

Kruskal's minimum spanning tree algorithm

High-level algorithm. Input: connected graph G , non-negative edge weights w_e .

Output: a tree connecting all vertices of G and of minimum weight.

Initialization: Let each vertex form a singleton tree and F be the collection of those trees.

For $i = 1, 2, \dots, m$:

Assume a forest $F = \{T_1, T_2, \dots, T_{n+1-i}\}$ is already defined.

Let $e = \{u, v\}$, $u \in T_i$, $v \in T_j$, $i \neq j$, be such that w_e is minimum.

Construct a new tree $T = T_i \cup T_j \cup \{e\}$.

$F \leftarrow F + \{T\} - \{T_i, T_j\}$.

Output the tree.

Proof of correctness. It is easy to see that the output is a tree (exercise). The rest of the proof is by contradiction. Assume that it is not of minimum cost, and let T^* be a minimum spanning tree (if there are several minimum spanning trees, pick one such that the first k edges picked by the algorithm are also in T^* , and k is as large as possible).

Consider the edges in the order in which they are chosen by the algorithm, let e be the first one which is not in T^* , and let F be the set just before e is added: $e = \{u, v\}$ with $u \in T_i$ and $v \in T_j$, $i \neq j$. In $T^* \cup \{e\}$, the edge e closes a cycle C , so there must exist some other edge $e^* = \{v^*, u^*\}$ on the cycle with $u^* \in T_i$ and $v^* \notin T_i$. Why did the algorithm choose e and not e^* at that point? Because $w_e \leq w_{e^*}$. But observe that $T' = T^* - \{e^*\} + \{e\}$ is still a spanning tree (any two vertices which in T^* were connected by a path going through e^* are still connected, replacing e^* by going around the cycle C). Since T^* is of minimum cost, it has cost less than T' , and so $w_e \geq w_{e^*}$. Thus $w_e = w_{e^*}$. But then, T' is also a minimum spanning tree, and has one more edge in common with the algorithm's choices: this contradicts the definition of T^* .

Implementation. We can view the algorithm as examining the edges one by one by order on increasing weight.

Initialization: Let $F = \{\{v_1\}, \{v_2\}, \dots, \{v_n\}\}$ be the initial partition of V into sets.

Sort the edge weights in increasing order.

For each edge $e = \{u, v\}$ in order:

Find the name of the set $S(u)$ containing u and of the set $S(v)$ containing v .

If the set $S(u)$ containing u differs from the set $S(v)$ containing v

Let $S = S(u) \cup S(v)$ be the union of the two sets.

$F = F + S - \{S(u), S(v)\}$.

Output the associated tree.

Data structure. Instead of retesting (say, by DFS) at each iteration whether the two endpoints of e are in different trees of the forest, we maintain a "union-find" data structure. That's more efficient.