The 2nd Annual
생각하는 프로그래밍 대회
Thinking Programming
Contest

CNU 컴퓨터공학과 A&A STARTLINK
국방ICT융합인력양성사업단 핀테크보안연구센터

Problem Set

2018-11-17

Problem A
괄호 없는 사칙연산

Time Limit: 1 Second

사칙연산에서 곱셈과 나눗셈은 덧셈과 뺄셈보다 먼저 계산한다. 덧셈과 뺄셈을 먼저 계산하고 싶을 때는 보통 가장 큰 연산 우선순위를 가지는 괄호를 사용하여 연산 순서를 지정한다. 예를 들어, 아래의 식은 연산 순서에 따라 두 가지 다른 결과가 나올 수 있다.

- (2 + 3) × 4 = 20
- 2 + (3 × 4) = 14

연산 우선순위가 같은 곱셈과 나눗셈 또는 덧셈과 뺄셈만 있는 식에서는 보통 왼쪽에서 오른쪽 순서로 연산을 한다. 하지만 이런 상황에도 연산 순서에 따라 아래와 같이 두 가지 다른 결과가 나올 수 있다.

- (6 ÷ 2) × 3 = 9
- 6 ÷ (2 × 3) = 1

만약 곱셈, 나눗셈, 덧셈, 뺄셈의 4가지 연산자의 연산 우선순위가 동등하다고 할 때, 괄호 없는 식에서 서로 다른 연산 순서의 계산 결과를 구하여라.

Input

첫 번째 줄에는 "K_1 O_1 K_2 O_2 K_3" 형태로 식이 주어진다. 정수 K_i ($1 \le K_i \le 1{,}000$)는 피연산자를, 문자열 O_i는 곱셈(*), 나눗셈(/), 덧셈(+), 뺄셈(-) 중 한 가지 연산자를 의미한다.

나눗셈 연산은 정수 나눗셈으로 몫만 취하며, 피연산자 중 하나가 음수이면 양수로 바꿔 계산한 결괏값에 음수를 취한다. 또한, 계산 과정에서 0으로 나누어야 하는 식은 주어지지 않는다.

Output

주어진 식에서 서로 다른 연산 순서의 계산 결과가 작은 것을 첫 번째 줄에, 큰 것을 두 번째 줄에 출력한다.

Sample Input 1	Output for the Sample Input 1
2 + 3 * 4	14 20

Sample Input 2	Output for the Sample Input 2
6 / 2 * 3	1 9

Sample Input 3	Output for the Sample Input 3
5 + 10 + 10	25 25

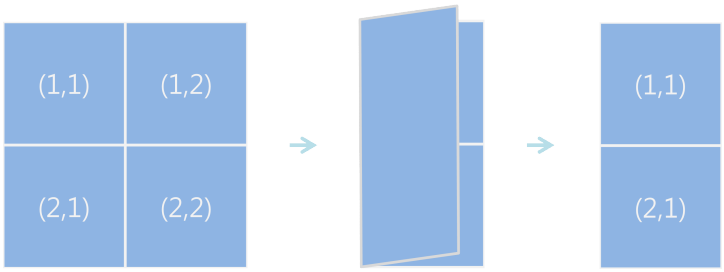
Sample Input 4	Output for the Sample Input 4
7 / 3 – 9	-7 -1

Problem B
종이접기

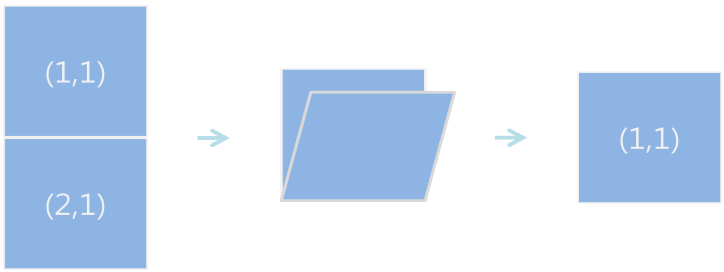
Time Limit: 1 Second

종이접기와 수학을 좋아하는 주성이는 종이접기와 수학을 한꺼번에 할 수 있는 놀이를 찾아냈다. 바로 *N*×*N* 크기를 가지는 색종이의 각 칸에 수를 적어놓고, 색종이를 반으로 접을 때마다 겹치는 부분의 수들을 더하는 것이다. 그리고 이 작업을 색종이를 더는 접을 수 없을 때까지 반복했을 때, 가장 마지막에 남는 수를 구하는 놀이이다.

아래의 예시는 위에서 설명한 놀이의 과정을 2×2 크기의 색종이를 사용하여 상세하게 나타낸 그림이다.



- 색종이를 **왼쪽에**서 **오른쪽**으로 정확히 반을 접는다.
- 겹치는 두 칸에 있는 수를 서로 더한다. 위 그림에서는 (1, 1)과 (1, 2)의 두 칸이 겹치고, (2, 1)과 (2, 2)의 두 칸도 겹치는 부분이다.
- 더한 값을 색종이 위에 다시 적는다.



- 색종이를 **아래쪽**에서 **위쪽**으로 정확히 반을 접는다.
- 겹치는 두 칸에 있는 수를 서로 더한다. 위 그림에서는 (1, 1)과 (2, 1)의 두 칸이 겹치는 부분이다.
- 더한 값을 색종이 위에 다시 적는다.

색종이를 반으로 접으면서 두꺼워지는 것을 고려하지 않고, 색종이를 더는 접을 수 없을 때까지 위 과정을 반복했을 때 가장 마지막에 남는 수를 구하여라.

Input

첫 번째 줄에는 색종이의 가로, 세로 길이를 의미하는 정수 N ($N = 2^m$, $1 \le m \le 10$)이 주어진다.

두 번째 줄부터 마지막 줄까지는 $N \times N$ 크기의 색종이 정보가 주어지며, 색종이의 각 칸에는 정수 K ($1 \le K \le 100,000$)가 주어진다.

Output

색종이를 반으로 접으면서 겹치는 부분의 수들을 더하는 과정을 반복했을 때, 가장 마지막에 남는 수를 출력한다.

Sample Input 1	**Output for the Sample Input 1**
4 2 6 5 4 1 5 7 6 9 8 8 7 1 4 7 8	88

Problem C
별

Time Limit: 1 Second

출력 예제를 보고 별 찍는 규칙을 유추하여 별을 찍어 보자.

Input

첫 번째 줄에는 정수 N ($0 \le N \le 10$)이 주어진다.

Output

별 찍는 규칙에 따라 별을 출력한다.

각 줄의 끝에는 필요 없는 공백을 출력하지 않는다.

Sample Input 1	Output for the Sample Input 1
0	*

Sample Input 2	Output for the Sample Input 2
1	** *

Sample Input 3	Output for the Sample Input 3
2	**** * * ** *

Sample Input 4	Output for the Sample Input 4
3	******** * * * * ** ** * * **** * * ** *

Sample Input 5	Output for the Sample Input 5
4	********** * * * * * * * * ** ** ** ** * * * * **** **** * * * * ** ** * * ******** * * * * ** ** * * **** * * ** *

Problem D
CPU

Time Limit: 1 Second

디지털하드웨어설계 과목의 최종 프로젝트는 16-bit CPU를 설계하고 Verilog 언어로 구현하는 것이다. 본인이 구현한 CPU가 제대로 동작하는지 테스트하기 위해서는 기계어 코드를 입력으로 주어야 한다. 하지만 대부분의 사람은 0과 1로만 이루어진 기계어 코드를 이해하기 힘들어서 C++, Java와 같은 프로그래밍 언어로 코드를 작성하고 컴파일러를 통해 기계어 코드로 번역하는 과정을 거친다.

여러 가지 프로그래밍 언어 중에서 어셈블리어는 사람이 이해하기 쉬우면서 기계어와 가장 유사한 언어이다. 어셈블리어 코드는 어셈블러를 통해 기계어 코드로 번역된다. 그리고 어셈블리어는 기계어와 일대일로 대응하는 특징이 있다. 예를 들면, 두 수의 합을 구하는 연산의 어셈블리어 코드가 ADD이고, 기계어 코드가 00000이면 어셈블러는 ADD를 읽어서 그대로 00000로 바꾸어주는 것이다.

입력과 출력은 항상 명령어 단위이며, 어셈블리어 코드는 *"opcode rD rA rB"* 또는 *"opcode rD rA #C"*의 형태이다. 기본적으로 레지스터 *rA*와 *rB*에 있는 두 수 또는 레지스터 *rA*에 있는 수와 상수 *#C*를 *opcode*에 해당하는 연산을 수행하고, 그 결괏값을 레지스터 *rD*에 저장하는 명령어이다. *rA*는 *opcode*에 따라 사용하지 않을 수도 있다. 어셈블러는 *opcode, rD, rA, rB, #C*를 각 bit의 자리에 맞게 2진수 0과 1로 이루어진 16-bit 기계어 코드로 변역한다. bit마다 자리의 의미는 아래와 같다.

- 0~4 : CPU가 수행해야 할 연산을 나타내는 *opcode*이다. 만약 4번 bit가 0일 경우 레지스터 *rB*를, 1일 경우 상수 *#C*를 사용한다.
- 5 : 사용하지 않는 bit이며, 항상 0이다.
- 6~8 : 결괏값을 저장하는 레지스터 *rD*의 번호이다.
- 9~11 : 연산에 사용되는 레지스터 *rA*의 번호이다. 사용하지 않을 경우 000이다.
- 12~15 : 만약 4번 bit가 0일 경우 12~14번 bit는 연산에 사용되는 레지스터 *rB*의 번호이며, 15번 bit는 항상 0이다. 만약 4번 bit가 1일 경우 12~15번 bit는 상수 *#C*이다.

디지털하드웨어설계 과목을 듣는 민호는 Verilog로 16-bit CPU 구현을 일찍 끝내 놓은 상태이다. 이 16-bit CPU를 테스트하기 위해서는 기계어를 매번 입력으로 줘야 하는데, 너무나 귀찮은 민호는 이에 맞는 어셈블러를 구현하려고 한다. 민호가 직접 설계한 16-bit CPU의 명령어 구조 표를 보고, 어셈블리어 코드가 주어졌을 때 이를 기계어 코드로 번역하는 어셈블러를 만들어보자.

아래의 그림은 민호가 설계한 CPU가 처리할 수 있는 16-bit 단위 명령어들의 구조를 모아놓은 표이다.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	opcode						rD			rA			rB			
ADD	0000				0	0	rD			rA			rB			0
ADDC					1								#C			
SUB	0001				0					rA			rB			0
SUBC					1								#C			
MOV	0010				0					000			rB			0
MOVC					1								#C			
AND	0011				0					rA			rB			0
ANDC					1								#C			
OR	0100				0					rA			rB			0
ORC					1								#C			
NOT	0101				0					000			rB			0
MULT	0110				0	0				rA			rB			0
MULTC					1								#C			
LSFTL	0111				0					rA			rB			0
LSFTLC					1								#C			
LSFTR	1000				0					rA			rB			0
LSFTRC					1								#C			
ASFTR	1001				0					rA			rB			0
ASFTRC					1								#C			
RL	1010				0					rA			rB			0
RLC					1								#C			
RR	1011				0					rA			rB			0
RRC					1								#C			

Input

첫 번째 줄에는 명령어의 개수를 의미하는 정수 *N* (1 ≤ *N* ≤ 500)이 주어진다.

다음 *N*개의 각 줄에는 명령어가 어셈블리어 코드로 *"opcode rD rA rB"* 또는 *"opcode rD rA #C"*의 형태로 주어진다. 문자열 *opcode*는 항상 대문자이다. 정수 *rD, rA, rB* (0 ≤ *rD, rA, rB* ≤ 7)는 레지스터 번호를 의미한다. 사용하는 레지스터 번호는 1부터 7까지이며, 사용하지 않을 경우에만 0이 주어진다. 정수 *#C* (0 ≤ *#C* ≤ 15)는 상수를 의미한다.

기계어 코드로 번역될 때 어긋나는 입력은 주어지지 않는다.

Output

*N*개의 각 줄에 어셈블리어 코드를 기계어 코드로 번역하여 출력한다.

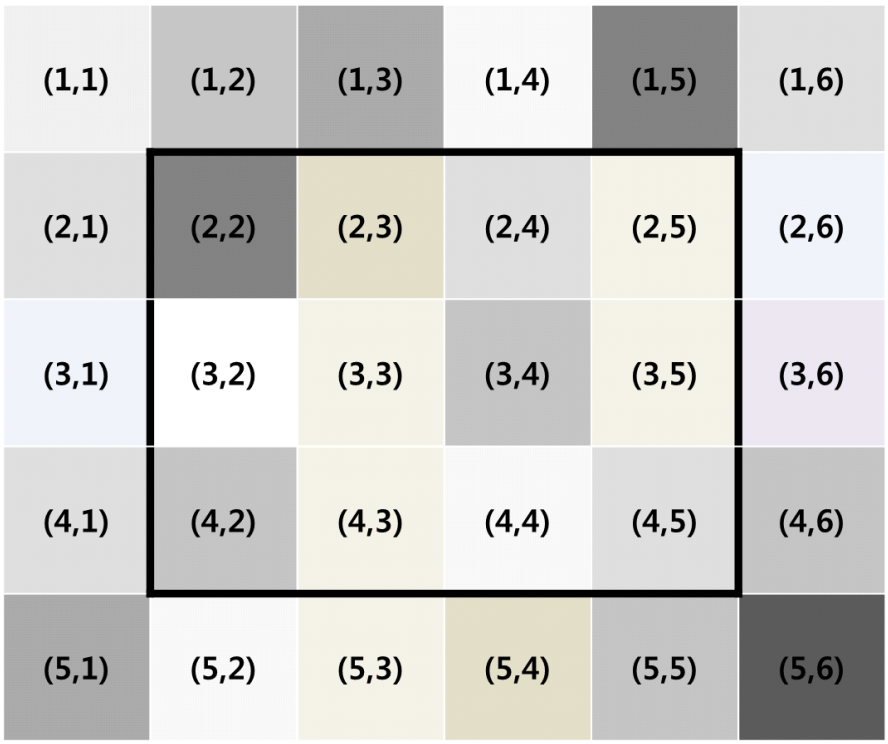
Sample Input 1	**Output for the Sample Input 1**
4 MOVC 1 0 5 MOVC 2 0 10 ADD 3 1 2 SUB 4 1 2	0010100010000101 0010100100001010 0000000110010100 0001001000010100

Sample Input 2	**Output for the Sample Input 2**
8 LSFTL 4 2 4 MULTC 3 7 12 NOT 2 0 4 SUB 4 4 3 ASFTR 6 4 1 MULT 7 7 5 RLC 6 4 14 RR 1 5 4	0111001000101000 0110100111111100 0101000100001000 0001001001000110 1001001101000010 0110001111111010 1010101101001110 1011000011011000

Problem E
어두운 건 무서워

Time Limit: 1 Second

호근이는 겁이 많아 어두운 것을 싫어한다. 호근이에게 어떤 사진을 보여주려는데 사진의 밝기가 평균 이상이 되지 않으면 일절 보려 하지 않는다. 호근이가 이 사진에서 일부분이라도 볼 수 있는 부분을 찾아주자.



위 그림은 호근이에게 보여줄 5×6 크기의 사진이며, 각 픽셀은 밝기를 나타낸다. 호근이가 사진의 일부분이라도 볼 수 있는지 알아보기 위해서는 두 점 (r_1, c_1)과 (r_2, c_2)를 꼭짓점으로 하는 직사각형의 밝기 평균을 구해야 한다. 예를 들어, 위 그림에서는 (2, 2)와 (4, 5)를 꼭짓점으로 하는 직사각형을 말한다.

호근이에게 보여줄 $R \times C$ 크기의 사진이 주어질 때, 사진의 일부분에 해당하는 밝기 평균을 구하여라.

Input

첫 번째 줄에는 사진의 크기를 의미하는 정수 R, C ($1 \le R, C \le 1{,}000$)와 사진 일부분의 밝기 평균을 알아볼 개수를 의미하는 정수 Q ($1 \le Q \le 10{,}000$)가 주어진다.

다음 R개의 줄에 걸쳐 $R \times C$ 크기의 사진 정보가 주어지며, 사진의 각 픽셀에는 밝기를 의미하는 정수 K ($1 \le K \le 1{,}000$)가 주어진다.

다음 Q개의 각 줄에는 사진의 일부분을 나타내기 위한 두 꼭짓점을 의미하는 정수 r_1, c_1, r_2, c_2 ($1 \le r_1 \le r_2 \le R$, $1 \le c_1 \le c_2 \le C$)가 주어진다.

Output

Q개의 각 줄에 주어진 사진에서 두 점 (r_1, c_1)과 (r_2, c_2)를 꼭짓점으로 하는 직사각형의 밝기 평균을 출력한다. 평균은 정수 나눗셈으로 몫만 취한다.

Sample Input 1

```
5 6 1
4 1 3 4 9 5
1 2 8 7 5 5
8 1 2 5 3 2
1 5 3 4 2 5
5 2 1 2 3 5
2 2 4 5
```

Output for the Sample Input 1

```
3
```

Sample Input 2

```
4 3 5
25 93 64
10 29 85
80 63 71
99 58 86
2 2 2 3
3 2 3 3
1 2 2 2
1 2 4 3
2 3 2 3
```

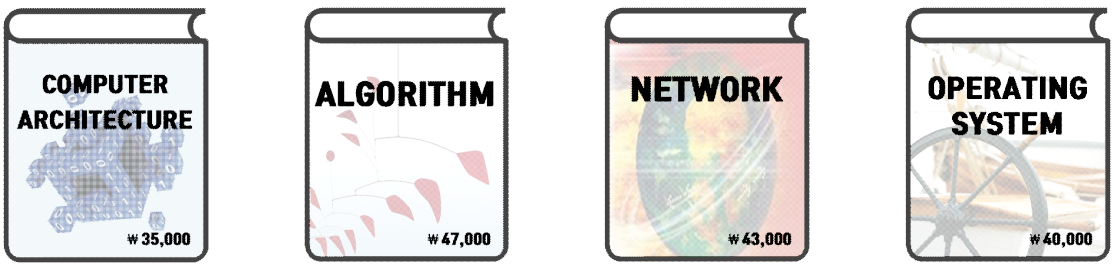
Output for the Sample Input 2

```
57
67
61
68
85
```

Problem F
전공책

Time Limit: 1 Second

곧 졸업을 앞둔 민호는 대학교 생활 동안 구매만 해놓고 한 번도 펴보지 않은 전공책에 먼지가 쌓여 있는 것을 보고는, 이 책들을 어떻게 처리해야 할지 고민 중이다. 열심히 고민한 끝에 민호는 결국 전공책을 모두 버리기로 마음먹었다. 하지만 그냥 버리기엔 심심한 민호는 전공책 제목에 있는 글자들을 오려서 단어 만들기 놀이를 하려고 한다. 단어 만들기 놀이는 아래 예시와 같다.



① ② ③ ④

- 1번 책 : COMPUTERARCHITECTURE (35,000원)
- 2번 책 : ALGORITHM (47,000원)
- 3번 책 : NETWORK (43,000원)
- 4번 책 : OPERATINGSYSTEM (40,000원)

만약 민호가 만들고 싶은 단어가 ALMIGHTY라고 하면, 위 4개의 책 중, 1번 책에서 A를, 2번 책에서 L, M, I, G, H, T를, 4번 책에서 Y를 오려내어 원하는 단어를 만들 수 있다. 이때 드는 비용은 1번, 2번, 4번 책 가격의 합인 122,000원이다.

만약 ANT라는 단어를 만들고 싶다고 하면, 2번 책에서 A를, 3번 책에서 N, T를 오려내어 원하는 단어를 만들 수 있다. 이때 드는 비용은 2번과 3번 책 가격을 합하여 90,000원이다. 그런데, ANT라는 단어에서 A를 2번 책에서 오려내지 않고, 4번 책에 있는 A를 오려낼 수도 있다. 만약 4번 책에서 A를 오려냈을 때 드는 비용은 3번과 4번 책 가격을 합하여 83,000원으로 2번과 3번 책을 고른 비용보다 작다. 하지만, 4번 책에는 ANT가 모두 포함되어 있으므로, 4번 책만 선택했을 때 드는 비용이 40,000원이다. 이는 ANT라는 단어를 만들기 위해서 가장 적은 비용이다.

민호는 여러 개의 전공책들을 나열해 놓고는, 심각한 고민 끝에 전공책 제목에 있는 글자를 오려내어 자신이 원하는 단어를 만들기 위해서는 여러 가지 경우가 있다는 것을 깨달았다. 매우 심심했던 민호는 가장 적은 비용으로 자신이 원하는 단어를 만들려면 어떤 전공책들을 선택해야 하는지 궁금했다. 하지만 일일이 가능한 조합을 만들어 보는 것은 매우 시간 낭비라고 생각한 민호는 컴퓨터공학과답게 프로그램을 만들려고 한다.

민호를 도와 각 전공책의 가격과 제목이 주어졌을 때, 가장 적은 비용으로 민호가 원하는 단어를 만들기 위해서 어떤 전공책을 선택해야 하는지 알아보자.

Input

첫 번째 줄에는 민호가 만들고자 하는 단어를 의미하는 문자열 T ($1 \le |T| \le 10$)가 주어진다. T는 항상 대문자이며, $|T|$는 문자열 T의 길이를 의미한다.

두 번째 줄에는 민호가 가진 전공책의 개수를 의미하는 정수 N ($1 \le N \le 16$)이 주어진다.

다음 N개의 각 줄에는 전공책 가격을 의미하는 정수 C_i ($10{,}000 \le C_i \le 100{,}000$)와 제목을 의미하는 문자열 W_i ($1 \le |W_i| \le 50$)가 주어진다. W_i는 항상 대문자이다.

Output

민호가 원하는 단어 T를 만들기 위해서 선택해야 하는 전공책의 가장 적은 가격의 합을 출력한다. 만약 단어를 만들 수 없다면 -1을 출력한다.

Sample Input 1	Output for the Sample Input 1
ANT 4 35000 COMPUTERARCHITECTURE 47000 ALGORITHM 43000 NETWORK 40000 OPERATINGSYSTEM	40000

Sample Input 2

```
ALMIGHTY
4
35000 COMPUTERARCHITECTURE
47000 ALGORITHM
43000 NETWORK
40000 OPERATINGSYSTEM
```

Output for the Sample Input 2

```
87000
```

Sample Input 3

```
BAKEJOON
3
25000 JAVA
10000 OOP
30000 BINARYCHECK
```

Output for the Sample Input 3

```
65000
```

Sample Input 4

```
JAVA
2
30000 CPLUSPLUS
25000 PYTHON
```

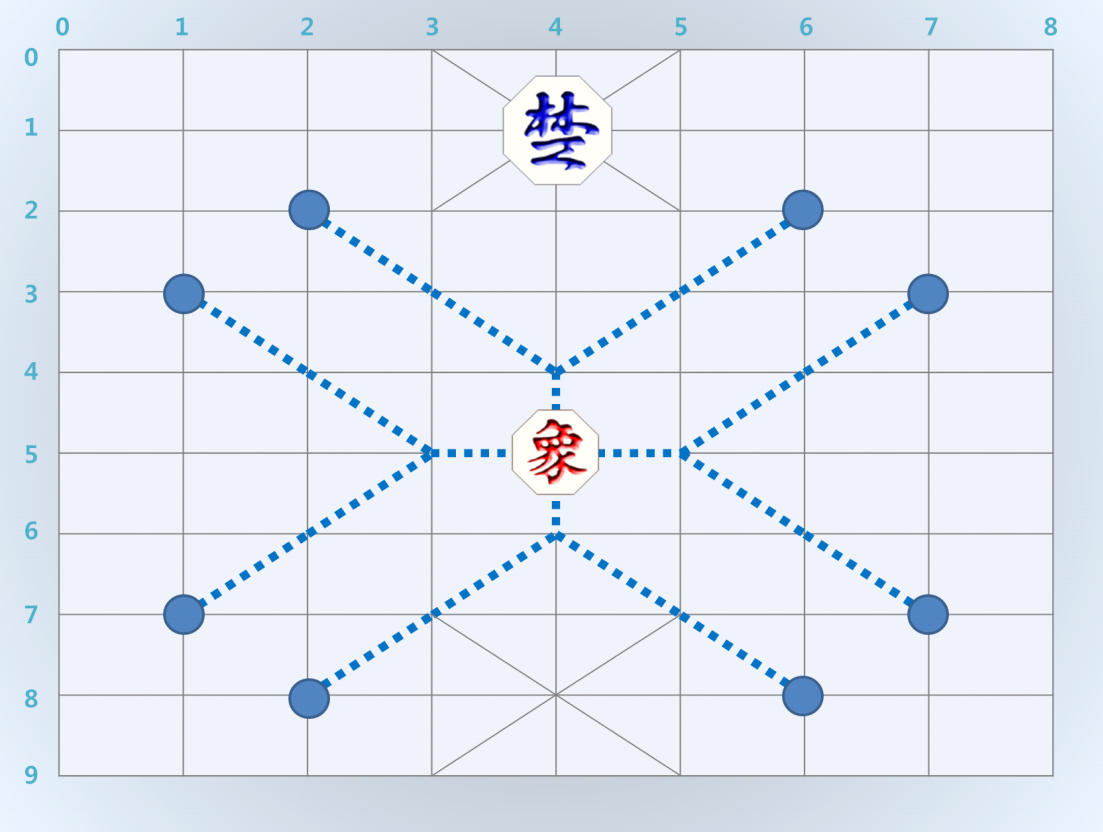
Output for the Sample Input 4

```
-1
```

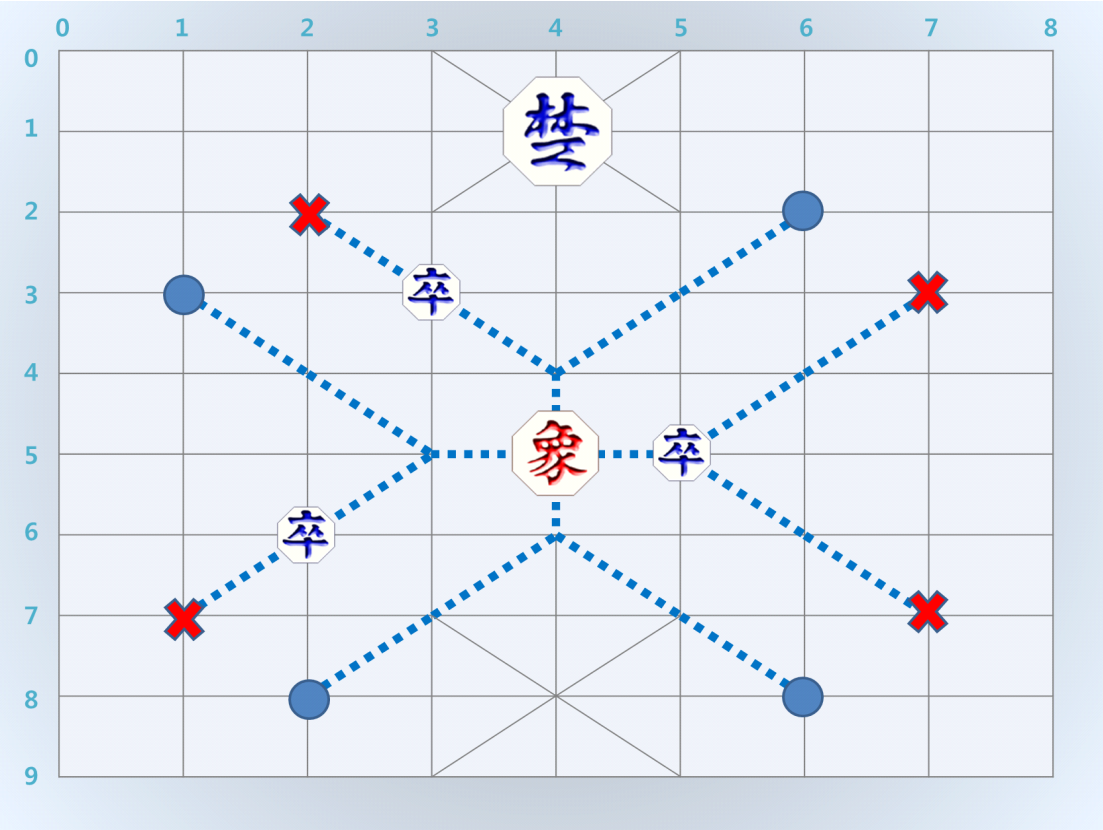
Problem G
장군

Time Limit: 1 Second

오랜만에 휴가를 나온 호근이는 문득 동아리방에 있는 장기가 하고 싶어졌다. 하지만 장기를 오랫동안 하지 않은 탓인지 예전에는 잘 쓰던 상을 제대로 쓰는 것이 너무 힘들었다. 호근이를 위해 상을 어떻게 써야 할지 도와주자.



위 그림은 10×9 크기의 장기판을 나타내며, 상은 (5, 4)에, 왕은 (1, 4)에 자리 잡고 있는 기물이다. (0, 3)과 (2, 5)를 꼭짓점으로 하는 사각형과, (7, 3)과 (9, 5)를 꼭짓점으로 하는 사각형은 왕이 위치할 수 있는 궁성이라고 한다. 상은 위 그림과 같이 8가지 방법으로 움직일 수 있는데, 상, 하, 좌, 우로 한 칸을 이동한 후에 같은 방향 쪽 대각선으로 두 칸 이동한다.



만약 상이 이동하는 경로에 위 그림과 같이 다른 기물이 있다면 상은 그쪽으로 이동할 수 없다. 또한, 상이 장기판을 벗어날 수도 없다.

10×9 크기의 장기판 위에 상과 왕의 처음 위치가 주어졌을 때, 상이 왕에게 도달할 수 있는 최소 이동 횟수를 구하여라.

Input

첫 번째 줄에는 상의 위치를 의미하는 정수 R_1, C_1이 주어진다.

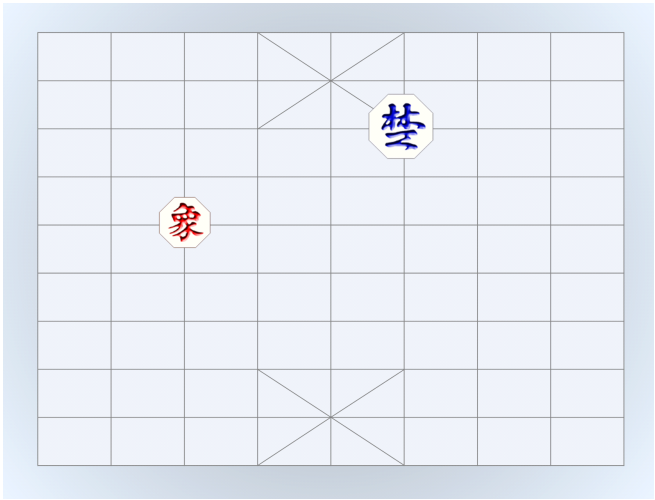
두 번째 줄에는 왕의 위치를 의미하는 정수 R_2, C_2가 주어진다. 장기판에서 R_i ($0 \le R_i \le 9$)는 행을, C_i ($0 \le C_i \le 8$)는 열을 의미한다.

왕은 항상 궁성에 자리 잡고 있으며, 상과 왕의 위치는 겹치지 않는다.

Output

상이 왕에게 도달할 수 있는 최소 이동 횟수를 출력한다. 만약 도달할 수 없다면 -1을 출력한다.

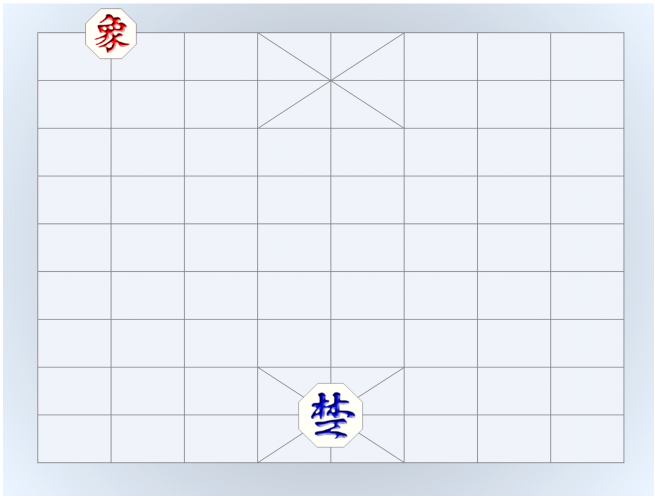
Sample Input 1	Output for the Sample Input 1
4 2 2 5	1



Sample Input 2

0 1 8 4	3

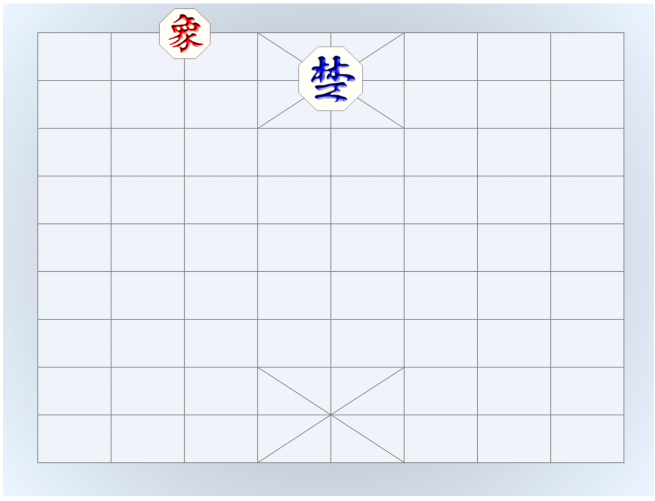
Output for the Sample Input 2



Sample Input 3

0 2 1 4	5

Output for the Sample Input 3



Problem H
Predictable Queue

Time Limit: 1 Second

혁진이는 일을 많이 벌여놓는 것을 좋아한다. 어느 날, 혁진이는 무작정 일을 많이 벌여놓고는 이 수많은 일을 어떻게 처리해야 할지 몰라 곤란해하고 있다. 어떤 순서로 일을 해야 가장 효율적일지를 고민하던 혁진이는 자료구조 시간에 공부한 내용을 바탕으로 좋은 방법을 생각해냈다. 혁진이가 생각한 방법은 매우 단순하다. 일마다 처리하는데 걸리는 시간을 저장해놓고, 일이 주어진 순서대로 한 개씩 처리하는 것이다.

똑똑한 혁진이는 자신이 생각한 방법을 가장 잘 다룰 수 있는 자료구조가 바로 큐(Queue)라는 것쯤은 이미 알고 있었다. 큐는 컴퓨터의 기본적인 자료구조의 한가지로, 먼저 넣은 데이터가 먼저 나오는 FIFO(First In First Out) 구조로 데이터를 저장하는 방식이다.

자신이 생각한 방법대로 일을 잘 처리하던 혁진이는 이 자료구조에 아주 중요한 문제점을 발견했다. 주어진 순서대로 일을 처리하다 보면 중간에 급한 일을 먼저 하고 싶어도 앞에 있는 모든 일을 처리해야만 하는 것이다. 혁진이는 일할 수 있는 시간이 정해져 있어서, 주어진 순서대로 일을 처리하다 보면 중간에 있는 급한 일을 처리할 수도 있고, 못할 수도 있는 상황이 생긴다. 혁진이는 중간에 있는 일을 처리할 수 있는지 알아보려면, 먼저 자신이 일할 수 있는 시간 동안 몇 개의 일을 처리할 수 있는지 예측할 수 있어야 한다고 생각했다.

혁진이는 이러한 기능을 갖춘 새로운 자료구조인 예측 큐(Predictable Queue)를 만들려고 한다. 혁진이를 도와 혁진이가 일할 수 있는 시간 동안 몇 개의 일을 처리할 수 있는지 알아보자.

Input

첫 번째 줄에는 혁진이가 벌여놓은 일의 개수와 일할 수 있는 시간 동안 몇 개의 일을 처리할 수 있는지 알아볼 개수를 의미하는 정수 N, M ($1 \le N, M \le 200{,}000$)이 주어진다.

두 번째 줄에는 공백으로 구분된 N개의 정수 $t_1, t_2, \ldots, t_N$ ($1 \le t_i \le 10{,}000$)이 주어진다. t_i는 큐에 들어가는 각 일의 시간을 의미하며, 가장 왼편에 있는 일이 가장 먼저 주어진 일이다.

다음 M개의 각 줄에는 혁진이가 일할 수 있는 시간을 의미하는 정수 T ($1 \le T \le 2 \times 10^9$)가 주어진다.

Output

M개의 각 줄에 혁진이가 일할 수 있는 시간 동안 처리할 수 있는 일의 개수를 출력한다.

Sample Input 1	Output for the Sample Input 1
7 4 1 2 3 1 2 3 1 1 8 11 14	1 4 5 7