

Kruskal's algorithm.

Let $E \leftarrow e_1, \dots, e_m$ be edges in increasing order of cost.

Let $T \leftarrow \emptyset$.

For $i \leftarrow 1$ to m do

Suppose $e_i = (u, v)$. If there is no path from ~~to~~ u to v in (V, T) ,

add e_i to T

endfor

Return T .

Claim 1 If $G = (V, E)$ is connected, so is (V, T) , where T is set returned by Kruskal.

Proof: Suppose (V, T) is not connected.

So there are two vertices u and v so that there is no path between them in (V, T) . Let $S \subseteq V$ be all the vertices to which there is a

path from u in (V, T) .

Clearly, $u \in S$ and $v \in V - S$.

In (V, T) , there is no edge from S to $V - S$. (otherwise, S would be larger.)

Since algo only adds edges to T , there is no edge in (V, T) from S to $V - S$ during at any point of algo.

Since there is a path from u to v in G , there is an edge ~~to~~ $f = (u', v')$

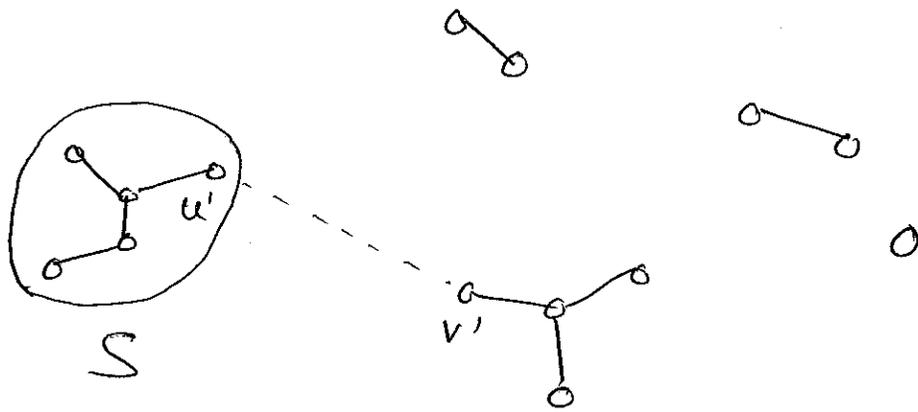
from $u' \in S$ to $v' \in V - S$ in $G = (V, E)$.

Evidently, f was considered by algo but was not added to T . This is a contradiction because there is no path from u' to v' in (V, T) at the time f was considered.

Claim 2 Suppose $e_i = (u', v')$ is added by algorithm to T . Then $e_i \in OPT$.

Remark: OPT is some optimal solution.

Proof: Consider (V, T) just before e_i is considered. Let S be all vertices to which there is a path from u' in (V, T) . Clearly, $u' \in S$ and $v' \in V - S$.



There is no edge from S to $V - S$ in (V, T) . So among edges from S to $V - S$, e_i is first one algo considers, so it is the cheapest edge.

Suppose $e_i \notin \text{OPT}$. There is a path π from u' to v' in OPT . Since $u' \in S$ and $v' \in V-S$, there must be some edge f in π that goes from S to $V-S$. We have

$$c_{e_i} < c_f.$$

Observe that $\text{OPT} - f + e_i$ is also a spanning subgraph, i.e., $(V, \text{OPT} - f + e_i)$ is connected.

$$\begin{aligned} \text{But } \text{cost}(\text{OPT} - f + e_i) &= \text{cost}(\text{OPT}) - c_f + c_{e_i} \\ &< \text{cost}(\text{OPT}), \end{aligned}$$

a contradiction to optimality of OPT .

Prim's Algorithm.

$S \leftarrow \{s\}$, where $s \in V$ is an arbitrary vertex.

$T \leftarrow \emptyset$.

While $(S \neq V)$ do

 Pick cheapest edge from S to $V-S$. (note that such edges exist.)

 Suppose this is $e = (u, v)$ where $u \in S$, $v \in V-S$.

 Add v to S and e to T .

endwhile

Return T

Claim 1 (V, T) is connected.

Proof: This follows from the fact that the while loop maintains the following invariant:

Any pair of vertices in S is connected by a path in (V, T) .

At the end, $S = V$. So at the end, any pair of vertices in ~~S~~ V is connected by a path in (V, T) . So (V, T) is connected.

The invariant itself is readily seen to hold.

Claim 2: ~~Suppose~~ If $e \in T$, then $e \in \text{OPT}$.

Remark: Recall that OPT is some optimal solution.

Proof: Suppose $e = (u, v)$ was added in some iteration of while loop because it was the cheapest edge

connecting S to $V-S$.

Suppose $e \notin \text{OPT}$. There is a simple path π from u to v in OPT , and this path has an edge f from S to $V-S$.

As in claim 2 of Kruskal,

$\text{OPT} - f + e$ is a spanning subgraph whose cost is less than OPT , a contradiction.

Graph Representation

The input graph $G = (V, E)$ is represented by an adjacency list representation.

$$V = \{1, 2, \dots, n\}$$

There is an array $Adj[1..n]$:

$Adj[u]$ is a list containing all the nodes v such that $(u, v) \in E$.

If $(u, v) \in E$, the list element in $Adj[u]$ corresponding to v also stores $c[u, v]$.

Implementing Prim's algo:

For each $v \in V - S$,

let $S(v)$ be the vertex u that minimizes $c(u, v)$ over all $(u, v) \in E$ such that $u \in S$.

For every $v \in V - S$,

we store the ~~pair~~ ~~(s, v)~~
triple $(v, \delta(v), c(v, \delta(v)))$

in a priority queue with key
 $c(v, \delta(v))$.

After adding s to S initialize,

we insert a priority queue element
corresponding to every ^{vertex} ~~element~~ incident
to s .

For vertices ^{v} ~~to~~ not incident to s ,
we add dummy triple $(v, -, \infty)$.

In each iteration of while loop,

extract-min will give us the
edge (u, v) we want.

Once we add v to S , we go through every $w \in V-S$ such that $(v, w) \in E$ and update the priority queue entry corresponding to w if necessary, by a Change-Key operation.

Running time:

$O(n)$ for initialization.
while loop takes $O(1)$ ~~per~~ $\deg(v)$
a single iteration not counting the
one extract-min operation and
multiple Change-key operations. ~~and~~

Note ~~that~~

There are $n-1$ iterations of while loop.
Total # of Change-key operations overall
 $\leq m$.

So running Time is

$$\begin{aligned} &= O(n) + n \times \text{Time for extract-min} \\ &\quad + m \times \text{Time for change-Key} \end{aligned}$$

$$= O(n + n \log n + m \log n)$$

$$= O(m \log n)$$