Meetings

There are N islands where beavers live, numbered from 0 through $N-1$. These islands are connected by $N-1$ bidirectional bridges. It is possible to travel between any islands using some bridges. **For each island, there are at most 18 bridges directly connected to it.** Each island is inhabited by a beaver.

Sometimes, some beavers gather at an island to hold a meeting. When exactly three beavers meet, they gather at the following island:

> The island which minimizes the total number of bridges the three beavers travel through when gathering (such island uniquely exists).

Note that this island may coincide with an island where one of the three beavers lives.

You are interested in how N islands are connected by bridges. You cannot go and check the islands directly, so you are going to issue some instructions to the beavers. An instruction is as follows:

- You specify three islands u, v and w ($0 \leq u \leq N-1, 0 \leq v \leq N-1, 0 \leq w \leq N-1, u \neq v, u \neq w, v \neq w$), and let the beavers living in the islands u, v and w hold a meeting.
- Then, you can see the island where the three beavers gather.

You want to determine how the islands are connected with few instructions.

Write a program which, given the number of islands, communicating with beavers, determine how the islands are connected.

Implementation Details

You need to submit one file.

The name of the file is `meetings.cpp`. It should implement the following function. The program should include `meetings.h`.

- `void Solve(int N)`

  This function is called exactly once for each test case.
  - The parameter `N` represents N, the number of islands.

  Your program can call the following functions.

  ★ `int Query(int u, int v, int w)`

  This function returns, for the indices of three islands you specify, the index of the island where the beavers living there gather for a meeting.

- ◇ You specify the indices u, v and w of the three islands via parameters `u`, `v` and `w`, respectively. These must satisfy $0 \leq u \leq N-1$, $0 \leq v \leq N-1$, $0 \leq w \leq N-1$, $u \neq v$, $u \neq w$ and $v \neq w$. Otherwise, your program is considered as **Wrong Answer [1]**.
- ◇ The function `Query` must not be called more than 100 000 times. Otherwise, your program is considered as **Wrong Answer [2]**.

★ `void Bridge(int u, int v)`

Using this function, you answer how the islands are connected by bridges.

- ◇ The parameters `u` and `v` represent that the islands u and v are directly connected by a bridge.
- ◇ If $0 \leq u < v \leq N-1$ does not hold, your program is considered as **Wrong Answer [3]**.
- ◇ If the islands u and v are not directly connected by a bridge, your program is considered as **Wrong Answer [4]**.
- ◇ If the function `Bridge` is called with the same parameters u and v multiple times, your program is considered as **Wrong Answer [5]**.
- ◇ Since there are $N-1$ bridges, the function `Bridge` must be called exactly $N-1$ times. If the number of calls to the function `Bridge` is not $N-1$ at the end of the execution of the function `Solve`, your program is considered as **Wrong Answer [6]**.

Important Notices

- Your program can implement other functions for internal use, or use global variables.
- Your program should not use the standard input and the standard output. Your program should not communicate with other files by any methods. But, your program may output debugging information to the standard error.

Compilation and Test Run

You can download an archive file from the contest webpage which contains the sample grader to test your program. The archive file also contains a sample source file of your program.

The sample grader is the file `grader.cpp`. In order to test your program, put `grader.cpp`, `meetings.cpp`, and `meetings.h` in the same directory, and run the following commands to compile your programs.

```
g++ -std=gnu++14 -O2 -o grader grader.cpp meetings.cpp
```

When the compilation succeeds, the executable file `grader` is generated.

Note that the actual grader is different from the sample grader. The sample grader will be executed as a single

process, which will read input data from the standard input and write the results to the standard output.

Input for the Sample Grader

The sample grader reads the following data from the standard input.

N
$A_0\ B_0$
$\vdots$
$A_{N-2}\ B_{N-2}$

A_i and B_i $(0 \leq i \leq N-2)$ represent that the islands A_i and B_i are directly connected by a bridge.

Output for the Sample Grader

When the program terminates successfully, the sample grader writes the following information to the standard output (quotes for clarity).

- If your program is considered correct, it writes the number of calls to the function `Query` as "`Accepted: 100`".
- If your program is considered as Wrong Answer, it writes its type as "`Wrong Answer [1]`".

If your program is considered as several types of Wrong Answer, the sample grader reports only one of them.

Constraints

Refer to the "Input for the Sample Grader" section for the definition of A_i and B_i.

- $3 \leq N \leq 2\,000$.
- $0 \leq A_i < B_i \leq N-1$ $(0 \leq i \leq N-2)$.
- It is possible to travel between any islands using some bridges.
- For each island, there are at most 18 bridges directly connected to it.

Subtasks

1. (7 points) $N \leq 7$.
2. (10 points) $N \leq 50$.
3. (12 points) $N \leq 300$.
4. (71 points) No additional constraints.

- For Subtasks 1, 2 and 3, if your program is correct for all test cases in a subtask, you are given the full score.
- For Subtask 4, if your program is correct for all test cases, your score will be the following, where X is the maximum number of calls to the function `Query` for a test case.
  - 49 points if $40\,000 < X \leq 100\,000$, and
  - 71 points if $X \leq 40\,000$.

Sample Communication

Here is a sample input for the sample grader and corresponding function calls.

Sample Input 1	Sample Calls		
	Call	Call	Return

`5`
`0 1`
`0 2`
`1 3`
`1 4`
	`Solve(5)`		

		`Query(0, 1, 2)`	`0`
		`Query(0, 3, 4)`	`1`
		`Bridge(1, 3)`	(none)
		`Bridge(0, 2)`	(none)
		`Bridge(1, 4)`	(none)
		`Bridge(0, 1)`	(none)