

Czech Technical University in Prague
ICPC Foundation
Czech ACM Chapter



Central Europe Regional Contest 2020

Bank Robbery

`bank.c`, `bank.cpp`, `Bank.java`, `bank.py`

Each day, robbers plan to rob exactly one bank in a region. Due to undercover informers' work, the detectives get to know which particular bank is being targeted by the robbers on each day just before 6 AM. The presence of a single detective in a bank is enough to deter the robbers, so the detectives want to plan their positions intelligently. Robbing is planned to happen during the daylight after 8 AM, when the banks open, and is always successful when there is no detective in the bank. Unfortunately, though, the number of detectives is not very big so there may not be enough of them to keep the banks safe. To ensure they are effective, there can be at most one detective in any bank at any time. A detective can leave a bank and travel to another bank only between 6 AM and 8 AM.

Given enough days, any detective would be able to move from any of the banks to any other via the 2-hour transits. Due to the travel restrictions (mainly time), they are not able to move freely but are restricted to move only between banks that are close to each other. The region is quite specific because there is a minimum number of bank pairs that are close that conform to the above restrictions. Additionally, no bank is close to exactly two other banks.

Now, there is a quest for a computer simulation: determining if the robbers can succeed in at least one robbery during one year. The simulation is run against the preprogrammed judge as a kind of computer game, with strong consequences in real life. In the game, the attacker represents the robbers, the defender manages the detectives.

In the beginning, the simulation is given the system of connections between close banks and the number of detectives available. Then, the simulation chooses whether it plays as an attacker or as a defender. The judge automatically adopts the opposite role. Next, the defender places the given number of detectives in banks according to its own choice.

Next, the game proceeds in turns. In one turn, the attacker announces a bank and then the defender moves each detective over at most one connection. The defender's aim is to choose the movements in such a way the announced bank is occupied by a detective at the end of the turn. If there is no detective in the announced bank after all movements in the turn, the attacker immediately wins the game. Otherwise, the defender defends the turn and the next turn ensues. If the defender can successfully defend for one year (365 turns), they win the game. During the game, the location of each detective is known to both the attacker and the defender.

The goal of the simulation you have to write is to choose a role smartly so that you are able to win the game.

Input Specification

The first input line contains values B and D ($4 \leq B \leq 100, 0 \leq D \leq B$), the number of banks and the number of detectives. The banks are labeled by integers $0, 1, \ldots, B-1$. Each of the next $B-1$ lines contains a pair of integers b_i and c_i ($0 \leq b_i, c_i \leq B-1$) representing a connection between two close banks b_i and c_i.

Output Specification

After reading the input, if you choose to attack, then prints a line with string `ATTACK`. Otherwise, print a line with string `DEFEND` and on the next line it print D different indices, in arbitrary order, of all the banks where a detective is initially located. The remainder of the exchange happens interactively.

Interactive Mode

The simulation is evaluated in so-called *interactive* mode, which means that the input received depends on the output produced so far. The output of the judge is the input of the simulation and vice versa. If you have no previous experience with such problems, just do not be afraid — you are still reading from the standard input and printing to the standard output. There are just a few things to pay attention to.

After printing each response to the input from the judge, the simulation has to flush the output buffer. For example, it may use `fflush(stdout)` or `cout.flush()` in C++, `System.out.flush()` in Java, or `stdout.flush()` in Python. Also, it should never try to read more data than what is guaranteed to be ready in the input, because that may cause it to hang indefinitely. In particular, be careful to *not* invoke anything like `scanf("%d ")` or `scanf("%d\n")`, as such formats try to scan forward for any further whitespace. Instead, use just `scanf("%d")` without trailing whitespace.

After you choose a role and print a corresponding output, the following exchange is repeated for up to 365 times: The attacker outputs the bank index $0 \leq v \leq B-1$ which they choose to attack. Then, the defender outputs an integer k and then k pairs b_i, c_i ($0 \leq b_i, c_i \leq B-1$), each of which represents a detective move from bank b_i to bank c_i. The detectives may only move along connections between close banks. The attacker ends the sequence of attacks by outputting -1. The defender may give up by ending the program.

In this exchange, the output of one player is always the input of the other player.

Sample Input 1

```
4 3
0 1
0 2
0 3

2

3

-1
```

Output for Sample Input 1

```
DEFEND
0 1 2

0

2 1 0 0 3
```

Sample Input 2	Output for Sample Input 2
4 1	
0 1	
0 2	
0 3	
	ATTACK
0	
	1
1 0 1	
	2

For clarity, the above data are interleaved to illustrate the order of interaction between the simulation and the judge. Note that there will be no empty lines in real data and there must not be any empty lines in the simulation output.