Task C23. MONOPOLY 2

⏳1 sec. 💾 256 MB

Author: Iliyan Yordanov

After you helped **D**eni to make the perfect playing map for monopoly in 2021, she needs your help again! We recall that the playing map for monopoly consists of N spaces (numbered from 1 to N) and M directed connections between them. Here $M = N - 1$. The directed connections fulfill the following conditions - there isn't a connection from a space to itself and there aren't different connections between the same pair of spaces. Also the connections have the property that every space is reachable from space 1.

Definition: Let us have a permutation of the numbers from 1 to N. We call the permutation an *appropriate* ordering of the spaces if the following condition holds - for every connection from space i to space j, i has to be **before** j in the permutation.

The problem for **D**eni is that she lost the perfect map for playing monopoly. Fortunately, her friend Bobi remembers the map but he decided to have some fun with her before letting her know the connections. Firstly, Bobi tells **D**eni that an *appropriate* ordering of the spaces is the sequence $1, 2, \dots, N$. After that Bobi will answer some questions from **D**eni to help her find out what are the connections between the spaces. Each question will be about how *appropriate* is some permutation of the numbers from 1 to N (for more details see section "Implementation details"). Our heroine requests your assistance to come up and implement a strategy with the fewest questions possible.

Task

Write the program **`monopoly2`** that determines the unknown connections with the fewest possible questions to Bobi. It must have the function `find_connections` which will be compiled with the jury's program (for the role of Bobi).

Implementation details

Your function `find_connections` must have the following format:

```
std::vector <std::pair <int, int> > find_connections (int N);
```

It will be called once by the jury's program with one parameter - the number of spaces. The function should return a list of ordered pairs describing the found connections between the spaces. The order of the ordered pairs in the list doesn't matter.

The function for asking questions to Bobi has the following format:

```
std::vector <bool> check (std::vector <int> p);
```

The parameter p is the permutation of the numbers from 1 to N in the question. The result is a boolean vector b of size N, where $b_i = 1$ ($0 \le i < N$) if and only if one of these holds:

- there is a connection from space p_j to space p_i with $i < j$;
- there is a space p_j for which $b_j = 1$ and you can walk along the connections from p_j to p_i.

Note that the unambiguity of the values follows from the properties of the connections.

If the sequence is not a valid permutation of the numbers from 1 to N, you will get `Wrong answer` for the test. The complexity of the function is $O(N)$. You can call this function at most $\frac{10^8}{N}$ times, otherwise you will get `Wrong answer`.

Your program **`monopoly2`** must implement the function `find_connections`. It can also contain other code, functions, and global variables, but it must not contain the `main` function. Also, you should not read from the standard input or print to the standard output. Your program must include the header file `monopoly2.h` by instruction to the preprocessor:

```
#include "monopoly2.h"
```

Constraints

♣ $1 \leq N \leq 1\ 000$;
♣ $M = N - 1$.

Subtasks

Subtask	Points	Required subtasks	N	Other constraints
0	0	–	–	The map from the sample communication.
1	5	–	$\leq 1\ 000$	$1, 2, \dots, N$ is the only *appropriate* ordering of the spaces.
2	6	0	≤ 6	-
3	17	1	$\leq 1\ 000$	$1, 2, \dots, N$ can be also obtained by: starting with space 1, after that listing the spaces with direct connections from 1, after that listing the spaces with direct connections from the second added space, and so on.
4	13	$0, 2$	≤ 300	-
5	59	$0 - 4$	$\leq 1\ 000$	If we start from space 1 and follow the connections, we cannot use more than 25 connections.

*The points for a given subtask are obtained only if all the tests for it and the required subtasks are **successfully** passed, and the points are equal to the minimum test score in it and the required subtasks, multiplied by the points of the subtask.*

Scoring

Each test receives a score that is a fractional number between 0 and 1 inclusive. If a test has a positive score, it is considered **successful** for your solution. A test has a positive score if you successfully find the connections between the spaces.

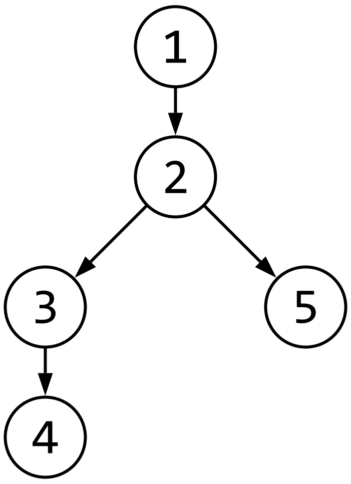
If cnt is the number of calls to the function `check` for a particular test, then the score of the test is calculated in the following way:

- For all subtasks: if $cnt > \frac{10^8}{N}$, then the score is equal to 0.
- Subtasks 0, 1, and 2.
  - If $cnt \leq \frac{10^8}{N}$, then the score is equal to 1.
- Subtask 3.

  - If $cnt \le 10\ 000$, then the score is equal to 1.
  - If $10\ 000 < cnt \le 15\ 000$, then the score is equal to $\min(\frac{1000}{cnt-9000}, 1)$.
  - If $15\ 000 < cnt \le \frac{10^8}{N}$, then the score is equal to 0.1.
- Subtask 4.
  - If $cnt \le 50\ 000$, then the score is equal to 1.
  - If $50\ 000 < cnt \le 200\ 000$, then the score is equal to $\min(\frac{25000}{cnt-25000}, 1)$.
  - If $200\ 000 < cnt \le \frac{10^8}{N}$, then the score is equal to 0.1.
- Subtask 5.
  - If $cnt \le 170$, then the score is equal to 1.
  - If $170 < cnt \le 1000$, then the score is equal to $\min((\frac{170+3000}{cnt+3000})^{\frac{3000}{cnt}}, 1)$.
  - If $1000 < cnt \le 20\ 000$, then the score is equal to $\min((\frac{170}{cnt})^{0.4}, 0.5)$.
  - If $20\ 000 < cnt \le \frac{10^8}{N}$, then the score is equal to 0.1.

Sample communication

Let we have the following illustration of a map with 5 spaces and 4 connections:



Actions of your program	Actions and answers of the jury
	`find_connections(5)`
`check({2, 3, 4, 5, 1})`	`return {1, 1, 1, 1, 0}`
`check({1, 5, 2, 3, 4})`	`return {0, 1, 0, 0, 0}`
`check({1, 4, 3, 2, 5})`	`return {0, 1, 1, 0, 0}`
`return {{1, 2}, {2, 3}, {3, 4}, {2, 5}}`	

Local testing

For local testing the following files are provided: `monopoly2.h`, `Lgrader.cpp`, sample file `monopoly2.cpp` for your program and file with the map from the sample communication. When the provided files are in the same folder, you can compile together your program `monopoly2.cpp` and `Lgrader.cpp`. This will make a program to check the correctness of your function.

The program will require from the standard input the following sequence of numbers:

- on the first line: one positive integer – the number of spaces N;
- on each of the following $N-1$ lines two positive integers which describe the directed connections.

If you don't follow the protocol for communication, you will get an appropriate error message. Otherwise, if the program is successful, you will get the message "Correctly found connections.".