The 7th Annual University of Akron Programming Competition

presented by

The University of Akron Computer Science Department

Association for Computing Machinery Student Chapter

March 11, 2017

Rules:

1. There are seven questions to be completed in four hours.
2. C, C++, and Java are the only languages available.
3. Data is read from Standard Input and output is sent to Standard Output. Do not prompt for input values in the code you submit to be judged. Do not attempt to read from or write to any files.
4. All programs are submitted as source code only. Submitting compiled information (binary, .class) will result in a Compilation Error penalty.
5. Non-standard libraries cannot be used in your solutions. The Standard Template Library (STL) and C++ string libraries are allowed. The standard Java API is available, except for those packages that are deemed dangerous by contest officials (e.g., that might generate a security violation).
6. The input to all problems will consist of multiple test cases. The Sample Input listed on the problem description is not intended to be a comprehensive input set. The input is guaranteed to adhere to the descriptions in each problem; you do not need to check for invalid input.
7. The output to all problems should conform to the format shown in the Sample Output for that problem, including but not limited to capitalization, whitespace, etc.
8. Use of personal electronics during the competition is cause for disqualification. Cell phones must be turned all the way off (not silent mode, not airplane mode, etc.). You may use any written notes, books, or reference materials you bring with you.
9. Programming style is not considered in this contest. You can code in any style or with any level of documentation your team prefers.
10. All communication with the judges will be handled through PC^2. All judges decisions are final.
11. Source code is compiled on the command line, C and C++ will be compiled with GNU G++ (`g++ sourcefile.cpp`) and Java with the standard Java RE (`javac sourcefile.java`).

A: Important Slime Research

Tanya is working in a lab researching slime specimens. She found that every second every slime will split in two and that they can do so indefinitely.

This means that if at the first second there were 4 slimes, in one additional second there would be 8, and after two more seconds there would be 32.

Help Tanya find the number of slimes there will be after a specified number of seconds have passed, assuming that she always starts with 1 slime.

Input

Input consists of one integer, n ($1 \leq n \leq 30$), the number of seconds Tanya waits before recording the new number of slimes.

Output

Output the number of slimes present after n seconds.

Sample Input	Sample Output
1	2

Sample Input	Sample Output
2	4

Sample Input	Sample Output
12	4096

B: The Polish Calculator

Polly loves polish notation. She loves it more than any other kind of notation. Integral notation, differential notation, standard notation... She loves polish notation. Polish notation is read from right to left. A number is pushed on to a virtual stack. Operations pop two numbers from the virtual stack (or the calculator will error if two numbers are not on the stack) and evaluates the operation. The first number on the virtual stack is the left operand and the second number is the right operand. An expression in polish notation is considered "well-founded" if the virtual stack has only one number after all operations are processed.

Polly wants you to implement a calculator for arithmetic expressions in polish notation. To make things a little easier for you the numbers are only from 1 to 9. Furthermore, you should take the floor of all division operations.

Input

The first line of input is one integer n ($1 \leq n \leq 25$), the number of symbols. The next line consists of n space-separated symbols. The symbols are either an operator ($+$, $-$, $\times$, or $/$) or a number t ($1 \leq t \leq 9$). All expressions are well-founded.

Output

Output one number, the result of the expression.

Sample Input	Sample Output
5 + 1 - 1 1	1

Sample Input	Sample Output
7 x 2 + 1 / 2 2	4

C: Favorite Words

Vanessa is trying to find her favorite collection of "words." A "word" for Vanessa is any sequence of four letters, where a letter is a lowercase letter from the English alphabet. The letters "a", "b", ..., "z" are the valid letters. The words "abdd" and "zzcd" are examples of valid words.

On some days Vanessa really dislikes certain letters. Sometimes she wants a lot of words, sometimes not a lot. Her dream app is to have a "word of the day" program that adheres to her specific interests. She's asked you to write a program to find her the collection of words she's most into on any particular day.

Specifically, she wants you to give her a contiguous list of words that do not have any letters she dislikes. To keep things interesting, she also wants the list of words to start after skipping some number of initial words so she doesn't get the same list every time for the same disliked letters.

Input

The first line consists of three integers, n ($1 \leq n \leq 25$), the number of characters Vanessa dislikes; m ($1 \leq m \leq 3 \times 10^5$), the mth word you should start recording; and k ($1 \leq k \leq 100$), the number of words in the collection. The next line consists of n alpha letters, the letters that Vanessa dislikes.

Output

Output k "words", one word on each line, the "words of the day."

Sample Input	Sample Output

`4 2 3`
`a b c e`
	`dddf`
	`dddg`
	`dddh`

Sample Input	Sample Output

`1 1000 10`
`z`
	`aboy`
	`abpa`
	`abpb`
	`abpc`
	`abpd`
	`abpe`
	`abpf`
	`abpg`
	`abph`
	`abpi`

D: Photons for Sale

Edward and his colleagues have been trying to transfer messages along photoluminescent wires. It's a brand new technology and just recently they've been able to encode bits with photons. A chip can hold a certain bit representation but there is a cost of moving the photons from one chip along the wire to a different chip. For every bit in the representation that changes between the initial chip and the target chip it costs one unit of energy. The cost of moving from a chip with the bit representation 101 to another with 100 is a single unit of energy.

Moving photons along the wire is particularly expensive but Edward only cares about getting to a certain bit representation from a starting chip. Your job is to figure out the minimum cost it would take to get to all other possible chips from a given chip. Not all chips are necessarily directly connected by wires and some chips might not be reachable at all.

Input

The first line of input consists of two integers, n ($1 \leq n \leq 100$), the number of chips, and m ($1 \leq m < \frac{n(n-1)}{2}$), the number of wires connecting chips. The next n lines consist of a bit representation, a sequence of 0s and 1s, that won't be longer than 100 characters. The next m lines consist of two integers, the wire connection between two chips. Note that the two integers are zero-based, so the first chip is represented by the number 0. The final line of input is the initial chip.

Output

Output n lines, the chips bit representation in order that they were given in the input, and the cost to reach that chip from the initial chip. If a chip can not be reached from the initial chip then output -1 as the cost.

Sample Input

```
5 8
1000
1010
1111
0000
1011
0 1
0 2
1 2
1 3
1 4
2 3
2 4
3 4
0
```

Sample Output

```
1000 0
1010 1
1111 3
0000 3
1011 2
```

Sample Input

```
5 5
0000000000
0000000001
0000000010
0000000100
1111111111
0 1
0 3
1 2
2 3
3 4
0
```

Sample Output

```
0000000000 0
0000000001 1
0000000010 3
0000000100 1
1111111111 10
```

E: Snek Game

Jastion likes playing the raddest games, like slither.io and agar.io. He's a bit of a hipster though, so he likes doing things before they are cool. He realizes that if he makes his own spin-off of one of these games that he'll definitely be playing it before it is cool.

Jastion decides to make a game called snek.io, where a bunch of sneks roam around and eat each other to get bigger. He made some pretty bad decisions with the game and it's not very fun. First, a snek can only eat the snek closest to its size and larger than it. Second, sneks grow each time they eat another snek, but the amount they grow is constant so it does not matter how big the sneks they eat are.

All of the other players' sneks start out at various sizes, but Jastion's snek always starts at size 0 (it's very small!). This means that the first snek that Jastion eats must be the next smallest one. Once that snek is eaten and Jastion's snek grows, he must then eat the next snek that is closest to its size but larger than him. He will continue to do this until he is the largest snek.

Because of Jastion's odd choices with this game, none of the other players can figure out how to eat the other sneks. Jastion is the only one who understands the rules properly, therefore his snek is the only one that will eat other sneks.

Jastion wants your help. Given a list of the other sneks' sizes, and the amount Jastion's snek will grow each time he eats a snek, he wants you to tell him which sneks he should eat and the order he should eat them.

Input

The first line contains two integers. The first, n ($1 \leq n \leq 10^5$), is the amount of sneks other than Jastion's in the game. The second, m ($1 \leq m \leq 10^5$), is the amount that Jastion's snek will grow after each snek he eats.

The second line contains integers s_i ($1 \leq s_i \leq 10^9$) from s_1 to s_n, where s_i corresponds to the size of the snek in the ith position.

Output

On a single line, output the positions of the sneks that Jastion's snek may eat in the order he may eat them.

Sample Input	**Sample Output**
10 1 1 2 3 4 5 6 7 8 9 10	1 2 3 4 5 6 7 8 9 10

Sample Input	**Sample Output**
10 2 1 2 3 4 5 6 7 8 9 10	1 3 5 7 9

F: Parents Loving Sequence

Johnny loves playing games with lists of numbers. One of his favorite games is to sum the numbers up in a list. Johnny is a bit of a prodigy. Gauss himself would be proud. His mother, Abigail, decided that Johnny needed to push his talents farther so that he would grow intellectually.

First Abigail decided that Johnny should start summing two lists at a time instead of just one. This seemed at first like a decent enough difficulty but Johnny quickly overcame it. Abigail decided that it was too time consuming to write out a whole list each time Johnny wanted a new problem. So, she modified the problem slightly to give Johnny two formulas for generating the lists instead of the lists themselves. The first formula would consist of n coefficients and n starting numbers. The second formula would consist of m coefficients and m starting numbers. The next number in the first list, the $n+1$st number, would be the linear combination of the coefficients and the preceding numbers. For example, given starting numbers a_i and coefficients c_i:

$a_0 = 1$, $a_1 = 2$, $a_3 = 3$, $c_0 = 0$, $c_1 = 1$, $c_2 = 2$

then, $c_3 = c_2a_2 + c_1a_1 + c_0a_0 = 2 \times 1 + 1 \times 2 + 0 \times 3 = 4$. c_4 would then be calculated with c_3, c_2, and c_1 (and the corresponding a_is). Moreover, Abigail would give Johnny a number of terms t_n for the first list and t_m for the second list that Johnny should compute. Thus, after computing the two lists up to t_n and t_m elements respectively, he would then sum the lists.

Much to Abigail's chagrin and amazement Johnny quickly conquered this task as well. It seemed that she would need to increase the difficulty with something less to do with numbers and more to do with manipulating the lists. Abigail's next trick was to have Johnny sum up the longest shared list between the two lists instead of the lists individually.

She explained to Johnny what she meant by shared in the following way: Johnny was allowed to remove elements from a list, but he had to find the fewest number of deletions such that the two lists were equal. Johnny then had to sum just one of the lists, since they would have the same length and the same numbers.

Abigail was proud of herself and her son. He didn't take to this problem as easily as the first, but all the same he managed it after a few months. Resolute, Abigail decided one final modification, but she's requested your help in making sure that she gets the right answers in order to check Johnny's work.

Abigail wants to ask Johnny to sum a range of the list instead of the entire list. Johnny should be able to answer q queries by range of the list, not just a few. You're job is to write a program that will give the answer to Abigail for each query so that she can double check Johnny's work.

Input

The input begins with two integers n ($1 \leq n \leq 100$), the number of coefficients, and t_n ($1 \leq t_n \leq 1000$), the total number of terms in the first list. The next line is $2n$ integers, the first n are the coefficients, c_i ($-10 \leq c_i \leq 10$), in order from first to last. The second n integers are the starting values for the list, a_i ($-10 \leq a_i \leq 10$), in order from first to last. The next line is another two integers m ($1 \leq m \leq 100$), the number of coefficients, and t_m ($1 \leq t_m \leq 1000$), the total number of terms in the second list. The next line is $2m$ integers, shaped similarly to the previous list of coefficients and starting values. The next line is one integer, q ($1 \leq q \leq 10000$), the number of queries. The following q lines are two integers, l_i ($l_i > 0$) and r_i ($r_i > 0$), which form a range $[l_i, r_i]$ (specifically, $l_i \leq r_i$) that Johnny should sum over. If l_i or r_i are larger than the length of the shared list then they should be rounded down to match only the largest valid range for the shared list. If the shared list is empty then the query range should be ignored and the result should be 0 for each query.

Output

For each query output one integer that is the sum from l_i to r_i of the shared list.

Sample Input	**Sample Output**
2 5 1 1 0 1 3 6 1 1 1 0 1 0 3 2 3 1 10 2 4	2 7 4

Sample Input	**Sample Output**
3 10 0 0 1 1 0 1 1 10 2 1 2 1 1 1 10	1 1

G: The Parenren Language

Searle has come up with a new *interesting* programming language. It consists of only parenthesis characters, (and). The left most parenthesis, (, will put the number 1 on a virtual stack. Having two left parenthesis characters together, like ((, will put the number 1 on a virtual stack and perform a `nand` operation. Having three left parenthesis characters together, like (((, will put the number 1 on the virtual stack and perform a `nor` operation. The program evaluator is greedy, so the `nor` operation consisting of three left parenthesis will be performed first always over the other two if possible.

A `nand` operation will pop off two numbers from the stack, "and" them, and then "not" the result. So for example, `1 nand 1` will give 0, and `0 nand 1` will give 1. A `nor` operation will pop off two numbers from the stack, "or" them, and then "not" the result.

After each `nand` or `nor` operation the result is put on the virtual stack.

A right most parenthesis will pop a number from the top of the virtual stack and print it to the screen. One caveat is that every left parenthesis must be paired with a right parenthesis to make parsing the language simple. If there is nothing on the virtual stack to be printed when a right parenthesis is evaluated then it does nothing. Here are some example programs:

((()()))

()()()

((()((()))()))

Searle is confident that this new *interesting* programming language is Turing complete but he is curious about how expressive it is. Given a number n, the number of paired parenthesis, Searle wants you to determine how many possible programs there are.

Input

The input consists of one integer n ($1 \leq n \leq 2000$).

Output

Output one integer, the number of possible programs, mod $10^7 + 14$.

Sample Input	**Sample Output**
`4`	`14`

Sample Input	**Sample Output**
`1`	`1`