2023 Team Formation Test

Task Overview

ID	Name	Time Limit	Memory Limit	Subtasks
T231	Facility Location	2.000 s	768 MB	5 + 13 + 18 + 15 + 30 + 13 + 6
T232	Challenge of Hanoi	2.000 s	768 MB	3 + 6 + 40 + 29 + 22
T233	The World Tour	1.000 s	768 MB	11 + 18 + 23 + 32 + 16
T234	Complete the Sequence	1.000 s	768 MB	14 + 36 + 27 + 23

Notice:

- All tasks are divided into subtasks. You need to pass all test cases in a subtask to get points.
- There is an attachment package that you can download from the contest system, containing sample graders, sample implementations, example test cases, and compile and run scripts.
- When testing your programs with the sample grader, your input should match the format and constraints from the task statement, otherwise, unspecified behaviors may occur.
- In sample grader inputs, every two consecutive tokens on a line are separated by a single space, unless another format is explicitly specified.
- The task statements specify signatures using generic type names `void`, `bool`, `int`, `int64`, `int[]` (array) and `int[][]` (array of array).
- In C++, the graders use appropriate data types or implementations, as listed below:

`void`	`bool`	`int`	`int64`	`int[]`	`int[][]`	length of array `a`
`void`	`bool`	`int`	`int64`	`std::vector<int>`	`std::vector<std::vector<int>>`	`a.size()`

T231 - FACILITY LOCATION

Time Limit: 2.000 s / Memory Limit: 768 MB

As population grows rapidly in Hackerland, a new town, Linearville, is being built. The town consists of a single road which we can regard as a number line, and the location of each house on the road is represented by a single integer.

More formally, let there be N houses, we can represent their locations as $A_0, A_1, \ldots, A_{N-1}$ in non-decreasing order. Note that it is allowed for two houses to be at the same location (Hackerland is very technologically advanced and can wrap space-time).

As part of the planning for the town, a water supply system is needed, and the council has scouted M possible locations to build groundwater pumping facilities, with their locations being $B_0, B_1, \ldots, B_{M-1}$ in strictly increasing order. Note that the potential locations of the facilities are distinct, but the location of a potential facility might coincide with that of a house (again, they can wrap space-time).

Of the M candidate locations, exactly K of them are to be chosen, and pumping facilities will be built at those locations. Afterwards, water pipes will need to be installed to connect the facilities to the houses. Each unit length of water pipe is able to transfer one unit of water for one unit along the number line, while each house requires one unit of water (so you may need parallel pipes for capacity). Assuming each pumping facility can produce infinite water, your task is to find a selection of K locations to minimise the total length of pipes required.

IMPLEMENTATION DETAILS

You should implement the following procedure:

```
pair<int64, int[]> choose_locations(int N, int M, int K, int[] A, int[] B)
```

- N: The number of houses.
- M: The number of candidate locations for the pumping facilities.
- A, B: Arrays of length N and M respectively, representing the locations of houses and candidate locations for pumping facilities, respectively.
- K: Number of locations to be chosen.

The procedure `choose_locations` is called once per test case. The procedure should return a pair containing an integer and an array of integers, denoting the minimum length of pipe required and the chosen locations, respectively. You may output the chosen locations in any order. If there are multiple ways to choose locations with the same minimum total length, return any.

SCORING

For each test case, if you return the correct minimum length and a correct optimal sequence of K locations, you score 100% on the test case.

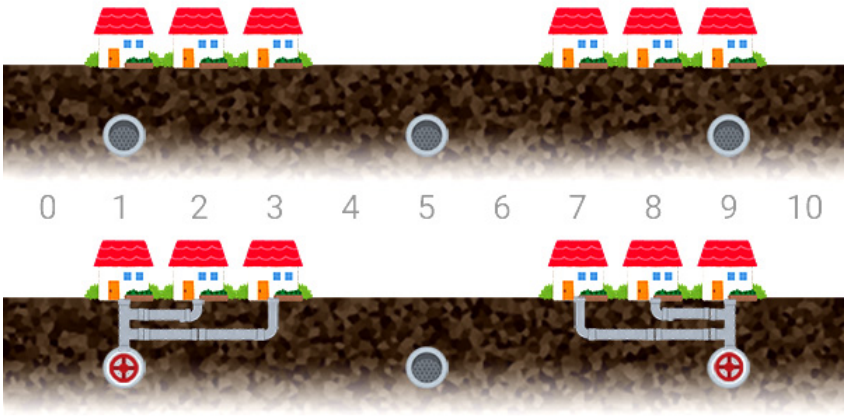
Otherwise, if you return the correct minimum length and *any array*, you score 80% on the test case. Otherwise, you score 0% on the test case.

The points you get on a subtask is the minimum points you score across all test cases in the subtask.

EXAMPLE

Consider the following call:

```
choose_locations(6, 3, 2, [1, 2, 3, 7, 8, 9], [1, 5, 9])
```



Here we can see the optimal way is to choose locations 1 and 9. Then, the pipe lengths required for the houses to connect to the nearest pumping station are $[0, 1, 2, 2, 1, 0]$, so the total pipe length is 6.

Hence, your program should return `(6, [1, 9])`.

SAMPLE GRADER

The sample grader reads the input in the following format:

- The first line consists of three integers N, M and K.
- The second line consists of N integers A_0 to A_{N-1}.
- The third line consists of M integers B_0 to B_{M-1}.

The sample grader outputs two lines, with the first line being the minimum length returned, and the second being the array of locations returned.

SAMPLE TESTS

	Input	Output
1	6 3 2 1 2 3 7 8 9 1 5 9	6 1 9

It is also fine to return `(6, [9, 1])`.

2	6 3 2 1 2 3 7 8 9 1 5 9	6 -1 123456789

This output gets 80% of the points.

3	5 3 1 1 8 11 14 14 8 10 12	20 10
4	6 2 2 1 2 3 4 5 6 0 100	21 0 100

SUBTASKS

For all cases:
$1 \le N \le 50000$
$1 \le M \le 50000$
$1 \le K \le \min(N, M)$
$0 \le A_0 \le \cdots \le A_{N-1} \le 10^9$
$0 \le B_0 < \cdots < B_{M-1} \le 10^9$

	Points	Constraints
1	5	$K = 1$
2	13	$N, M \le 40$
3	18	$N, M \le 100$
4	15	$N, M \le 400$
5	30	$N, M \le 3000$
6	13	$N, M \le 20000$
7	6	No additional constraints

T232 - CHALLENGE OF HANOI

Time Limit: 2.000 s / Memory Limit: 768 MB

Once upon a time, there was a master who was known for his great wisdom. One day, he decided to challenge his disciple, a monk, with a puzzle - the Tower of Hanoi.

The master presented the monk with three rods and a series of disks of varying sizes. He then introduced the following rules:

- The 3 rods are labelled X, Y, and Z.
- There are N disks, and all of the disks are different in size.
- Initially, the disks were arranged in a neat stack on rod X, starting with the largest disk at the bottom and gradually decreasing in size towards the top.
- The goal of the puzzle is to transfer the entire stack of disks to rod Z.
- The monk can only move one disk at a time. A disk can only be moved when it is on the top of a stack, and a larger disk is never placed on top of a smaller one. It could be moved from any rod to any rod as long as it satisfies this constraint.

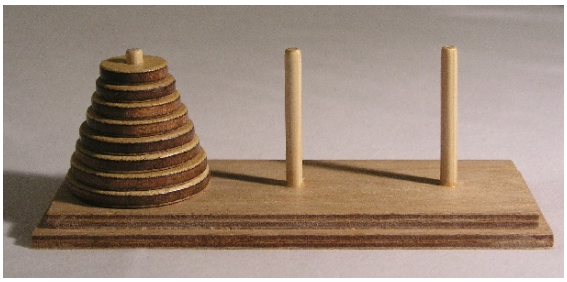


Figure 1: A setup of Tower of Hanoi (from Wikipedia)

The master planned to teach a lesson about how Tower of Hanoi represented the challenges that we face in life, and that they must be tackled one step at a time with patience. However, the monk has a programming background, he quickly devises a plan to solve the problem with the power of recursion.

The monk explained the concept of recursion to the master. He told the master, "the key to solving this puzzle is to break it down into smaller sub-problems", and then proceeded to write the following pseudocode:

```
function Hanoi(N, Start, End, Other)
  if (N == 1):
    Move(Start, End)
  else:
    Hanoi(N - 1, Start, Other, End)
    Move(Start, End)
    Hanoi(N - 1, Other, End, Start)
```

This recusion program will generate a list of $M = 2^N - 1$ commands with the `Move` procedure. For example, when $N = 3$ (i.e. calling `Hanoi(3, X, Z, Y)`), the list of commands are:

```
1: X → Z
2: X → Y
3: Z → Y
4: X → Z
5: Y → X
6: Y → Z
7: X → Z
```

The monk first placed 3 disks on rod X following the rules. Simply applying the commands one by one to move the disks, from command 1 to command 7, the monk successfully transferred all the disks from rod X to rod Z.

"You are a master on recursion", the master laughed, "but are you a master on handling query and update operations?"

First, the master clear all rods and place all N disks on rod X following the rules. After that, the master called `Hanoi(N, X, Z, Y)` to generate the list of $M = 2^N - 1$ commands, and numbered them from 1 to $2^N - 1$ in order. Then, the monk needs to handle Q operations that happen in chronological order. Initially, all M commands are visible.

1. `execute(l)`: Start applying the commands starting from the l-th command, find whether all remaining commands can be successfully executed, or find the position of the first invalid command.
    - A command is invalid when the monk tries to take a disk from an empty rod, or when some larger disks are placed on other smaller disks after the command.
    - The execution process stops after the first invalid command.
    - A valid command executed will actually transfer the disk, affecting future commands and also future call to `execute`.
2. `update(l, r)`: Toggle the visibility of commands $[l..r]$, i.e. For each command i where $l \le i \le r$, change command i to invisible if it was visible, and vice versa. An invisible command will not be applied during `execute`.

"I understand that the problem can be challenging, but do not be discouraged", the master added, "tackle one step at a time with patience, and you can solve every challenge on your way."

You are the monk, please solve the master's challenge.

IMPLEMENTATION DETAILS

You should implement the following procedures:

`void init(int N)`

- N: the number of disks.
- This procedure is called exactly once, before any calls to `execute` and `update`.

`int execute(int l)`

- l: the starting position to apply the commands.
- You should return the position of the first invalid command. If all remaining commands can be successfully executed, return 0 instead.
- Note that the commands executed in this call will affect the initial state of future calls.
- This procedure and `update` are called exactly Q times combined. It is guaranteed that this procedure will be called at least once.

`void update(int l, int r)`

- l: the starting position of commands to be toggled.
- r: the ending position of commands to be toggled.
- Toggle the visibilities of commands $[l..r]$.
- This procedure and `execute` are called exactly Q times combined.

EXAMPLE

Consider the following call:

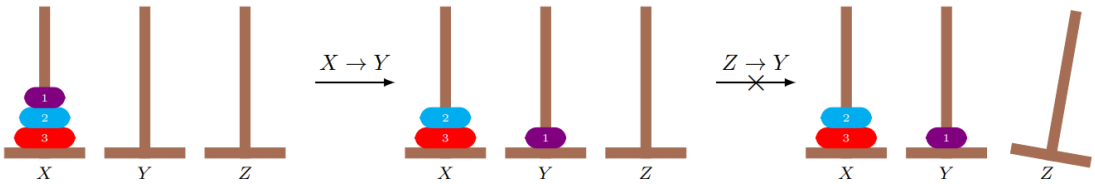
`init(3)`

When $N = 3$, the command sequence is $X \to Z$, $X \to Y$, $Z \to Y$, $X \to Z$, $Y \to X$, $Y \to Z$, $X \to Z$.

First, 3 disks are placed in rod X. For explaining, let's number the smallest disk with 1, the middle-sized disk with 2, and the largest disk with 3. Initially, disk 1 is on the top of the stack, while disk 3 is on the bottom of the stack.

Let's say the grader called `execute(2)`.

- The 2-nd command $X \to Y$ is successful.
- But the 3-rd command $Z \to Y$ is invalid since rod Z is empty.

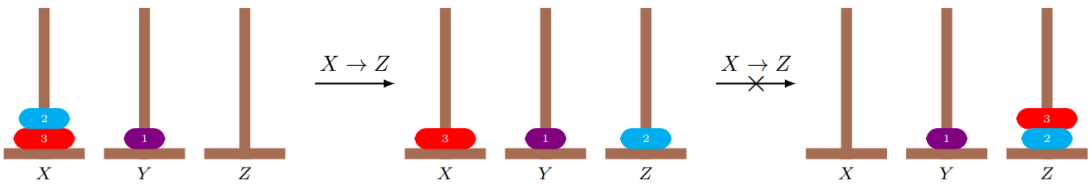


- Therefore, the call should return 3.

Let's say the grader called `update(5, 6)` next. The command sequence becomes $X \to Z$, $X \to Y$, $Z \to Y$, $X \to Z$, n/a, n/a, $X \to Z$.

Let's say the grader called `execute(4)` next.

- The 4-th command $X \to Z$ is successful.
- The 5-th and 6-th commands are not applied.
- But the 7-th command $X \to Z$ is invalid since a bigger disk (disk 3) is placed on a smaller disk (disk 2).

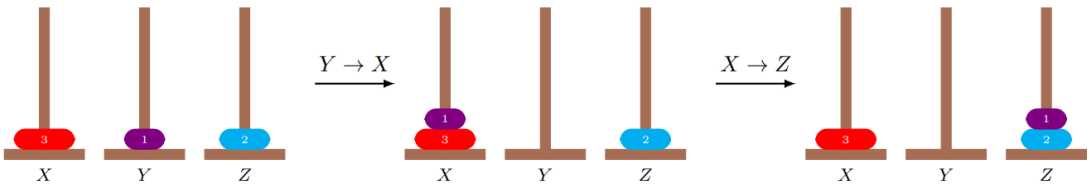


- Therefore, the call should return 7.

Let's say the grader called `update(2, 5)` next. The command sequence becomes $X \to Z$, n/a, n/a, n/a, $Y \to X$, n/a, $X \to Z$.

Let's say the grader called `execute(3)` next.

- The 3-rd and 4-th commands are not applied.
- The 5-th command $Y \to X$ is successful.
- The 6-th command is not applied.
- The 7-th command $X \to Z$ is successful.



- Therefore, the call should return 0.

SAMPLE GRADER

The sample grader reads the input in the following format:

- The first line consists of two integers, N, Q.
- The next Q lines each consist of a description of an operation. On the i-th line, the first integer t_i denotes the type of operation i.
  - If $t_i = 1$, then an integer l_i follows, the grader will call `execute(l_i)`.
  - If $t_i = 2$, then two integers l_i, r_i follow, the grader will call `update(l_i, r_i)`.

The sample grader prints your answer in Q' lines: (if `execute` is called Q' times)

- On the i-th line, the answer for the i-th call to `execute`.

SAMPLE TESTS

	Input	Output
1	3 5 1 2 2 5 6 1 4 2 2 5 1 3	3 7 0

SUBTASKS

For all cases:
$1 \leq N \leq 18$
$1 \leq Q \leq 10^5$
$1 \leq l_i \leq 2^N - 1$ if $t_i = 1$
$1 \leq l_i \leq r_i \leq 2^N - 1$ if $t_i = 2$

	Points	Constraints
1	3	$N \leq 10$ $Q \leq 3000$
2	6	There are at most 10 calls for `execute`
3	40	There are no calls for `update`
4	29	There are no calls for `update` after any call for `execute`
5	22	No additional constraints

HINT

Be aware of the potentially tight memory limit.

T233 - THE WORLD TOUR

Time Limit: 1.000 s / Memory Limit: 768 MB

Loudly Talk is a famous Cantopop boy group in Hackerland. They are organizing their first ever world tour concert in 2023!

There are N countries in the world. Countries are numbered from 0 to $N-1$. There are M one-way direct flights between countries, the i-th flight departs from country U_i and arrives at country V_i. No two flights share the same pair of departing country and arriving country. Members of Loudly Talk can only travel between countries using flights.

The first concert of the world tour must be held at Hackerland (country 0), and the last concert must be held at Byteland (country $N-1$) as these two countries have special meanings to them. During the world tour, they must hold exactly one concert per day, but they cannot hold concerts in the same country for two **consecutive** days.

At the same time, Loudly Talk does not want to spend too much time on transport. Therefore, they demand that they can only take at most one flight between every two concerts. In other words, the country where the next concert is held must be directly connected to the country where the current concert is held by a flight.

Unfortunately, the manager of Loudly Talk has one more requirement. She wants to hold **exactly** C_1 and C_2 concerts in countries P_1 and P_2 respectively.

Please help the manager to determine whether there exists a plan for the world tour such that all requirements above are satisfied. You are not required to find the plan, determining whether it is possible is sufficient.

IMPLEMENTATION DETAILS

You should implement the following procedure:

```
bool tour_possible(int N, int M, int[] U, int[] V, int P1, int C1, int P2, int C2)
```

- N: The number of countries.
- M: The number of one-way direct flights.
- U, V: Arrays of length M, representing the flights.
- P_1, C_1, P_2, C_2: Integers denoting the manager's requirement.

The procedure `tour_possible` is called once per test case. The procedure should return a boolean value, which is `true` if it is possible to plan the world tour, `false` otherwise.

EXAMPLE

Consider the following call:

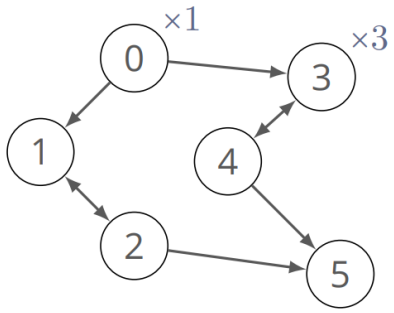
```
tour_possible(2, 2, [0, 1], [1, 0], 0, 3, 1, 5)
```

Observe that the number of concerts held in country 0 and country 1 must be identical. Therefore, it is impossible to hold exactly 3 concerts in country 0 and exactly 5 concerts in country 1.

Hence, your program should return `false`.

Consider the following call:

```
tour_possible(6, 8, [0, 1, 2, 2, 0, 3, 4, 4], [1, 2, 1, 5, 3, 4, 3, 5], 0, 1, 3, 3)
```



One possible tour plan would be $0 \to 3 \to 4 \to 3 \to 4 \to 3 \to 4 \to 5$, where 1 concert is held in country 0 and 3 concerts are held in country 3.

Hence, your program should return `true`.

SAMPLE GRADER

The sample grader reads the input in the following format:

- The first line consists of two integers N and M.
- The next M lines each contains two integers U_i and V_i.
- The next line consists of two integers P_1 and C_1.
- The next line consists of two integers P_2 and C_2.

The sample grader outputs either `Yes` or `No` on a single line, which corresponds to the return values `true` and `false` respectively.

SAMPLE TESTS

	Input	Output
1	2 2 0 1 1 0 0 3 1 5	No
2	6 8 0 1 1 2 2 1 2 5 0 3 3 4 4 3 4 5 0 1 3 3	Yes

SUBTASKS

For all cases:
$2 \leq N \leq 10^5$
$1 \leq M \leq 2 \times 10^5$
$0 \leq U_i, V_i < N$
$U_i \neq V_i$
$(U_i, V_i) \neq (U_j, V_j)$ for $i \neq j$
$0 \leq P_1, P_2 < N$
$P_1 \neq P_2$
$0 \leq C_1 \leq C_2 \leq 10^9$

	Points	Constraints
1	11	$C_1 = C_2 = 0$
2	18	$C_1 = C_2 = 1$
3	23	$C_1 = 0$ $C_2 > 1$
4	32	$P_1 = 0$ $P_2 = N - 1$
5	16	No additional constraints

T234 - COMPLETE THE SEQUENCE

Time Limit: 1.000 s / Memory Limit: 768 MB

"Fill in the blank to complete the sequence: ___, 2, 3, 4." This is a question that every one of us has encountered during our primary school days. To revive your childhood memories, you are also solving this type of problems today. The problem is as follows:

There is an unknown function f, which takes an integer input from 0 to 255 (inclusive) and outputs an integer from 0 to 255 (inclusive). You are given with an integer X ($0 \leq X \leq 255$), denoting that the value of $f(X)$ is unknown. You are free to call the function f with any inputs from 0 to 255, except X. Your job is to deduce the value of $f(X)$. There are two types of tasks (corresponding to two different types of functions f) you have to solve.

- In the first task type ($S = 1$), it is guaranteed that $f(x) = (Ax) \bmod B$ for some integral constants $0 \leq A < B \leq 256$.
- In the second task type ($S = 2$), it is guaranteed that $f(x) = (A^x) \bmod B$ for some integral constants $1 \leq A < B \leq 251$, **where B is a prime number**.

However, there is a twist - you are solving this problem on a new type of computer processor!

The processor has access to $M = 10000$ different 16-bit memory cells, which are called **registers**, and are numbered from 0 to $M - 1$. We denote the registers by $r[0], r[1], \ldots, r[M - 1]$. Each register stores an integer value from 0 to $2^{16} - 1 = 65535$.

Initially, the register $r[0]$ stores the integer X, while all other registers are initialized to zero. The processor supports 10 types of **basic instructions** to modify the values in the registers. Each instruction operates on one or more registers and stores the output in one of the registers. The operations performed by each type of instruction are described below.

- $\text{move}(t, y)$: Copy the value in register y to register t.
- $\text{store}(t, v)$: Set register t to be equal to v, where v is an integer.
- $\text{and}(t, x, y)$: Take the bitwise-AND of registers x and y, and store the result in register t.
- $\text{or}(t, x, y)$: Take the bitwise-OR of registers x and y, and store the result in register t.
- $\text{xor}(t, x, y)$: Take the bitwise-XOR of registers x and y, and store the result in register t.
- $\text{not}(t, x)$: Take the bitwise-NOT of register x, and store the result in register t.
- $\text{left}(t, x, p)$: Shift all bits in register x to the left by p, and store the result in register t, the bit positions that have been vacated by the shift operation are zero-filled.
- $\text{right}(t, x, p)$: Shift all bits in register x to the right by p, and store the result in register t, the bit positions that have been vacated by the shift operation are zero-filled.
- $\text{add}(t, x, y)$: Add the integer values stored in register x and register y, and store the result in register t. The addition is carried out modulo 2^{16}.
- $\text{f}(t, x)$: Execute the unknown function f on the value stored in register x, and store the result (that is, $f(r[x])$) in register t. This operation is invalid if $r[x] \geq 256$ or $r[x] = X$, **and the program will be judged as incorrect if such invalid operation occurs**.

Additionally, the processor supports 3 types of **expensive instructions**. The use of each type of expensive instructions incurs a great cost.

- $\text{mult}(t, x, y)$: Multiply the integer values stored in register x and register y, and store the result in register t. The multiplication is carried out modulo 2^{16}.
- $\text{mod}(t, x, y)$: Divide the integer value stored in register x by that in register y, and store the remainder in register t. This operation is invalid if $r[y] = 0$, **and the program will be judged as incorrect if such invalid operation occurs**.
- $\text{cmp}(t, x, y)$: If the integer value stored in register x is less than or equal to that in register y, store the value 65535 in register t. Otherwise, store the value 0 in register t.

For both types of tasks, you need to produce a **program**, that is a sequence of instructions defined above. After executing the program, the register $r[0]$ should contain the value $f(X)$, while the contents in all other registers can be arbitrary. **It is guaranteed that for all input data, there exists a unique solution $f(X)$ that is consistent with all other outputs returned by f.**

Provide programs consisting of at most 10^5 instructions each, that can solve these tasks.

IMPLEMENTATION DETAILS

You should implement the following procedure:

```
void construct_instructions(int S)
```

- S: the type of task.
- This procedure is called exactly once and should construct a sequence of instructions to perform the required task.

This procedure should call one or more of the following procedures to construct a sequence of instructions:

```
void append_move(int t, int y)
void append_store(int t, int v)
void append_and(int t, int x, int y)
void append_or(int t, int x, int y)
void append_xor(int t, int x, int y)
void append_not(int t, int x)
void append_left(int t, int x, int p)
void append_right(int t, int x, int p)
void append_add(int t, int x, int y)
void append_f(int t, int x)
void append_mult(int t, int x, int y)
void append_mod(int t, int x, int y)
void append_cmp(int t, int x, int y)
```

- Each procedure appends a $\text{move}(t, y)$, $\text{store}(t, v)$, $\text{and}(t, x, y)$, $\text{or}(t, x, y)$, $\text{xor}(t, x, y)$, $\text{not}(t, x)$, $\text{left}(t, x, p)$, $\text{right}(t, x, p)$, $\text{add}(t, x, y)$, $\text{f}(t, x)$, $\text{mult}(t, x, y)$, $\text{mod}(t, x, y)$ or $\text{cmp}(t, x, y)$ instruction to the program, respectively.
- For all relevant instructions, t, x, y must be at least 0 and at most $M - 1$.
- For all relevant instructions, t, x, y are not necessarily pairwise distinct.
- For left and right instructions, p must be at least 0 and at most 16.
- For store instructions, v must be at least 0 and at most $2^{16} - 1 = 65535$.

You may also call the following procedures to help you in testing your solution:

```
void append_print(int t)
void append_message(string msg)
```

- Any call to these procedures will be ignored during the grading of your solution.
- In the sample grader, this procedure appends a $\text{print}(t)$ or $\text{message}(msg)$ instruction to the program, respectively.
- When the sample grader encounters a $\text{print}(t)$ instruction during the execution of a program, it prints the contents of the register t to standard error stream. t must satisfy $0 \leq t \leq M - 1$.
- When the sample grader encounters a $\text{message}(msg)$ instruction during the execution of a program, it prints the contents of the string msg to standard error stream.
- Any call to these procedures does not add to the number of constructed instructions.
- **Note that the "Code" function in this online judge does NOT display the outputs in the standard error stream.**

After appending the last instruction, `construct_instructions` should return. The program is then evaluated on some number of test cases, each specifying an input consisting of 3 integers A, B and X. X is the initial value of $r[0]$ and A, B are parameters describing the sequence.

SCORING

If your program is incorrect, the score of your solution will be 0 in the subtask. Otherwise, let K be the number of instructions in your program. Then, your score (in percent) for the subtask will be calculated according to the respective table for $S = 1$ or $S = 2$:

$S = 1$:

Condition	Score
$2000 < K \leq 10^5$	$(158.291 - 7.669 \ln K)\%$
$K \leq 2000$	100%

$S = 2$:

Condition	Score
$1500 < K \leq 10^5$	$(152.239 - 7.143 \ln K)\%$
$K \leq 1500$	100%

In particular, you will get about 70% of the score if $K = 10^5$.

Let E be the number of types of expensive instructions used. If $E > 0$, some percentage of the score in the subtask will be further deducted according to the following table:

Condition	Score Adjustment
$E = 1$	-25%
$E = 2$	-35%
$E = 3$	-45%

Note that the deduction here is based on the number of types of expensive instructions used, not on the number of times they are used. For example, if `append_mult` is called multiple times, only 25% of the points will be deducted; but if `append_mult` and `append_mod` are both called, 35% of the points will be deducted, even if each function is called only once.

EXAMPLE

Suppose it is guaranteed that $S = 1$, $0 \leq A \leq 1$, $B = 256$ and $100 \leq X \leq 101$. Then, a possible solution is to construct a program by making the following calls:

1. `append_store(1, 255)`, which appends an instruction to store the value 255 in $r[1]$.
2. `append_f(2, 1)`, which appends an instruction to execute the function f on $r[1]$ and store the value $f(r[1])$ in $r[2]$. Therefore, $r[2]$ stores $f(255)$.
3. `append_and(0, 0, 2)`, which appends an instruction to take the bitwise-AND of $r[0]$ and $r[2]$, then store the result in $r[0]$.

If $A = 0$, $f(255)$ returns 0 and 0 will be stored in register $r[0]$ after the third instruction. If $A = 1$, $f(255)$ returns 255 and X will be stored in register $r[0]$ after the third instruction. Therefore, this program is correct.

SAMPLE GRADER

The sample grader reads input from standard input in the following format:

- The first line consists of a single integer S, the type of the task.
- The second line consists of a single integer Q, the number of test cases.
- The next Q lines each contains three integers A, B and X, representing a test case.

If the sequence of instructions is incorrect, or it gives an incorrect output for any of the test cases, the sample grader outputs a line `Wrong Answer: Reason`. Otherwise, it outputs a line `Accepted: %d instructions used`.

The outputs to the $\text{print}(t)$ and $\text{message}(msg)$ instructions will be outputted as well.

If an invalid input is supplied to the sample grader, the behaviour of the grader is undefined.

SAMPLE TESTS

	Input	Output
1	1 2 5 256 8 123 234 123	Test 1: OK 40 Test 2: OK 153 Accepted: 100000 instructions used
2	2 2 5 113 8 123 251 123	Test 1: OK 97 Test 2: OK 211 Accepted: 100000 instructions used

SUBTASKS

For all cases:
$1 \leq S \leq 2$
$M = 10000$
For $S = 1$, $0 \leq A < B \leq 256$
For $S = 2$, $1 \leq A < B \leq 251$ and B is a prime number
There exists a unique solution $f(X)$ that is consistent with all other outputs returned by f

	Points	Constraints
1	14	$S = 1$ $B = 256$
2	36	$S = 1$
3	27	$S = 2$ $B < 128$
4	23	$S = 2$