Sublinear Algorithms: Minimum Spanning Forest

Problem ID: kth.avalg.spanningfores
CPU Time limit: 3 secor
Memory limit: 1024 ME

Source: Jan van den Bra

Usually a graph algorithm requires $\Omega(|E|)$ time, because the algorithm has to read the entire input, i.e. every edge of the graph. As a result classical graph algorithms can often not be used for extremely large graphs. Interestingly, for many problems we do not need to read the entire graph in order to give an approximate solution.

The task

Write a program that estimates the weight of a minimum spanning forest, such that:

- The program only needs to know small fraction of the input.
- The program outputs a $(1 \pm \varepsilon)$-approximation of the weight of a minimum spanning forest. This means $(1-\varepsilon)\text{opt} \leq \text{output} \leq (1+\varepsilon)\text{opt}$.
- Even though the optimal value is an integer, your program is allowed to return a float value as approximation. E.g. you could output 9.5 if you don't know if the optimal value is 9 or 10.
- The program is allowed to be randomized.

You may assume the following about the input graphs:

- The graph is undirected.
- The input graphs will have less than 2^{31} nodes. So you do not have to worry about integer overflows when using signed 32 bit integers.
- The input graphs will have small degree, i.e. bounded by some constant.
- The weights c_{ij} of the edges are natural numbers.
- You may assume the optimal spanning forest has weight at least $n/2$.
- The nodes are indexed as $0, 1, \ldots, n-1$.

Interaction

As you are supposed to only read a small fraction of the input, your program is not given the entire graph at once. At first, you are only given the number of nodes n, the desired approximation ratio $(1+\varepsilon)$ and the maximum edge weight W. Your program is then allowed to request more data from Kattis.

Requesting data:

If your program wants to know the neighbors of node number v, then it should print the number v to `stdout`. Kattis will then give the neighbor information on `stdin`. The format of the answer is as follows: The answer is given by a single line, starting with the number of neighbors. The rest of the line is a list of neighbors and the cost of the respective edges. For example `3 5 2 9 4 1 1` means the node v has 3 neighbors 5, 9 and 1. The cost of these edges is $c_{v,5} = 2$, $c_{9,5} = 4$ and $c_{v,1} = 1$.

Reporting your result:

If your program wants to report a result, it should print `end` followed by the weight of the spanning forest. For example you should print `end 10.312`, when your program computed a weight of 10.312.

Communication example

Your program's output (stdout)	Kattis' input/answer (stdin)	Interpretation
	`10`	The graph has 10 nodes.
	`1.2`	The output is supposed to be a 1.2 approximation.
	`4`	The edge weights are at most $c_{i,j} \leq 4$.
`7`		Your program asks for the neighbors of node 7.
	`3 1 3 6 2 3 1`	Node 7 has 3 incident edges: $(7,1)$ with weight 3, $(7,6)$ with weight 2 and $(7,3)$ with weight 1.

`1`		Your program asks for the neighbors of node 1.
	`1 7 3`	Node 1 has only one neighbor: 7 and the corresponding edge has weight 3.
`2`		Your program asks for the neighbors of node 2.
	`0`	Node 2 has no neighbors
`end 4.12`		Your program thinks 4.12 is a 1.2-approximation for the weight of the minimum spanning forest. Your program should exit now.

Note that the empty lines in the table above are just for formatting. The actual content of `stdin`/`stdout` looks like this:

Your program's output (stdout)	Kattis' input/answer (stdin)
`7`	`10`
`1`	`1.2`
`2`	`4`
`end 4.12`	`3 1 3 6 2 3 1`
	`1 7 3`
	`0`

Score

There are six test groups, each of which tests how well your algorithm works on a certain type of graph/input (e.g. fixed weight, varying weight, disconnected graphs, connected graphs etc.). Your submission will get a score ranging from 0 to 10 for each test group, where a lower score means that your submission requested more nodes. The fewer nodes your submission requests from Kattis, the better your score. The total score will be the sum of the scores of each test group.

When your submission returns a value that is not a valid $(1 \pm \varepsilon)$-approximation, or when your submission reads far too many nodes, then the entire test group will be graded with 0.

Remarks on performance:

The best score is obtained by reading as little information of the graph as possible. The actual runtime of your submission does not matter, so you do not need to write your program in C++ or other low-level languages in order to get a good score.