Lëtzebuerger Informatiksolympiad 2023

Semi-Finals

Task descriptions

Instructions

- The allowed programming languages are Python 3, Java and C/C++.
- All the programs must be console applications. For instructions how to write a console application in the allowed programming languages, please refer to the remarks on the site `www.infosolympiad.lu` under the heading *The tasks*.
- The input of a program can mean either the direct entry of data from the keyboard (referred to as ”standard input”) or input via specific function calls described in the task description (”no reading”). Similarly, the output of a program can mean either printing to the console (referred to as ”standard output”) or output via specific functions described in the task description (”no writing”).
- The formats of the input and output data shown in the execution examples must be respected. Other formats will not be accepted.
- For testing, submitting and evaluating a program, the source file with the correct file extension `py`, `java` or `c/cpp` must be uploaded to the automated online judge CMS (Contest Management System), accessible via the homepage `www.infosolympiad.lu` or directly via the URL `http://158.64.46.20:81`. Please use your personal login (username and password) to access your account on the CMS. The filename of the single source file should be the same than the task name. Please refer to the CMS for technical details on how to test and submit a program.
- Time limits and memory constraints are described both in the task statements and the CMS. Please refer to the CMS for compilation commands.
- You are allowed to ask questions via the CMS, however no hints concerning the use of programming languages, the implementation of algorithms or the solutions to the tasks will be given. Questions should be about CMS or seek clarification concerning the task descriptions.

{LIO;}

Find the fake coin

Notice

This is an interactive task. Please read the task statement carefully, and respect the template which can be found on CMS as an attachment.

Description

You were given N coins, labelled from 1 to N. A numismatist friend (someone who collects coins) tells you that exactly one of those coins is a fake and that it is heavier than the others. To teach you a lesson about reputable sources, they do not tell you which one is the fake. Instead they hand you double sided scales to compare your coins. This means that you can put coins on both sides of the scales, and it will tip to the heavier side.

Task

Write a program that returns the fake coin's label while using the scales as little as possible.

Functions

Use the provided template, where a skeleton solution is already implemented.
Implement a function `int guess(int N)` where N is the number of coins. At the end of its execution, the function should return the fake coin's label.
In you implementation of `int guess(int N)` you can make calls to the function:

- `query(vector<int> coinLeft, vector<int> coinRight)` in C++.
- `grader.query(Vector<Integer> coinLeft, Vector<Integer> coinRight)` in Java.
- `def query(coinLeft:list, coinRight:list)-> int` in Python.

Here, `coinLeft` and `coinRight` should be vectors or lists containing the labels of the coins you wish to place on either side of the scales. The call will return 1 if the left side is heavier, -1 if the right side is heavier or 0 if the scales are in equilibrium. Please note that both vectors or lists must be of the same size and each label cannot occur more than once. Otherwise your code will be terminated for a **Protocol Violation**.

Call Example

The function guess is called with $N = 6$. Assume coin number 2 is fake, and coins $1, 3, 4, 5, 6$ are genuine.

- The call `query({1,2},{4,3})` will return 1 as the left side is heavier, as it contains the fake coin.
- The call `query({1},{3})` will return 0 as both sides weigh the same amount.
- The calls `query({1},{2,3})`, `query({4,4},{5,6})` and `query({4,5},{4,6})` will trigger a **Protocol Violation** as the first call has different amount of coins on both sides and the two last contain the same coin twice.

Constraints

- $2 \leq N \leq 120$

Input and output of the program

Your implementation should not read from standard input nor write to standard output. This is done entirely by the grader.
If you choose to create your own test files to test your solution with the local grader or the CMS test tool, respect the following format for the input: a single line containing the amount of coins and the label of the fake coin, separated by a whitespace.

Input

```
6 2
```

Distribution of points

The score or your program depends on the maximum number of queries across all test cases Q:

- If $5 \leq Q \leq 120$: your score is $\lfloor 30 * \exp\left(-0.029575(Q-5)\right)\rfloor$, meaning $Q = 5$ gives 30 points and $Q = 120$ gives 1 point.
- If $Q < 5$, you receive 30 points.
- If $Q > 120$, you receive no points.

Technical constraints

Task name	fakecoin
Input file	no reading
Output file	no writing
Time limit	1 second
Memory limit	256 megabytes

Zeckendorf Representation

Description

Lea just started her journey in competitive programming. After learning about Fibonacci numbers and how to compute them efficiently, she heard another interesting fact. Each positive number can be uniquely represented as the sum of non-consecutive Fibonacci numbers. This is the Zeckendorf representation of a positive integer.

Formally, given a (shifted) Fibonacci sequence defined as $F_1 = 1,\ F_2 = 2,\ F_n = F_{n-1} + F_{n-2}\ \forall n > 2$, the Zeckendorf representation of an positive integer N is the unique sequence of m binary digits $d_i \in \{0, 1\}$, written as $d_m d_{m-1} \dots d_2 d_1$ such that $N = \sum_{i=1}^{m} d_i F_i$ and $d_m = 1$.

Task

Write a program to help Lea find the Zeckendorf representation of the Q numbers $N_1, \dots, N_Q$.

Constraints

- $1 \le Q \le 10^4$
- $1 \le N_i \le 10^8$

Input and output of program

Input data

A single line containing Q. The following Q lines contain the numbers $N_1, \dots, N_Q$, one per line.

Output data

For each number $N_1, \dots, N_Q$ output the Zeckendorf reprensentation of that number, one per line.

Execution example

Input

```
4
5
7
4
1
```

Output

```
1000
1010
101
1
```

The first 4 Fibonacci numbers are $F_1 = 1, F_2 = 2, F_3 = 3, F_4 = 5$ and we can write $5 = 1 \cdot F_4 + 0 \cdot F_3 + 0 \cdot F_2 + 0 \cdot F_1$, hence its representation is 1000. Moreover, we can write $7 = 1 \cdot F_4 + 0 \cdot F_3 + 1 \cdot F_2 + 0 \cdot F_1$, hence its representation is 1010.

Note: 110 is not a valid representation for 5 even though $F_2 + F_3 = 5$, as it contains two consecutive 1 digits.

Distribution of points

Subtask	Points	Constraints/Description
1	5	$N_i \le 10$
2	10	$Q = 1, N_i \le 10^5$
3	15	No additional constraints

Technical constraints

Task name	zeckendorf
Input file	standard input
Output file	standard input
Time limit	1 second
Memory limit	256 megabytes

Botanical garden

Description

Jempi is visiting a botanical garden, which can be represented as a collection of M unidirectional paths that connect N intersections, numbered from 1 to N. Unfortunately, he arrived rather late and the garden is about to close up for the day. This puts him at odds with the gardener, who would like to go home. Jempi starts his stroll at intersections S. Each time he arrives at an intersection, the gardener gets in his way and closes off one of the outgoing paths. Jempi then chooses one of the available paths and follows it to the next intersection. The gardener follows along to the next intersection and yet again closes of one of the outgoing paths. The path she previously blocked is now open again. They both continue in this fashion until one of two things happens. The first possibility is that the gardener has forced Jempi into a dead end, meaning that there is no open path available to him. In this case, she escorts Jempi back to the exit and closes up the garden. The second possibility is that Jempi is never forced into a dead end. Once the gardener realises this, she goes off to take care of flowers somewhere else in the garden, letting Jempi finish his walk without further interruption.

Task

Write a program that given the shape of the garden as well as Jempi's starting intersection determines whether he can be forced into a dead end. You need to assume that both Jempi and the gardener always make optimal choices. This means that Jempi always chooses the path that will make his walk as long as possible, while the gardener will choose to block a path that would make Jempi's walk as long as possible. If Jempi can be forced into a dead end, compute the maximum number of intersections he can visit before he is escorted out of the garden.

Example

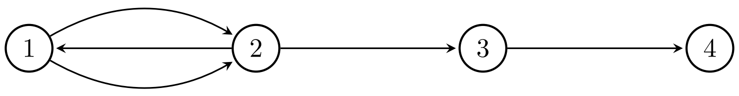

Figure 1: First example

In the first example, Jempi starts at intersection 1. The gardener will then close of one of the two paths leading to intersection 2. It does not matter which intersection she chooses. Once Jempi arrives at intersection 2, the gardener will close off the path to intersection 1. Jempi then has to continue on to intersection 3, at which point there are no more paths for him to take (as the gardener will close off the path to intersection 3), and he is escorted to the exit.

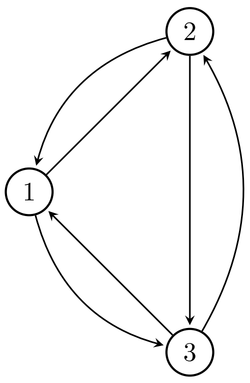

Figure 2: Second example

In the second example, no matter which paths the gardener closes off, Jempi can never be forced into a dead end. Each path has at least two outgoing edges, and so Jempi can never be cornered.

Constraints

- $1 \leq S \leq N \leq 1000$
- $0 \leq M \leq 3N(N-1)$
- There can be multiple paths between two intersections.
- There can be loops, i.e. paths where the start and end intersection are the same.

Input and output of program

Input data

The first line contains the natural numbers N, M and S, separated by spaces. The M following lines contain two integers each, the start and end intersections of a path in the garden.

Output data

If Jempi can be forced into a dead end, output the maximum number of intersections (including the start intersection) that he can visit if both he and the gardener choose their actions optimally. If the gardener cannot force Jempi into a dead end, output 0.

Execution example

Input

```
4 5 1
1 2
1 2
2 1
2 3
3 4
```

Output

```
3
```

Input

```
3 6 1
1 2
1 3
2 1
2 3
3 1
3 2
```

Output

```
0
```

Distribution of points

Subtask	Points	Constraints/Description
1	3	Intersections i and $i+1$ are connected by exactly one incoming and one outgoing edge
2	7	Any two intersections are connected by exactly one incoming and one outgoing edge
3	10	$N \leq 7$
4	20	No further constraints

Technical constraints

Task name	garden
Input file	standard input
Output file	standard input
Time limit	1 second
Memory limit	256 megabytes