CIS 575. Introduction to Algorithm Analysis
Material for April 10, 2024

Minimum-Cost Spanning Trees: Kruskal's Algorithm

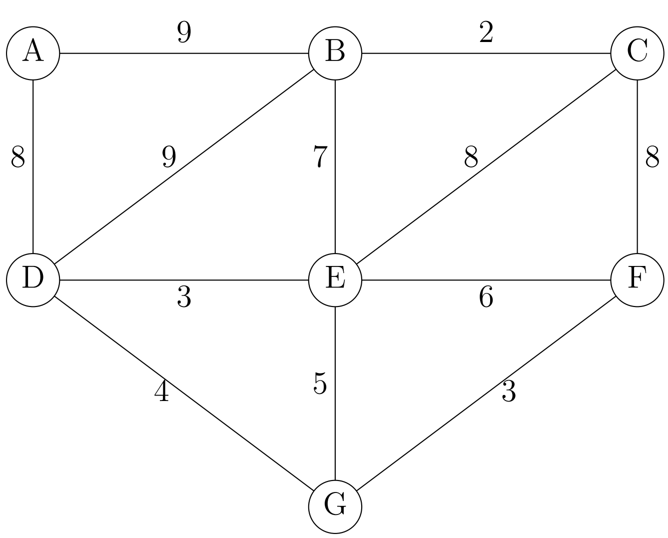
The topic of this note is presented in *Cormen*'s Section 21.2.

1 Kruskal's Algorithm

The main idea behind this algorithm is to build the minimum spanning tree T "piecemeal". As it grows at various places, more and more nodes will be joined together into **connected components** where two nodes are in the same connected component if and only if there is a path between them using only edges from T. By the result stated and proved in the previous note, it is safe to repeatedly add a minimum-weight inter-component edge, as is done by the pseudo code

$T \leftarrow \emptyset$
sort the edges so their weights are non-decreasing (lowest ones first)
foreach edge e
  if e has end points in two different connected components
    $T \leftarrow T \cup \{e\}$

Example Recall our running example:

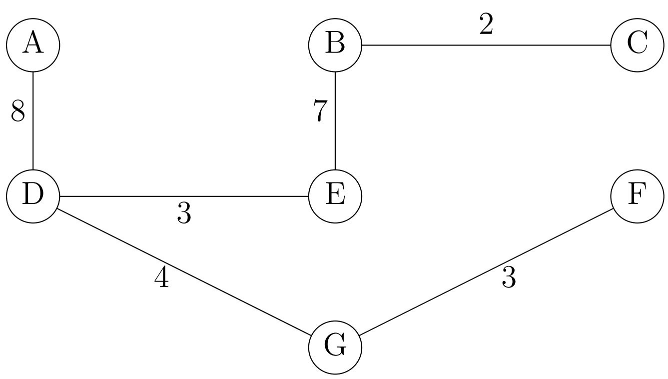


Kruskal's algorithm may examine the edges in the order:

edge	weight	pick?	non-singleton connected components
BC	2	Yes	BC
DE	3	Yes	BC, DE
FG	3	Yes	BC, DE, FG
DG	4	Yes	BC, DEFG
EG	5	No	BC, DEFG
EF	6	No	BC, DEFG
BE	7	Yes	BCDEFG
CF	8	No	BCDEFG
CE	8	No	BCDEFG
AD	8	Yes	ABCDEFG
AB	9	No	ABCDEFG
BD	9	No	ABCDEFG

We see that the first edge not to be included in the minimum spanning tree is the one between node E and node G; this is because these two nodes are already in the same connected component.

The resulting spanning tree has weight 27:



Implementation and Running Time One question remains: how to represent connected components so that one can efficiently

1. check if two nodes are in the same connected component
2. if they are in two different connected components, combine those into one.

But we have already seen that a *Union-Find* structure allows us (if we do "Union by Rank") to do both in time *logarithmic* in the number of nodes[1]. We can thus

1. sort the edges in time $\Theta(a \lg(a))$ (for example by heapsort)
2. process the `foreach` loop in time $\Theta(a \lg(n))$.

Since we assume the graph to be connected, we have $n - 1 \leq a \leq n(n-1)/2$ and thus $\lg(a) \in \Theta(\lg(n))$. We conclude that the **total running time** of Kruskal's algorithm is in $\boldsymbol{\Theta}(\mathbf{a} \lg(\mathbf{n}))$. (This assumes that the graph is represented using adjacency lists; if an adjacency matrix representation is used then we must add $\Theta(n^2)$ to retrieve the edges, which is worse for sparse graphs.)

[1]By "path compression" one can do even better, but that doesn't change the total asymptotic running time of Kruskal's algorithm.