Project Programming Competition

In an **optional**, just-for-fun competition, we'd like to test the performance of the programs you've implemented for the project.

The program you've implemented should:

1. Read the problem instance from the UWG file specified in a command-line argument.
2. Find the minimum-cost mirror-friendly spanning tree.
3. Output the minimum value B to standard output on a single line, followed by $n - 1$ space-separated integers specifying which edges form a spanning tree satisfying MFMST requirements for the given B.
   If the program does not finish searching within **20 seconds**, output the best solution found by that time and terminate.

Scoring

Your programs will be run on a number of instances of varying size.

For each instance, we'll rank the implementations by the B values they output (if a solution is invalid or not output in time, set $B = \infty$). The k implementations with the lowest B value earn n points[1]; implementations with the second-lowest B value earn $n - k$, etc[2].

The final score is the sum of all points earned for all test instances; the team with the highest score wins.

[1] Where n is the number of teams participating.

[2] If all B's are distinct, this corresponds to $n, n-1, n-2, \ldots, 1$ points. $B = \infty$ solutions always earn exactly 1 point.

Platform details

Some details about the competition environment:

- A **VirtualBox** instance running **OpenBSD 5.3** (amd64).
- 1 core (1.8 GHz Core i5), 2048 GB RAM.
- No network connection during the competition.

You should submit your source code, instructions specifying how to produce an executable, and how to run it on this platform[3].

Submissions are handled through an assignment on CampusNet; the deadline is **November 10th, 23:59**.

[3] You can assume that `gcc`/`g++` and `javac`/`java` are present.

Input details

Your implementations will be tested on instances satisfying all of the limits below:

$$1 < n \leq 500$$

$$2^{31} > \sum_{i=1}^{m} w(e_i)$$

Furthermore, all inputs are *valid* and follow the described UWG file specification; they specify no duplicate edges, no self-loops, and reference only vertices in $\{1, \ldots, n\}$.