제 31 회

한국정보올림피아드

중등부 문제

자리배정

어떤 공연장에는 가로로 C개, 세로로 R개의 좌석이 $C \times R$격자형으로 배치되어 있다. 각 좌석의 번호는 해당 격자의 좌표 (x,y)로 표시된다.

예를 들어보자. 아래 그림은 가로 7개, 세로 6개 좌석으로 구성된 7×6격자형 좌석배치를 보여주고 있다. 그림에서 각 단위 사각형은 개별 좌석을 나타내며, 그 안에 표시된 값 (x,y)는 해당 좌석의 번호를 나타낸다. 가장 왼쪽 아래의 좌석번호는 $(1,1)$이며, 가장 오른쪽 위 좌석의 번호는 $(7,6)$이다.

(1,6)						(7,6)
			(4,4)			(7,4)
(1,3)					(6,3)	
(1,2)						
(1,1)	(2,1)	(3,1)				(7,1)

이 공연장에 입장하기 위하여 많은 사람이 대기줄에 서있다. 기다리고 있는 사람들은 제일 앞에서부터 1, 2, 3, 4, ... 순으로 대기번호표를 받았다. 우리는 대기번호를 가진 사람들에 대하여 $(1,1)$위치 좌석부터 시작하여 시계방향으로 돌아가면서 비어있는 좌석에 관객을 순서대로 배정한다. 이것을 좀 더 구체적으로 설명하면 다음과 같다.

먼저 첫 번째 사람, 즉 대기번호 1인 사람은 자리 $(1,1)$에 배정한다. 그 다음에는 위쪽 방향의 좌석으로 올라가면서 다음 사람들을 배정한다. 만일 더 이상 위쪽 방향으로 빈 좌석이 없으면 오른쪽으로 가면서 배정한다. 오른쪽에 더 이상 빈자리가 없으면 아래쪽으로 내려간다. 그리고 아래쪽에 더 이상 자리가 없으면 왼쪽으로 가면서 남은 빈 좌석을 배정한다. 이 후 왼쪽으로 더 이상의 빈 좌석이 없으면 다시 위쪽으로 배정하고, 이 과정을 모든 좌석이 배정될 때까지 반복한다.

아래 그림은 7×6공연장에서 대기번호 1번부터 42번까지의 관객이 좌석에 배정된 결과를 보여주고 있다.

6	*7*	*8*	*9*	*10*	*11*	*12*
5	*26*	*27*	*28*	*29*	*30*	*13*
4	*25*	*38*	*39*	*40*	*31*	*14*
3	*24*	*37*	*42*	*41*	*32*	*15*
2	*23*	*36*	*35*	*34*	*33*	*16*
1	*22*	*21*	*20*	*19*	*18*	*17*

여러분은 공연장의 크기를 나타내는 자연수 C와 R이 주어져 있을 때, 대기 순서가 K인 관객에게 배정될 좌석 번호 (x,y)를 찾는 프로그램을 작성해야 한다.

소스파일의 이름은 seat.c 또는 seat.cpp이며 수행시간은 1초를 넘을 수 없다.

입력 형식

입력파일의 이름은 input.txt로 한다. 입력파일의 첫 줄에는 공연장의 격자 크기를 나타내는 정수 C와 R이 하나의 공백을 사이에 두고 차례대로 주어진다. 두 값의 범위는 $5 \le C, R \le 1{,}000$이다. 그 다음 줄에는 어떤 관객의 대기번호 K가 주어진다. 단 $1 \le K \le 100{,}000{,}000$이다.

출력 형식

출력파일의 이름은 output.txt이다. 여러분은 입력파일에 제시된 대기번호 K인 관객에게 배정될

좌석번호 (x,y)를 구해서 두 값, x와 y를 하나의 공백을 사이에 두고 출력해야 한다. 만일 모든 좌석이 배정되어 해당 대기번호의 관객에게 좌석을 배정할 수 없는 경우에는 0(숫자 영)을 출력해야 한다.

입력과 출력의 예

입력(1) (input.txt)

```
7 6
11
```

출력(1) (output.txt)

```
6 6
```

입력(2) (input.txt)

```
7 6
87
```

출력(2) (output.txt)

```
0
```

입력(3) (input.txt)

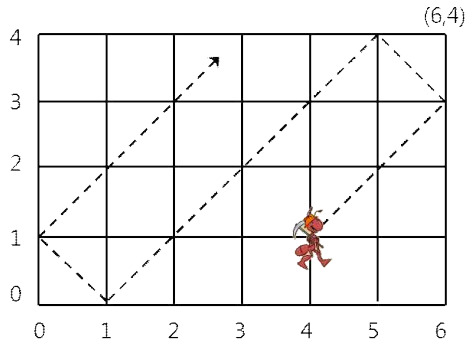
```
100 100
3000
```

출력(3) (output.txt)

```
9 64
```

개미

계면에 부딪치면 같은 속력으로 반사되어 움직인다.



위 그림은 6×4 격자에서 처음에 $(4,1)$에서 출발한 개미가 움직인 길을 보여주고 있다. 처음에 $(4,1)$에 있는 개미는 2시간 후에 $(6,3)$에 있으며 8시간 후에 $(0,1)$에 있다. 만일 그 개미가 처음에 $(5,3)$에 있었다면 매 시간마다 $(6,4)$, $(5,3)$, $(4,2)$, $(3,1)$로 움직인다.

여러분은 크기 $w\times h$인 격자 공간에서 처음에 (p,q)에서 출발하는 개미의 t시간 후의 위치 (x,y)를 계산하여 출력해야 한다. 개미는 절대 지치지 않고 같은 속력으로 이동한다고 가정한다.

문제에서 w와 h는 자연수이며 범위는 $2\le w,h\le 40{,}000$이다. 그리고 개미의 초기 위치 p와 q도 자연수이며 범위는 각각 $0<p<w$과 $0<q<h$이다. 그리고 계산할 시간 t의 범위는 $1\le t\le 2{,}000{,}000{,}000$이다. 단, 채점 데이터의 40%는 $1\le t\le 100{,}000{,}000$이다.

소스파일의 이름은 ant.c 또는 ant.cpp이며 수행 시간은 1초를 넘을 수 없다.

입력 형식

입력파일의 이름은 input.txt로 한다. 입력파일의 첫줄에는 w와 h가 공백을 사이에 두고 주어진다. 그 다음 줄에는 초기 위치의 좌표값 p와 q가 공백을 사이에 두고 주어진다. 3번째 줄에는 개미가 움직일 시간 t가 주어진다.

출력 형식

출력파일의 이름은 output.txt이다. 출력은 t 시간 후에 개미의 위치 좌표 (x,y)의 값 x와 y를 공백을 사이에 두고 출력한다.

입력과 출력의 예

입력(1) (input.txt)

```
6 4
4 1
8
```

출력(1) (output.txt)

```
0 1
```

입력(2) (input.txt)

```
6 4
5 3
4
```

출력(2) (output.txt)

```
3 1
```

저울

무게가 서로 다른 N개의 물건이 있다. 각 물건은 1부터 N까지 번호가 매겨져 있다. 우리는 일부 물건 쌍에 대해서 양팔 저울로 어떤 것이 무거운 것인지를 측정한 결과표를 가지고 있다. 이 결과표로부터 직접 측정하지 않은 물건 쌍의 비교 결과를 알아낼 수도 있고 알아내지 못할 수도 있다.

예를 들어, 총 6개의 물건이 있고, 다음 5개의 비교 결과가 주어졌다고 가정하자. ([1]은 1번 물건의 무게를 의미한다.)

[1]>[2], [2]>[3], [3]>[4], [5]>[4], [6]>[5]

우리는 [2]>[3], [3]>[4]로부터 [2]>[4]라는 것을 알 수 있다. 하지만, 물건 2와 물건 6을 비교하는 경우, 앞서의 결과만으로는 어느 것이 무거운지 알 수 없다. 이와 같이, 물건 2는 물건 1, 3, 4와의 비교 결과는 알 수 있지만, 물건 5, 6과의 비교 결과는 알 수 없다. 물건 4는 모든 다른 물건과의 비교 결과를 알 수 있다.

비교 결과가 모순되는 입력은 없다고 가정한다. 위 예제의 기존 측정 결과에 [3]>[1]이 추가되었다고 가정하자. 이 경우 [1]>[2], [2]>[3]이므로 우리는 [1]>[3]이라는 것을 예측할 수 있는데, 이는 기존에 측정된 결과 [3]>[1]과 서로 모순이므로 이러한 입력은 가능하지 않다.

물건의 개수 N과 일부 물건 쌍의 비교 결과가 주어졌을 때, 각 물건에 대해서 그 물건과의 비교 결과를 알 수 없는 물건의 개수를 출력하는 프로그램을 작성하시오.

소스파일의 이름은 bal.c 또는 bal.cpp이며 수행 시간은 0.2초를 넘을 수 없다. 부분점수는 없다.

입력 형식

입력파일의 이름은 input.txt로 한다. 입력파일의 첫 줄에는 물건의 개수 N이 주어지고, 둘째 줄에는 미리 측정된 물건 쌍의 개수 M이 주어진다. 단, $5 \le N \le 100$이고, $0 \le M \le 2{,}000$이다. 다음 M개의 줄에 미리 측정된 비교 결과가 한 줄에 하나씩 주어진다. 각 줄에는 측정된 물건 번호를 나타내는 두 개의 정수가 공백을 사이에 두고 주어지며, 앞의 물건이 뒤의 물건보다 더 무겁다.

출력 형식

출력파일의 이름은 output.txt이다. 여러분은 N개의 줄에 결과를 출력해야 한다. i번째 줄에는 물건 i와 비교 결과를 알 수 없는 물건의 개수를 출력한다.

입력과 출력의 예

입력(1) (input.txt)

```
6
5
1 2
2 3
3 4
5 4
6 5
```

출력(1) (output.txt)

```
2
2
2
0
3
3
```

입력(2) (input.txt)

```
9
11
2 1
3 1
2 8
2 9
7 8
4 5
6 7
6 3
1 7
6 2
1 9
```

출력(2) (output.txt)

```
2
3
3
7
7
2
3
3
4
```

암호

새로 바뀐 KOI 웹사이트의 암호는 N개의 영문 알파벳 대문자로 이루어진다. 특별히 암호는 영문 알파벳 중 처음 K개를 사용해서 만든다. 예를 들어, $K=5$이면, 'A', 'B', 'C', 'D', 'E'만으로 암호를 만들게 된다. 하지만 문자가 중복되어 나타날 수도 있고 전혀 안 나타날 수도 있다.

최근 연구에 의해서 2가지의 특정 패턴이 암호에 상당히 많이 나타난다는 사실이 알려졌다. 이 특정 패턴은 **ABCBC** 와 **ABABC** 이다. 해커들이 이 정보를 이용할 수 있기 때문에 암호를 만들 때 이 두 패턴 중 어떤 것도 암호 안에 나타나는 것을 피하는 것이 안전한 암호를 만드는 방법이 될 수 있다. 따라서 위의 패턴이 나타나지 않는 암호를 안전한 암호라고 부르고, KOI 웹사이트는 사용자들이 안전한 암호만을 사용하도록 결정하였다.

예를 들어, $N=6$, $K=3$일 때, 위 패턴을 포함하는 암호들은 다음과 같은 12가지가 있다.

ABCBCA, ABCBCB, ABCBCC, AABCBC, BABCBC, CABCBC, ABABCA, ABABCB, ABABCC, AABABC, BABABC, CABABC

가능한 총 암호의 개수 $3^6=729$에서 위의 12가지를 제외하면 717가지의 안전한 암호를 만들 수 있다.

암호의 길이 N, 문자의 가지 수 K가 주어질 때, 만들 수 있는 안전한 암호의 총 개수를 구하는 프로그램을 작성하시오.

소스 파일의 이름은 pass.c 혹은 pass.cpp이며 수행 시간은 1초를 넘을 수 없다.

입력 형식

입력파일의 이름은 input.txt로 한다. 입력파일의 첫 줄에는 각각 암호의 길이와 문자의 가지 수를 나타내는 정수 N과 K가 공백을 사이에 두고 주어진다. 이 두 정수 값의 범위는 $5 \le N \le 1{,}000{,}000$, $3 \le K \le 26$이다.

출력 형식

출력파일의 이름은 output.txt이다. 출력은 한 줄로 이루어진다. 안전한 암호의 총 개수를 $1{,}000{,}000{,}009$으로 나눈 나머지를 출력한다. 계산 과정에서 32비트 정수 변수가 표현할 수 있는 범위를 넘어서 64비트 정수 변수를 사용해야 할 수도 있음에 주의하라.

입력 분포

- 채점 데이터 중 10%에서 $K=3, N \le 10$이다.
- 채점 데이터 중 40%에서 $K \le 10$, $10 < N \le 300$이다.
- 채점 데이터 중 20%에서 $300 < N \le 10{,}000$이다.
- 채점 데이터 중 20%에서 $10{,}000 < N \le 50{,}000$이다.
- 채점 데이터 중 10%에서 $50{,}000 < N \le 1{,}000{,}000$이다.

입력과 출력의 예

입력(1) (input.txt)

```
6 3
```

출력(1) (output.txt)

```
717
```

입력(2) (input.txt)

```
10 3
```

출력(2) (output.txt)

```
56245
```

중등부 1번 문제(seat)

```
#include <stdio.h>

int yy[]={1,0,-1,0},xx[]={0,1,0,-1};
int T,R,C,A[1001][1001],Y[1000001],X[1000001];

int main()
{
	int y,x,n,d;
	freopen("input.txt","r",stdin), freopen("output.txt","w",stdout);
	scanf("%d%d",&C,&R);
	for (n=x=1,y=d=0;n<=R*C;n++){
		int ny = y+yy[d], nx = x+xx[d];
		while (!(1 <= ny && ny <= R && 1 <= nx && nx <= C &&
!A[ny][nx])){
			d = (d+1)%4;
			ny = y+yy[d], nx = x+xx[d];
		}
		y = ny, x = nx;
		A[y][x] = n;
		Y[n] = y, X[n] = x;
	}
	scanf("%d",&n);
	if (1 <= n && n <= R*C) printf("%d %d\n",X[n],Y[n]);
	else puts("0");
}
```

중등부 2번 문제(ant)

```
#include <stdio.h>

int W,H,X,Y,T,P,Q;

int main()
{
        freopen("input.txt","r",stdin), freopen("output.txt","w",stdout);
        scanf("%d%d%d%d%d",&W,&H,&X,&Y,&T);
        X += T, Y += T;
        P = X%W, Q = Y%H;
        if ((X/W)&1) P = W-P;
        if ((Y/H)&1) Q = H-Q;
        printf("%d %d\n",P,Q);
}
```

중등부 3번 문제(bal)

```
#include <stdio.h>

int f[101][101];

int main()
{
	freopen ("input.txt","r",stdin);
	freopen ("output.txt","w",stdout);

	int N,M;
	scanf ("%d %d",&N,&M);

	for (int i=0,x,y;i<M;i++){
		scanf ("%d %d",&x,&y);
		f[x][y] = 1;
	}

	for (int k=1;k<=N;k++){
		for (int i=1;i<=N;i++){
			if (f[i][k]){
				for (int j=1;j<=N;j++){
					if (f[k][j]) f[i][j] = 1;
				}
			}
		}
	}

	for (int i=1;i<=N;i++){
		int count = 0;
		for (int j=1;j<=N;j++) if (i != j && f[i][j] + f[j][i] == 0) count++;
		printf ("%d\n",count);
	}

	return 0;
}
```

중등부 4번 문제(pass)

```
#include <stdio.h>

#define MAXN 1000006
#define MOD 1000000009
typedef long long lld;

int N,K,D[7],E[7],ans;

int main()
{
	int i,j;
	freopen("input.txt","r",stdin), freopen("output.txt","w",stdout);
	scanf("%d%d",&N,&K);
	E[0] = 1;
	for (i=1;i<=N;i++){
		D[0] = ((lld)E[0]*(K-1)%MOD + (lld)E[1]*(K-2)%MOD + (lld)E[2]*(K-2)%MOD + (lld)E[3]*(K-2)%MOD + (lld)E[4]*(K-2)%MOD + (lld)E[5]*(K-2)%MOD + (lld)E[6]*(K-2)%MOD)%MOD;
		D[1] = ((lld)E[0] + E[1] + E[3] + E[5] + E[6])%MOD;
		D[2] = E[1];
		D[3] = (E[2]+E[4])%MOD;
		D[4] = E[3];
		D[5] = E[2];
		D[6] = E[5];
		for (j=0;j<7;j++) E[j] = D[j];
	}
	for (i=0;i<7;i++) ans = (ans+D[i])%MOD;
	printf("%d\n",ans);
}
```

나. 중등부 해법

(1) 자리배정 (seat)

의도한 풀이는 $O(RC)$이지만, 보다 더 좋은 별해들을 별도로 설명한다.

1) $O(RC)$

대기 번호는 (1,1)에서 시작하여 (1,2), (1,3), (1,4), (1,5) ,.... 순으로 증가하다가 (1,R)에서 방향을 바꿔 (2,R) , (3,R) ... 순으로 나아가게 된다. 이렇게 대기번호를 한 단계씩 늘려가면서 시뮬레이션을 통해 대기번호에 따른 좌표를 구할 수 있다.

2) $O(R+C)$

위에서 설명한 규칙을 보면, 처음에 위로 $R-1$만큼 올라간 뒤, 오른쪽으로 $C-1$만큼 가고, 밑으로 $R-1$만큼 내려온 뒤 왼쪽으로 $C-2$만큼 가고, …, 규칙적이게 번호가 매겨진다. 대기번호가 $C-1$, $R-1$, $C-2$, $R-2$, $C-3$, $R-3$, … 씩 지났을 때 방향을 바꾸게 된다.

이를 이용하여 1)에서 설명한 시뮬레이션을 묶음 단위로 할 수 있다. 이를 통해 보다 더 빨리 답을 구해낼 수 있다.

3) $O(\lg(Max(R, C)))$

대기 번호의 방향이 $C-1$, $R-1$,$C-2$, $R-2$, $C-3$, $R-3$, … 묶음 단위로 바뀐다는 특징을 보았을 때, 이는 1씩 줄어듦을 확인할 수 있다. 이를 통해, 주어진 K가 몇 번째 묶음 단위에 속하는지 이진 탐색(Binary Search)을 통해 빠르게 구할 수 있다. K가 어느 묶음에 속하는지 알아낸 뒤에는, 그 묶음 속에서 해당 대기번호가 어디에 속하는지 계산할 수 있다.

4) $O(1)$

3)과 거의 유사하지만, 주어진 K가 몇 번째 묶음 단위에 속하는지를 이진 탐색(Binary Search) 대신 이차방정식을 풀어 구할 수 있다. 이차방정식은 상수 시간안에 해결할 수 있으므로 상수 시간 안에 해결 가능하다.

(2) 개미 (ant)

1) $O(t)$ 방법 (40점)

개미의 좌표 (x,y)를 1시간 지날 때 마다 변화시키는 시뮬레이션을 돌려본다.

문제에 주어진 입출력 예제 1에서 예를 들어보면 다음과 같다.

(4,1) → (5,2) → (6,3) → (5,4) → (4,3) → (3,2) → (2,1) → (1,0) → (0,1)

0시간 1시간 2시간 3시간 4시간 5시간 6시간 7시간 8시간

2) 70점 방법

개미의 좌표 (x,y)를 격자 공간의 모서리, 꼭짓점에 닿을 상황으로만 변화시키는 시뮬레이션을 돌린다.

문제에 주어진 입출력 예제 1에서 예를 들어보면 다음과 같다.

(4,1) → (6,3) → (5,4) → (1,0) → (0,1)

0시간 2시간 3시간 7시간 8시간

3) $O(1)$ 해법 (100점)

x좌표와 y좌표는 서로 독립적이므로, 서로에게 영향이 없고 따로 계산할 수 있다.

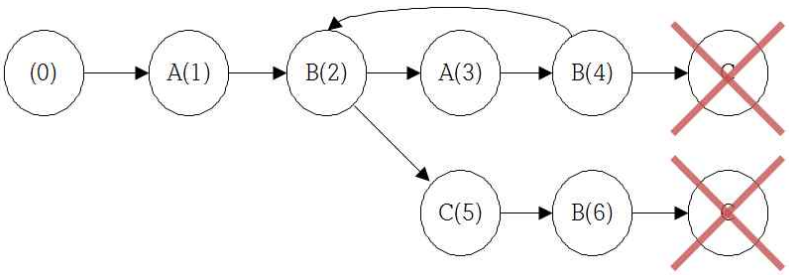
우선, 개미의 최종 x 좌표를 구하는 방법은 다음과 같다.

개미가 격자 공간에서 x 좌표의 경계($x=0$ or $x=w$)에 닿는 횟수를 계산한다. c를 그 횟수라고 할 때 값은 $c=(x+t)/w$ 이며, t 시간이 지난 뒤 개미의 최종 x 좌표 x_f는 아래와

같다.
c가 홀수일 때 : $x_f = w - ((x+t) \bmod w)$
c가 짝수일 때 : $x_f = (x+t) \bmod w$
마찬가지의 방법으로 t 시간이 지난 뒤 개미의 최종 y 좌표를 구할 수 있다.

(3) 저울 (bal)
이 문제는 얼핏 보기에는 까다로운 문제처럼 보이지만, 문제의 상황을 그래프로 표현하여 풀면 보다 간단해진다. N개의 물건들을 각각 노드로 표현하고, 각 물건 무게 사이의 관계를 아래와 같은 규칙을 통해 단방향 간선으로 표현하자.
[x] > [y]와 같은 입력이 주어졌을 때, 이것을 x번 노드에서 y번 노드로 가는 단방향 간선으로 표현한다. 즉, 그래프에는 N개의 노드와 M개의 단방향 간선이 있다.
이 그래프에서 두 물건 사이의 무게 관계를 쉽게 알 수 있는데, 두 물건 p, q의 무게 관계를 알 수 있는 경우는 p번 노드에서 q번 노드로 가는 경로가 있거나, q번 노드에서 p번 노드로 가는 경로가 있는 경우다.
위에서 설명한 것과 같이 그래프를 만들고, DFS, BFS나 Floyd-Warshall 알고리즘을 사용하여 $O(N^3)$ 시간복잡도에 해결할 수 있다.

(4) 암호 (pass)
1) $O(K^N N)$
K개의 알파벳으로 길이 N인 문자열들을 모두 구한 뒤, 패턴 “ABABC”, “ABCBC”이 나타나는지 확인하여, 가능한 암호의 개수를 세는 방법이다. 이 방법으로 문제를 해결할 시 10%의 점수를 받을 수 있다.
2) $O(NK^5)$
동적계획법으로 해결 가능하다. 아래와 같이 2차원 배열 D를 정의한다.
$D_{i\,p}$ = 길이가 i인 유효한 암호문이며 문자열 뒤 네 글자가 p인 문자열의 개수
정의에 따라 p는 길이가 4인 문자열이다. 뒤에 K개 종류의 알파벳을 넣어 “ABABC”나 “ABCBC”가 되는지 확인하고, 패턴이 나오지 않은 경우 $D_{i+1\,p'}$에 $D_{i\,p}$를 더해주면 된다. 여기서, p'는 문자열 p에 새로 알파벳을 하나 뒤에 붙인 문자열의 뒤 네 글자로 구성 된 문자열이다. 이 방법으로 문제를 해결할 시 50%의 점수를 받을 수 있다.
3) $O(N \cdot 4^5)$
$O(NK^5)$ 방법과 유사한 동적계획법으로 해결한다. 다만 정의의 문자열 p에서 K개 종류 알파벳을 모두 사용하는 것이 아니라, 네 종류의 알파벳을 사용하며 문제를 해결할 수 있다. 이 방법으로 문제를 해결할 시 90%의 점수를 받을 수 있다.
4) $O(N)$
위에서 설명한 방법에서 뒤 네 글자를 동적계획법의 상태로 가지고 있다. 이는 $4^4 = 256$가지의 상태를 가지고 있는 것인데, 여기서 상태의 개수를 7개로 줄일 수 있다.
$D_{i\,state}$ = 길이가 i인 유효한 암호문이며 현재 상태가 state인 문자열의 개수
state들의 이동은 아래 그림에 의해 설명된다.



※ 괄호 안의 수는 모범코드에서 state의 번호를 의미함

점화식에 대한 자세한 설명은 첨부된 모범코드를 참고한다.

5) $O(\lg N)$

추가적으로 효과적인 방법으로는 $O(N)$의 점화식을 행렬 곱셈식으로 표현한 뒤, repeated squaring으로 $O(\lg N)$에 계산하는 방법이 있다.