ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institute of Theoretical Computer Science
Rasmus Kyng, Bernd Gärtner, David Steurer, Vera Traub

Algorithms, Probability, and Computing **Solutions KW39** **HS25**

Solution 1

Assume for a contradiction that the graph has two distinct minimum spanning trees T and T'. Let $e = \{u, v\}$ be an edge in T which is not in T'. Removing edge e cuts the tree T into two components (trees). Let T_u and T_v be the vertices in the component containing u and v, respectively. Consider the edge boundary $\partial(T_v)$ and let e' be the unique light edge in $\partial(T_v)$. If $e \neq e'$ then $w(e') < w(e)$ and the spanning tree obtained by removing e and adding e' has a smaller weight.

Hence, assume that $e' = e$, i.e., the unique light edge e in $\partial(T_v)$ does not belong to T'. Consider the path p from u to v in T'. Path starts in T_u and ends in T_v, hence there must be an edge e'' on it which is in $\partial(T_v)$ (there might be several of them, but take one). As e is the unique light edge in $\partial(T_v)$, $w(e) < w(e'')$. If we add e to T', we get a cycle composed of e and path p. By removing any edge from the cycle we get again a spanning tree. Hence, removing e'' from T' and adding e into it, gives us a lighter spanning tree, which is a contradiction.

If no two edges have the same weight, then for every non-empty vertex set $S \subset V$, the edge with the minimum weight in $\partial(S)$ is unique. This implies that the graph has exactly one MST.

Solution 2

We assume in both subtasks that $c_0 > 0$ is a constant such that the running time of our algorithm (without counting the contributions of the two recursive calls) on any input graph $G = (V, E)$ is bounded from above by the function $D(n, m) := c_0(n + m)$, where $n = |V|$ and $m = |E|$.

(i) Let $T(n, m)$ denote the worst-case possible running time of the algorithm when given a graph on n vertices and m edges as input. The key observation that we will use for the analysis is the following inequality.

$$T(n, m) \leq \max_{\substack{k_1, k_2 \geq 0 \\ k_1 + k_2 \leq m + n/8}} \left(T\left(\frac{n}{8}, k_1\right) + T\left(\frac{n}{8}, k_2\right) + D(n, m) \right) \tag{1}$$

Naturally, the above formula reflects the fact that the running time consists of two recursive calls and the additional non-recursive work, which is assumed to be no more than $D(n, m)$. Given that the algorithm uses Boruvka three times, it is also clear that

the number of vertices drops from n to at most $n/8$ in both recursive subproblems. As for the numbers k_1 and k_2 of edges, we do not have any specific information about either subproblem; each one of them could have close to m edges. Nevertheless, we do know that only a small number of edges can find their way into both subproblems at the same time, namely those edges that are contained in the spanning tree (or forest) returned by the first recursive call. Since that spanning tree contains at most $n/8 - 1$ edges, we can conclude that $k_1 + k_2 \leq m + n/8$ holds in any case, and we simply maximize over all such possibilities.

We deal with the remaining parts of this subtask in the following two claims. Simply observe that in order to prove $f = O(\min\{g, h\})$ for functions f, g and h, it suffices to prove both $f = O(g)$ and $f = O(h)$. Also note that the bound proved in Claim 2 immediately yields the stronger bound of $O(m \log n)$ if we assume that the input graph is connected and that, hence, $m \geq n - 1$.

Claim 1. *There exists a constant $c_1 > 0$ such that $T(n, m) \leq c_1 \cdot n^2$.*

Proof. The proof is by induction on n. In the following derivation, let k_1 and k_2 be the numbers that maximize the formula in equation (1).

$$\begin{aligned} T(n, m) &\leq T\left(\frac{n}{8}, k_1\right) + T\left(\frac{n}{8}, k_2\right) + D(n, m) \\ &\leq c_1 \cdot \left(\frac{n}{8}\right)^2 + c_1 \cdot \left(\frac{n}{8}\right)^2 + c_0 \cdot (\underbrace{n}_{\leq n^2} + \underbrace{m}_{\leq n^2}) \\ &\leq \frac{1}{32} c_1 \cdot n^2 + 2c_0 \cdot n^2 \leq c_1 \cdot n^2 \end{aligned}$$

The last inequality goes through if we choose the constant $c_1 := 3c_0$, for example. □

Claim 2. *There exists a constant $c_2 > 0$ such that $T(n, m) \leq c_2 \cdot (n + m) \log n$.*

Proof. The proof is again by induction on n. Let k_1 and k_2 be as before.

$$\begin{aligned} T(n, m) &\leq T\left(\frac{n}{8}, k_1\right) + T\left(\frac{n}{8}, k_2\right) + D(n, m) \\ &\leq c_2 \cdot \left(\frac{n}{8} + k_1\right) \log\left(\frac{n}{8}\right) + c_2 \cdot \left(\frac{n}{8} + k_2\right) \log\left(\frac{n}{8}\right) + c_0 \cdot (n + m) \\ &= c_2 \cdot (\underbrace{k_1 + k_2}_{\leq m + \frac{n}{8}} + \frac{n}{4}) \log\left(\frac{n}{8}\right) + c_0 \cdot (n + m) \\ &\leq c_2 \cdot (n + m) \log\left(\frac{n}{8}\right) + c_0 \cdot (n + m) \\ &= c_2 \cdot (n + m) \log n - 3c_2 \cdot (n + m) + c_0 \cdot (n + m) = c_2 \cdot (n + m) \log n \end{aligned}$$

The last equality holds if we choose the constant $c_2 := c_0/3$. □

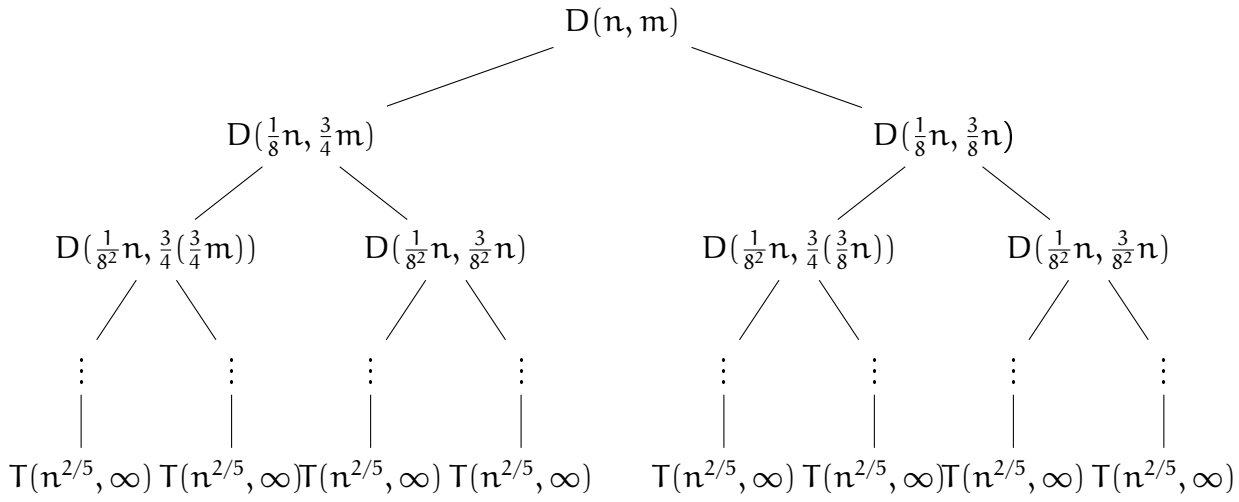

Figure 1: A binary tree of very specific running times that we use in order to prove an upper bound on the actually observed running time of the algorithm.

(ii)

(a). Let us start by imagining what could happen when the algorithm is invoked with some input graph G. Clearly, we spend at most $D(n, m)$ time if we neglect the two recursive calls. As for the first recursive call, it might happen that we construct a new graph G_1 with exaclty $\frac{n}{8}$ vertices and a randomly selected set of $\frac{3m}{4}$ edges. Thus, in that first recursive call we would spend at most $D(\frac{n}{8}, \frac{3m}{4})$ time plus however much time is needed for the ensuing two recursive calls. As for the second recursive call, we might construct a graph G_2 with $\frac{n}{8}$ vertices again, but only a set of $\frac{3n}{8}$ edges composed of a spanning tree of size $\frac{n}{8}$ and another $\frac{n}{4}$ edges that turned out to be non-heavy. Thus, in this second recursive call we would spend at most $D(\frac{n}{8}, \frac{3n}{8})$ time plus however much time is needed for the ensuing two recursive calls.

The imagined situation is depicted in Figure 1 as a binary tree of running times. In the depicted tree it is assumed that the number of vertices of G is divided by exactly 8 at every level of the recursion. Furthermore, the number of edges in each first (i.e., left) subproblem is assumed to shrink exactly by a factor of $\frac{3}{4}$; and the number of edges in each second (i.e., right) subproblem is assumed to be exactly 3 times the number of vertices in that subproblem. As soon as the number of vertices of G has shrunk down to $n^{2/5}$ we make an arbitrary cut-off and use the worst-case function $T(n^{2/5}, \infty)$ from subtask (i) to express the remaining running times.

Of course, it is highly unlikely that this special tree of running times will be observed in an actual run of the algorithm. Nevertheless, we can prove that this model tree is not too bad and that with high probability it dominates the actually observed tree.

Claim 3. *There exists a constant $c_3 > 0$ such that the sum of all running times in the nodes of the tree depicted in Figure 1 is bounded from above by $c_3 \cdot (n + m)$.*

Proof. Let us first deal with the nodes at the bottom level of the tree. Observe that these nodes must be at depth $3/5 \cdot \log_8 n$, and also recall Claim 1. Then, we get that the sum over all such nodes is at most

$$\sum T(n^{2/5}, \infty) \leq 2^{3/5 \cdot \log_8 n} \cdot c_2 \left(n^{2/5}\right)^2 = c_2 \cdot n. \tag{2}$$

As for all remaining nodes that are not at the bottom level, we will again have to solve a recursion similar to what we did in subtask (i). Let thus $W(n, m)$ denote the sum of all nodes contained in the subtree rooted at a node $D(n, m)$, which hence has the following recursive definition.

$$W(n, m) = W(\frac{n}{8}, \frac{3m}{4}) + W(\frac{n}{8}, \frac{3n}{8}) + D(n, m) \tag{3}$$

Starting from equation (3), it can be shown by induction on n again that we have in fact $W(n, m) \leq 4c_0 \cdot (n + m)$. Combining this with equation (2) yields the desired claim if we choose the constant $c_3 := 4c_0 + c_2$, for example. □

(b). Observe, again, that the bound obtained from Claim 3 is in fact $O(m)$ for all connected graphs G. The exercise is hence concluded by proving the following final claim.

Claim 4. *For any connected input graph G, the running time of the algorithm is dominated by the sum over all nodes in the tree depicted in Figure 1 with probability $1 - o(1)$ (i.e., a number that tends to 1 as $n \to \infty$).*

Proof. For every inner node $D(n', m')$ of the tree in Figure 1 we simulate the following random experiment. We let G be any graph on n' vertices and m' edges and then run the algorithm on G, but without actually executing the recursive calls. Instead, we only observe the sizes of the produced graphs G_1 and G_2 that would have been used as the inputs for the two respective subproblems. We say that G_1 is *bad* if it has more than $\frac{n'}{8}$ vertices or more than $\frac{3m'}{4}$ edges, and we say that G_2 is *bad* if it has more than $\frac{n'}{8}$ vertices or more than $\frac{3n'}{8}$ edges. From the definition of the algorithm it is clear that neither graph can ever be bad because it has too many vertices.

For G_1 to be bad we would therefore need at least $\frac{3m'}{4}$ heads out of m' coin flips. This probability is upper bounded by Chernoff as follows, where $X_1, X_2, \ldots$ is a sequence of indicator random variables that are 1 iff the corresponding coin flip is heads.

$$\Pr[G_1 \text{ is bad}] = \Pr\left[\sum_{i=1}^{m'} X_i \geq \frac{3m'}{4}\right] \leq \exp\left(-\frac{(\frac{1}{2})^2 \cdot \frac{m'}{2}}{2}\right) = e^{-\frac{m'}{16}}$$

For G_2 to be bad we would need, in addition to the $\frac{n'}{8}$ edges coming from the spanning tree, at least $\frac{n'}{4}$ non-heavy edges. In other words, we would need at least $\frac{n'}{4}$ tails before getting $\frac{n'}{8}$ heads, which is upper bounded by the probability of getting at least $\frac{n'}{4}$ tails out of $\frac{3n'}{8}$ coin flips. This probability is bounded again by Chernoff as follows, where

$Y_1, Y_2, \ldots$ is a sequence of indicator random variables that are 1 iff the corresponding coin flip is tails.

$$\Pr[G_2 \text{ is bad}] = \Pr\left[\sum_{i=1}^{3n'/8} Y_i \geq \frac{n'}{4}\right] \leq \exp\left(-\frac{(\frac{1}{3})^2 \cdot \frac{3n'/8}{2}}{2}\right) = e^{-\frac{n'}{96}}$$

By using the union bound and the fact that $m' \geq n' \geq n^{2/5}$ in all nodes[1] we then get

$$\Pr[G_1 \text{ or } G_2 \text{ is bad}] \leq e^{-\frac{m'}{16}} + e^{-\frac{n'}{96}} \leq 2e^{-\frac{n^{2/5}}{96}}.$$

Finally, the union bound allows us to sum over all simulations for all nodes of the tree.

$$\Pr[\text{Any generated graph in any node is bad}] \leq n^{1/5} \cdot 2e^{-\frac{n^{2/5}}{96}} \to 0 \quad (\text{as } n \to \infty)$$

It remains to observe that in the run of the actual algorithm, as opposed to our simulation, the probabilities of generating a bad G_1 or G_2 at any step of the recursion are even smaller than what we computed, since there might be even fewer vertices and fewer edges present to start with than what we assumed in our derivation.

□

Solution 3

(i) Assume that $n = |V|$ and $m = |E|$. Consider an arbitrary labeling $e_1, \cdots, e_m$ on the edges. Partition the vertex set V into two sets V_1 and V_2 of size $n/2$ at random. Define Bernoulli random variable x_i for $1 \leq i \leq m$ to be 1 if and only if $e_i \in \partial(V_1)$. There are $\binom{n}{n/2}$ possible ways to partition the vertex set into two sets of size $n/2$. Furthermore, for an edge $e = \{v, u\}$, the number of ways to partition the vertex set into two sets of size $n/2$ such that v and u are in different sets is equal to $2\binom{n-2}{n/2-1}$. Therefore,

$$\Pr[x_i = 1] = \frac{2\binom{n-2}{\frac{n}{2}-1}}{\binom{n}{\frac{n}{2}}} = \frac{2(n-2)!\ (\frac{n}{2})!\ (\frac{n}{2})!}{n!\ (\frac{n}{2}-1)!\ (\frac{n}{2}-1)!} = \frac{n^2}{2n(n-1)} \geq \frac{1}{2}.$$

Let random variable $X = \sum_{i=1}^{m} x_i = |\partial(V_1)|$ be the number of edges between V_1 and V_2. By linearity of expectation, we have

$$\mathbb{E}[X] = \mathbb{E}[\sum_{i=1}^{m} x_i] = \sum_{i=1}^{m} \mathbb{E}[x_i] = \sum_{i=1}^{m} \Pr[x_i = 1] \geq \sum_{i=1}^{m} \frac{1}{2} = \frac{m}{2}.$$

This implies that there exists a partitioning of V into two sets of size $n/2$ with at least $m/2$ edges in between. In other words, there is a balanced cut of size at least $m/2$.

[1] This is the only step that requires us to have made the arbitrary cut-off of the tree in Figure 1, as otherwise we would not be able to prove that the failure probability vanishes for every node.

(ii) By moving a vertex from one side to the other, the number of edges between the two parts increases at least by one. Thus, after at most m swaps, the process terminates.

Furthermore, by applying $m = \sum_{v \in V} d(v)/2$, where $d(v)$ is the degree of vertex v, we have

$$\partial(V_1) = \frac{\sum_{v \in V_1} d_{V_2}(v) + \sum_{v \in V_2} d_{V_1}(v)}{2} \geq \frac{\sum_{v \in V} \frac{d(v)}{2}}{2} = \frac{m}{2}.$$

Solution 4

Create a graph G with n vertices $v_1, \cdots, v_n$ representing the houses and edges between every pair of vertices representing the potential pipes. Then add an additional "source" vertex v which connects to vertex v_i with cost C_i, so that the new graph G' has $n+1$ vertices.

Now, compute an MST T in G'. If vertex v_i is connected to v in T, provide house h_i with a well and if there is an edge between v_i and v_j in T, set a pipe between h_i and h_j. We claim this construction provides all houses with water and has the minimum cost.

First, we discuss it provides all houses access to water. Since T is a spanning tree, for each vertex v_i there is a path to v. Thus, a vertex v_i is directly connected to v, which implies that h_i has its own well, or v_i has a path to a vertex which is connected to v, which implies h_i has a pipe connection to a house with well.

Regarding the optimality, assume that there is a solution S with lower cost. We prove then there exists a connected spanning subgraph, consequently a spanning tree, which is lighter than T. Consider the subgraph $G' = (V, E')$, where

$$E' := \{\{v_i, v\} : \text{ if } h_i \text{ has its own well in S for } 1 \leq i \leq n\} \cup$$
$$\{\{v_i, v_j\} : \text{if } h_i \text{ has a pipe to } h_j \text{ in S for } 1 \leq i, j \leq n\}.$$

It suffices to show that this subgraph is connected. We show that each vertex has a path to v. If h_i has a well, then v_i has an edge to v. Otherwise h_i has a pipe connection to a house which has its own well; in other words, v_i has a path to a vertex which is connected to v.