Asian MathSci League, Inc (AMSLI)
Website: amsliphil.com
Email address: amsliphil@yahoo.com

Student Copy
AIEP Sendoff HS Session 10

ASSORTMENT OF PROGRAMMING CHALLENGES (v. 2.0)

"If we wish to count lines of code, we should not regard them as 'lines produced' but as 'lines spent.' "

~ Edsger Dijkstra

In this handout, we present some programming challenges. The top priority is always to produce a correct algorithm, written in syntactically and semantically correct Python code. The next step is to overhaul the algorithm with the goal of making it more efficient although this does require a bit more time, insight, and practice.

YOUR SOLUTION MUST STRICTLY ADHERE TO THE SPECIFICATIONS IN EACH PROBLEM. Unless otherwise stated, your program must accept input from the standard input device (keyboard) and display the required output on the standard output device (monitor screen). Failure to follow the instructions will invalidate your submission.

PROBLEM 8: There's a Kind of Hash

Dictionaries in Python are built via a technique known as hashing, a procedure with the objective of (ideally) reducing the complexity of searching to the absolute fastest possible time: $O(1)$. To be honest, this is an excellent example of the idiom "easier said than done."

Suppose that our hash table consists of 100 slots and uses the simple modulo-based hash function $h(x) = x \bmod 97$. Such a hash function is called deterministic since it maps a value to a precise slot. If the key leaves a remainder of 2 when divided by 97, then it gets mapped to slot 2; if the remainder is 3, then it lands on slot 3.

Before continuing with the remainder of this introduction, let us pause for a few seconds and imagine ourselves as *adversaries* (the term used in computer security to refer to malicious attackers) trying to find the weaknesses of this scheme.

*Prepared by **Mark Edward M. Gonzales***

We can attempt to "overload" a single slot by deliberately inputting entries that leave the same remainder when divided by 97. For example, 1, 98, 195, 292, etc. are "chained" and mapped to slot 1 (as seen in the table below). Another weakness is exposed when we try to store 10,000 entries; there is at least one slot that has to accommodate at least 100 entries.

Slot	Keys Stored in a "Chain"
0	(none)
1	1, 98, 195, 292, ... VERY BAD!
2	(none)
...	(none)
98	(none)
99	(none)

Collisions can thus be "manufactured" so that searching degrades to linear time — a far cry from our goal of constant time. A possible solution to mitigate this is that, instead of relying on a single deterministic function only, we sprinkle an "element of surprise" and randomly choose from a "good" bunch of functions. In this programming challenge, we are going to take a look at one such example: a universal family of hash functions.

A property which makes a family of hash functions "universal" is that, if we randomly pick a hash function from this set, then the probability of a collision occurring between any two distinct keys is at most the reciprocal of the size of the hash table.

In 1979, IBM researchers J. Lawrence Carter and Mark N. Wegman published the well-known universal family consisting of functions h that are of the form

$$h(x) = ((ax + b) \bmod p) \bmod M$$

Note that x is the key, M is the size of the table, p is a prime number greater than or equal to M, a is a positive integer less than p, and b is a nonnegative integer less than p.

Your task is to compute for the return value of this function. But there's a twist. Getting the modulo is expensive in some machines, especially those with limited computational power, and this function features two modulo operations — not really efficient!

In practice, we choose *p* to be a Mersenne prime. Recall from our AIEP session 6 handout that it is a prime of the form $2^q - 1$, where *q* is also prime. As you might have guessed, the advantage of using Mersenne primes lies in its relation to bitwise operators.

RESTRICTIONS

Ergo, your task is to compute for the return value of Carter and Wegman's proposed hash function given *x, a, b,* and *M*, subject to the following requirements:

a. *p* is the smallest Mersenne prime that is greater than or equal to *M.*
b. In the interest of efficiency, you are required to get the inner modulo (*ax* + *b*) mod *p* using only three binary operators: two distinct bitwise operators and addition (+). You are not allowed to use the modulo operator (%) to calculate the inner modulo.
c. You can use the modulo operator (%) to calculate the outer modulo (mod *M*).

To reiterate, Carter and Wegman's hash function is ***h*(*x*) = ((*ax* + *b*) mod *p*) mod *M***.

INPUT

The input consists of a single line:

a. **Line 1**: Values of *x, a, b,* and *M* in this order (separated by a space). For simplicity, assume that all values are valid and in conformity with the definition discussed. Moreover, *M* is guaranteed to be at most $2^{31} - 1$.

OUTPUT

The output consists of a single line:

a. **Line 1:** Return value of Carter and Wegman's proposed hash function

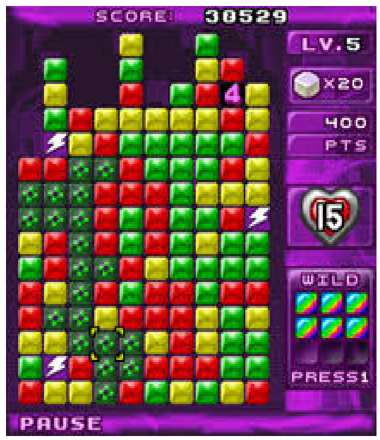
SAMPLE RUN

Input	Output
`295 99 3 100`	`25`

Prepared by ***Mark Edward M. Gonzales***

PROBLEM 9: Block'd

Back when the Nokia C3 line of cellphones was released near the start of the last decade, one of the pre-installed games is Block'd. Its core gameplay is relatively simple: we select a block and all contiguous blocks with the same color are cleared along with it. Note that, by "contiguous," we are referring to a "chain" of blocks that share at least one edge.



https://www.gameguru.in/puzzle/2007/10/blockd-available-all-over-europe-says-ea-mobile/

There are three types of colored blocks: red, green, and yellow. For simplicity, assume that they are always arranged in a complete rectangular grid. Sandra is an avid player of this mobile game; she also happens to be a diligent AIEP trainee. She becomes interested in finding the largest set of contiguous blocks that can be cleared given a specific color.

The following figure shows a simplified version of the game. The largest set of contiguous red (R), green (G), and yellow (Y) blocks are shaded accordingly:

G	G	R	R	Y	Y	Y
Y	G	G	Y	Y	R	R
R	R	G	G	G	G	Y
G	R	R	R	G	Y	G
G	G	R	Y	G	Y	G
R	R	Y	G	G	G	Y

Prepared by ***Mark Edward M. Gonzales***

INPUT *(blockd-input.txt)*

The input file consists of the following lines:

a. **Line 1:** First letter of the color of the contiguous blocks. For simplicity, assume that it is given in uppercase and that it can only be R, G, or Y.
b. **Line 2:** Number of rows and number of columns in the rectangular grid of blocks (separated by a space)
c. **Line 3 onwards:** Colored blocks (blocks in each row are separated by a space and each line corresponds to a row). R stands for a red block; G, green; and Y, yellow.

OUTPUT *(blockd-output.txt)*

The output consists of a single line:

a. **Line 1:** Number of blocks comprising the largest set of contiguous blocks given the specified color

SAMPLE RUN

Input	Output
G 6 7 G G R R Y Y Y Y G G Y Y R R R R G G G G Y G R R R G Y G G G R Y G Y G R R Y G G G Y	13

PROBLEM 10 STARTS ON THE NEXT PAGE

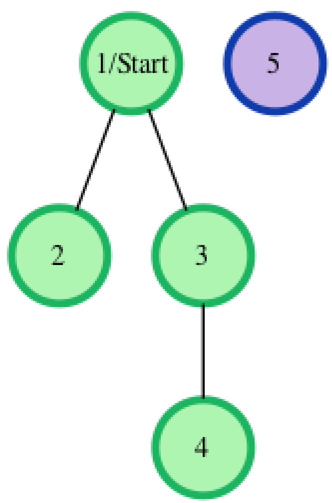
PROBLEM 10: Breadth-First Search Shortest Reach

(Hackerrank)

Consider an undirected graph where each edge is the same weight. Each of the nodes is labeled consecutively.

You will be given a number of queries. For each query, you will be given a list of edges describing an undirected graph. After you create a representation of the graph, you must determine and report the shortest distance to each of the other nodes from a given starting position using the breadth-first search algorithm (BFS). Distances are to be reported in node number order, ascending. If a node is unreachable, print **-1** for that node. Each of the edges weighs 6 units of distance.

For example, given a graph with 5 nodes and 3 edges, [1, 2], [1, 3], [3, 4], a visual representation is:



The start node for the example is node 1. Outputs are calculated for distances to nodes 2 through 5: [6, 6, 12, –1]. Each edge is 6 units, and the unreachable node 5 has the required return distance of –1.

INPUT

The first line contains an integer q, the number of queries. Each of the following q sets of lines has the following format:

- The first line contains two space-separated integers n and m, the number of nodes and edges in the graph.
- Each line i of the m subsequent lines contains two space-separated integers, u and v, describing an edge connecting node u to node v.
- The last line contains a single integer, s, denoting the index of the starting node.

OUTPUT

For each of the q queries, print a single line of $n-1$ separated integers denoting the shortest distances to each of the $n-1$ other nodes from starting position s. These distances should be listed sequentially by node number (i.e., 1, 2, ..., n), but should not include node s. If some node is unreachable from s, print **-1** as the distance to that node.

SAMPLE RUN

Input	Output
2 4 2 1 2 1 3 1 3 1 2 3 2	6 6 -1 -1 6

PROBLEM 11 STARTS ON THE NEXT PAGE

PROBLEM 11: An Industrial Spy

(2009 Europe Northwestern ACM-ICPC)

Industrial spying is very common for modern research labs. I am such an industrial spy — don't tell anybody! My recent job was to steal the latest inventions from a famous math research lab. It was hard to obtain some of their results but I got their waste out of a document shredder.

I have already reconstructed that their research topic is fast factorization. But the remaining paper snippets only have single digits on it and I cannot imagine what they are for. Could it be that those digits form prime numbers? Please help me find out how many prime numbers can be formed using the given digits.

<u>INPUT</u> ***(spy-input.txt)***

The first line of the input holds the number of test cases c ($1 \le c \le 200$). Each test case consists of a single line. This line contains the digits (at least one, at most seven) that are on the paper snippets.

<u>OUTPUT</u> ***(spy-output.txt)***

For each test case, print one line containing the number of different primes that can be reconstructed by shuffling the digits. You may ignore digits while reconstructing the primes (e.g., if you get the digits 7 and 1, you can reconstruct three primes 7, 17, and 71). Reconstructed numbers that (regarded as strings) differ just by leading zeros, are considered identical (see the fourth case of the sample input).

<u>SAMPLE RUN</u>

Input	Output
4 17 1276543 9999999 011	3 1336 0 2

Prepared by ***Mark Edward M. Gonzales***

RESTRICTION *(added to meet the time requirement of the original problem)*

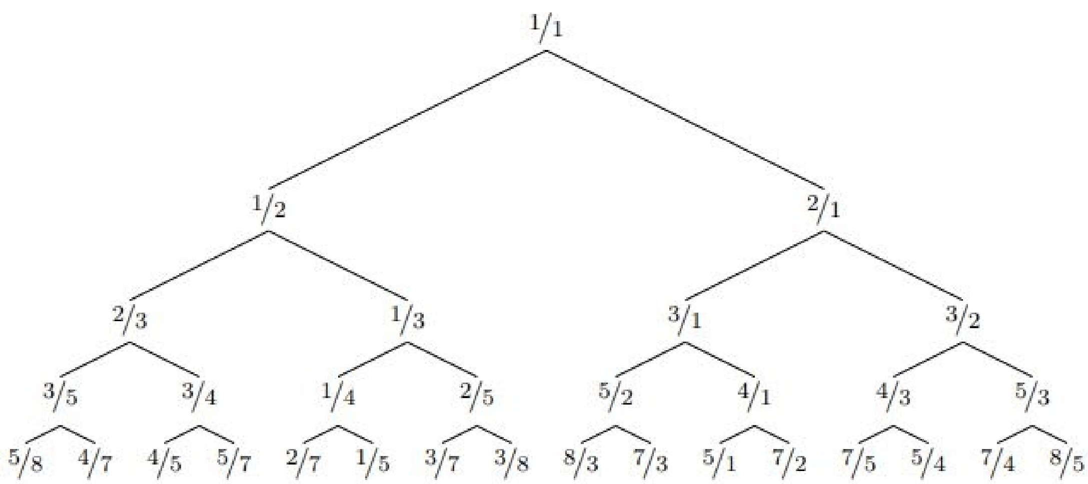
a. To test whether a number is prime or not, you are not allowed to use naive trial division (that is, testing possible divisors up to the square root of the number) in the interest of efficiency. The actual time limit to solve this problem is 1 second.
b. You are given the liberty to write your own implementation of any other (faster) primarily test, such as the $6k \pm 1$ optimization discussed in one of our sessions.

PROBLEM 12 STARTS ON THE NEXT PAGE

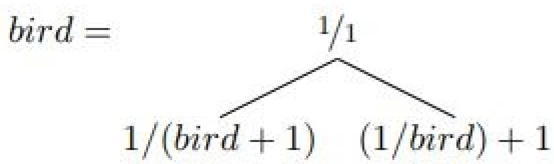
PROBLEM 12: Bird Tree

(2011 Europe Northwestern ACM-ICPC)

The Bird tree [Hinze, R. (2009). The Bird tree. *J. Funct. Program.,* 19:491-508] is an infinite binary tree, whose first 5 levels look as follows:



It can be defined as follows:



This is a *co-recursive* definition in which both occurrences of *bird* refer to the full (infinite) tree. The expression *bird* + 1 means than 1 is added to every fraction in the tree, and 1/*bird* means that every fraction in the tree is inverted (so $\frac{a}{b}$ becomes $\frac{b}{a}$).

Surprisingly, the tree contains every positive rational number exactly once, so every reduced fraction is at a unique place in the tree. Hence, we can also describe a rational number by giving directions (**L** for left subtree, **R** for left subtree) in the Bird tree. For example, $\frac{2}{5}$ is represented by **LRR**. Given a reduced fraction, returns a string consisting of **L**'s and **R**'s: the directions to locate this fraction from the top of the tree.

INPUT *(bird-input.txt)*

On the first line a positive integer: the number of test cases, at most 100. After that per test case:

- One line with two integers *a* and *b* ($1 \leq a, b \leq 10^9$), separated by a '/'. These represent the numerator and denominator of a reduced fraction. The integers *a* and *b* are not both equal to 1, and they satisfy gcd(*a*, *b*) = 1.

For every test case the length of the string with directions will be at most 10 000.

OUTPUT *(bird-output.txt)*

Per test case:

- One line with the string representation of the location of this fraction in the Bird tree

SAMPLE RUN

Input	Output
3 1/2 2/5 7/3	L LRR RLLR

-- return 0; --