Minimum Spanning Tree (MST)

Input: (undirected connected) $G = (V, E, c)$
- $c_e$ is the cost of the edge $e$.
- $c_e > 0$ and w.l.o.g all $c_e$'s are distinct.

Output: a subset $T \subseteq E$ so that $G = (V, T)$ is connected
and the total cost $\sum_{e \in T} c_e$ is minimized.

Claim (4.16): $T$ is a tree.
Proof: suppose $T$ contains a cycle $C$,
then if we any edge $e \in C$, the graph
$(V, T - \{e\})$ is still connected.

Kruskal's algorithm:
1. Start with $T = \emptyset$.
2. Sort edges in $E$ by cost in ascending order.
3. Iterate over all edges, add $e$ to $T$ as long as it
does not create a cycle.

Example:

Correctness:
Theorem (4.18): Kruskal's algorithm produces an MST.
Lemma (4.17): Assume $c_e$'s are distinct.
Fix $S \subseteq V$.
Let $e$ be the minimum cost edge with
one end in $S$ and one end in $V - S$.
Then $e$ must appear in every MST.
Proof:

Suppose $e = (v, w)$ is the cheapest edge between $S$ and $V-S$.

Suppose there is an MST $T$ s.t. $e \in T$.

Because $T$ is a spanning tree, there is a path $P$ from $v$ to $w$ in $T$. Since $v \in S$ and $w \notin S$, $P$ must leave $S$ at some point.

Let $e'$ be the first edge in $P$ that leaves $S$.

$(v', w)$

$T + \{ e \} - \{ e' \}$ is a spanning tree with smaller cost.

- the cost is smaller because $C_e < C_{e'}$.
- $T + \{ e \}$. contains a unique cycle, $C$ and moreover $e' \in C$.

$\Rightarrow T + \{ e \} - \{ e' \}$ is connected.

$T + \{ e \} - \{ e_3 \}$ has $(n-1)$ edges.

Proof (correctness of Kruskal):

Consider an edge $e = (v, w)$ added by Kruskal's algorithm.

Let $S$ be $v$'s connected component just before adding $e$.

- $w \notin S$ (o.w. adding $e$ creates a cycle).
- no edge from $S$ to $V-S$ has been considered (any such edge would have been added.)

Thus, $e$ must be the cheapest edge that leaves $S$. So it is safe to include $e$.

Moreover, Kruskal's output must be connected.
Prim's algorithm.

1. Pick a root $s \in V$.
2. $S = \{s\}$ (All nodes in $S$ are connected.)
3. Repeat the following until $S = V$:
   
   Add $v$ to $S$ where $v \not\in S$ minimizes the "attachment cost".
   
   $$\min_{e=(u,v), u \in S} C_e.$$ (breaking ties arbitrarily)
   
   Add $e=(u,v)$ to $T$.
4. Return $T$.

Example:

```
1 7
2
3
4 9
5 15
6 12
7 11

Example Graph
```

Correctness:

- Every time the algorithm adds an edge $e$, $e$ is the cheapest edge between $S$ and $V \setminus S$.
- $T$ is a spanning tree at the end.

Runtime of Prim: $O(m \log n)$.

Similar to Dijkstra's algorithm.

We will use a min-heap $H$.

Overall: \[
  \begin{cases}
    O(n), & H.\min( ) \text{ each call takes } O(1) \text{ time} \\
    O(m), & H.\changekey( ) \text{ each call takes } O(\log n) \text{ time}
  \end{cases}
\]

Overall runtime $= O(n \cdot 1 + m \cdot \log n) = O(m \log n)$.

Runtime of Kruskal:

1. Sort all edges: $O(m \log m) = O(m \log n) \quad (m \leq n^2)$
2. Check (in total $O(m)$ times) whether adding $e$ would cause a cycle. How? ($O(n)$ via DFS) Can we do it faster?

The Union-Find data structure. (page 152)

- Init(): all element are in separate sets.
- Union($x$, $y$): merge the set containing $x$ with the set containing $y$.
- Find($x$): return the "name" of the set that $x$ is in.

We use this data structure to maintain the set of connected components (which are changing over time) in Kruskal's algorithm.
When we consider any edge $e=(u,v)$

If $\text{find}(u) \neq \text{find}(v)$ then

Add $e$ to $T$

Union $(u,v)$.

End if.

### Union-Find First Attempt

- **name** $[i]$: name of the set that contains $i$
- **size** $[j]$: size of the set with name $j$

**Init $(i)$:** $\text{name}[i] = i$, $\text{size}[i] = 1$

**Find $(x)$:** return $\text{name}[x]$

**Union $(x, y)$:**

$x = \text{name}[x]$

$y = \text{name}[y]$

If $\text{size}[x] < \text{size}[y]$ swap $(x, y)$

*(so w.l.o.g $\text{size}[x] \geq \text{size}[y]$)*

For every element $i$ in set "y",

$name[i] = x$.

End For

$\text{size}[x] = \text{size}[x] + \text{size}[y]$.

### Example:

<table>
<thead>
<tr>
<th>Name</th>
<th>Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
</tr>
</tbody>
</table>

Before

$\text{size}[1] = 2$, $\text{size}[2] = 3$, $\text{size}[4] = 1$, $\text{size}[5] = 1$

$\text{Union}(4, 5)$

$x=4$, $y=5$ $\Rightarrow$ $x=1$, $y=2$.

$x=2$, $y=1$ *(swapped so that size[x] $\geq$ size[y])*.

For $i \in \{1, 4\}$,

$name[i] = 2$.

End For

$\text{size}[2] = \text{size}[2] + \text{size}[1]$.

After

$\text{size}[1] = 5$, $\text{size}[2] = 2$, $\text{size}[4] = 2$, $\text{size}[5] = 1$, $\text{size}[6] = 1$, $\text{size}[7] = 1$

We always merge the smaller set into the bigger one!

### Runtime

**Init:** $O(n \times k)$

**Find:** $O(1) \times O(m)$

**Union:** $O(1)$

### Claim

The first $k$ union operations run in total time $O(k \log k)$.

$\Rightarrow$ Kruskal runs in time $O(m \log n + m + n \log n) = O(m \log n)$.
Proof sketch:

After \( k > 1 \) union operations, the largest set has size \( O(k) \).

How many times can \( \text{name}[i] \) change?

Once per union \((A, B)\) if \( i \in B \) and \( |B| \leq |A| \).

At most \( O(\log k) \) times because the size of the set \( \text{name}[i] \) at least doubles each time.

At most \( 2^k \) \( \text{name}[i] \) are changed.

\[ O(k \log k). \]

Not very ideal because sometimes \( \text{Union}(x, y) \) can take \( \Omega(\log n) \) time.

Example: \( \text{name} = 11 \ldots 11 22 \ldots 2 \) (\( n/2 \) \( n/2 \))

\[
\begin{array}{c|cc}
\text{Union-Find} & \text{Second Attempt} \\
\hline
1 & 2 & 6 \\
4 & 3 & 5 \\
\end{array}
\]

\[
\text{Index} \: 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad (-1 \text{ if no parent})
\]

\[
\text{Parent} \: -1 \quad -1 \quad 2 \quad 1 \quad 2 \quad -1 \quad -1
\]

\[
\text{Find}(x) : \quad \text{while} \ (\text{parent}[x] \neq -1)
\]

\[
\text{End while} \quad x = \text{parent}[x] \\
\text{Return} \ x
\]

\[
\text{Union}(x, y) : \quad x = \text{Find}(x), \ y = \text{Find}(y).
\]

merge the smaller set into the bigger one.

\[
\text{If} \ \text{size}[x] \geq \text{size}[y]
\]

\[
\text{parent}[y] = x, \quad \text{size}[x] = \text{size}(x) + \text{size}[y]
\]

\[
\text{Else} \quad \text{parent}[x] = y, \quad \text{size} \ldots
\]

Example: \( \text{Union}(4, 5) \).

\[
\begin{array}{c|cc}
\text{Runtime} & \text{Init()} : O(n) \\
\hline
\text{Find()} : O(\log n) \\
\text{Union()} : O(1)
\end{array}
\]

Runtime of Kruskal under this \text{Union-Find} implementation = \( O(\log n + m \log n + n) \) after \( x = \text{find}(x) \) and \( y = \text{find}(y) \).
Union-find Third Attempt (the real version)

Building on the second attempt, we add one more optimization.

Path compression: After \( \text{find}(x) \), point all the elements we traversed to the root.

\[
\begin{array}{c}
\text{Before} \\
\text{After}
\end{array}
\]

Implementation:

\[
\text{int} \quad \text{Find}(x) \quad \{
\text{If} \quad (x == \text{parent}[x]) \quad \text{return} \; x; \\
\text{root} = \text{Find}(\text{parent}[x]); \quad \leftarrow \text{recursion}, \\
\text{parent}[x] = \text{root}; \\
\text{return} \; \text{root}
\}
\]

\[
\text{parent}[7] = 2 \leftarrow \text{parent}[4] = 2 \leftarrow \text{parent}[1] = 2
\]

Upshot: Next time we call \( \text{find}(7) \) before any union operation takes \( O(1) \) time.

Runtime: First \( k \) Union/Find operations

A "simple" proof (available on Union-Find Wikipedia page) \footnote{Hopcroft and Ullman '73}
shows that these \( k \) operations runs in \( O(k \cdot \log^* n) \).

\( \log^* n \): iterated logarithm: \# of times \( \log \) needs to be applied for \( n \) to become 1.

\[
\log_2^\ast \left(2 \cdot \log^\ast 2 \right) = 5
\]

\( O(m \cdot \alpha(n)) \quad \text{[Tarjan '75]} \quad \Theta (m \cdot \alpha(n)) \quad \text{[Fredman Saks '89]}

\uparrow \text{inverse Ackermann function (grows even slower than } \log^* n)\).