Minimum Spanning Tree

Given $G$, an undirected, weighted graph, a spanning tree $T \subseteq G$ is a connected tree on $V$.

Weight of a tree: $\sum_{e \in T} \omega(e)$

Given $G$, find a spanning tree with minimum weight.

E.g.

Not unique.
**Prim's Algorithm** (1930 → 1957).
Sornik → Prim.

**Algorithm**

1) Start from an empty set of vertices: $S$.
2) Choose a random root & add it to $S$.
3) Start growing a tree from $S$ by adding the least weighted edge going out of $S$.

E.g.,

- Choose $A$ as root.

![Diagram of Prim's Algorithm example](image-url)
for $v \in V$
\[ H[v] = \infty \]
($H$ is a priority queue with change operations).
\[ S = \emptyset, \quad \pi[v] = -1 \]
\[ H[v] = 0 \] for some random $v$.

while ($S \neq \emptyset$)
\[ \nu = H.\text{deleteMin}() \]
\[ S.\text{add}(\nu) \]
for each $\text{ubh of } \nu$
\[ \text{if } H[\text{ubh}] > \omega(\nu, \text{ubh}) \]
\[ H[\text{ubh}] = \omega(\nu, \text{ubh}) \]
\[ \pi[\text{ubh}] = 0 \]

---

Running time analysis.

- each edge is examined once in the for loop.
  (twice when ubh is added).

- at each point, we set $H[\text{ubh}]$ which is a change operation in the heap.
  (also called decrease-key) \( \rightarrow O(\log n) \).

- each vertex is added & deleted once from $H$.
\[ n - O(\log n) \]
\[ = O((n + m) \log n) \]
if we use a FibonacciHeap

\[ T'(y) = O(n \log u + m) \]

(good for dense graphs).

---

Proof of correctness.

Suppose that \( T_p \) is the tree given by Prim's Algorithm & \( T_0 \) is another tree with \( \omega(T_0) \leq \omega(T_p) \).

Suppose the edges in \( T_p \) were added in the order
\[ e_1, e_2, \ldots, e_{n-1} \]

As \( e_i \) is the first edge \( i, e_i \notin T_0 \), \( e_i = (x, y) \)

\( \Rightarrow \) \( T_0 \) is of the form

Suppose that in Prim's algorithm, \( x \in S \), \( y \notin S \).

then along the path \( x \leadsto v_1 \leadsto v_2 \cdots \leadsto v_k \leadsto y \)

\( \Rightarrow \) \( k \), \( k+1 \in S \), \( v_k \in S \), \( v_k+1 \notin S \)

(because \( S \) is a connected tree)
now in Prim's algorithm
we did not choose $v_x \rightarrow v_{x+1}$ over $x \rightarrow y$ 

$\Rightarrow \quad \omega(v_x, v_{x+1}) \geq \omega(x, y)$

if $\omega(v_x, v_{x+1}) > \omega(x, y)$
swap the edges in the tree $T_0$

i.e. remove $v_x \rightarrow v_{x+1}$

and insert $x \rightarrow y$

to get $T_1'$

$T_1'$ is a tree, $\mathcal{B}$ has smaller weight than $T_0$

$\Rightarrow \quad \omega(v_x, v_{x+1}) = \omega(x, y)$

we can swap $x$ and keep the weight same.

Inductively we can change $T_0$ to $T_p$ without changing the weight

$\Rightarrow \quad \omega(T_0) = \omega(T_p)$
Kruskal's Algorithm

1) Start with an empty set of vertices/edges.
2) Choose the minimum edge which joins two disconnected vertices & add it to the forest.

E.g.

- E:
  - A
  - B
  - C
  - D
  - E
  - F

- Diagram: Graph with edges connecting vertices.
operations needed

- find minimum edge [sort the edges]
- check if two vertices are in the same component.
- join two components.

For the last two operations, union-find data structure

**Time complexity**

- sorting $\rightarrow \mathcal{O}(m \log m)$
- find $\rightarrow \text{foreach edge} \rightarrow \mathcal{O}(m \text{find}(u))$
- join $\rightarrow n-1$ join operations $\rightarrow \mathcal{O}(n \text{join}(u))$

$\Rightarrow T(d) = \mathcal{O}(m \log m + m \text{find}(u) + n \text{join}(u))$

* Dijonct set forest (union-find with path compression)
  
  find$(u)$, join$(u) = \mathcal{O}(\alpha(u)) 
  \quad \alpha(u) \leq \log(n)$

$\Rightarrow T(d) = \mathcal{O}(m \log n)$
Proof of correctness.

we only join components which are disconnected.

$\Rightarrow$ we get a spanning tree $T_k$

Suppose that $\exists T_0 \ni \omega(T_0) \leq \omega(T_k)$

let the edges added in $T_k$ be in order

$e_1, e_2, \ldots, e_m$

& let $e_i$ be the first edge $\notin T_0$, $\forall e_i = (x,y)$

look at $T_0 \cup \{e_i\}$

\[
\begin{array}{c}
V_0 & \vdots & V_k \\
\hline
X & e_i & Y
\end{array}
\]

before adding $e_i$, in Kruskal, we would have added
any other edge $e$ (including $v_j \rightarrow v_{j+1}$), if they were
of smaller weight $\omega$ unless they were in one component.

Get both cases:

1) $x, v_i, \ldots, v_k, y$ are not all in the same
   component.

   (a) of the form $v_j \rightarrow v_{j+1}$ or $x \rightarrow v_i$ or $v_k \rightarrow y$

2) $\exists$ edge $v_j \rightarrow v_{j+1}$ or $x \rightarrow v_i$ or $v_k \rightarrow y$

$\Rightarrow \omega(a,b) \leq \omega(e_i)$

By same argument as Prim's