

Solution description for minmaxtree

Author: Tamio-Vesa Nakajima

Some notation:

- $M(x, y) = \{ \text{the set of weights of all maximum type restrictions that pass through edge } (x, y) \} \cup \{ \infty \}$
- $m(x, y) = \{ \text{the set of weights of all minimum type restrictions that pass through edge } (x, y) \} \cup \{ -\infty \}$
- $CM(x, y) = \min(M(x, y))$
- $Cm(x, y) = \max(m(x, y))$

The main observations used to solve this problem are that:

- For each edge (x, y) one can only assign a weight included in the interval $[Cm(x, y), CM(x, y)]$
- A solution is valid if and only if, for each restriction (x, y, w) there exists at least one edge on the path from x to y which takes value w .

First, how can Cm or CM be found? There are several ways of doing this ; I will describe one way of finding Cm , as CM is found identically. To calculate $Cm(x, y)$ for all edges (x, y) , first sort the minimum restrictions in decreasing order of w . Consider each restriction (x, y, w) in this order ; it is necessary to set the value of $Cm(p, q)$ to w for all edges (p, q) on the path from x to y which have not been set so far. At this point it is important to find an efficient way to “skip over” the edges which have already been set. To do this, set an arbitrary node as the root of the tree, and initialise a union-find data structure whose keys coincide with the nodes of the tree, maintaining the node of least height for each set of nodes in the union find-data structure. Now, to process path (p, q) , simply move the deeper of p, q to either its father f (if $Cm(p, f)$ hasn't yet been set, setting $Cm(p, f)$ to w and joining p and f in the union-find data structure), or to the node of minimal height in the set in which p belongs.

Note that in general any solution where a weight different from both $Cm(x, y)$ and $CM(x, y)$ is attributed to edge (x, y) can be turned into one in which one of those values is attributed to the edge. Thus consider only this kind of solution.

The problem now asks us to attribute at least one of $Cm(x, y)$ or $CM(x, y)$ to each edge (x, y) such that edge non-infinite restriction value appears at least once throughout the assignment. A simple way to do this is to create a bipartite graph where the nodes in the left part represent restriction values, nodes in the right part represent tree edges, and an edge in the bipartite graph is added between the node that represents (x, y) and the nodes that represent $Cm(x, y)$, $CM(x, y)$. Finding the maximal matching in this graph solves the problem — this solution is faster than expected due to the fact that each node on the right side of the graph has degree at most 2.

There also exists a linear solution for this final part, which I now sketch. First, immediately and match all nodes on the right side whose degree is exactly 1. Continue doing this until no such nodes exist. Now, ignore nodes on the right side whose degree is 0. Only nodes with degree 2 remain. Create yet another graph whose node set coincides with the nodes on the left side of the previous bipartite graph, and which includes an edge (x, y) if and only if there exists some node z such that there exist edges (x, z) and (z, y) in the bipartite graph. Now, consider some way of orienting the edges of this graph such that at least each node has out degree at least one (this can be done in $O(N)$ with a depth first search). If we orient edge (x, y) like so: $x \rightarrow y$, and there exists edges (x, z) and (z, y) in the original bipartite graph, the add edge (x, z) to the bipartite matching. Do this for all edges in the new graph. Now, note that the “matching” is no longer exactly a “matching”, as some nodes on the left side might be “matched” to several on the right — however, since each node on the left is matched to at least one on the right,

assigning weights using the result of this algorithm leads to an assignment of weights where each restriction's weight is used by at least one edge — which is exactly the desired result.

In conclusion, 100 points can be achieved both with a solution that first sorts the edges in $O(n \log(n))$, then finds C_m and c_m in $O(n \log^*(n))$, and then assigns weights to edges in either $O(n \cdot \sqrt{n})$ or $O(n)$ respectively.