:
- \text{Union}(S_1, S_2): Merge subsets \(S_1\) and \(S_2\) into one.
- \text{Find}(x): Output the subset \(S\) containing \(x\).
Find

Each set is represented as a tree of pointers, where every vertex is labeled with longest path ending at the vertex.

To check whether A, Q are in the same connected component, follow pointers and check if the root is the same.

So, the cost of $\text{Find}(x)$ is at most the height of the set.
**Union**

**Union**: To merge two connected components, make the root with the smaller label point to the root with the bigger label (adjusting labels if necessary). Runs in $O(1)$ time.

You can get the same result by “Union by size”.

At most one label must be adjusted.
Depth vs Size: Correctness

Claim: If the label of a root is $k$, there are at least $2^k$ elements in the set.

Proof: By induction on $k$.

Base Case ($k = 0$): this is true. The set has size 1.

Inductive Step: If we merge roots with labels $k_1 > k_2$, the number of vertices only increases while the label (height) stays the same. If $k_1 = k_2$, the merged tree has label (height) $k_1 + 1$, and by induction, it has at least

$$2^{k_1} + 2^{k_2} = 2^{k_1+1}$$

elements.
**Claim:** If the label of a root is $k$, there are at least $2^k$ elements in the set.

Therefore, the depth of any tree in algorithm is at most $\log_2 n$

So, we can check if $u, v$ are in the same component in time $O(\log n)$
Kruskal’s Algorithm with Union Find

Implementation. Use the union-find data structure.

- Build set $T$ of edges in the MST.
- Maintain a set for each connected component.
- $O(m \log n)$ for sorting and $O(m \log n)$ for union-find

```cpp
Kruskal(G, c) {
    Sort edges weights so that $c_1 \leq c_2 \leq \cdots \leq c_m$.
    $T \leftarrow \emptyset$

    foreach $(u \in V)$ make a set containing singleton \{u\}

    for $i = 1$ to $m$
        Let $(u, v) = e_i$
        if $(u$ and $v$ are in different sets) {
            $T \leftarrow T \cup \{e_i\}$
            merge the sets containing $u$ and $v$
        } \\
    return $T$
}
```

Find roots and compare

Merge at the roots
5. Extra Credit: Let \([n] = \{1, 2, 3, \cdots, n\}\) and \(\mathcal{I}\) be a collection of subsets of \([n]\). We call any set \(I \in \mathcal{I}\) is nice.

We know that \(\mathcal{I}\) satisfy two main axioms:

(a) If \(X \subset Y\) and \(Y \in \mathcal{I}\), then \(X \in \mathcal{I}\). Namely, any subset of a nice set is nice.
(b) If \(X \in \mathcal{I}, Y \in \mathcal{I}\) and \(|Y| > |X|\), then there exists \(i \in Y \setminus X\) such that \(X \cup \{i\} \in \mathcal{I}\). Namely, if \(X\) is nice and there exists a larger nice set \(Y\), then \(X\) can be extended to a larger nice set by adding an element of \(Y \setminus X\).

The collection \(\mathcal{I}\) may have exponentially size and is only defined implicitly. However, we assume that we can test if a set \(I\) is nice or not in polynomial time.

Given a cost \(c_1, c_2, c_3, \cdots, c_n\), design a greedy polynomial time algorithm to find a nice set \(X\) with maximum total cost \(c(X) = \sum_{x \in X} c_x\).

- Let \([n]\) be all edges of \(G\).
- Let \(I\) be the collection of subsets of edges without cycles.
- Any nice set \(X \in I\) is a forest.
- Maximum \(c(X)\) over nice set \(X\) is same as finding maximum spanning tree.
- The answer for this question is basically Kruskal’s Algorithm
Find: Path Compression

After finding the root $r$ of the tree containing $x$, Change the parent points of all nodes along the path to $r$ directly.

\[
\text{Find}(x) \{ \\
\quad \text{If } x \neq \text{parent}(x) \\
\quad \quad \text{parent}(x) \leftarrow \text{Find}(\text{parent}(x)) \\
\quad \text{return parent}(x) \\
\}
\]
In 1972, Fischer proved this it takes $O(n \log \log n)$ time to do $O(n)$ find and $n - 1$ union.

In 1973, Hopcroft-Ullman improve the bound to $O(n \log^* n)$.

In 1975, Tarjan improved the bound to $O(n \alpha(n))$.

Finally, in 1989, Fredman-Saks proved this is optimal.
What is $\alpha(n)$?

Inverse Ackermann function $\alpha(n)$.

Define

- $\alpha_1(n) = \lfloor n/2 \rfloor$.
- $\alpha_2(n) = \lfloor \log n \rfloor = \# \text{ of times we divide } n \text{ by two until we reach } 1$.
- $\alpha_3(n) = \log^* n = \# \text{ of times we apply } \log \text{ until we reach } 1$.
- $\alpha_4(n) = \# \text{ of times we apply } \log^* \text{ until we reach } 1$.

Define $\alpha(n) = \min\{k: \alpha_k(n) \leq 3\}$. Any number $n$ I can write down satisfies $\alpha(n) \leq 3$. 

| $\alpha_1(n)$ | 1 | 1 | 2 | 2 | 3 | 3 | 4 | 4 | 5 | 5 | 6 | 6 | 7 | 7 | 8 | 8 | ... | $2^{15}$ | ... | $2^{65535}$ | ... | huge |
| $\alpha_2(n)$ | 1 | 1 | 2 | 2 | 3 | 3 | 3 | 3 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | ... | 16 | ... | 65536 | ... | $2 \uparrow \ 65536$ |
| $\alpha_3(n)$ | 1 | 1 | 2 | 2 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | ... | 4 | ... | 5 | ... | 65536 |
| $\alpha_4(n)$ | 1 | 1 | 2 | 2 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | ... | 3 | ... | 3 | ... | 4 |