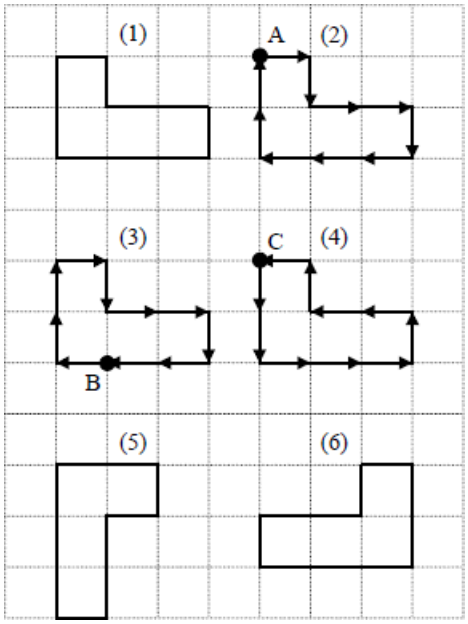
다각형 그리기

모눈종이에 다각형을 그리려고 한다. 그리는 방법은 **모양수열**로 표시된다. 모양수열은 1과 4사이의 숫자가 연속되어 나열된 것으로 1은 오른쪽으로, 2는 윗쪽으로, 3은 왼쪽으로 4는 아래쪽으로 한 칸씩 그리는 것을 말한다.

예를 들어 아래 그림의 다각형 (2)는 점 A에서 시작하여 화살표 방향으로 모양수열 1411433322를 따라서 그린 것이다. 다각형 (3)은 점 B에서 시작하여 화살표 방향으로 모양수열 3221411433을 따라서 그린 것이다. 또한 다각형 (4)는 점 C에서 시작하여 화살표 방향으로 모양수열 4411123323을 따라서 그린 것이다. 다각형 (2), (3), (4)는 다각형 (1)과 같으므로 모양수열들 1411433322, 3221411433, 4411123323은 모두 같은 다각형을 그릴 수 있다. 단, 다각형이 회전된 것이나 뒤집어진 것은 같은 다각형이 아니다. 그러므로 아래 그림의 다각형 (5)와 (6)은 다각형 (1)과는 다르다.



한 개의 표본 모양수열과 여러 모양수열들이 주어졌을 때 표본 모양수열과 같은 다각형을 그릴 수 있는 모양수열들을 모두 찾는 프로그램을 작성하시오.

실행 파일의 이름은 POLY.EXE로 하고, 프로그램의 실행시간은 10초를 초과할 수 없다.

입력 형식

입력 파일의 이름은 INPUT.TXT이다. 첫째 줄에 표본 모양수열의 길이(숫자의 개수)가 주어지고 둘째 줄에는 표본 모양수열이 주어진다. 셋째 줄에는 모양수열의 개수가 주어지고 넷째 줄부터는 각 줄에 표본 모양수열과 같은 길이의 모양수열이 하나씩 주어진다. 단, 모양수열들의 개수는 최대 100개이고 모양수열의 길이는 최대 50이다. 모양수열의 각 숫자 사이에는 빈칸이 하나 있다.

출력 형식

출력 파일의 이름은 OUTPUT.TXT이다. 첫째 줄에는 입력된 표본 모양수열과 같은 다각형을 그리는 모양수열들의 개수를 출력한다. 둘째 줄부터는 각 줄에 표본 모양수열과 같은 다각형을 그릴 수 있는 모양수열을 출력한다. 출력되는 모양수열의 숫자들은 한 칸 띄고 출력한다.

입력과 출력의 예

입력(INPUT.TXT)

```
10
1 4 1 1 4 3 3 3 2 2
3
3 2 2 1 4 1 1 4 3 3
1 4 4 3 3 3 2 1 1 2
4 4 1 1 1 2 3 3 2 3
```

출력(OUTPUT.TXT)

```
2
3 2 2 1 4 1 1 4 3 3
4 4 1 1 1 2 3 3 2 3
```

풀이

이 문제는 주어진 두 개의 '모양 수열'이 같은 다각형을 나타내는지를 판단하는 문제이다. 같은 다각형의 모양 수열은 여러 가지 종류가 가능한데, 이는 시작점의 위치와 회전 방향에 따라 달라진다. 따라서 이 두 가지에 대해서 모든 경우를 고려해 보고, 그 때 일치하는 수열이 만들어지는지 확인하면 다각형을 구성하지 않고도 판단할 수 있다.

이러한 알고리즘을 위해서 먼저 한 개의 수열을 기준으로 잡고 다른 하나의 수열을 한 칸씩 옆으로 돌려가며 같은지 비교해 주어야 한다. 이를 위해서 실제로 회전된 형태의 수열을 만들어 볼 필요는 없고, 배열을 탐색하는 순서를 바꿔주는 방식으로 해결할 수 있다. 또한, 다각형을 반대 방향으로 그린 경우도 있을 수 있으므로, 수열을 뒤집은 뒤 1과 3을 바꾸고 2와 4를 바꾼 다음에 위의 과정을 한 번 더 거치며 비교해 주어야 한다.

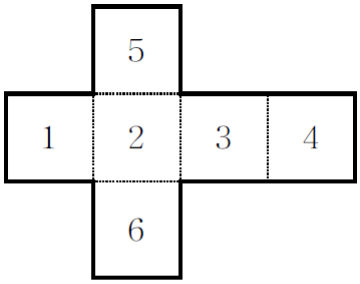
이러한 과정을 수행하면서 일치하는 경우가 한 번이라도 있다면 답으로 체크해 두고, 최종적으로는 체크된 수열의 개수를 출력한 후 해당하는 수열을 출력하면 된다.

```
#include<stdio.h>

int a[50], b[100][50], c[100], d[50];
int n, m, ret;

int Match(void)
{
	int l1, l2;
	for(l1=0;l1<n;l1++)
	{
		for(l2=0;l2<n;l2++)
			if(a[l2] != d[(l1+l2)%n])
				break;
		if(l2 == n) return 1;
	}
	for(l1=0;l1<n;l1++)
	{
		for(l2=0;l2<n;l2++)
			if(a[n-1-l2] != d[(l1+l2)%n])
				break;
		if(l2 == n) return 1;
	}
	return 0;
}

int main(void)
{
	int l1, l2;
	freopen("input.txt","r",stdin);
	freopen("output.txt","w",stdout);

	scanf("%d",&n);
	for(l1=0;l1<n;l1++) scanf("%d",&a[l1]);
	scanf("%d",&m);
	for(l1=0;l1<m;l1++)
	{
		for(l2=0;l2<n;l2++) scanf("%d",&b[l1][l2]);
		for(l2=0;l2<n;l2++) d[l2] = b[l1][l2];
		if(Match()) c[l1] = 1;
		for(l2=0;l2<n;l2++)
		{
			if(d[l2] == 1) d[l2] = 3;
			else if(d[l2] == 2) d[l2] = 4;
			else if(d[l2] == 3) d[l2] = 1;
			else if(d[l2] == 4) d[l2] = 2;
		}
		if(Match()) c[l1] = 1;
		if(c[l1]) ret++;
	}

	printf("%d\n",ret);
```

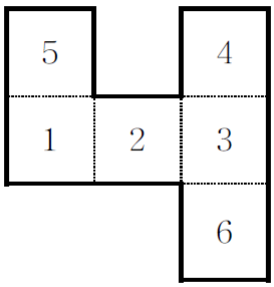
```
        for(l1=0;l1<m;l1++)
        {
                if(c[l1])
                {
                        for(l2=0;l2<n;l2++)
                        {
                                if(l2) printf(" ");
                                printf("%d",b[l1][l2]);
                        }
                        printf("\n");
                }
        }

        return 0;
}
```

전개도

아래에 주어진 전개도의 점선 부분을 접어서 주사위 모양의 정육면체를 만들 수 있는지를 생각해보자. 전개도의 각 면은 1에서 6까지 서로 다른 정수로 표시되어 있다.



전개도 (1)



전개도 (2)

전개도 (1)은 정육면체로 접을 수 있지만, 전개도 (2)는 정육면체로 접을 수 없다. 입력으로 주어진 전개도를 정육면체로 접을 수 있는지를 알아보는 프로그램을 작성하시오.

실행 파일의 이름은 FOLD.EXE로 하고, 프로그램의 실행시간은 5초를 초과할 수 없다.

입력 형식

입력 파일의 이름은 INPUT.TXT이다. 입력 파일은 여섯 줄로 되어 있으며 각 줄에는 0에서 6까지의 정수들이 여섯 개 있고, 숫자 사이에는 빈칸이 하나씩 있다. 1에서 6까지의 숫자는 전개도의 면을 나타내고, 0은 전개도의 바깥 부분을 나타낸다.

출력 형식

출력 파일의 이름은 OUTPUT.TXT이다. 입력된 전개도를 정육면체로 접을 수 있으면, 정육면체에서 1번으로 표시된 면의 맞은 편 면의 번호를 출력하고, 정육면체로 접을 수 없으면 0을 출력한다.

입력과 출력의 예

전개도 (1)의 입력 예

입력(INPUT.TXT)

```
0 0 0 0 0 0
0 0 0 0 0 0
0 0 5 0 0 0
0 1 2 3 4 0
0 0 6 0 0 0
0 0 0 0 0 0
```

출력(OUTPUT.TXT)

```
3
```

전개도 (2)의 입력 예

입력(INPUT.TXT)

```
0 0 0 0 0 0
0 5 0 4 0 0
0 1 2 3 0 0
0 0 0 6 0 0
0 0 0 0 0 0
0 0 0 0 0 0
```

출력(OUTPUT.TXT)

```
0
```

풀이

주어진 전개도를 접어서 정육면체를 만들 수 있는지, 만들 수 있다면 마주보는 면은 어느 면인지를 판단하는 문제로, 여러 가지 방법으로 해결할 수 있다. 가장 간단한 방법은 모든 가능한 전개도를 미리 저장하는 방식이다. 정육면체의 가능한 전개도의 종류를 헤아려보면 몇 가지밖에 되지 않음을 확인할 수 있다. 이러한 성질을 이용해서 손으로, 혹은 프로그램으로 모든 종류의 전개도를 만들고, 이를 프로그램에 저장한다. 프로그램은 입력으로 주어진 전개도가 저장된 종류에 들어가는지 확인하고, 해당 면의 번호를 출력하면 된다.

또 다른 방법으로는 전개도가 갖추어야 할 성질을 이용하는 것이 있다. 문제에서 요구하는 바를 따져 보면, 전개도 위에서 반대편 면을 찾아서 저장하는 방식이 적합함을 알 수 있다. 즉, 전개도 위에서 어떤 두 면이 서로 마주보는지 아닌지를 알아낼 수 있는 조건을 이용해서 문제를 풀면 된다. 이러한 면들은 일정 조건을 만족하면 되는데, 모범 답안에서는 다음과 같은 성질을 이용하였다. 먼저 정육면체의 전개도 상에서 두 면이 마주보기 위해서는 x좌표나 y좌표의 차이 중 하나가 정확히 2여야 한다. 이는 두 면이 한 칸 건너에 있어야 서로 마주보게 되기 때문이다. 또한 전체가 하나의 전개도로 구성되기 위해서, 두 면의 사이가 다른 면들로 연결되어 있어야 한다. 이러한 성질을 확인한 후, 조건들이 만족되면 올바른 전개도로 판단하여 찾아낸 면을 출력하면 된다.

```
#include<stdio.h>

#define Swap(aa,bb) {cc=aa;aa=bb;bb=cc;}

int a[6][6], b[7], c[7], d[7], cc;

int Match(int x1, int y1, int x2, int y2)
{
	int l1;

	if(x1 > x2) Swap(x1, x2);
	if(y1 > y2) Swap(y1, y2);

	if(x2 == x1 + 2)
	{
		for(l1=y1;l1<=y2;l1++)
			if(a[x1+1][l1] == 0) break
		if(l1 > y2) return 1;
	}
	if(y2 == y1 + 2)
	{
		for(l1=x1;l1<=x2;l1++)
			if(a[l1][y1+1] == 0) break
		if(l1 > x2) return 1;
	}

	return 0;
}

int main(void)
{
	int l1, l2, l3;

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

	for(l1=0;l1<6;l1++) for(l2=0;l2<6;l2++)
	{
		scanf("%d",&a[l1][l2]);
		b[a[l1][l2]]=l1;
		c[a[l1][l2]]=l2;
	}

	for(l1=1;l1<=6;l1++)
		if(d[l1] == 0)
			for(l2=l1+1;l2<=6;l2++)
				if(d[l2] == 0)
					if(Match(b[l1], c[l1], b[l2], c[l2]))
					{
						d[l1] = l2;
						d[l2] = l1;
```

```
                                                    break
                                                }
        for(l1=1;l1<=6;l1++) if(d[l1] == 0) break
        if(l1 > 6) printf("%d\n",d[1]);
        else printf("0\n");
}
```

색종이 올려 놓기

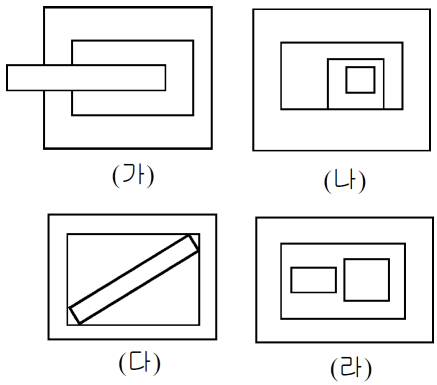
크기가 모두 다른 직사각형 모양의 색종이가 여러 장 있다. 색종이를 하나씩 올려 놓아, 되도록 많은 장수의 색종이들을 쌓으려고 한다.

새로 한 장의 색종이를 올려 놓을 때는 지금까지 쌓아 놓은 색종이들 중 맨 위의 색종이 위에 올려놓아야 하며 아래의 두 조건을 모두 만족해야 한다.

조건 1 : 새로 올려 놓는 색종이는 맨 위의 색종이보다 크지 않아야 한다. 즉, 맨 위의 색종이 밖으로 나가지 않아야 한다.

조건 2 : 새로 올려 놓는 색종이와 맨 위의 색종이의 변들은 서로 평행해야 한다. (색종이를 90° 돌려 놓을 수 있다.)

예를 들어, 아래의 그림 중에서 위의 두 조건을 모두 만족하는 경우는 (나) 뿐이다.



색종이는 두 변의 길이로 표현된다. (3, 5)는 두 변의 길이가 각각 3과 5인 색종이를 나타낸다. 예를 들어, 다음과 같이 7장의 색종이가 주어졌다고 하자 : (1, 2), (8, 7), (20, 10), (20, 20), (15, 12), (12, 14), (11, 12). 위의 두 조건을 만족하면서 가장 많이 쌓을 수 있는 색종이들의 순서는 (20, 20), (15, 12), (12, 14), (11, 12), (8, 7), (1, 2)이다.

입력으로 색종이들이 주어졌을 때, 위의 조건 1과 조건 2를 만족하면서 쌓을 수 있는 최대 색종이 장수를 구하는 프로그램을 작성하시오.

실행 파일의 이름은 PAPER.EXE로 하고, 프로그램의 실행시간은 10초를 초과할 수 없다.

입력 형식

입력 파일의 이름은 INPUT.TXT이다. 첫 번째 줄에는 색종이의 장수가 주어진다. 다음 줄부터 각 줄에는 색종이의 두 변의 길이가 주어진다. 두 변의 길이는 한 칸 띄어 주어진다. 색종이의 최대 장수는 100이고, 각 변의 길이는 1000보다 작은 자연수이다.

출력 형식

출력 파일의 이름은 OUTPUT.TXT이다. 쌓을 수 있는 최대 색종이 장수를 출력한다.

입력과 출력의 예

입력(INPUT.TXT)

```
7
1 2
8 7
20 10
20 20
15 12
12 14
11 12
```

출력(OUTPUT.TXT)

```
6
```

풀이

색종이들을 쌓을 때 위의 색종이가 아래의 색종이를 벗어나지 않게 쌓을 수 있는 최대 색종이 장수를 구하는 문제로, 동적 계획법을 이용하여 해결할 수 있다.

먼저 두 색종이를 올려놓을 수 있는 조건을 살펴보자. 색종이 (a, b)를 색종이 (c, d) 위에 올려놓을 수 있으려면, a<=c 그리고 b<=d를 만족하거나 a<=d 그리고 b<=c를 만족해야 한다. 이 조건을 조금 더 정리해 보면 색종이 (c, d)의 짧은 변이 색종이 (a, b)의 짧은 변보다 작거나 같아야 하고, 색종이 (c, d)의 긴 변도 마찬가지로 색종이 (a, b)의 긴 변보다 작거나 같아야 한다는 것을 알 수 있다.

이제 각 색종이에 대해 두 변의 길이 중 짧은 것을 앞으로 오도록 한 뒤, 모든 색종이를 두 변의 길이 중 짧은 변의 길이가 증가하는 순서로 정렬한다. 만일 짧은 변의 길이가 같은 경우에는 다른 변의 길이가 작은 것이 앞에 오도록 한다. 이렇게 하여 색종이 (a, b)를 색종이 (c, d) 위에 올려놓을 수 있는 한 가지 경우, 즉 a<=c 그리고 b<=d를 만족하는지만 확인하면 된다.

이와 같이 색종이를 정렬하여 순서대로 색종이가 쌓여 있는 경우를 생각하면, 짧은 변의 길이에 대한 조건은 자동적으로 해결된다. 다음으로 긴 변의 길이에 대한 조건을 처리하기 위해서는 긴 변의 길이를 가지고 만들어지는 수열에서 증가하는 가장 긴 부분 수열을 구하면 된다.

긴 변의 길이를 나타내는 수열을 L[1], ..., L[n]이라고 놓자. 그리고 동적 계획법을 위한 배열 D[i]를 '1번부터 i번까지의 색종이를 이용하여 색종이를 쌓아 나가되 i번 색종이를 가장 아래에 놓을 때 최대로 쌓을 수 있는 색종이의 수'로 정의하자. 그러면 자연스럽게 D[1]=1임을 알 수 있고, D[i]를 채우기 위해서는 L[k]<L[i]인 1<=k<i들에 대해 D[k]+1 중 최대값을 선택하면 된다. 이렇게 하면 k의 색종이를 위에 놓는 경우의 조건이 만족되고, 이 때 그 위로는 D[k]만큼의 색종이를 더 쌓을 수 있기 때문이다. 따라서 이와 같이 배열을 채운 후, 우리가 구하고자 하는 답은 D[i]들 중 가장 큰 값이 된다.

```
#include<stdio.h>

#define Swap(aa,bb) {cc=aa;aa=bb;bb=cc;}

int a[100], b[100], D[100], n, cc, ret;

int main(void)
{
	int l1, l2;

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

	scanf("%d",&n);
	for(l1=0;l1<n;l1++)
	{
		scanf("%d %d",&a[l1],&b[l1]);
		if(a[l1] > b[l1]) Swap(a[l1], b[l1]);
	}

	for(l1=0;l1<n;l1++)
		for(l2=l1+1;l2<n;l2++)
			if(a[l1] > a[l2] || (a[l1] == a[l2] && b[l1] > b[l2]))
			{
				Swap(a[l1], a[l2]);
				Swap(b[l1], b[l2]);
			}

	for(l1=0;l1<n;l1++)
	{
		for(l2=0;l2<l1;l2++)
			if(a[l2] <= a[l1] && b[l2] <= b[l1] && D[l1] < D[l2])
				D[l1] = D[l2];
		D[l1]++;
		if(D[l1] > ret) ret = D[l1];
	}

	printf("%d\n",ret);
}
```

수 이어가기

다음과 같은 규칙에 따라 수들을 만들려고 한다.

규칙 1. 첫 번째 수로 양의 정수가 주어진다.

규칙 2. 두 번째 수는 양의 정수 중에서 하나를 선택한다.

규칙 3. 세 번째부터 이후에 나오는 모든 수는 앞의 앞의 수에서 앞의 수를 빼서 만든다. 예를 들어, 세 번째 수는 첫 번째 수에서 두 번째 수를 뺀 것이고, 네 번째 수는 두 번째 수에서 세 번째 수를 뺀 것이다.

규칙 4. 음의 정수가 만들어지면, 이 음의 정수를 버리고 더 이상 수를 만들지 않는다.

첫 번째 수로 100이 주어질 때, 두 번째 수로 60을 선택하여 위의 규칙으로 수들을 만들면 7개의 수들 100, 60, 40, 20, 20, 0, 20이 만들어진다. 그리고 두 번째 수로 62를 선택하여 위의 규칙으로 수들을 만들면 8개의 수들 100, 62, 38, 24, 14, 10, 4, 6이 만들어진다. 위의 예에서 알 수 있듯이, 첫 번째 수가 같더라도 두 번째 수에 따라 만들어지는 수들의 개수가 다를 수 있다.

입력으로 첫 번째 수가 주어질 때, 이 수에서 시작하여 위의 규칙으로 만들어지는 최대 개수의 수들을 구하는 프로그램을 작성하시오. 최대 개수의 수들이 여러 개일 때, 그 중 하나의 수들만 출력하면 된다.

실행파일의 이름은 SEQ.EXE로 하고, 프로그램의 실행시간은 10초를 초과할 수 없다. 여러분들이 구한 수들의 개수에 따라 부분점수가 주어질 수 있다.

입력 형식

입력 파일명은 INPUT.TXT로 한다. 입력 파일에 첫 번째 수가 주어진다. 이 수는 30000보다 같거나 작은 양의 정수이다.

출력 형식

출력 파일명은 OUTPUT.TXT로 한다. 첫째 줄에는 입력된 첫 번째 수로 시작하여 위의 규칙에 따라 만들 수 있는 수들의 최대 개수를 출력한다. 둘 때 줄에 그 최대 개수의 수들을 차례대로 출력한다. 이들 수 사이에는 빈칸을 하나씩 둔다.

입력과 출력의 예

입력(INPUT.TXT)

```
100
```

출력(OUTPUT.TXT)

```
8
100 62 38 24 14 10 4 6
```

풀이

첫 번째 수가 주어지고 수를 만들어나가는 규칙이 정해져 있을 때, 최대 개수의 수들을 만들어내는 방법을 찾는 문제이다. 두 번째 수를 정하는 규칙을 발견하여 풀 수도 있겠지만, 가능한 모든 경우를 고려해 보는 방법으로도 충분히 해결할 수 있는 문제이다.

규칙을 잘 살펴보면 두 번째 수를 무엇으로 할지를 결정했을 때, 그 이후에 만들어지는 수들이 모두 확정됨을 알 수 있다. 즉, 가능한 모든 경우에 대해서 두 번째 수를 결정한 후, 문제에서 주어진 규칙에 의해 세 번째 수, 네 번째 수 등을 순서대로 모두 만들어 본다. 이러한 과정을 가능한 모든 두 번째 수에 대해서 반복하면서, 가장 수가 많이 만들어지는 경우를 찾으면 된다.

실제로 두 번째에 놓일 수 있는 수는 무한히 많다. 그러나 두 번째 수가 첫 번째 수보다 큰 경우에는 세 번째 수가 음의 정수가 되므로 이러한 경우는 고려하지 않아도 된다. 이러한 경우에는 어차피 두 개의 수밖에 만들 수 없으므로, 다른 경우에 이보다 많은 수가 만들어지게 된다.

```
#include<stdio.h>

int prev, now, next, X, max_len, max_val, len;

int main(void)
{
	int l1;

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

	scanf("%d",&X);

	for(l1=1;l1<=X;l1++)
	{
		prev = X; now = l1; len = 2;
		while(1)
		{
			next = prev - now;
			if(next < 0) break
			len++; prev = now; now = next;
		}
		if(len > max_len)
		{
			max_len = len;
			max_val = l1;
		}
	}
	printf("%d\n",max_len);
	prev = X; now = max_val;
	printf("%d %d",prev,now);
	while(1)
	{
		next = prev - now;
		if(next < 0) break
		printf(" %d",next);
		prev = now; now = next;
	}
	printf("^\n");
}
```

스위치 상태

1부터 연속적으로 번호가 붙어있는 스위치들이 있다. 스위치는 켜져 있거나 꺼져있는 상태이다. <그림 1>에 스위치 8개의 상태가 표시되어 있다. '1' 은 스위치가 켜져 있음을, '0' 은 꺼져 있음을 나타낸다. 그리고 학생 몇 명을 뽑아서, 학생들에게 1 이상이고 스위치 개수 이하인 자연수를 하나씩 나누어주었다. 학생들은 자신의 성별과 받은 수에 따라 아래와 같은 방식으로 스위치를 조작하게 된다.

남학생은 스위치 번호가 자기가 받은 수의 배수이면, 그 스위치의 상태를 바꾼다. 즉, 스위치가 켜져 있으면 끄고, 꺼져 있으면 켠다. <그림 1>과 같은 상태에서 남학생이 3을 받았다면, 이 학생은 <그림 2>와 같이 3번, 6번 스위치의 상태를 바꾼다.

여학생은 자기가 받은 수와 같은 번호가 붙은 스위치를 중심으로 좌우가 대칭이면서 가장 많은 스위치를 포함하는 구간을 찾아서, 그 구간에 속한 스위치의 상태를 모두 바꾼다. 이 때 구간에 속한 스위치 개수는 항상 홀수가 된다.

예를 들어 <그림 2>에서 여학생이 3을 받았다면, 3번 스위치를 중심으로 2번, 4번 스위치의 상태가 같고, 1번, 5번 스위치의 상태가 같으므로, <그림 3>과 같이 1번부터 5번까지 스위치의 상태를 모두 바꾼다. 만약 <그림 2>에서 여학생이 4를 받았다면, 3번, 5번 스위치의 상태가 서로 다르므로 4번 스위치의 상태만 바꾼다.

스위치 번호	①	②	③	④	⑤	⑥	⑦	⑧
스위치 상태	0	1	0	1	0	0	0	1

<그림 1>

스위치 번호	①	②	③	④	⑤	⑥	⑦	⑧
스위치 상태	0	1	1	1	0	1	0	1

<그림 2>

스위치 번호	①	②	③	④	⑤	⑥	⑦	⑧
스위치 상태	1	0	0	0	1	1	0	1

<그림 3>

입력으로 스위치들의 처음 상태가 주어지고, 각 학생의 성별과 받은 수가 주어진다. **학생들은 입력되는 순서대로** 자기의 성별과 받은 수에 따라 스위치의 상태를 바꾸었을 때, 스위치들의 마지막 상태를 출력하는 프로그램을 작성하시오.

실행파일의 이름은 SWITCH.EXE로 하고, 프로그램의 실행시간은 10초를 초과할 수 없다.

입력 형식

입력 파일명은 INPUT.TXT로 한다. 첫째 줄에는 스위치 개수가 주어진다. 스위치 개수는 100 이하인 양의 정수이다. 둘째 줄에는 각 스위치의 상태가 주어진다. 켜져 있으면 1, 꺼져있으면 0이라고 표시하고 사이에 빈칸이 하나씩 있다. 셋째 줄에는 학생수가 주어진다. 학생 수는 100이하인 양의 정수이다. 넷째 줄부터 마지막 줄까지 한 줄에 한 학생의 성별, 학생이 받은 수가 주어진다. 남학생은 1로, 여학생은 2로 표시하고, 학생이 받은 수는 스위치 개수 이하

인 양의 정수이다. 학생의 성별과 받은 수 사이에 빈칸이 하나씩 있다.

출력 형식

출력 파일의 이름은 OUTPUT.TXT이다. 스위치의 상태를 1번 스위치에서 시작하여 마지막 스위치까지 한 줄에 20개씩 출력한다. 예를 들어 21번 스위치가 있다면 이 스위치의 상태는 둘째 줄 맨 앞에 출력한다. 켜진 스위치는 1, 꺼진 스위치는 0으로 표시하고, 스위치 상태 사이에 빈칸을 하나씩 둔다.

입력과 출력의 예

입력(INPUT.TXT)

```
8
0 1 0 1 0 0 0 1
2
1 3
2 3
```

출력(OUTPUT.TXT)

```
1 0 0 0 1 1 0 1
```

풀이

스위치의 상태를 바꾸는 방법과 스위치의 상태가 주어졌을 때, 스위치의 최종 상태를 구하는 문제이다. 문제에서 주어진 규칙에 따라서 스위치의 상태가 변해가는 과정을 그대로 시뮬레이션 한 후, 최종적으로 만들어지는 스위치의 상태를 출력하면 된다.

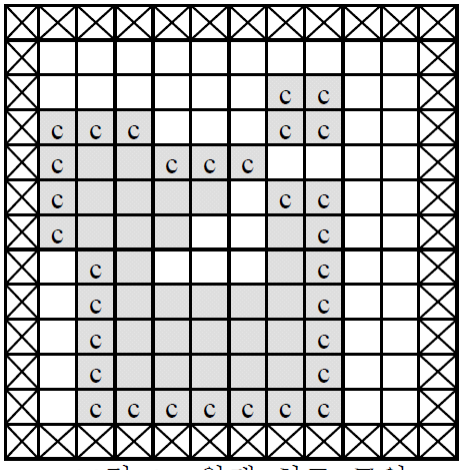
일차원 배열을 이용하여 스위치의 상태를 저장한 후, 문제에서 주어진 방법대로 순서대로 스위치의 상태가 변해가는 과정을 처리해주면 된다. 남학생의 경우에는 배열을 일정 간격으로 뛰어 넘으면서 상태를 바꾸면 된다. 여학생의 경우에는 반복문을 사용하여 좌우 대칭이 어디까지 이어지는지를 확인하고, 확인된 구간의 스위치를 모두 바꾸면 된다. 경계조건에 대하여 주의가 필요한데, 문제에서 주어진 배열의 바깥으로는 넘어가는 일이 없도록 해야 한다.

```
#include<stdio.h>

int a[100], n, m;

int main(void)
{
	int l1, l2, l3, t1, t2;

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

	scanf("%d",&n);
	for(l1=0;l1<n;l1++) scanf("%d",&a[l1]);
	scanf("%d",&m);
	for(l1=0;l1<m;l1++)
	{
		scanf("%d %d",&t1,&t2);
		if(t1 == 1)
		{
			for(l2=t2;l2<=n;l2+=t2)
				a[l2 - 1] = 1 - a[l2 - 1];
		}
		else if(t1 == 2)
		{
			t2--;
			l2 = l3 = t2;
			while(1)
			{
				if(l2 < 0 || l3 >= n) break
				if(a[l2] != a[l3]) break
				l2--; l3++;
			}
			for(l2++;l2<l3;l2++) a[l2] = 1 - a[l2];
		}
	}
	for(l1=0;l1<n;l1++)
	{
		if(l1 > 0)
		{
			if(l1 % 20 == 0) printf("\n");
			else printf(" ");
		}
		printf("%d",a[l1]);
	}
	printf("\n");
}
```

치즈

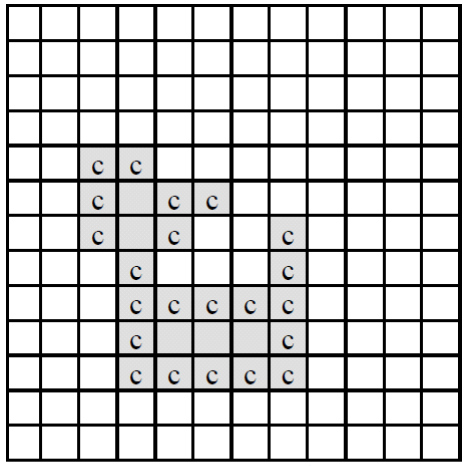
아래 <그림 1>과 같이 정사각형 칸들로 이루어진 사각형 모양의 판이 있고, 그 위에 얇은 치즈(회색으로 표시된 부분)가 놓여 있다. 판의 가장자리(<그림 1>에서 네모 칸에 X친 부분)에는 치즈가 놓여 있지 않으며 치즈에는 하나 이상의 구멍이 있을 수 있다.

이 치즈를 공기 중에 놓으면 녹게 되는데 공기와 접촉된 칸은 한 시간이 지나면 녹아 없어진다. 치즈의 구멍 속에는 공기가 없지만 구멍을 둘러싼 치즈가 녹아서 구멍이 열리면 구멍 속으로 공기가 들어가게 된다. <그림 1>의 경우, 치즈의 구멍을 둘러싼 치즈는 녹지 않고 'c'로 표시된 부분만 한 시간 후에 녹아 없어져서 <그림 2>와 같이 된다.

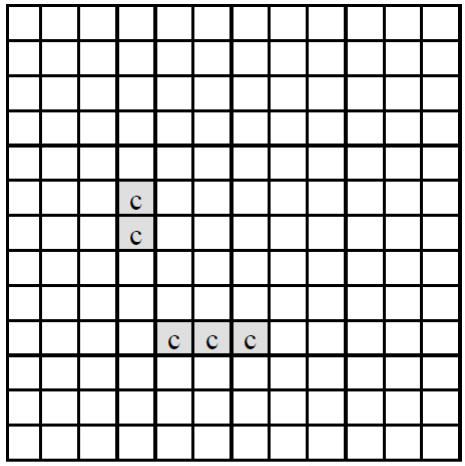


<그림 1> 원래 치즈 모양

다시 한 시간 후에는 <그림 2>에서 'c'로 표시된 부분이 녹아 없어져서 <그림 3>과 같이 된다.



<그림 2> 한 시간 후의 치즈 모양



<그림 3> 두 시간 후의 치즈 모양

<그림 3>은 원래 치즈의 두 시간 후 모양을 나타내고 있으며, 남은 조각들은 한 시간이 더 지나면 모두 녹아 없어진다. 그러므로 처음 치즈가 모두 녹아 없어지는 데는 세 시간이 걸린다. <그림 3>과 같이 치즈가 녹는 과정에서 여러 조각으로 나누어 질 수도 있다.

입력으로 사각형 모양의 판의 크기와 한 조각의 치즈가 판 위에 주어졌을 때, 공기 중에서 치즈가 모두 녹아 없어지는 데 걸리는 시간과 모두 녹기 한 시간 전에 남아있는 치즈조각이 놓여 있

는 칸의 개수를 구하는 프로그램을 작성하시오.

실행파일의 이름은 CHEESE.EXE로 하고, 프로그램의 실행시간은 10초를 초과할 수 없다.

입력 형식

입력 파일명은 INPUT.TXT로 한다. 입력 파일의 첫째 줄에는 사각형 모양 판의 세로와 가로의 길이가 양의 정수로 주어진다. 세로와 가로의 길이는 최대 100이다. 판의 각 가로줄의 모양이 윗줄부터 차례로 입력 파일의 둘째 줄부터 마지막 줄까지 주어진다. 치즈가 없는 칸은 0, 치즈가 있는 칸은 1로 주어지며 각 숫자 사이에는 빈칸이 하나씩 있다.

출력 형식

출력 파일명은 OUTPUT.TXT로 한다. 첫째 줄에는 치즈가 모두 녹아서 없어지는 데 걸리는 시간을 출력하고, 둘째 줄에는 모두 녹기 한 시간 전에 남아있는 치즈조각이 놓여 있는 칸의 개수를 출력한다.

입력과 출력의 예

입력(INPUT.TXT)

```
13 12
0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 1 0 0 0
0 1 1 1 0 0 0 1 1 0 0 0
0 1 1 1 1 1 1 0 0 0 0 0
0 1 1 1 1 1 0 1 1 0 0 0
0 1 1 1 1 0 0 1 1 0 0 0
0 0 1 1 0 0 0 1 1 0 0 0
0 0 1 1 1 1 1 1 1 0 0 0
0 0 1 1 1 1 1 1 1 0 0 0
0 0 1 1 1 1 1 1 1 0 0 0
0 0 1 1 1 1 1 1 1 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
```

출력(OUTPUT.TXT)

```
3
5
```

풀이

치즈가 녹는 규칙과 치즈의 초기 상태가 주어졌을 때, 치즈의 최종 상태와 이에 걸리는 시간을 출력하는 문제이다. 한 번에 치즈의 최종 상태를 알 수는 없으므로, 시간이 진행되면서 치즈가 어떻게 녹는지를 시뮬레이션 하는 방법으로 풀 수 있다. 즉, 매 시간마다 치즈의 상태가 변해가는 모습을 기록한 후, 최종 상태의 모양과 그 때까지 걸린 시간을 답으로 출력하면 된다.

시뮬레이션을 하는 과정에서 깊이우선탐색(DFS) 또는 너비우선탐색(BFS)을 이용하여 바깥쪽의 공기와 치즈의 구멍을 구분하는 것이 이 문제의 핵심이다. 각 시간마다의 상태는 2차원 배열 형태로 표현되는데, 입력으로 주어진 판의 가장 가장자리의 칸들이 항상 비어 있으므로 이 부분을 모두 공기로 볼 수 있다. 먼저 DFS 또는 BFS를 이용해서 공기와 인접하면서 치즈가 없는 칸들을 쭉 탐색해 나간다. 그러면 이처럼 이어진 모든 칸들이 공기에 해당된다. 이와 같이 공기에 해당하는 모든 칸들을 파악한 후에는 공기와 인접해 있는 치즈들을 모두 지운다.

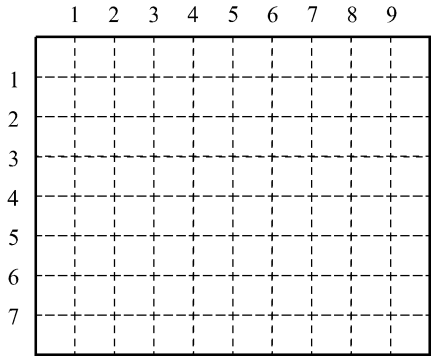
위와 같은 과정을 모든 치즈가 없어질 때까지 계속 반복하면 된다. 매 상태를 구할 때마다 치즈가 남아있는 칸의 개수를 센 후, 이 값이 0일 경우에 시뮬레이션을 종료한다. 0이 아닐 경우에는 따로 기록을 하고, 마지막으로 기록되었던 값을 답으로 출력한다. 또한 실제 프로그래밍을 할 때에는 모든 시간마다 상태를 저장할 필요는 없고, 이전 상태와 그 다음 상태를 저장하는 배열 두 개만 사용해도 된다.

```
#include<stdio.h>

int a[100][100], n, m;
int qx[10000], qy[10000], head, tail, c[100][100];
int dx[4] = {-1, 1, 0, 0};
int dy[4] = { 0, 0,-1, 1};
int ret;

int main(void)
{
	int l0, l1, l2, l3, t1, t2;

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

	scanf("%d %d",&n,&m);
	for(l1=0;l1<n;l1++) for(l2=0;l2<m;l2++) scanf("%d",&a[l1][l2]);

	for(l0=0;;l0++)
	{
		for(l1=0;l1<n;l1++) for(l2=0;l2<m;l2++) c[l1][l2] = 0;
		qx[0] = qy[0] = 0;
		c[0][0] = 1;
		head = 0; tail = 1;
		while(head < tail)
		{
			l1 = qx[head]; l2 = qy[head];
			head++;
			for(l3=0;l3<4;l3++)
			{
				t1 = l1 + dx[l3];
				t2 = l2 + dy[l3];
				if(t1 >= 0 && t2 >= 0 && t1 < n && t2 < m && c[t1][t2] == 0
&& a[t1][t2] == 0)
				{
					c[t1][t2] = 1;
					qx[tail] = t1; qy[tail] = t2;
					tail++;
				}
			}
		}
		if(tail == n*m) break
		ret = 0;
		for(l1=0;l1<n;l1++) for(l2=0;l2<m;l2++)
			if(a[l1][l2] == 1)
			{
				for(l3=0;l3<4;l3++)
				{
					t1 = l1 + dx[l3];
					t2 = l2 + dy[l3];
					if(t1 >= 0 && t2 >= 0 && t1 < n && t2 < m &&
```

```
c[t1][t2] == 1)
                                                    break
                                        }
                                        if(l3 < 4)
                                        {
                                                a[l1][l2] = 0;
                                                ret++;
                                        }
                                }
        }

        printf("%d\n%d\n",l0,ret);
}
```

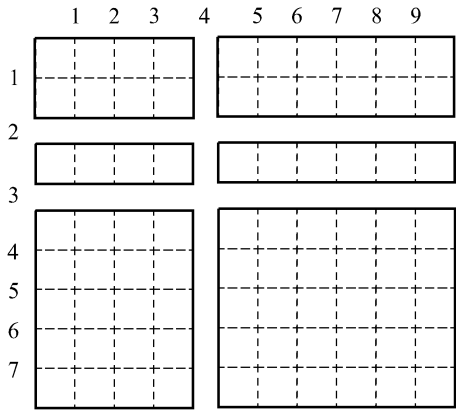
종이 자르기

아래 <그림 1>과 같이 직사각형 모양의 종이가 있다. 이 종이는 가로 방향과 세로 방향으로 1㎝마다 점선이 그어져 있다. 가로 점선은 위에서 아래로 1번부터 차례로 번호가 붙어 있고, 세로 점선은 왼쪽에서 오른쪽으로 번호가 붙어 있다.



<그림 1>

점선을 따라 이 종이를 칼로 자르려고 한다. 가로 점선을 따라 자르는 경우는 종이의 왼쪽 끝에서 오른쪽 끝까지, 세로 점선인 경우는 위쪽 끝에서 아래쪽 끝까지 한 번에 자른다. 예를 들어, <그림 1>의 가로 길이 10㎝이고 세로 길이 8㎝인 종이를 3번 가로 점선, 4번 세로 점선, 그리고 2번 가로 점선을 따라 자르면 <그림 2>와 같이 여러 개의 종잇조각으로 나뉘게 된다. 이 때 가장 큰 종잇조각의 넓이는 30㎠이다.



<그림 2>

입력으로 종이의 가로와 세로 길이, 그리고 잘라야할 점선들이 주어질 때, 가장 큰 종잇조각의 넓이가 몇 ㎠인지를 구하는 프로그램을 작성하시오.

실행파일의 이름은 AA.EXE로 하고, 프로그램의 실행시간은 5초를 넘을 수 없다. 부분점수는 없다.

입력 형식

입력 파일명은 INPUT.TXT로 한다. 입력 파일의 첫째 줄에는 종이의 가로와 세로의 길이가 차례로 자연수로 주어진다. 가로와 세로의 길이는 최대 100㎝이다. 둘째 줄에는 칼로 잘라야하는 점선의 개수가 주어진다. 입력 파일의 셋째 줄부터 마지막 줄까지 한 줄에 점선이 하나씩 아래와 같은 방법으로 입력된다. 가로로 자르는 점선은 0과 점선 번호가 차례로 주어지고, 세로로 자르는 점선은 1과 점선 번호가 주어진다. 입력되는 두 숫자 사이에는 빈칸이 하나씩 있다.

출력 형식

출력 파일명은 OUTPUT.TXT로 한다. 첫째 줄에 가장 큰 종이 조각의 넓

이를 출력한다. 단, 넓이의 단위는 출력하지 않는다.

입력과 출력의 예

입력(INPUT.TXT)

```
10 8
3
0 3
1 4
0 2
```

출력(OUTPUT.TXT)

```
30
```

풀이

종이를 자르는 규칙이 주어졌을 때, 가장 큰 종이의 크기를 구하는 문제이다. 가로와 세로 방향 모두 같은 방식으로 종이를 자르므로, 각 방향에서 점선 사이의 길이가 제일 긴 경우를 조합할 때가 가장 커짐을 알 수 있다. 따라서 각 방향에 대해서 따로 처리를 한 후, 두 경우의 답을 곱하면 그것이 최대의 면적이 된다.

먼저 주어진 종이를 가로 방향으로 자르는 점선을 정렬하고, 이웃한 점선들 사이의 가운데 거리가 가장 큰 구간을 찾는다. 이 때 구현상의 편의를 위해서 맨 위와 맨 아래에도 점선이 있다고 가정하고 정렬한다. 마찬가지로 세로 방향 점선에 대해서도 이웃한 점선들 사이 가운데 거리가 가장 큰 구간을 찾는다. 이 경우에는 맨 왼쪽과 맨 오른쪽에도 점선이 있다고 생각해 주어야 한다. 이제 각각의 경우에서 구한 두 개의 최장 구간의 길이를 곱해서 답으로 출력한다.

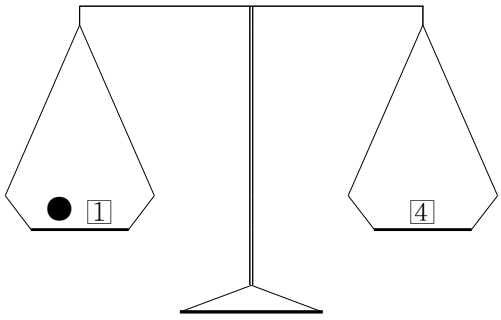
```
#include<stdio.h>

int a[101], b[101], n, m, k;

int Go(int a[], int n)
{
	int l1, t1 = 0, ret = 0;
	for(l1=0;l1<=n;l1++) if(a[l1])
	{
		if(ret < l1 - t1) ret = l1 - t1;
		t1 = l1;
	}
	return ret;
}

int main(void)
{
	int l1, t1, t2, v1 = 0, v2 = 0;
	freopen("input.txt","r",stdin);
	freopen("output.txt","w",stdout);

	scanf("%d %d",&m,&n);
	scanf("%d",&k);
	a[0] = a[n] = b[0] = b[m] = 1;
	for(l1=0;l1<k;l1++)
	{
		scanf("%d %d",&t1,&t2);
		if(t1 == 0) a[t2] = 1;
		else if(t1 == 1) b[t2] = 1;
	}
	printf("%d\n",Go(a, n) * Go(b, m));
}
```

양팔 저울

양팔 저울과 몇 개의 추가 주어졌을 때, 이를 이용하여 입력으로 주어진 구슬의 무게를 확인할 수 있는지를 결정하려고 한다.

무게가 각각 1g과 4g인 두 개의 추가 있을 경우, 주어진 구슬과 1g 추 하나를 양팔 저울의 양쪽에 각각 올려놓아 수평을 이루면 구슬의 무게는 1g이다. 또 다른 구슬이 4g인지를 확인하려면 1g 추 대신 4g 추를 올려놓으면 된다.

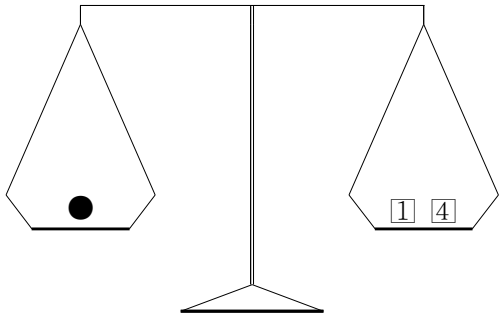
구슬이 3g인 경우 아래 <그림 1>과 같이 구슬과 추를 올려놓으면 양팔 저울이 수평을 이루게 된다. 따라서 각각 1g과 4g인 추가 하나씩 있을 경우 주어진 구슬이 3g인지도 확인해 볼 수 있다.



<그림 20> 구슬이 3g인지 확인하는 방법
(1은 1g인 추, 4는 4g인 추,
●은 무게를 확인할 구슬)

<그림 2>와 같은 방법을 사용하면 구슬이 5g인지도 확인할 수 있다. 구슬이 2g이면 주어진 추를 가지고는 확인할 수 없다.

추들의 무게와 확인할 구슬들의 무게가 입력되었을 때, 주어진 추만을 사용하여 구슬의 무게를 확인할 수 있는지를 결정하는 프로그램을 작성하시오.



<그림 21> 구슬이 5g인지 확인하는 방법

실행파일의 이름은 BB.EXE로 하고, 프로그램의 실행시간은 5초를 넘을 수 없다. 부분점수는 없다.

입력 형식

입력 파일명은 INPUT.TXT로 한다. 입력 파일의 첫째 줄에는 추의 개수가 자연수로 주어진다. 추의 개수는 30 이하이다. 둘째 줄에는 추의 무게들이 자연수로 가벼운 것부터 차례로 주어진다. 같은 무게의 추가 여러 개 있을 수도 있다. 추의 무게는 500g이하이며, 입력되는 무게들 사이에는 빈칸이 하나씩 있다. 세 번째 줄에는 무게를 확인하고자 하는 구슬들의 개수가 주어진다. 확인할 구슬의 개수는 7이하이다. 네 번째 줄에는 확인하고자 하는 구슬들의 무게가 자연수로 주어지며, 입력되는 무게들 사이에는 빈칸이 하나씩 있다.

출력 형식

출력 파일명은 OUTPUT.TXT로 한다. 주어진 각 구슬의 무게에 대하여 확인이 가능하면 Y, 아니면 N을 차례로

출력한다. 출력 파일은 한 개의 줄로 이루어지며, 각 구슬에 대한 답 사이에는 빈칸을 하나씩 둔다.

입력과 출력의 예(1)

입력(INPUT.TXT)

```
2
1 4
2
3 2
```

출력(OUTPUT.TXT)

```
Y N
```

입력과 출력의 예(2)

입력(INPUT.TXT)

```
4
2 3 3 3
3
1 4 10
```

출력(OUTPUT.TXT)

```
Y Y N
```

풀이

구슬과 추에 대한 정보가 주어졌을 때, 구슬의 무게를 확인할 수 있는지를 알아내는 문제이다. 먼저, 무게를 잴 구슬을 양팔저울의 한쪽에 올린다. 구슬을 놓은 쪽을 A, 다른 쪽을 B라고 하자. 이제 각각의 추는 다음의 세 가지 중 하나로 이용할 수 있다. (1) 이 추를 A에 올리는 경우 (2) 이 추를 B에 올리는 경우 (3) 이 추를 사용하지 않을 경우. 따라서 각 추에 대해서 이 중 하나를 결정하여 모든 가능성에 대해 검사해 보면서 평형을 이루는 경우가 있는지를 보면 된다. 이 때 추의 개수를 n개라고 하면 모든 가능한 경우가 총 3^n가지나 생기므로, 적절한 가지치기를 추가해야 한다.

가지치기에는 다양한 방법이 있을 수 있다. 예를 들어, 지금까지 이용하고 남은 추들 전부를 A나 B 중 가벼운 쪽에 모두 올려놓더라도 평형을 이룰 수 없으면 더 이상 탐색을 하지 않아도 된다. 즉, 이처럼 앞으로 계속 탐색을 수행하더라도 답에 이를 수 없음을 미리 확인할 수 있다면 이후의 탐색은 수행하지 않아도 된다.

당시 대회 환경에서는 배열 크기의 제한 때문에 사용하기 힘들지만, 현재 대회 환경에서는 동적 계획법을 이용한 풀이도 가능하다. 어떤 추의 무게가 W라고 하면, 이 추를 A에 올리면 W가 더해지고, B에 올리면 W가 빼지는 것으로 이해할 수 있다. 이러한 아이디어를 이용하여 주어진 추들로 확인할 수 있는 모든 구슬의 무게를 구하고, 각 구슬의 무게가 이에 속하는지 판단하여 답을 낼 수도 있다.

```
#include<stdio.h>

int a[33], b[33], sum[33], gcd[33], n, m, S, flag;

int gcd_(int x, int y)
{
	if(x > y) return gcd_(y, x);
	if(x == 0) return y;
	return gcd_(y%x, x);
}

void Go(int Dep, int curr)
{
	if(curr == S)
	{
		flag = 1;
		return
	}
	if((flag == 0) && (Dep <= n) && (curr - sum[Dep] <= S)
		&& (S <= curr + sum[Dep]) && ((S - curr) % gcd[Dep] == 0))
	{
		if(a[Dep] != a[Dep - 1] || b[Dep - 1] == 1)
		{
			b[Dep] = 1; Go(Dep+1, curr + a[Dep]);
		}
		b[Dep] = 2; Go(Dep+1, curr);
		if(a[Dep] != a[Dep - 1] || b[Dep - 1] == 3)
		{
			b[Dep] = 3; Go(Dep+1, curr - a[Dep]);
		}
	}
}

int main(void)
{
	int l1;

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

	scanf("%d",&n);
	for(l1=n;l1>=1;l1--)
	{
		scanf("%d",&a[l1]);
		sum[l1] = sum[l1+1] + a[l1];
		if(l1 == n)
			gcd[n] = a[n];
		else if(l1 < n)
			gcd[l1] = gcd_(gcd[l1+1], a[l1]);
	}
```

```
        scanf("%d",&m);
        for(l1=0;l1<m;l1++)
        {
                scanf("%d",&S);
                flag = 0;
                Go(1, 0);

                if(l1) printf(" ");

                if(flag) printf("Y");
                else printf("N");
        }
        printf("\n");
}
```

카드 게임

운재는 오늘 재미있는 카드 게임을 배우고 있다. 카드는 빨간색, 파란색, 노란색, 녹색의 네 가지 색이 있고, 색깔별로 1부터 9까지 숫자가 쓰여진 카드가 9장씩 있다. 카드는 모두 36(=4×9)장이다. 운재가 배운 카드 게임은 36장의 카드에서 5장을 뽑고, 아래와 같은 규칙으로 점수를 계산하는 것이다.

각 카드는 다음과 같이 나타낸다. 카드의 색깔은 영어 대문자 R, B, Y, G로 나타내는데, R은 빨간색, B는 파란색, Y는 노란색, G는 녹색을 뜻한다. 예를 들어서 Y8은 노란색 8을 나타내고, B5는 파란색 5를 나타낸다.

<점수를 정하는 규칙>

① 카드 5장이 모두 같은 색이면서 숫자가 연속적일 때, 점수는 가장 높은 숫자에 900을 더한다. 예를 들어, 카드가 Y4, Y3, Y2, Y5, Y6일 때 점수는 906(=6+900)점이다.

② 카드 5장 중 4장의 숫자가 같을 때 점수는 같은 숫자에 800을 더한다. 예를 들어, 카드가 B3, R3, B7, Y3, G3일 때 점수는 803(=3+800)점이다.

③ 카드 5장 중 3장의 숫자가 같고 나머지 2장도 숫자가 같을 때 점수는 3장이 같은 숫자에 10을 곱하고 2장이 같은 숫자를 더한 다음 700을 더한다. 예를 들어, 카드가 R5, Y5, G7, B5, Y7일 때 점수는 757(=5×10+7+700)점이다.

④ 5장의 카드 색깔이 모두 같을 때 점수는 가장 높은 숫자에 600을 더한다. 예를 들어, 카드가 Y3, Y4, Y8, Y6, Y7일 때 점수는 608(=8+600)점이다.

⑤ 카드 5장의 숫자가 연속적일 때 점수는 가장 높은 숫자에 500을 더한다. 예를 들어, 카드가 R7, R8, G9, Y6, B5일 때 점수는 509(=9+500)점이다.

⑥ 카드 5장 중 3장의 숫자가 같을 때 점수는 같은 숫자에 400을 더한다. 예를 들어 R7, Y7, R2, G7, R5 일 때 점수는 407(=7+400)점이다.

⑦ 카드 5장 중 2장의 숫자가 같고 또 다른 2장의 숫자가 같을 때 점수는 같은 숫자 중 큰 숫자에 10을 곱하고 같은 숫자 중 작은 숫자를 더한 다음 300을 더한다. 예를 들어, R5, Y5, Y4, G9, B4일 때 점수는 354(=5×10+4+300)점이다.

⑧ 카드 5장 중 2장의 숫자가 같을 때 점수는 같은 숫자에 200을 더한다. 예를 들어, R5, Y2, B5, B3, G4일 때 점수는 205(=5+200)점이다.

⑨ 위의 어떤 경우에도 해당하지 않을 때 점수는 가장 큰 숫자에 100을 더한다. 예를 들어, R1, R2, B4, B8, Y5일 때 점수는 108(=8+100)점이다.

입력으로 카드 5장이 주어질 때, 카드 게임의 점수를 구하는 프로그램을 작성하시오. 두 가지 이상의 규칙을 적용할 수 있는 경우에는 가장 높은 점수가 카드 게임의 점수이다.

실행파일의 이름은 AA.EXE로 하고, 프로그램의 실행시간은 5초를 넘을 수 없다.

입력 형식

입력 파일의 이름은 INPUT.TXT로 한다. 입력 파일의 첫째 줄부터 다섯째

줄까지 한 줄에 카드 하나씩 입력된다. 카드의 색깔과 숫자 사이에는 빈 칸이 하나 있다.

출력 형식

출력 파일의 이름은 OUTPUT.TXT로 한다. 한 줄에 입력으로 주어진 카드의 점수를 출력한다.

입력과 출력의 예

입력(INPUT.TXT)

```
B 3
B 7
R 1
B 2
Y 7
```

출력(OUTPUT.TXT)

```
207
```

풀이

카드의 점수를 계산하는 다양한 규칙이 주어졌을 때, 주어진 카드로 얻을 수 있는 점수를 구하는 문제이다. 별다른 풀이가 있는 문제는 아니며, 각각의 규칙을 모두 적용해 보면서 점수를 계산하는 식으로 해결해야 한다. 주어진 규칙을 차례로 적용해 보면서, 해당하는 경우에는 그 때의 점수가 몇 점인지 계산한다. 이를 쭉 반복하면서, 얻을 수 있는 점수 중 가장 높은 점수를 답으로 출력한다. 여러 가지 규칙의 종류가 있으므로, 꼼꼼하게 프로그래밍 하는 것이 중요하다.

규칙을 확인할 때에는 크게 두 가지의 경우를 고려하게 되는데, 하나는 색깔이나 숫자의 일치 여부를 판단하는 것이고, 다른 하나는 숫자의 연속성을 판단하는 것이다. 일치 여부를 판단하는 것은 순서에 상관없이 수행할 수 있지만, 연속성을 판단할 때에는 정렬을 이용하면 편하게 처리할 수 있다. 즉, 각각의 규칙을 확인할 때마다 카드를 숫자 순서대로 정렬하고, 이를 바탕으로 연속성을 확인하면 쉽게 해결할 수 있다.

```
#include<stdio.h>

#define Swap(aa,bb) {cc=aa;aa=bb;bb=cc;}

int a[5], b[5], ret, cc;

void Read(int l1)
{
	char s[2];
	scanf("%s",s);
	if(s[0] == 'R') a[l1] = 1;
	else if(s[0] == 'B') a[l1] = 2;
	else if(s[0] == 'Y') a[l1] = 3;
	else if(s[0] == 'G') a[l1] = 4;
	scanf("%d",&b[l1]);
}

void SortA(void)
{
	int l1, l2;
	for(l1=0;l1<5;l1++) for(l2=l1+1;l2<5;l2++)
		if(a[l1] > a[l2] || (a[l1] == a[l2] && b[l1] > b[l2]))
		{
			Swap(a[l1], a[l2]);
			Swap(b[l1], b[l2]);
		}
}

void SortB(void)
{
	int l1, l2;
	for(l1=0;l1<5;l1++) for(l2=l1+1;l2<5;l2++)
		if(b[l1] > b[l2] || (b[l1] == b[l2] && a[l1] > a[l2]))
		{
			Swap(a[l1], a[l2]);
			Swap(b[l1], b[l2]);
		}
}

void Try(int score)
{
	if(score > ret) ret = score;
}

int Flush(void)
{
	SortA();
	if(a[0] == a[1] && a[0] == a[2] && a[0] == a[3] && a[0] == a[4]) return b[4];
	return 0;
}
```

```
int Straight(void)
{
	SortB();
	if(b[0]+1 == b[1] && b[0]+2 == b[2] && b[0]+3 == b[3] && b[0]+4 == b[4]) return b[4];
	return 0;
}

int Four(void)
{
	SortB();
	if(b[1] == b[2] && b[1] == b[3] && b[1] == b[4]) return b[1];
	if(b[0] == b[1] && b[0] == b[2] && b[0] == b[3]) return b[0];
	return 0;
}

int Three(void)
{
	SortB();
	if(b[2] == b[3] && b[2] == b[4]) return b[2];
	if(b[1] == b[2] && b[1] == b[3]) return b[1];
	if(b[0] == b[1] && b[0] == b[2]) return b[0];
	return 0;
}

int Two(void)
{
	SortB();
	if(b[3] == b[4]) return b[3];
	if(b[2] == b[3]) return b[2];
	if(b[1] == b[2]) return b[1];
	if(b[0] == b[1]) return b[0];
	return 0;
}

int Two2(void)
{
	SortB();
	if(b[0] == b[1]) return b[0];
	if(b[1] == b[2]) return b[1];
	if(b[2] == b[3]) return b[2];
	if(b[3] == b[4]) return b[3];
	return 0;
}

int main(void)
{
	int l1,l0;

	freopen("input.txt","r",stdin);
	freopen("output.txt","w",stdout);
```

```
        for(l1=0;l1<5;l1++) Read(l1);

        if(Flush() && Straight()) Try(900 + Flush());
        if(Four()) Try(800 + Four());
        if(Three() && Two() && Three() != Two()) Try(700 + 10*Three() + Two());
        if(Three() && Two2() && Three() != Two2()) Try(700 + 10*Three() + Two2());
        if(Flush()) Try(600 + Flush());
        if(Straight()) Try(500 + Straight());
        if(Three()) Try(400 + Three());
        if(Two() && Two2() && Two() != Two2()) Try(300 + 10*Two() + Two2());
        if(Two()) Try(200 + Two());
        for(l1=0;l1<5;l1++) Try(100 + b[l1]);

        printf("%d\n",ret);
}
```

삼각형 만들기

같은 길이의 성냥개비가 여러 개 주어져 있다. 이것들을 평면에 늘어놓아서 삼각형을 만들려고 한다. 삼각형의 한 변은 여러 개의 성냥개비를 직선으로 이어서 만들 수 있지만, 성냥개비를 꺾거나 잘라서 변의 한 부분을 만들 수는 없다. 성냥개비의 개수가 주어졌을 때, 이들 성냥개비를 사용하여 만들 수 있는 서로 다른 삼각형의 개수를 구하는 프로그램을 작성하시오.

예를 들어서 9개의 성냥개비로 만들 수 있는 서로 다른 삼각형은 그림 1과 같이 3 가지이다.

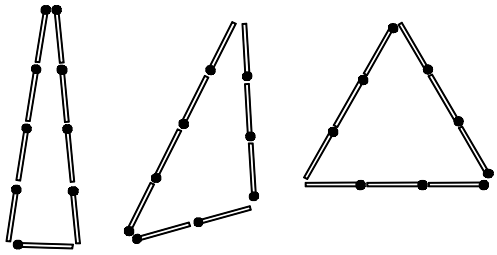


그림 1

주의사항

① 주어진 성냥개비는 모두 사용하여 하나의 삼각형을 만들어야 한다.

② 삼각형을 한 개도 만들 수 없으면 0을 출력한다. 예를 들어서, 주어진 성냥개비의 개수가 1, 2, 또는 4인 경우에는 삼각형을 한 개도 만들 수 없다.

③ 합동인 삼각형들은 같은 삼각형으로 본다. 예를 들어서 성냥개비 5개를 사용하여 만들 수 있는 그림 2의 삼각형들은 같은 삼각형으로 본다.

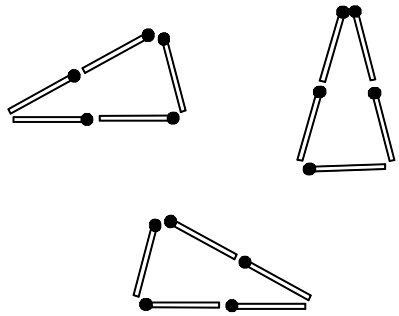


그림 2

실행파일의 이름은 BB.EXE로 하고, 프로그램의 실행시간은 2초를 넘을 수 없다.

입력 형식

입력 파일의 이름은 INPUT.TXT로 한다. 입력 파일의 첫째 줄에 성냥개비의 개수가 주어진다. 성냥개비의 개수는 1 이상 50,000 이하이다.

출력 형식

출력 파일의 이름은 OUTPUT.TXT로 한다. 첫째 줄에 만들 수 있는 삼각형의 개수를 출력한다.

입력과 출력의 예

입력(INPUT.TXT)

```
9
```

출력(OUTPUT.TXT)

```
3
```

풀이

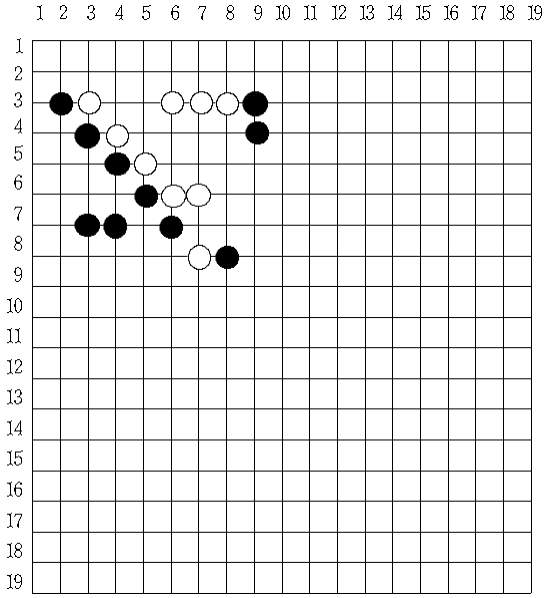
n개의 성냥개비를 가지고 만들 수 있는 서로 다른 모양의 삼각형 개수를 계산하는 문제이다. 먼저 삼각형의 변에 길이에 대한 조건을 생각해 보면, 가장 짧은 변의 길이가 될 수 있는 값은 1 이상 n/3 이하라는 것을 알 수 있다. 따라서 우선 가장 짧은 변의 길이를 i로 결정한다. 이제 가장 긴 변의 길이를 j로 두면, j가 될 수 있는 값의 종류 수가 곧 이 경우에 만들 수 있는 삼각형의 개수가 됨을 알 수 있다.

가장 짧은 변을 제외한 성냥개비의 개수가 n-i이므로 j는 (n-i)/2보다 크거나 같아야 함을 알 수 있다. 또한 j가 다른 두 변의 길이인 i나 n-i-j 보다도 모두 크거나 같아야 함도 알 수 있다. 이 두 가지 조건을 만족하는 j의 개수는 대소비교를 통해 구할 수 있다. 따라서 이러한 과정을 모든 i에 대해 반복해서 구한 뒤, 각각의 경우의 수를 전체적으로 누적해서 더한 값을 출력하면 된다.

```
#include<stdio.h>

int n, ret;

int max(int x, int y)
{
	if(x > y) return x;
	return y;
}

int main(void)
{
	int i, jmax, jmin;

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

	scanf("%d",&n);

	for(i=1;i<=n/3;i++)
	{
		jmax = n-i;
		jmin = max(2*i, n-2*i+1);
		ret += jmax/2 - (jmin+1)/2 + 1;
	}
	printf("%d\n",ret);
}
```

오목

오목은 바둑판에 검은 바둑알과 흰 바둑알을 교대로 놓아서 겨루는 게임이다. 바둑판에는 19개의 가로줄과 19개의 세로줄이 그려져 있는데 가로줄은 위에서부터 아래로 1번, 2번, …, 19번의 번호가 붙고 세로줄은 왼쪽에서부터 오른쪽으로 1번, 2번, …, 19번의 번호가 붙는다.



위의 그림에서와 같이 같은 색의 바둑알이 연속적으로 다섯 알이 놓이면 그 색이 이기게 된다. 여기서 연속적이란 가로, 세로 또는 대각선 방향 모두를 뜻한다. 즉, 위의 그림은 검은색이 이긴 경우이다. 하지만 여섯 알 이상이 연속적으로 놓인 경우에는 이긴 것이 아니다.

입력으로 바둑판의 어떤 상태가 주어졌을 때, 검은색이 이겼는지, 흰색이 이겼는지 또는 아직 승부가 결정되지 않았는지를 판단하는 프로그램을 작성하시오. 단, 검은색과 흰색이 동시에 이기거나 검은색 또는 흰색이 두 군데 이상에서 동시에 이기는 경우는 입력으로 들어오지 않는다.

실행 파일의 이름은 AA.EXE로 하고, 프로그램의 실행시간은 1초를 넘을 수 없다. 부분 점수는 없다.

입력 형식

입력 파일의 이름은 INPUT.TXT로 한다. 입력 파일은 19줄에 각 줄마다 19개의 숫자로 표현되는데, 검은 바둑알은 1, 흰 바둑알은 2, 알이 놓이지 않은 자리는 0으로 표시되며, 숫자는 한 칸씩 띄어서 표시된다.

출력 형식

출력 파일의 이름은 OUTPUT.TXT로 한다. 첫줄에 검은색이 이겼을 경우에는 1을, 흰색이 이겼을 경우에는 2를, 아직 승부가 결정되지 않았을 경우에는 0을 출력한다. 검은색 또는 흰색이 이겼을 경우에는 둘째 줄에 연속된 다섯 개의 바둑알 중에서 가장 왼쪽에 있는 바둑알(연속된 다섯 개의 바둑알이 세로로 놓인 경우, 그중 가장 위에 있는 것)의 가로줄 번호와 세로줄 번호를 순서대로 출력한다.

입력과 출력의 예

입력(INPUT.TXT)

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 2 0 0 2 2 2 1 0 0 0 0 0 0 0 0 0 0
0 0 1 2 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
0 0 0 1 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 2 2 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 2 1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

출력(OUTPUT.TXT)

```
1
3 2
```

풀이

바둑판의 상태가 주어졌을 때, 오목의 승자가 누구인지 판단하는 문제이다. 바둑판을 2차원 배열 형태로 생각하고, 이러한 배열에서 오목의 패턴이 나타나는지 확인하는 식으로 처리할 수 있다.

주어진 2차원 배열을 죽 살펴보면서, '1' 또는 '2'가 가로, 세로, 대각선 방향으로 다섯 번 연속하는 경우를 찾으면 된다. 이 때 문제에서 주어진 조건에 따라 여섯 번 이상 연속하는 경우는 제외해야 한다. 즉, 정확히 다섯 번만 연속하여 나타나는 경우를 찾아서 확인하면 된다. 이를 확인하는 것은 여섯 번의 비교로 해결할 수도 있고, 반복문을 사용할 수도 있다. 어느 방법을 사용하든 배열의 범위를 벗어나는 경우에 주의해야 한다. 구현상의 편의를 위해서 배열의 바깥 테두리에 다른 값을 저장하는 방식으로 처리할 수도 있다.

```
#include <iostream>
#include <fstream>

using namespace std;

class Omok
{
	private:
		int board[19][19];
		int result;
		int sx, sy;
		int ex, ey;
	public:
		Omok();
		void process();
		void output();
		int getBoard(int x, int y);
};

Omok::Omok()
{
	ifstream inf("input.txt");
	int i,j;

	for(i=0;i<19;i++)
		for(j=0;j<19;j++)
			inf >> board[i][j];

	inf.close();
	result = 0;
}

int
Omok::getBoard(int x, int y)
{
	if (x<0) return 0;
	if (y<0) return 0;
	if (x>=19) return 0;
	if (y>=19) return 0;
	return board[y][x];
}

void
Omok::process()
{
	const int direction[4][2] =
	{
		{ 1, 0},
		{ 0, 1},
		{ 1, 1},
		{ 1,-1}
```

```
        };
        int x,y,dir;
        int currentColor;
        for(dir=0;dir<4;dir++)
                for(y=0;y<19;y++)
                        for(x=0;x<19;x++)
                                if ((currentColor = getBoard( x, y )) &&
                                        getBoard( x, y ) !=
                                        getBoard( x - direction[dir][0], y - direction[dir][1] ))
                                {
                                        int count = 0;
                                        int xh=x, yh=y;
                                        while(getBoard(xh,yh)==currentColor)
                                        {
                                                xh+=direction[dir][0];
                                                yh+=direction[dir][1];
                                                count++;
                                        }
                                        if (count==5)
                                        {
                                                result = currentColor;
                                                sx = x;
                                                sy = y;
                                                ex = xh;
                                                ey = yh;
                                        }
                                }
}

void
Omok::output()
{
        ofstream ouf("output.txt");
        ouf << result << endl;
        if (result)
        {
                ouf << sy+1 << ' ' << sx+1 << endl;
        }
        ouf.close();
}

int main()
{
        Omok b;
        b.process();
        b.output();
        return 0;
}
```

주사위 쌓기

천수는 여러 종류의 주사위를 가지고 쌓기 놀이를 하고 있다. 주사위의 모양은 모두 크기가 같은 정육면체이며 각 면에는 1 부터 6 까지의 숫자가 하나씩 적혀있다. 그러나 보통 주사위처럼 마주보는 면에 적혀진 숫자의 합이 반드시 7이 되는 것은 아니다.

주사위 쌓기 놀이는 아래에서부터 1번 주사위, 2번 주사위, 3번 주사위, … 의 순서로 쌓는 것이다. 쌓을 때 다음과 같은 규칙을 지켜야 한다: 서로 붙어 있는 두 개의 주사위에서 아래에 있는 주사위의 윗면에 적혀있는 숫자는 위에 있는 주사위의 아랫면에 적혀있는 숫자와 같아야 한다. 다시 말해서, 1번 주사위 윗면의 숫자는 2번 주사위 아랫면의 숫자와 같고, 2번 주사위 윗면의 숫자는 3번 주사위 아랫면의 숫자와 같아야 한다. 단, 1번 주사위는 마음대로 놓을 수 있다.

이렇게 쌓아 놓으면 긴 사각 기둥이 된다. 이 사각 기둥에는 4개의 긴 옆면이 있다. 이 4개의 옆면 중에서 어느 한 면의 숫자의 합이 최대가 되도록 주사위를 쌓고자 한다. 이렇게 하기 위하여 각 주사위를 위 아래를 고정한 채 옆으로 90도, 180도, 또는 270도 돌릴 수 있다. 한 옆면의 숫자의 합의 최대값을 구하는 프로그램을 작성하시오.

실행 파일의 이름은 BB.EXE로 하고 프로그램의 실행 시간은 1초를 넘을 수 없다. 부분점수는 없다.

입력 형식

입력 파일의 이름은 INPUT.TXT로 한다. 첫줄에는 주사위의 개수가 입력된다. 그 다음 줄부터는 한 줄에 하나씩 주사위의 종류가 1번 주사위부터 주사위 번호 순서대로 입력된다. 주사위의 종류는 각 면에 적혀진 숫자가 그림1에 있는 주사위의 전개도에서 A, B, C, D, E, F 의 순서로 입력된다. 입력되는 숫자 사이에는 빈 칸이 하나씩 있다. 주사위의 개수는 10,000개 이하이며 종류가 같은 주사위도 있을 수 있다.

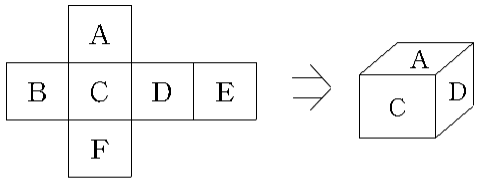


그림 1

출력 형식

출력 파일의 이름은 OUTPUT.TXT로 한다. 한 옆면의 숫자의 합이 가장 큰 값을 출력한다.

입력과 출력의 예

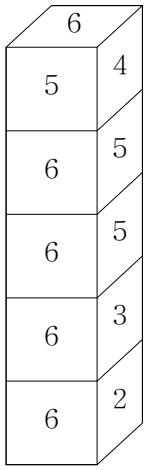
입력(INPUT.TXT)

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

출력(OUTPUT.TXT)

```
29
```

※ 입력 예의 주사위들을 쌓아서 출력 예와 같은 합을 얻으려면 아래 그림과 같이 쌓으면 된다.


6
5
4
6
5
6
5
6
3
6
2

풀이

주사위의 닿는 면이 같은 눈을 가져야 한다는 조건이 주어졌을 때, 여러 개의 주사위를 쌓아 올릴 때 어느 한 옆면의 최대 합을 구하는 문제이다. 어떤 주사위의 아랫면을 어느 면으로 할지를 결정하면, 이 면과 마주 보는 면이 윗면이 된다. 즉, 아랫면이 무엇인지를 결정하면 윗면은 자동으로 결정되고, 마찬가지 방식으로 그 위에 놓이는 주사위의 아랫면도 자동으로 결정된다. 즉, 1번 주사위의 아랫면을 무엇으로 할지만 결정하면 위에 놓일 모든 주사위의 아랫면이 순서대로 결정된다.

이제 주사위의 아래 위만 고정되면, 총 네 가지 상태로 각각의 주사위가 회전이 가능하다는 점을 알 수 있다. 이를 이용해서, 모든 주사위에 대해 앞에서 구한 아랫면과 윗면을 제외한 나머지 네 개의 눈 중에서 가장 큰 값을 더하면 이 경우에 합이 최대가 되는 경우를 구할 수 있다. 이제 1번 주사위의 아랫면을 고정하는 작업을 모든 면에 대해 6번 반복한 뒤 전체의 최대값을 답으로 출력하면 된다.

```
#include <iostream>
#include <fstream>

using namespace std;

#define MAX 10000
#define ASSERT(x) if (!(x)) { cout << "Assertion Fail (" << __LINE__ << "): " << #x << endl; }

const int NOT_REACHED = 0;
int n;
int dice[MAX][6];
int result;

void input()
{
	ifstream inf("input.txt");
	inf >> n;
	int i,j;
	for(i=0;i<n;i++)
		for(j=0;j<6;j++)
		{
			inf >> dice[i][j];
		}
}

int across(int face)
{
	int aface[6] =
		{ 5, 3, 4, 1, 2, 0 };
	ASSERT(0<=face && face<6)
	return aface[face];
}

int sidemax(int n, int top)
{
	int bottom = across(top);
	int max=0;
	for(int i=0;i<6;i++)
	{
		if (i==top || i==bottom) continue;
		if (max < dice[n][i])
			max = dice[n][i];
	}
	return max;
}

int find(int n, int value)
{
	ASSERT(1<=value && value<=6)
	for(int i=0;i<6;i++)
	{
```

```
                if (dice[n][i] == value)
                        return i;
        }
        ASSERT(NOT_REACHED);
        return -1;
}

void process()
{
        result = 0;
        for(int first = 0; first <6; first++)
        {
                int prevbottom = dice[0][across(first)];
                int sum = sidemax(0, first);
                for(int i=1;i<n;i++)
                {
                        int currenttop = find(i,prevbottom);
                        sum+=sidemax(i,currenttop);
                        prevbottom = dice[i][across(currenttop)];
                }
                if (sum>result)
                        result = sum;
        }
}

void output()
{
        ofstream ouf("output.txt");
        ouf << result << endl;
        ouf.close();
}

int main(int argc, char* argv[])
{
        input();
        process();
        output();
        return 0;
}
```

소형 기관차

기차는 맨 앞에 있는 기관차 1대가 손님이 탄 객차 여러 칸을 끌고 간다. 기관차가 고장나면 기차를 운행할 수 없게 되므로 최근 철도청은 기관차 고장에 대비하여 몇몇 역에 소형 기관차 3대를 배치하기로 결정하였다. 소형 기관차는 평소에 이용하는 기관차보다 훨씬 적은 수의 객차만을 끌 수 있다.

기관차가 고장났을 때 끌고 가던 객차 모두를 소형 기관차 3대가 나누어 끌 수는 없기 때문에, 소형 기관차들이 어떤 객차들을 끌고 가는 것이 좋을까하는 문제를 고민하다가 다음과 같이 하기로 결정하였다.

① 소형 기관차가 최대로 끌 수 있는 객차의 수를 미리 정해 놓고, 그보다 많은 수의 객차를 절대로 끌게 하지 않는다. 3대의 소형 기관차가 최대로 끌 수 있는 객차의 수는 서로 같다.

② 소형 기관차 3대를 이용하여 최대한 많은 손님을 목적지까지 운송하도록 한다. 각 객차마다 타고 있는 손님의 수는 미리 알고 있고, 다른 객차로 손님들이 이동하는 것은 허용하지 않는다.

③ 각 소형 기관차는 번호가 연속적으로 이어진 객차를 끌게 한다. 객차는 기관차 바로 뒤에 있는 객차부터 시작하여 1번부터 차례로 번호가 붙어 있다.

예를 들어 기관차가 끌고 가던 객차가 7칸이고, 소형 기관차 1대가 최대로 끌 수 있는 객차 수는 2칸이라고 하자. 그리고 1번부터 7번까지 각 객차에 타고 있는 손님의 수가 아래 표와 같다고 하자. 괄호 속에 있는 숫자는 객차 번호를 나타낸다.

(1)	(2)	(3)	(4)	(5)	(6)	(7)
35	40	50	10	30	45	60

소형 기관차 3대는 각각 1-2번, 3-4번, 그리고 6-7번 객차를 끌고 가면 손님 240명을 운송할 수 있고, 이보다 많은 수의 손님을 운송할 수 없다.

기관차가 끌고 가던 객차의 수와 각 객차에 타고 있던 손님의 수, 그리고 소형 기관차가 최대로 끌 수 있는 객차의 수가 주어질 때, 소형 기관차 3대를 이용하여 최대로 운송할 수 있는 손님 수를 구하는 프로그램을 작성하시오.

실행 파일의 이름은 CC.EXE로 하고, 프로그램의 실행시간은 1초를 넘을 수 없다. 부분 점수는 없다.

입력 형식

입력 파일의 이름은 INPUT.TXT로 한다. 입력 파일의 첫째 줄에 기관차가 끌고 가던 객차의 수가 입력된다. 그 수는 50,000 이하이다. 둘째 줄에는 기관차가 끌고 가던 객차에 타고 있는 손님의 수가 1번 객차부터 차례로 입력된다. 한 객차에 타고 있는 손님의 수는 100명 이하이고, 입력되는 숫자들 사이에 빈칸이 하나씩 있다. 셋째 줄에는 소형 기관차가 최대로 끌 수 있는 객차의 수가 입력된다. 그 수는 기관차가 끌고 가던 객차 수의 $\frac{1}{3}$보다 적다.

출력 형식

출력 파일의 이름은 OUTPUT.TXT로 한다. 한 줄에 소형 기관차 3대를 이용하여 최대로 운송할 수 있는 손님 수를 출력한다.

입력과 출력의 예

입력(INPUT.TXT)

```
7
35 40 50 10 30 45 60
2
```

출력(OUTPUT.TXT)

```
240
```

풀이

여러 개의 객차들을 세 대의 소형 기관차로 나누어 끌게 하는데 가장 많이 끌 수 있게 나누는 문제로, O(n) 동적 계획법으로 해결할 수 있다.

객차의 개수를 n, 각 객차의 손님 수를 A[1], ..., A[n]으로 두자. 먼저 문제에서 한 개의 소형 기관차가 최대로 끌 수 있는 객차의 수를 k라고 두면, 주어진 조언에 따라서 항상 3k는 n 이하임을 알 수 있다. 그런데 손님이 가장 많아지려면 최대한 많은 객차를 끄는 것이 이득임을 알 수 있으므로, 세 개의 소형 기관차가 모두 정확히 k개의 객차를 끄는 것이 항상 답을 최대로 함을 알 수 있다.

이제 D[i][j]를 1번부터 i번 객차에 대해, 소형 기관차를 j개 이하로 이용해서 최대로 운송할 수 있는 손님 수로 정의한다. 기관차가 한 개도 없는 경우에는 손님이 없는 경우이므로 D[i][0] = 0이다. 이제 i번 객차가 마지막 소형 기관차에 포함되는 경우를 생각하자. 그러면 i, i-1, …, i-k+1 번째의 객차가 포함되게 된다. 따라서 D[i][j]는 D[i-k][j-1] + (A[i-k+1] + A[i-k+2] + ... + A[i])의 합으로 구할 수 있다. 이 때 D[i-k][j-1]는 k개의 객차를 제외한 앞부분에서의 최적해를 의미한다. 다음으로, 마지막 소형 기관차에 이 객차가 포함되지 않는 경우의 D[i][j]는 D[i-1][j]임을 알 수 있다. 따라서 D[i][j]는 둘 중 더 큰 값으로 값을 구해 나갈 수 있다. 이를 차례대로 구하는 과정을 수행한 후, D[n][3]이 우리가 구하고자 하는 답이 된다.

```
#include <stdio.h>
#include <stdlib.h>
#define size 100000
#define k 3
#define in_file "input.txt"
#define out_file "output.txt"

int s[size+1][4];
int a[size+1];
int sum[size+1];

FILE *fi,*fo;

int l,n,j;

void main(void)
{
	fi = fopen(in_file,"rt");

	fscanf(fi,"%d",&n);

	for(int i=1;i<=n;i++)
	{
		fscanf(fi,"%d",&a[i]);
		sum[i]=sum[i-1]+a[i];
	}
	fscanf(fi,"%d",&l);

	for(i=1;i<=n;i++)
	{
		for(j=1;j<=k;j++)
		{
			s[i][j]=s[i-1][j];

			if (i-1>=0)
				if (s[i][j] < s[i-l][j-1]+(sum[i]-sum[i-l]))
					s[i][j] = s[i-l][j-1]+(sum[i]-sum[i-l]);
		}
	}


	//printf("%d\n",s[n][k]);

	fo = fopen(out_file,"wt");
	fprintf(fo,"%d\n",s[n][k]);
	fcloseall();

}
```

비밀 편지

홍진이는 지현이에게 문자 A, B, C, D, E, F, G, H 로 쓰인 편지를 날마다 보내는데, 컴퓨터로 보내는 비밀편지로, 한 문자마다 0 또는 1인 숫자 여섯 개를 사용하여 보낸다. 둘 사이의 약속은 다음과 같다.

A	000000
B	001111
C	010011
D	011100
E	100110
F	101001
G	110101
H	111010

홍진이가 어느날 001111000000011100를 보내면 지현이는 이것을 BAD로 이해하게 된다. 그런데 둘 사이에 약속이 잘 만들어져 있기 때문에, 통신에 문제가 생겨서 한 문자를 표시하는 여섯 숫자 중 어느 한 숫자만 틀리게 오는 경우, 지현이는 원래 보내려는 문자를 알아 낼 수가 있다.

예를 들어 지현이가 000100을 받았을 때, A와는 숫자 한 자만 다르고, 다른 문자들과는 각각 숫자 두 자 이상이 다르므로 지현이는 이것이 A라고 알아보게 된다.

다만 111111과 같이 모든 문자의 표현과 숫자 두 자 이상이 다른 경우에는 무슨 문자인지 알 수가 없게 된다. 예를 들어 지현이가 011111000000111111000000111111를 받았을 때, BA 다음에 알아 볼 수 없는 문자가 나오는데, 이 경우 이런 것이 처음 나오는 문자의 위치인 3을 출력한다.

지현이가 받은 편지를 보고 문자들을 알아내어 출력하거나, 모르는 문자가 있는 경우, 이것이 처음 나오는 위치를 출력하는 프로그램을 작성하시오.

실행 파일의 이름은 AA.EXE로 하고, 프로그램의 실행 시간은 1초를 넘을 수 없다. 부분 점수는 없다.

입력 형식

입력 파일의 이름은 INPUT.TXT로 한다. 첫줄에는 보낸 문자의 개수(10개보다 작다)가 입력된다. 다음 줄에는 문자의 개수의 여섯 배 만큼의 숫자 입력이 주어진다.

출력 형식

출력 파일의 이름은 OUTPUT.TXT로 한다. 주어진 입력에서 지현이가 이해한 문자들을 출력하거나, 모르는 문자가 나오는 경우 그런 것이 처음 나오는 위치를 출력한다.

입력과 출력의 예(1)

입력(INPUT.TXT)

```
3
001110000100011100
```

출력(OUTPUT.TXT)

```
BAD
```

입력과 출력의 예(2)

입력(INPUT.TXT)

```
5
011111000000111111000000111111
```

출력(OUTPUT.TXT)

```
3
```

풀이

A부터 F까지의 각 문자를 여섯 숫자로 표시하는데, 여섯 숫자 중 어느 한 숫자가 달라진 경우에도 어떤 문자인지 알 수 있도록 표현되어 있다. 따라서 입력을 차례로 앞에서부터 보면서, 각각이 어느 문자에 대응되는지를 확인하는 식으로 처리하면 된다.

먼저 입력된 숫자들을 여섯 개씩 끊어서 관리한다. 그러면 각각은 한 개의 문자에 해당하게 되므로, 각각을 여덟 개의 문자를 나타내는 여섯 개의 숫자와 비교해 본다. 만일 여섯 숫자가 모두 같은 경우가 있다면 이는 그 문자에 대응되는 경우이고, 또는 하나만 다른 경우에도 대응되는 경우이다. 이럴 때에는 해당하는 문자를 문자열 형태로 저장해 둔다. 이런 식으로 쭉 처리하면서 문제되는 경우가 없다면 저장된 문자열을 출력한다. 하지만, 만일 두 숫자 이상이 다른 경우가 있다면 바로 해당되는 위치를 출력하면 된다.

```
#include <stdio.h>

#define MAXN 60
#define INFILE "input.txt"
#define OUTFILE "output.txt"

int n, sw;
int a[MAXN];
char c[MAXN / 6];
char data[8][6] = {{'0', '0', '0', '0', '0', '0'}
                  , {'0', '0', '1', '1', '1', '1'}
                  , {'0', '1', '0', '0', '1', '1'}
                  , {'0', '1', '1', '1', '0', '0'}
                  , {'1', '0', '0', '1', '1', '0'}
                  , {'1', '0', '1', '0', '0', '1'}
                  , {'1', '1', '0', '1', '0', '1'}
                  , {'1', '1', '1', '0', '1', '0'}};

FILE *in = fopen(INFILE, "rt");
FILE *out = fopen(OUTFILE, "wt");

void input()
{
        fscanf(in, "%d", &n);
        for (int i=0;i<n*6;i++)
                fscanf(in, "%1d", &a[i]);
  fclose(in);
}

void process()
{
        int comp;
        for (int i=0;i<n;i++)
        {
                for (int j=0;j<8;j++)
                {
                        comp = 0;
                        for (int k=0;k<6;k++)
                        {
                                if (a[i * 6 + k] != data[j][k] - 48) comp++;
                        }
                        if (comp <= 1)
                        {
                                c[i] = j + 65;
                                sw = -1;
                        }
                }
                if (sw == -1)
                        sw = 0;
                else
                {
```

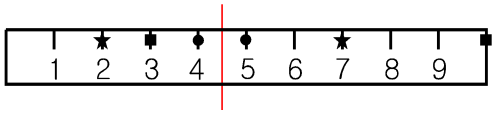
```
                                sw = i + 1;
                                break;
                        }
                }
}

void output()
{
        if (sw == 0)
        {
                for (int i=0;i<n;i++)
                        fprintf(out, "%c", c[i]);
                fprintf(out, "\n");
        }
        else
                fprintf(out, "%d\n", sw);
  fclose(out);
}

void main()
{
        input();
        process();
        output();
}
```

줄자 접기

준성이는 1 cm 간격으로 눈금이 매겨져 있는 줄자를 가지고 있다. 그 줄자에 있는 서로 다른 눈금 6개에 한 눈금에 하나씩 점이 찍혀 있는데, 빨간 점, 파란 점, 노란 점이 각각 두 개씩 있다.

준성이는 먼저 두 빨간 점이 만나도록 줄자를 접었다. 그런 후 두 파란 점이 만나도록 줄자를 접고, 또다시 두 노란 점이 만나도록 줄자를 접었다. 줄자는 투명하여 접더라도 점들을 잘 볼 수 있다. 어떤 색깔의 두 점이 만나도록 줄자를 접었을 때, 그 다음에 접으려는 색깔의 두 점이 이미 만나고 있으면, 그 두 점에 대해서는 줄자를 접지 않는다.

예를 들어 길이 10 cm인 줄자에 아래 그림과 같이 2 cm와 7 cm 위치에 두 빨간 점이 찍혀 있고, 5 cm와 4 cm 위치에 파란 점이, 10 cm와 3 cm 위치에 노란 점이 찍혀 있다고 하자. (그림에서 빨간 점은 별표로, 파란 점은 동그라미, 그리고 노란 점은 네모로 표시되어 있다.) 빨간 두 점이 만나도록 줄자를 접으면 줄자의 4.5 cm 위치에서 접히고 4 cm와 5 cm 눈금이 서로 만나게 된다. 그러면 줄자의 왼쪽 부분의 길이는 4.5 cm이고 오른쪽 부분의 길이는 5.5 cm가 되어, 접힌 줄자의 길이는 5.5 cm가 된다. 파란 두 점은 이미 만나므로 줄자를 접지 않고, 그런 다음 노란 두 점이 만나도록 접으면 줄자의 길이는 3.5 cm가 된다.



줄자의 길이와 각 색깔의 점들이 찍혀 있는 위치가 주어질 때, 준성이가 빨간 색, 파란 색, 노란 색의 순서로 두 점이 만나도록 줄자를 접으면 줄자의 길이가 얼마가 되는지를 구하는 프로그램을 작성하시오.

실행 파일의 이름은 BB.EXE로 하고, 프로그램의 실행시간은 1초를 넘을 수 없다. 부분 점수는 없다.

입력 형식

입력 파일의 이름은 INPUT.TXT로 한다. 입력 파일의 첫째 줄에 줄자의 길이가 입력된다. 줄자의 길이는 10 cm 이상 1,000 cm 이하이고 단위를 나타내는 cm은 입력되지 않는다. 둘째 줄에는 두 빨간 점의 위치를 나타내는 정수가 빈칸을 사이에 두고 입력된다. 셋째 줄에는 두 파란 점의 위치가, 넷째 줄에는 두 노란 점의 위치를 나타내는 정수가 빈칸을 사이에 두고 입력된다. 모든 점들의 위치는 서로 다르다.

출력 형식

출력 파일의 이름은 OUTPUT.TXT로 한다. 한 줄에 접은 후의 줄자의 길이를 소수점 이하 한자리까지 출력한다.

입력과 출력의 예

입력(INPUT.TXT)

```
10
2 7
5 4
10 3
```

출력(OUTPUT.TXT)

```
3.5
```

풀이

줄자의 길이 N과 여섯 개의 점의 위치가 주어지면, 문제에서 주어진 규칙에 따라 빨간 색, 파란 색, 노란 색의 순서로 해당되는 두 점이 각각 만나도록 줄자를 접어 나가면 된다. 이 때 줄자를 접는 순서가 주어져 있으므로, 별다른 처리 없이 단순히 줄자를 접는 과정만을 프로그래밍 하면 된다.

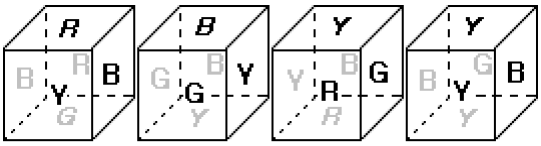
줄자를 한 번 접는 과정은 다음과 같이 프로그래밍 할 수 있다. 먼저 마주보아야 할 두 점의 좌표의 평균을 구하면, 이 점이 줄자가 접혀야 할 위치가 된다는 것을 알 수 있다. 이런 식으로 접을 위치를 계산한 후, 실제로 그 위치에서 접는 과정을 수행한다. 접는 방법에는 두 가지가 있는데, 편의상 그 중 하나를 선택하면 된다. 예를 들어 오른쪽에서 왼쪽으로 접었다고 생각할 수 있는데, 그러면 접힐 위치의 오른쪽에 있던 점들이 왼쪽으로 넘어갔다고 생각을 할 수 있다. 따라서 점들이 접힌 뒤의 좌표를 새롭게 구할 수 있으므로, 이를 세 번 반복한 뒤에 접힌 줄자의 길이를 출력해 주면 된다. 이 때 줄자의 끝의 위치도 기억해 둔 후 접는 과정을 수행하며, 접힐 때마다 줄자의 끝의 위치를 갱신해 나간다.

```
#include <stdio.h>

FILE *fi;
FILE *fo;
double ed,rl,rr,bl,br,yl,yr;
int n;
double lm;

void swap(double &a,double &b)
{
        double t;
        t=a;
        a=b;
        b=t;
}

void main(void)
{
        fi = fopen("input.txt","rt");
        fo = fopen("output.txt","wt");
        fscanf(fi,"%d",&n);
        ed = n;
        fscanf(fi,"%lf %lf",&rl,&rr);
        fscanf(fi,"%lf %lf",&bl,&br);
        fscanf(fi,"%lf %lf",&yl,&yr);

        if (rl > rr) swap(rl,rr);
        if (bl > br) swap(bl,br);
        if (yl > yr) swap(yl,yr);

        if (rl!=rr)
        {
                lm = (rl + rr)/2.0;

                if (lm > bl)
                        bl = (lm-bl); else bl = bl - lm;

                if (lm > br)
                        br = (lm-br); else br = br - lm;

                if (lm > yl)
                        yl = (lm-yl); else yl = yl - lm;

                if (lm > yr)
                        yr = (lm-yr); else yr = yr - lm;

                if (lm > (ed/2) )
                        ed = lm;
                 else ed = ed - lm;
        }
```

```
        if (bl!=br)
        {
                lm = (bl + br)/2.0;

                if (lm > yl)
                        yl = (lm-yl); else yl = yl - lm;

                if (lm > yr)
                        yr = (lm-yr); else yr = yr - lm;

                if (lm > (ed/2) )
                        ed = lm;
                 else ed = ed - lm;
        }

        if (yl!=yr)
        {
                lm = (yl + yr)/2.0;

                if (lm > (ed/2) )
                        ed = lm;
                 else ed = ed - lm;
        }

        fprintf(fo,"%.1lf\n",ed);
}
```

기둥 만들기

주사위 모양의 정육면체에 각 면이 빨강(R), 초록(G), 파랑(B), 노랑(Y) 가운데 어떤 색으로 칠해져 있다. 이러한 정육면체 4개를 기둥 모양으로 쌓아 올려서 기둥의 각 옆면에 4가지 색이 모두 나타나게 하고 싶다. 이러한 기둥을 모두 몇 개나 만들 수 있는지 구하는 프로그램을 작성하시오.



정육면체1 정육면체2 정육면체3 정육면체4

그림 1

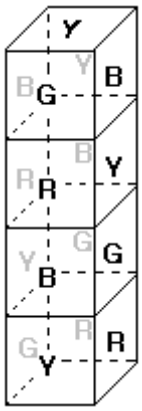


그림 2

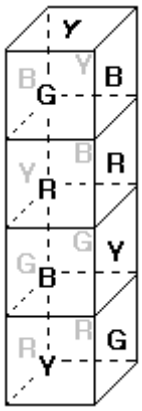


그림 3

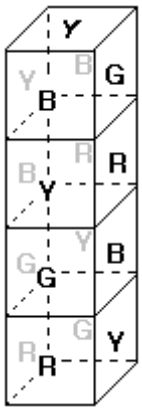


그림 4

정육면체를 쌓을 때 1번 정육면체를 맨 아래에 놓고, 그 위에 2번 정육면체, 3번 정육면체, 맨 위에 4번 정육면체를 놓는다. 각 정육면체는 마음대로 위치를 바꾸어서 놓을 수 있다. 예를 들어서, 그림 1과 같은 4개의 정육면체를 쌓아서 그림 2와 그림 3의 두 개의 기둥을 만들 수 있다.

하지만 기둥을 옆으로 회전시켜서 같은 모양이 되는 것은 같은 기둥으로 본다. 예를 들어서 그림 3에 있는 기둥과 그림 4에 있는 기둥은 같은 기둥이다. 기둥의 윗면의 색이 다른 것은 다른 기둥이며, 기둥의 밑면은 보이지 않으므로 고려하지 않는다.

실행 파일의 이름은 CC.EXE로 하고, 프로그램의 실행 시간은 1초를 넘을 수 없다. 부분 점수는 없다.

입력 형식

입력 파일의 이름은 INPUT.TXT로 한다. 첫줄에 1번 정육면체, 두 번째 줄에 2번 정육면체, 세 번째 줄에 3번 정육면체, 네 번째 줄에 4번 정육면체가 입력된다. 각 줄은 6개의 영문자로 이루어진다. 영문자는 R, G, B, Y 중의 하나이다. 6개의 영문자는 순서대로 그림 5의 가, 나, 다, 라, 마, 바 면의 색을 나타낸다.

그림 5

출력 형식

출력 파일의 이름은 OUTPUT.TXT로 한다. 조건을 만족하는 기둥의 개수를 출력한다. 조건을 만족하는 기둥이 하나도 만들어지지 않으면 0을 출력한다.

입력과 출력의 예

입력(INPUT.TXT)

```
YBRBRG
GYBGBY
RGBYYR
YBGBYY
```

출력(OUTPUT.TXT)

```
2
```

풀이

정육면체들을 주어진 조건을 만족하도록 쌓는 방법의 가짓수를 구하는 문제로, 상태 공간을 탐색하는 문제이다. 문제에서 모든 가능한 경우를 구하라고 되어 있는데, 실제로 가능한 경우의 수가 얼마 되지 않으므로 하나씩 해 보면서 그 중 조건을 만족하는 경우가 몇 가지인지를 세면된다.

우선 한 개의 정육면체를 쌓는 가짓수는 총 6*4=24가지가 있다. 따라서 이러한 경우를 모든 정육면체에 대해서 따져주면 되는데, 좀 더 생각해 보면 각 기둥의 옆면에 네 가지 색이 모두 나타나는 경우만 검사하면 충분함을 알 수 있다. 따라서 이러한 조건을 벗어나는 경우를 제외하도록 하면 불필요한 탐색을 줄일 수 있다.

또한 문제에서 회전을 통해서 같은 모양이 되는 것은 제외해야 한다고 하였으므로, 조건에 맞는 정육면체를 쌓은 후에는 이를 비교하는 과정을 수행해야 한다. 즉, 이렇게 네 개의 정육면체를 모두 쌓은 뒤 만들어진 기둥이 지금까지 만들어진 기둥과 회전하여 같아지지 않는지 검사해 주면 된다. 만일 같아지는 경우가 없다면 새로 만들어진 경우를 답에 추가하고, 경우의 수를 하나 증가시킨다.

```
#include <fstream>

using namespace std;

#define INFILE "input.txt"
#define OUTFILE "output.txt"
#define code_length 17

ifstream fin(INFILE);
ofstream fout(OUTFILE);

char cube[4][6], code[18], save[1000][18];

int pattern[6][6] = {{4, 5, 0, 1, 2, 3}, {5, 4, 0, 3, 2, 1}, {0, 2, 1, 4, 3, 5}, {2, 0, 1, 5, 3, 4},{1, 3, 0, 5, 2, 4}, {3, 1, 0, 4, 2, 5}};
int m;

void swap(char &a, char &b)
{
	char c = a;
	a = b, b = c;
};

void input()
{
	char s[100];
	for (int i=0;i<4;i++)
	{
		fin.getline(s, 255);
		for (int j=0;j<6;j++)
			cube[i][j] = s[j];
	}
}

void code_shift()
{
	for (int i=0;i<4;i++)
	{
		swap(code[i * 4 + 0], code[i * 4 + 1]);
		swap(code[i * 4 + 1], code[i * 4 + 2]);
		swap(code[i * 4 + 2], code[i * 4 + 3]);
	}
}

void code_check()
{
	int i, sw, same = 0;
	for (i=0;i<4;i++)
	{
		if (same == 0)
			for (int j=0;j<m;j++)
```

```
                        {
                                sw = 0;
                                for (int k=0;k<code_length;k++)
                                        if (code[k] != save[j][k]) sw = 1;
                                if (sw == 0) same = 1;
                        }
                code_shift();
        }
        if (same == 0)
        {
                for (i=0;i<code_length;i++)
                        save[m][i] = code[i];
                m++;
        }
}

void process(int p)
{
        int i, j, k;
        for (i=0;i<4;i++)
                for (j=0;j<p-1;j++)
                        if (code[(p - 1) * 4 + i] == code[j * 4 + i]) return;
        if (p == 4)
                code_check();
        else
                for (i=0;i<6;i++) // pattern
                        for (j=0;j<4;j++)
                        {
                                for (k=0;k<4;k++)
                                        code[p * 4 + k] = cube[p][pattern[i][2 + (j + k) % 4]];
                                if (p == 3) code[16] = cube[p][pattern[i][0]];
                                process(p + 1);
                                if (cube[p][2] == cube[p][3] && cube[p][3] == cube[p][4] &&
cube[p][4] == cube[p][5]) break;
                                if (cube[p][2] == cube[p][4] && cube[p][3] == cube[p][5] &&
cube[p][2] != cube[p][3] && j == 1) break;
                        }
}

void output()
{
        fout << m << endl;
        fout.close();
}

void main()
{
        input();
        process(0);
        output();
}
```

숫자 게임

N명이 모여 숫자 게임을 하고자 한다. 각 사람에게는 1부터 10사이의 숫자가 적혀진 다섯 장의 카드가 주어진다. 그 중 세 장의 카드를 골라 합을 구한 후 일의 자리 수가 가장 큰 사람이 게임을 이기게 된다. 세 장의 카드가 (7, 8, 10)인 경우에는 합은 7+8+10 = 25가 되고 일의 자리 수는 5가 된다. 어떤 사람이 받은 카드가 (7, 5, 5, 4, 9)인 경우 (7, 4, 9)를 선택하면 합이 20이 되어 일의 자리 수는 0이 되고, (5, 5, 9)를 선택하면 합이 19가 되어 일의 자리 수는 9가 된다. 게임을 이기기 위해서는 세 장의 카드를 선택할 때 그 합의 일의 자리 수가 가장 크게 되도록 선택하여야 한다.

예를 들어, N=3일 때, 1번 사람이 (7, 5, 5, 4, 9), 2번 사람이 (1, 1, 1, 1, 1), 3번 사람이 (2, 3, 3, 2, 10)의 카드들을 받았을 경우, 세 수의 합에서 일의 자리 수가 가장 크게 되도록 세 수를 선택하면 1번 사람은 (5, 5, 9)에서 9, 2번 사람은 (1, 1, 1)에서 3, 3번 사람은 (2, 3, 3)에서 8의 결과를 각각 얻을 수 있으므로 첫 번째 사람이 이 게임을 이기게 된다.

N명에게 각각 다섯 장의 카드가 주어졌을 때, 세 장의 카드를 골라 합을 구한 후 일의 자리 수가 가장 큰 사람을 찾는 프로그램을 작성하시오. 가장 큰 숫자를 갖는 사람이 두 명 이상일 경우에는 번호가 가장 큰 사람의 번호를 출력한다.

실행 파일의 이름은 AA.EXE로 하고, 실행시간은 1초를 넘을 수 없다. 부분 점수는 없다.

입력 형식

입력 파일의 이름은 INPUT.TXT이다. 첫 줄에는 사람의 수를 나타내는 정수 N이 주어진다. N은 2이상 1,000이하이다. 그 다음 N 줄에는 1번부터 N번까지 각 사람이 가진 카드가 주어지는 데, 각 줄에는 1부터 10사이의 정수가 다섯 개씩 주어진다. 각 정수 사이에는 한 개의 빈칸이 있다.

출력 형식

출력 파일의 이름은 OUTPUT.TXT이다. 게임에서 이긴 사람의 번호를 첫 번째 줄에 출력한다. 이긴 사람이 두 명 이상일 경우에는 번호가 가장 큰 사람의 번호를 출력한다.

입력과 출력의 예

입력(INPUT.TXT)

```
3
7 5 5 4 9
1 1 1 1 1
2 3 3 2 10
```

출력(OUTPUT.TXT)

```
1
```

풀이

각 사람이 받은 카드에 대한 정보가 주어졌을 때, 누가 승자가 되는지 알아내는 문제이다. 이 때 각 사람에 대해서 그 사람이 받을 점수(일의 자리 수)의 최대값을 미리 각각 계산한 후, 누구의 점수가 제일 큰지를 확인하면 된다.

다섯 개의 정수 중에서 세 개를 골라 일의 자리 수가 가장 크게 되도록 하는 것은 반복문을 이용하면 쉽게 해결할 수 있다. 이는 경우의 수가 몇 되지 않기 때문으로, 3중 반복문 등을 이용하여 모든 가능한 경우를 따져보고, 그 중 점수가 가장 커지는 경우를 판단하면 된다. 일의 자리 수를 구하기 위해서는 10으로 나눈 나머지를 이용해도 되고, 10이나 20을 빼는 식으로도 구할 수 있다.

이제 이러한 과정을 총 N번 반복한 뒤, 일의 자리 수가 가장 큰 사람의 번호를 출력하면 된다. 문제의 조건에 의해서 두 명 이상일 경우에는 번호가 가장 큰 경우를 출력해야 함에 주의한다.

```
#include <stdio.h>


int sol=0,n,i,j,k,l,s,vmax,t;
int a[6];

FILE *fi=fopen("input.txt","rt");
FILE *fo=fopen("output.txt","wt");

void main(void)
{
	fscanf(fi,"%d",&n);
	vmax = -1;
	t = 0;
	for(i=1;i<=n;i++)
	{
		for(j=1;j<=5;j++)
			fscanf(fi,"%d",&a[j]);
		for(j=1;j<=5;j++)
		{
			for(k=j+1;k<=5;k++)
			{
				for(l=k+1;l<=5;l++)
				{
					s = a[j]+a[k]+a[l];
					s = s % 10;
					if (vmax <= s)
					{
						vmax=s;
						t = i;
					}
				}
			}
		}
	}
	fprintf(fo,"%d\n",t);
}
```

창고 다각형

N개의 막대 기둥이 일렬로 세워져 있다. 기둥들의 폭은 모두 1m이며 높이는 다를 수 있다. 이 기둥들을 이용하여 양철로 된 창고를 제작하려고 한다. 창고에는 모든 기둥이 들어간다. 이 창고의 지붕을 다음과 같이 만든다.

(1) 지붕은 수평 부분과 수직 부분으로 구성되며, 모두 연결되어야 한다.
(2) 지붕의 수평 부분은 반드시 어떤 기둥의 윗면과 닿아야 한다.
(3) 지붕의 수직 부분은 반드시 어떤 기둥의 옆면과 닿아야 한다.
(4) 지붕의 가장자리는 땅에 닿아야 한다.
(5) 비가 올 때 물이 고이지 않도록 지붕의 어떤 부분도 오목하게 들어간 부분이 없어야 한다.

그림 1은 창고를 옆에서 본 모습을 그린 것이다. 이 그림에서 굵은 선으로 표시된 부분이 지붕에 해당되고, 지붕과 땅으로 둘러싸인 다각형이 창고를 옆에서 본 모습이다. 이 다각형을 창고 다각형이라고 하자.

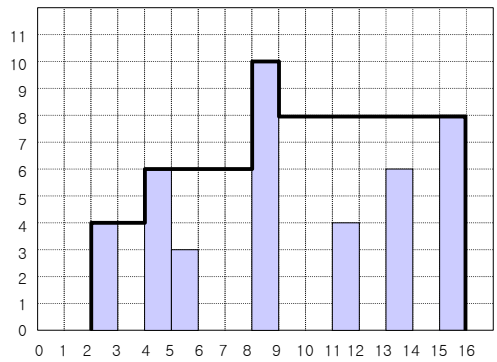


그림 1. 기둥과 지붕(굵은 선)의 예

창고 주인은 창고 다각형의 면적이 가장 작은 창고를 만들기를 원한다. 그림 1에서 창고 다각형의 면적은 98 ㎡이고, 이 경우가 가장 작은 창고 다각형이다.

기둥들의 위치와 높이가 주어질 때, 가장 작은 창고 다각형의 면적을 구하는 프로그램을 작성하시오.

실행 파일의 이름은 BB.EXE로 하고, 실행시간은 1초를 넘을 수 없다. 부분점수는 없다.

입력 형식

입력 파일의 이름은 INPUT.TXT이다. 첫 줄에는 기둥의 개수를 나타내는 정수 N이 주어진다. N은 1 이상 1,000 이하이다. 그 다음 N 개의 줄에는 각 줄에 각 기둥의 왼쪽 면의 위치를 나타내는 정수 L과 높이를 나타내는 정수 H가 한 개의 빈 칸을 사이에 두고 주어진다. L과 H는 둘 다 1 이상 1,000 이하이다.

출력 형식

출력 파일의 이름은 OUTPUT.TXT이다. 첫 줄에 창고 다각형의 면적을 나타내는 정수를 출력한다.

입력과 출력의 예

입력(INPUT.TXT)

```
7
2 4
11 4
15 8
4 6
5 3
8 10
13 6
```

출력(OUTPUT.TXT)

```
98
```

풀이

주어진 조건을 만족하는 다각형을 찾는 문제이다. 여러 가지 방법이 있을 수 있지만, 모범 답안에서는 기둥의 높이가 감소하는 순서대로 진행하면서 넓이를 계산해 나가는 방법을 이용하였다.

이를 위해서는 먼저 각 기둥을 높이가 감소하는 순서대로 정렬해야 한다. 그 후에는 가장 높은 기둥부터 순서대로 처리해 나간다. 이 때 몇 가지 경우가 발생할 수 있는데, 다각형의 넓이가 증가하는 경우를 판단하여 이때 증가하는 면적을 누적시켜 나가는 방식으로 처리할 수 있다. 즉, 다음으로 고려하는 기둥 현재까지 처리한 영역의 왼쪽에 있거나 오른쪽에 있는 경우에 해당하는 만큼 추가되는 넓이를 누적해서 더해 나가는 방식을 이용한다.

이런 식으로 계산을 수행하는 과정에서 현재까지 처리된 영역의 범위와 각각의 높이를 저장해 둔다. 즉, 영역이 넓어지게 될 때마다 왼쪽과 오른쪽 각각에 대해서 현재 어디까지가 처리된 영역이며 그 영역의 높이는 얼마인지를 저장해 둔다. 이와 같이 다각형을 저장해 둔 후, 최종적으로 구해진 다각형의 면적을 계산하여 출력한다.

```
#include <stdio.h>

/* 상수 정의 */
#define MAXSIZE (1000)
#define INPUTFILE "input.txt"
#define OUTPUTFILE "output.txt"

/* 입력 구조체 */
typedef struct {
        int pos;
        int height;
} INPUT;

/* 입력 */
int input(int *size, INPUT *data)
{
        int i;
        FILE *fp;

        fp = fopen(INPUTFILE, "r");
        /* 기둥 수 입력 */
        if (fp==NULL) return -1;
        fscanf(fp, "%d", size);
        if (*size<=0 || *size>MAXSIZE) return -1;
        /* 기둥 데이터 입력 */
        for (i=0; i<*size; i++) {
                fscanf(fp, "%d %d", &data[i].pos, &data[i].height);
                if (data[i].height<=0 || data[i].height>1000) return -1;
        }
        fclose(fp);

        return 0;
}

/* 높이순으로 정렬 */
void sort(int size, INPUT* data)
{
        int i, j;
        INPUT tmp;

        for (i=1; i<size; i++) {
                tmp.pos = data[i].pos;
                tmp.height = data[i].height;
                for (j=i-1; j>=0; j--) {
                        if (tmp.height<data[j].height) break;
                        data[j+1].pos = data[j].pos;
                        data[j+1].height = data[j].height;
                }
                data[j+1].pos = tmp.pos;
                data[j+1].height = tmp.height;
        }
```

```
}

/* 넓이 계산 */
int area(int size, INPUT* data)
{
	int i;
	int left, right, ret;
	ret = data[0].height;
	left = data[0].pos;
	right = data[0].pos+1;
	for (i=1; i<size; i++) {
		if (data[i].pos<left) {
			ret += (left-data[i].pos)*data[i].height;
			left = data[i].pos;
		} else if (data[i].pos+1>right) {
			ret += (data[i].pos+1-right)*data[i].height;
			right = data[i].pos+1;
		}
	}
	return ret;
}

int main(void)
{
	int size;
	INPUT data[MAXSIZE];
	FILE *fp;

	if (input(&size, data)<0) {
		printf("ERROR : INVALID INPUT FILE\n");
		return 0;
	}
	sort(size, data);
	size = area(size, data);
	fp = fopen(OUTPUTFILE, "w");
	if (fp==NULL) {
		printf("ERROR : CANNOT OPEN OUTPUT FILE\n");
		return 0;
	}
	fprintf(fp, "%d", size);
	fclose(fp);
	return 0;
}
```

극장 좌석

어떤 극장의 좌석은 한 줄로 되어 있으며 왼쪽부터 차례대로 1번부터 N번까지 번호가 매겨져 있다. 공연을 보러 온 사람들은 자기의 입장권에 표시되어 있는 좌석에 앉아야 한다. 예를 들어서, 입장권에 5번이 쓰여 있으면 5번 좌석에 앉아야 한다. 단, 자기의 바로 왼쪽 좌석 또는 바로 오른쪽 좌석으로는 자리를 옮길 수 있다. 예를 들어서, 7번 입장권을 가진 사람은 7번 좌석은 물론이고, 6번 좌석이나 8번 좌석에도 앉을 수 있다. 그러나 5번 좌석이나 9번 좌석에는 앉을 수 없다.

그런데 이 극장에는 "고정석 회원"들이 있다. 이 사람들은 반드시 자기 좌석에만 앉아야 하며 옆 좌석으로 자리를 옮길 수 없다.

오늘 공연은 입장권이 매진되어 1번 좌석부터 N번 좌석까지 모든 좌석이 다 팔렸다. 고정석 회원들의 좌석 번호들이 주어졌을 때, 사람들이 좌석에 앉는 서로 다른 방법의 가짓수를 구하는 프로그램을 작성하시오.

예를 들어서, 그림과 같이 좌석이 9개이고, 4번 좌석과 7번 좌석이 고정석인 경우에 <123456789>는 물론 가능한 배치이다. 또한 <213465789> 와 <132465798> 도 가능한 배치이다. 그러나 <312456789> 와 <123546789> 는 허용되지 않는 배치 방법이다.

실행 파일의 이름은 CC.EXE로 하고, 실행시간은 1초를 넘을 수 없다. 부분점수는 없다.

입력 형식

입력파일의 이름은 INPUT.TXT이다. 첫째 줄에는 좌석의 개수 N이 입력된다. N은 1 이상 40 이하이다. 둘째 줄에는 고정석의 개수 M이 입력된다. M은 0 이상 N 이하이다. 다음 M 개의 줄에는 고정석의 번호가 작은 수부터 큰 수의 순서로 한 줄에 하나씩 입력된다.

출력 형식

출력파일의 이름은 OUTPUT.TXT이다. 주어진 조건을 만족하면서 사람들이 좌석에 앉을 수 있는 방법의 가짓수를 출력한다. 방법의 가짓수는 2,000,000,000을 넘지 않는다(2,000,000,000 $< 2^{31} - 1$).

입력과 출력의 예

입력(INPUT.TXT)

```
9
2
4
7
```

출력(OUTPUT.TXT)

```
12
```

풀이

주어진 조건을 만족하는 경우의 수를 구하는 문제이다. 먼저 고정석이 없는 경우를 생각해 보고 일반화를 시켜 볼 수 있다.

D[n]를 자리가 n개일 때 n명이 앉을 수 있는 경우의 수로 정의하자. n번 입장권을 가진 사람은 n번 자리에 앉거나 n-1번 자리에 앉아야 한다. 첫 번째 경우의 가짓수는 D[n-1]과 같고, 두 번째의 경우는 n-1번 입장권을 가진 사람이 반드시 n번 자리에 앉아야 하므로 D[n-2]와 같다. 즉, D[n]=D[n-1]+D[n-2]의 관계식으로 구할 수 있고, 이를 프로그램 상에서 사전에 구하는 과정을 수행한다.

다음으로는 고정석이 있는 경우를 생각해 보자. 먼저 알 수 있는 것은, 고정석으로 인해 분리되는 양쪽 좌석 간에는 최대 두 칸 이상 차이가 나게 된다는 점이다. 이러한 경우를 생각해 보면, 그 사이로는 사람들이 자리를 바꾸어 앉을 수 없다는 것을 알 수 있다. 즉, 전체 좌석을 고정석에 따라서 나눠줄 수 있으며, 나눠준 각각에 대해서 경우의 수를 구하면 전체 경우의 수를 구할 수 있다. 즉, 고정석으로 나뉜 연속된 좌석들 각각에 대해 경우의 수를 앞에서 구한 D[n]을 이용하여 계산하고, 이 때 계산된 경우의 수를 곱해 주면 전체 경우의 수를 구할 수 있다.

```
#include <stdio.h>
#include <stdlib.h>
#define Big_N 40

int n, n_fixed;
int fixed[Big_N+1];
int fibonaci[Big_N];
FILE *in, *out;

void attain_input() {
	int i;

	in = fopen("input.txt", "r");

	fscanf(in, "%d", &n);
	fscanf(in, "%d", &n_fixed);

	for (i=0; i<n_fixed; i++)
		fscanf(in, "%d", &fixed[i]);

	fixed[n_fixed] = n+1;
}

void sort() {
	int i, j;
	for (i=0; i<n_fixed; i++)
		for (j=i+1; j<n_fixed; j++)
			if (fixed[i] > fixed[j]) {
				int temp = fixed[i];
				fixed[i] = fixed[j];
				fixed[j] = temp;
			}
}

int algorithm() {
	int i, max_interval = fixed[0]-1;
	int result;

	sort();

	for (i=1; i<=n_fixed; i++)
		if ((fixed[i] - fixed[i-1] - 1) > max_interval)
			max_interval = (fixed[i] - fixed[i-1] - 1);

	fibonaci[0] = 1;
	fibonaci[1] = 1;
	for (i=2; i<=max_interval; i++)
		fibonaci[i] = fibonaci[i-1] + fibonaci[i-2];

	result = fibonaci[fixed[0]-1];
	for (i=1; i<=n_fixed; i++)
```

```
                result *= fibonaci[fixed[i]-fixed[i-1]-1];

        return result;
}

void print_output(int value) {
        printf("%d\n", value);
        out = fopen("output.txt", "w");
        fprintf(out, "%d\n", value);
        fclose(out);
}

int main() {
        attain_input();
        print_output(algorithm());
        return 0;
}
```

임시 반장 정하기

김갑동 선생님은 올해 6학년 1반 담임을 맡게 되었다. 김갑동 선생님은 우선 임시로 반장을 정하고 학생들이 서로 친숙해진 후에 정식으로 선거를 통해 반장을 선출하려고 한다. 그는 자기반 학생 중에서 1학년부터 5학년까지 지내오면서 한번이라도 같은 반이었던 사람이 가장 많은 학생을 임시 반장으로 정하려 한다.

그래서 김갑동 선생님은 각 학생들이 1학년부터 5학년까지 몇 반에 속했었는지를 나타내는 표를 만들었다.

예를 들어 학생 수가 5명일 때의 표를 살펴보자.

	1학년	2학년	3학년	4학년	5학년
1번 학생	2	3	1	7	3
2번 학생	4	1	9	6	8
3번 학생	5	5	2	4	4
4번 학생	6	5	2	6	7
5번 학생	8	4	2	2	2

위 경우에 4번 학생을 보면 3번 학생과 2학년 때 같은 반이었고, 3번 학생 및 5번 학생과 3학년 때 같은 반이었으며, 2번 학생과는 4학년 때 같은 반이었음을 알 수 있다. 그러므로 이 학급에서 4번 학생과 한번이라도 같은 반이었던 사람은 2번 학생, 3번 학생과 5번 학생으로 모두 3명이다. 이 예에서 4번 학생이 전체 학생 중에서 같은 반이었던 학생 수가 제일 많으므로 임시 반장이 된다.

각 학생들이 1학년부터 5학년까지 속했던 반이 주어질 때, 임시 반장을 정하는 프로그램을 작성하시오.

실행 파일의 이름은 AA.EXE로 하고 실행시간은 1초를 넘을 수 없다. 부분점수는 없다.

입력 형식

입력 파일의 이름은 INPUT.TXT이다. 첫째 줄에는 반의 학생 수를 나타내는 정수가 주어진다. 학생 수는 3 이상 1000 이하이다. 둘째 줄부터는 1번 학생부터 차례대로 각 줄마다 1학년부터 5학년까지 몇 반에 속했었는지를 나타내는 5개의 정수가 빈칸 하나를 사이에 두고 주어진다. 주어지는 정수는 모두 1 이상 9 이하의 정수이다.

출력 형식

출력 파일의 이름은 OUTPUT.TXT이다. 첫 줄에 임시 반장으로 정해진 학생의 번호를 출력한다. 단, 임시 반장이 될 수 있는 학생이 여러 명인 경우에는 그 중 가장 작은 번호만 출력한다.

입력과 출력의 예

입력(INPUT.TXT)

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

출력(OUTPUT.TXT)

```
4
```

풀이

문제에서 요구하는 조건대로 가장 많은 학생들과 같은 반을 했었던 학생을 찾아주면 되는 문제이다. 각 학생들에 대해서, 그 학생과 한 번이라도 같은 반을 했었던 학생의 수를 세고, 그러한 학생이 가장 많은 학생들을 찾아서 출력하면 된다. 이 때 이러한 학생이 여러 명일 때에는 문제의 조건에 따라서 가장 작은 번호를 출력해야 한다.

이 때 주의할 것은, 어떤 두 학생이 두 번 이상 같은 반을 했어도 한 번만 세어야 한다는 것이다. 따라서 매 학년마다 같은 반이었던 학생을 찾는 방법을 사용하지 말고, 반대로 각 학생마다 같은 반이었던 해가 있었는지를 확인하는 방식이 좋다. 즉, 각 학생을 차례로 살펴보면서, 그 학생과 반장 후보 학생이 한 번이라도 같은 반이었던 학년이 있었는지를 검사하는 것이다.

```
#include <cstdio>
#include <algorithm>
using namespace std;

int ban[1001][6];
int cou,i,j,k,n,vmax,sol;

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

	scanf("%d",&n);

	for(i=1;i<=n;i++)
		for(j=1;j<=5;j++)
			scanf("%d",&ban[i][j]);

	vmax=-1;
	for(i=1;i<=n;i++)
	{
		cou=0;
		for(j=1;j<=n;j++)
			for(k=1;k<=5;k++)
				if (ban[i][k]==ban[j][k])
				{
					cou++;
					break;
				}
		if (cou>vmax)
		{
			vmax=cou;
			sol=i;
		}
	}

	printf("%d\n",sol);
	return 0;
}
```

빙산

지구 온난화로 인하여 북극의 빙산이 녹고 있다. 빙산을 그림 1과 같이 2차원 배열에 표시한다고 하자. 빙산의 각 부분별 높이 정보는 배열의 각 칸에 양의 정수로 저장된다. 빙산 이외의 바다에 해당되는 칸에는 0이 저장된다. 그림 1에서 빈칸은 모두 0으로 채워져 있다고 생각한다.

	2	4	5	3		
	3		2	5	2	
	7	6	2	4		

그림 1. 행의 개수가 5이고 열의 개수가 7인 2차원 배열에 저장된 빙산의 높이 정보

빙산의 높이는 바닷물에 많이 접해있는 부분에서 더 빨리 줄어들기 때문에, 배열에서 빙산의 각 부분에 해당되는 칸에 있는 높이는 일년마다 그 칸에 동서남북 네 방향으로 붙어있는 0이 저장된 칸의 개수만큼 줄어든다. 단, 각 칸에 저장된 높이는 0보다 더 줄어들지 않는다. 바닷물은 호수처럼 빙산에 둘러싸여 있을 수도 있다. 따라서 그림 1의 빙산은 일년 후에 그림 2와 같이 변형된다.

그림 3은 그림 1의 빙산이 2년 후에 변한 모습을 보여준다. 2차원 배열에서 동서남북 방향으로 붙어있는 칸들은 서로 연결되어 있다고 말한다. 따라서 그림 2의 빙산은 한 덩어리이지만, 그림 3의 빙산은 세 덩어리로 분리되어 있다.

		2	4	1		
	1		1	5		
	5	4	1	2		

그림 2

			3			
				4		
	3	2				

그림 3

한 덩어리의 빙산이 주어질 때, 이 빙산이 두 덩어리 이상으로 분리되는 최초의 시간(년)을 구하는 프로그램을 작성하시오. 그림 1의 빙산에 대해서는 2가 답이다. 만일 전부 다 녹을 때까지 두 덩어리 이상으로 분리되지 않으면 프로그램은 0을 출력한다.

실행 파일의 이름은 BB.EXE로 하고, 실행시간은 1초를 넘을 수 없다. 부분점수는 없다.

입력 형식

입력 파일의 이름은 INPUT.TXT이다. 첫 줄에는 이차원 배열의 행의 개수와 열의 개수를 나타내는 두 정수 N과 M이 한 개의 빈칸을 사이에 두고 주어진다. N과 M은 3 이상 300 이하이다. 그 다음 N개의 줄에는 각 줄마다 배열의 각 행을 나타내는 M개의 정수가 한 개의 빈 칸을 사이에 두고 주어진다. 각 칸에 들어가는 값은 0 이상 10 이하이다. 배열에서 빙산이 차지하는 칸의 개수, 즉, 1 이상의 정수가 들어가는 칸의 개수는 10,000개 이하이다. 배열의 첫 번째 행과 열, 마지막 행과 열에는 항상 0으로 채워진다.

출력 형식

출력 파일의 이름은 OUTPUT.TXT이

다. 첫 줄에 빙산이 분리되는 최초의 시간(년)을 출력한다. 만일 빙산이 다 녹을 때까지 분리되지 않으면 0을 출력한다.

입력과 출력의 예

입력(INPUT.TXT)

```
5 7
0 0 0 0 0 0 0
0 2 4 5 3 0 0
0 3 0 2 5 2 0
0 7 6 2 4 0 0
0 0 0 0 0 0 0
```

출력(OUTPUT.TXT)

```
2
```

풀이

빙산의 모양과 빙산이 녹는 규칙이 주어졌을 때 언제 분리가 일어나는지 계산하는 문제이다. 이는 바로 답을 구할 수 있는 것은 아니고, 시간이 진행되면서 빙산의 모습이 어떻게 바뀌어 나가는지를 시뮬레이션 하는 방식으로 처리할 수 있다.

먼저 각각의 칸에 대해서 동서남북 네 방향의 0의 개수를 세어서 1년 후 빙산의 모습을 구할 수 있다. 이 때 주의해야 할 점은 모든 빙산이 한꺼번에 녹아야 한다는 점이다. 따라서 배열을 하나 유지하면서 순차적으로 값을 변경하는 방식으로 처리해서는 안 되고, 두 개의 배열 등을 사용하여 빙산의 높이를 일괄적으로 낮추어 주는 방식을 사용해야 한다.

이러한 과정을 반복할 때마다 빙산이 두 부분으로 분리되었는지를 알아보아야 한다. 이는 2차원 배열 상에서 깊이우선탐색(DFS)이나 너비우선탐색(BFS)을 이용해서 해결을 할 수 있다. 먼저 배열을 한 번 살펴보면서 빙산에 해당하는 임의의 칸을 하나 찾는다. 이제 이 칸에 대해서 탐색을 수행하는데, 이 칸과 이웃한 빙산 칸을 차례대로 모두 체크해 나가면서 더 이상 이웃한 빙산이 없을 때까지 반복한다. 이러한 탐색 과정이 종료된 후, 먼저 빙산에 해당하는 칸이 모두 체크되었는지 살펴본다. 만일 모든 빙산이 다 체크되었으면 빙산이 한 덩어리로 이루어진 것이고, 아직 탐색되지 않은 칸이 있다면 빙산이 두 부분 이상으로 분리되었다는 것이다. 이와 같이 빙산이 두 부분 이상으로 분리된 경우에는 지금까지 걸린 시간을 출력하고 끝내면 된다.

이러한 과정을 수행하다 보면, 항상 한 덩어리의 빙산만이 유지되는 경우가 있다. 즉, 빙산이 한 덩이로 유지되고 모두 녹아버리는 경우가 발생할 수 있는데, 앞의 과정에서 빙산 칸이 하나도 발견되지 않은 경우에 이에 해당한다. 이러한 경우가 발생하였을 때에는 0을 출력해 주어야 한다.

```
#include <stdio.h>
#include <conio.h>

#define N 1100

int n,m;
int q[N][N];
int visit[N][N];
int ans;
int dir[4][2]={{0,1},{1,0},{-1,0},{0,-1}};
int coord[N*N][3];
int cc;
int queue[N*N][2];
int qh,qc;

void input() {
        FILE *in;
        int i,j;
        in=fopen("input.txt","rt");
        fscanf(in,"%d %d",&n,&m);
        for(i=0;i<n;i++) {
                for(j=0;j<m;j++) {
                        fscanf(in,"%d",&q[i][j]);
                        if (q[i][j]!=0) {
                                coord[cc][0]=i;
                                coord[cc][1]=j;
                                cc++;
                        }
                }
        }
        fclose(in);
}

void melt() {
        int i,j;
        int y,x;
        int px,py;
        for(i=0;i<cc;i++) {
                y=coord[i][0];
                x=coord[i][1];
                coord[i][2]=0;
                for(j=0;j<4;j++) {
                        px=x+dir[j][0];
                        py=y+dir[j][1];
                        if (q[py][px]==0) {
                                coord[i][2]++;
                        }
                }
        }
        for(i=0;i<cc;i++) {
                y=coord[i][0];
```

```
                x=coord[i][1];
                if (q[y][x]<coord[i][2]) q[y][x]=0;
                else q[y][x]-=coord[i][2];
        }
}

void bfs(int y,int x,int p) {
        int py,px;
        int i;
        queue[0][0]=y;
        queue[0][1]=x;
        visit[y][x]=p;
        qc=1;
        qh=0;
        for(;;) {
                y=queue[qh][0];
                x=queue[qh][1];
                for(i=0;i<4;i++) {
                        px=x+dir[i][0];
                        py=y+dir[i][1];
                        if (q[py][px]==0) continue;
                        if (visit[py][px]==p) continue;
                        visit[py][px]=p;
                        queue[qc][0]=py;
                        queue[qc][1]=px;
                        qc++;
                }
                qh++;
                if (qh==qc) break;
        }
}

int component(int p) {
        int i;
        int x,y;
        int cnt;
        cnt=0;
        for(i=0;i<cc;i++) {
                y=coord[i][0];
                x=coord[i][1];
                if (q[y][x]==0) continue;
                if (visit[y][x]==p) continue;
                if (cnt==1) return 2;
                bfs(y,x,p);
                cnt++;
        }
        return cnt;
}

void proc() {
        int i,j;
```

```
		int v;
		for(i=0;i<n;i++) {
			for(j=0;j<m;j++) {
				visit[i][j]=0;
			}
		}
		for(i=1;;i++) {
			melt();
			v=component(i);
			if (v==0) {
				ans=0;
				break;
			}
			else if (v==2) {
				ans=i;
				break;
			}
		}
}

void output() {
		FILE *out;
		out=fopen("output.txt","wt");
		fprintf(out,"%d",ans);
		fclose(out);
}

int main() {
		input();
		proc();
		output();
		return 0;
}
```

마법 색종이

$1cm$ 간격으로 수평선과 수직선이 그어진 직사각형 모양의 마법 색종이를 가로 방향과 세로 방향으로 잘라서 작은 직사각형 조각들로 나누려고 한다. 처음에는 흰색 색종이 한 조각이 주어진다. 이 후로 한 점씩 주어질 때마다 아래의 규칙을 따르면서 색종이 조각을 자른다.

(1) 주어진 점이 흰색 색종이 조각에 포함되면, 그 점을 지나는 '수평선'을 따라 자른다. 이때 잘려진 두 조각 모두 '검정색'으로 색깔이 변한다.
(2) 주어진 점이 검정색 색종이 조각에 포함되면, 그 점을 지나는 '수직선'을 따라 자른다. 이때 잘려진 두 조각 모두 '흰색'으로 색깔이 변한다.

흰색 마법 색종이의 가로 길이와 세로 길이, 그리고 색종이 안에 점들이 순서를 가지고 주어질 때, 위의 규칙을 따라 색종이를 모두 자른 후 만들어지는 색종이 조각들 중에서 넓이가 가장 큰 조각과 넓이가 가장 작은 조각의 넓이를 각각 구하는 프로그램을 작성하시오. 단, 입력으로 주어지는 점들 중 어떤 두 점도 같은 수평선 위에 놓이거나 같은 수직선 위에 놓이지 않는다. 또한 맨 처음 주어진 색종이의 둘레에 위치하는 점들도 입력되지 않는다.

색종이의 가로 길이와 세로 길이는 모두 양의 정수로 주어진다. 그리고 색종이를 자르기 위해서 주어지는 점들의 위치는 자르기 전 색종이의 왼쪽 아래 꼭지점에서 가로 방향으로 떨어진 거리와 세로 방향으로 떨어진 거리를 나타내는 두개의 정수로 주어진다. 예를 들어 아래 그림들에서 점 1의 위치는 (5,4)이다.

만약 가로의 길이와 세로의 길이가 각각 8과 7인 색종이를 자르기 위한 점들의 위치가 (5,4), (2,3), (3,1), (7,6), (6,2)의 순서로 주어진다고 하자. 먼저 첫째 점에 의해서 색종이는 그림 1과 같이 두 검정색 조각으로 잘려진다.

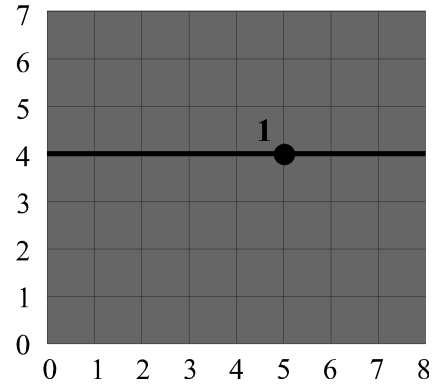


그림 1

이어서 두 번째 점에 의해서 그림 1의 아래쪽 검정색 조각이 그림 2와 같이 두 흰색 조각으로 나뉘어 진다.

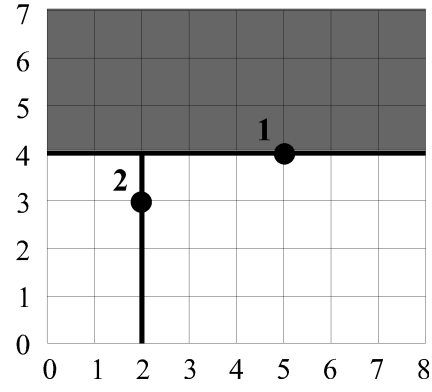


그림 2

나머지 점들에 대해서도 모두 위와 같은 방식으로 색종이를 자르면 그림 3과

같이 된다. 이 경우 가장 큰 색종이 조각의 넓이는 $21cm^2$ 이고 가장 작은 색종이 조각의 넓이는 $3cm^2$이다.

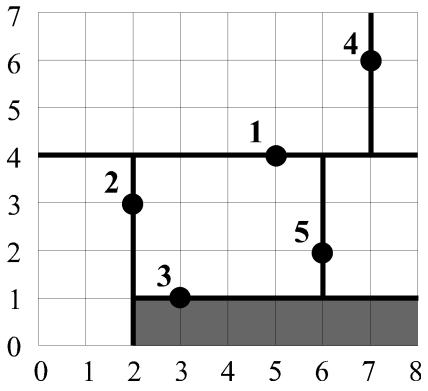


그림 3

실행 파일의 이름은 CC.EXE로 하고, 실행시간은 1초를 넘을 수 없다. 부분점수는 없다.

입력 형식

입력 파일의 이름은 INPUT.TXT이다. 첫째 줄에는 색종이의 가로의 길이와 세로의 길이를 나타내는 양의 정수가 빈칸을 사이에 두고 주어진다. 가로와 세로의 길이는 모두 40,000 이하이다. 둘째 줄에는 색종이를 자르기 위한 점의 개수를 나타내는 양의 정수 N이 주어진다. N은 30,000 이하이다. 셋째 줄부터 마지막 줄까지 색종이를 자르기 위한 점들의 위치가 한 줄에 하나씩 순서대로 주어진다. 점들의 위치는 빈칸을 사이에 두고 가로 방향의 거리가 주어진 다음 세로 방향의 거리가 주어진다.

출력 형식

출력 파일의 이름은 OUTPUT.TXT이다. 첫째 줄에 잘려진 색종이 조각 중에서 가장 넓이가 큰 조각의 넓이와 가장 작은 조각의 넓이가 몇 cm^2인지를 빈칸을 사이에 두고 순서대로 출력한다.

입력과 출력의 예

입력(INPUT.TXT)

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

출력(OUTPUT.TXT)

```
21 3
```

※ 비주얼 베이직 사용자 주의 사항

비주얼 베이직에서 색종이 조각의 넓이를 저장하기 위한 변수를 INTEGER 형으로 선언하면 그 변수에 큰 정수를 저장할 수 없어서 오류가 발생할 수 있다. 이 경우 LONG 형으로 선언하거나 혹은 변수형을 지정하지 않으면 해결된다.

풀이

이 문제는 색종이를 자르는 규칙이 주어졌을 때, 잘려진 색종이의 모양이 어떻게 되는지 판단하는 문제이다. 주어지는 점을 순서대로 처리하면서, 각각의 경우에 규칙에 따라서 어떤 형태로 색종이를 자르는지 확인하고, 이를 처리하는 방식으로 프로그래밍 하면 된다.

이러한 과정은 분할 정복 방법을 이용하여 해결할 수 있다. 먼저 현재 추가되는 점을 기준으로 수평선 또는 수직선을 따라 색종이를 잘라야 한다. 따라서 색종이가 두 개로 쪼개지게 되는데, 이제 나머지 점들이 잘려진 두 개의 색종이 조각 중 어느 쪽에 속하는지를 판별하여 점들을 두 그룹으로 나눌 수 있다. 이렇게 점을 두 개의 그룹으로 나누게 되면, 원래의 문제가 크기가 작은 두 개의 문제로 나뉘었다는 것을 알 수 있다.

이제 크기가 작은 각각의 문제에 대해서 같은 방식으로 계속 처리를 해 주면 된다. 즉, 각각의 경우에도 똑같은 방법으로 문제를 해결하는 재귀적인 방법을 사용하면 평균 $O(n\log n)$의 시간복잡도에 해결할 수 있다. 최초의 경우에는 전체 색종이에 대한 처리를 수행하도록 하면 된다.

```
#include <stdio.h>

int w,h,n,m;
int x[35000],y[35000],b[35000];
int min,max;

void divide(int m,int a[],int x1,int y1,int x2,int y2,int sw) {
	int i,l,r;
	int p=0,cnt=0x7fffffff;
	if(m==0) {
		if(max<(x2-x1)*(y2-y1))max=(x2-x1)*(y2-y1);
		if(min>(x2-x1)*(y2-y1))min=(x2-x1)*(y2-y1);
		return;
	}
	for(i=0;i<m;i++)
		if(cnt>a[i]) cnt=a[i],p=i;
	cnt=a[0],a[0]=a[p],a[p]=cnt;
	l=1,r=m-1;
	switch(sw){
	case 0:
		while(l<m && y[a[l]]<y[a[0]])l++;
		while(r>=0 && y[a[r]]>y[a[0]])r--;
		while(l<r) {
			while(l<m && y[a[l]]<y[a[0]])l++;
			while(r>=0 && y[a[r]]>y[a[0]])r--;
			if(l>=r)break;
			cnt=a[l],a[l]=a[r],a[r]=cnt;
		}
		divide(l-1,a+1,x1,y1,x2,y[a[0]],1);
		divide(m-l,a+l,x1,y[a[0]],x2,y2,1);
		break;
	case 1:
		while(l<m && x[a[l]]<x[a[0]])l++;
		while(r>=0 && x[a[r]]>x[a[0]])r--;
		while(l<r) {
			while(l<m && x[a[l]]<x[a[0]])l++;
			while(r>=0 && x[a[r]]>x[a[0]])r--;
			if(l>=r)break;
			cnt=a[l],a[l]=a[r],a[r]=cnt;
		}
		divide(l-1,a+1,x1,y1,x[a[0]],y2,0);
		divide(m-l,a+l,x[a[0]],y1,x2,y2,0);
		break;
	}
}

int main() {
	int i;
	FILE *in;
	in=fopen("input.txt","rt");
	fscanf(in,"%d%d",&w,&h);
```

```
	fscanf(in,"%d",&n);
	for(i=0;i<n;i++) {
		fscanf(in,"%d%d",&x[i],&y[i]);
		b[i]=i;
	}
	fclose(in);
	min=0x7fffffff,max=0x80000000;
	divide(n,b,0,0,w,h,0);
	in=fopen("output.txt","wt");
	fprintf(in,"%d %d",max,min);
	fclose(in);
	return 0;
}
```

후보 추천하기

정보초등학교 학생회장 후보는 일정 기간 동안 전체 학생의 추천에 의하여 정해진 수만큼 선정된다. 그래서 학교 홈페이지에 추천받은 학생의 사진을 게시할 수 있는 사진틀을 후보의 수만큼 만들었다. 추천받은 학생의 사진을 사진틀에 게시하고 추천받은 횟수를 표시하는 규칙은 다음과 같다.

(규칙 1) 학생들이 추천을 시작하기 전에 모든 사진틀은 비어있다.
(규칙 2) 어떤 학생이 특정 학생을 추천하면, 추천받은 학생의 사진이 반드시 사진틀에 게시되어야 한다.
(규칙 3) 비어있는 사진틀이 없는 경우에는 현재까지 추천 받은 횟수가 가장 적은 학생의 사진을 삭제하고, 그 자리에 새롭게 추천받은 학생의 사진을 게시한다. 이 때, 현재까지 추천 받은 횟수가 가장 적은 학생이 두 명 이상일 경우에는 게시된 지 가장 오래된 사진을 삭제한다.
(규칙 4) 현재 사진이 게시된 학생이 다른 학생의 추천을 받은 경우에는 추천받은 횟수만 증가시킨다.
(규칙 5) 사진틀에 게시된 사진이 삭제되는 경우에는 해당 학생이 추천받은 횟수는 0으로 바뀐다.

후보의 수 즉, 사진틀의 개수와 전체 학생의 추천 결과가 추천받은 순서대로 주어졌을 때, 최종 후보가 누구인지 결정하는 프로그램을 작성하시오.

실행 파일의 이름은 AA.EXE로 하고, 실행시간은 1초를 넘을 수 없다. 부분 점수는 없다.

입력 형식

입력 파일의 이름은 INPUT.TXT이다. 첫째 줄에는 사진틀의 개수를 나타내는 자연수가 주어진다. 둘째 줄에는 전체 학생의 총 추천 횟수가 주어지고, 셋째 줄에는 추천받은 학생을 나타내는 번호가 빈 칸 하나를 사이에 두고 추천받은 순서대로 주어진다. 단, 사진틀의 개수는 20개 이하이고, 총 추천 횟수는 1000번 이하이다. 학생을 나타내는 번호는 1부터 100까지의 자연수이다.

출력 형식

출력 파일의 이름은 OUTPUT.TXT이다. 사진틀에 사진이 게재된 최종 후보의 학생 번호를 빈 칸 하나를 사이에 두고 출력한다. 단, 출력순서는 상관없다.

입력과 출력의 예

입력(INPUT.TXT)

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

출력(OUTPUT.TXT)

```
2 7 6
```

풀이

각 사진틀에 사진이 게시된 시각, 추천 수, 학생의 번호를 기록해 두면서, 추천 받은 학생 번호를 순서대로 처리해 나가면 되는 문제이다. 각 학생들의 추천을 차례대로 살펴보면서, 문제에 주어진 규칙에 맞춰서 처리하는 방식으로 프로그래밍 하면 된다.

이러한 과정을 수행하다 보면, 사진틀에 더 이상 빈자리가 없을 때 삭제할 기존 사진을 선택해야 한다. 이는 일반적인 순차 검색으로 처리할 수 있는데, 추천 수가 가장 낮은 것을 선택하고, 같은 것이 있으면 그 중에 사진이 게시된 시간이 먼저인 것으로 고르면 된다. 따라서 각각의 사진에 대해서 그 사진이 어떤 후보의 사진인지를 저장하고, 추천 수에 대한 정보와 게시된 시간에 대한 정보를 함께 저장해야 한다.

몇 가지 주의할 점이 있는데, 먼저추천을 받은 학생이 사진 틀 수보다 적을 때 출력을 조심해야 한다. 또한 여러 종류의 배열에 정보를 유지해야 하므로, 초기화와 관련된 부분을 유의해야 한다. 예를 들어, 사진틀에서 사진을 삭제할 때 추천수를 초기화하는 것을 잊어서는 안 된다.

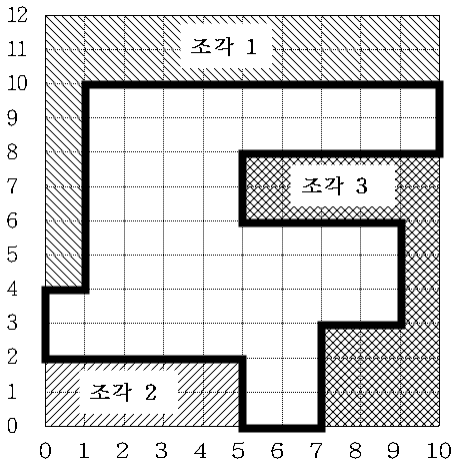
```
#define _CRT_SECURE_NO_DEPRECATE
#include <stdio.h>

struct { int std,vote,time; } cand[100001];

int main()
{
	FILE* fin = fopen("input.txt","r");
	int n,m,k,i,j;
	fscanf(fin,"%d",&n);
	m=0;
	fscanf(fin,"%d",&k);
	for (i=0;i<k;i++)
	{
		int std;
		fscanf(fin,"%d",&std);
		for (j=0;j<m;j++) if (cand[j].std==std)
		{
			cand[j].vote++;
			break;
		}
		if (j<m) continue;

		if (m<n)
		{
			j=m;
			m++;
		}
		else
		{
			j=0;
			for (int t=1;t<m;t++)
				if (cand[t].vote<cand[j].vote ||
					cand[t].vote==cand[j].vote && cand[t].time<cand[j].time)
				{
					j=t;
				}
		}
		cand[j].std=std;
		cand[j].vote=1;
		cand[j].time=i;
	}
	fclose(fin);
	FILE* fout = fopen("output.txt","w");
	for (i=0;i<m;i++)
		if (i<m-1) fprintf(fout,"%d ",cand[i].std);
		else fprintf(fout,"%d\n",cand[i].std);
	fclose(fout);
	return 0;
}
```

모눈종이 자르기

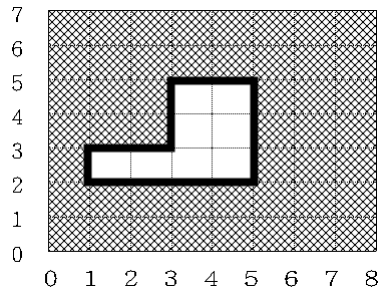
1㎝ 간격으로 가로선과 세로선이 그어진 직사각형 모양의 모눈종이 위에 한 개의 다각형이 그려져 있다. 다각형의 모든 변은 모눈종이에 그어진 선 위에 있다. 다각형의 변을 따라 모눈종이에서 다각형을 오려내고 나면 하나 이상의 조각들이 남는다.

모눈종이의 가로 길이, 세로 길이와 다각형 한 개가 주어질 때, 이 다각형을 오려낸 후 남은 조각들의 개수와 둘레가 가장 긴 조각의 둘레 길이를 출력하는 프로그램을 작성하시오.

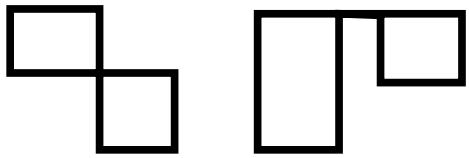
예를 들어, 다음 그림과 같이 모눈종이에 그려진 다각형을 오려내면, 세 개의 조각이 만들어 진다. 조각 1, 조각 2, 조각 3의 둘레 길이는 각각 36㎝, 14㎝, 30㎝이므로, 가장 둘레가 긴 조각의 둘레 길이는 36㎝이다.



모눈종이에서 다각형을 오려낸 후 남은 조각에 구멍이 있는 경우, 이 조각의 둘레 길이는 모눈종이의 둘레 길이와 다각형의 둘레 길이의 합이다. 예를 들어, 다음 그림에서 조각의 둘레의 길이는 44㎝이다.



단, 아래 그림과 같이 두 꼭지점의 위치가 같거나 두 변이 서로 교차하는 경우, 두 변이 일부라도 서로 겹치는 경우는 모두 다각형이 아니다.



모눈종이에 그려진 다각형은 임의의 꼭지점부터 시작하여 시계반대방향으로 차례로 주어진 꼭지점들의 위치에 의하여 결정된다. 다각형의 각 꼭지점의 위치는 모눈종이의 왼쪽 아래 꼭지점으로부터 오른쪽으로 몇 칸, 위쪽으로 몇 칸 떨어져 있는 지를 나타내는 두 수로 주어진다.

실행 파일의 이름은 BB.EXE로 하고, 실행시간은 0.5초를 넘을 수 없다. 부분점수는 없다.

입력 형식

입력 파일의 이름은 INPUT.TXT이다. 첫째 줄에는 모눈종이의 가로 길이와 세로 길이를 나타내는 자연수가 빈칸을 사이에 두고 주어진다. 가로와 세로의 길이는 모두 200000 이하이다. 둘째 줄에는 오려내고자 하는 다각형의 꼭지점

개수를 나타내는 500000 이하의 자연수가 주어진다. 셋째 줄부터 마지막 줄까지 다각형의 꼭지점의 위치가 시계반대방향으로 한 줄에 하나씩 차례대로 주어진다. 다각형 꼭지점의 위치는 모눈종이의 왼쪽 아래 꼭지점으로부터 오른쪽으로 몇 칸, 위쪽으로 몇 칸 떨어져 있는 지를 나타내는 두 수가 빈칸을 사이에 두고 차례로 주어진다.

출력 형식

출력 파일의 이름은 OUTPUT.TXT이다. 첫째 줄에 다각형을 오려낸 후 남은 조각들의 개수와 이들 중 가장 긴 둘레의 길이를 출력한다. 단 길이의 단위인 ㎝는 출력하지 않는다.

입력과 출력의 예(1)

입력(INPUT.TXT)

```
10 12
14
9 6
5 6
5 8
10 8
10 10
1 10
1 4
0 4
0 2
5 2
5 0
7 0
7 3
9 3
```

출력(OUTPUT.TXT)

```
3 36
```

입력과 출력의 예(2)

입력(INPUT.TXT)

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

출력(OUTPUT.TXT)

```
1 44
```

풀이

모눈종이에서 수직선, 수평선으로 이루어진 다각형을 잘라낸 후 남은 조각들 중 가장 큰 것을 구하는 문제이다. 다각형의 꼭지점의 수에 비례하는 시간 복잡도에 해결할 수 있다.

크게 두 가지 경우로 나눠서 생각할 수 있는데, 먼저 오려내는 다각형의 꼭지점들 중 모눈종이의 변 위에 있지 않은 경우가 있다. 이러한 경우를 잘 따져보면, 다각형을 잘라 낸 후 남은 조각은 한 개라는 사실을 알 수 있다. 또한, 이 때 남게 되는 조각의 둘레의 합은 모눈종이의 둘레의 길이와 다각형의 둘레의 길이의 합이 된다는 점도 알 수 있다.

다음으로, 오려내는 다각형의 꼭지점들 중 모눈종이의 변 위에 있는 꼭지점이 있는 경우가 있다. 이 경우의 처리는 앞의 경우보다 복잡한데, 먼저 다각형의 변을 따라 가면서 모눈종이 내부로 처음 들어가는 변을 찾아야 한다. 다음으로는 이 변에서부터 시작하여 이 다각형의 변을 반시계방향으로 따라가는 과정을 수행한다. 이러한 과정을 수행할 때, 만일 모눈종이의 변을 중간에 만나게 되면 하나의 다각형 조각이 만들어짐을 알 수 있다. 이제 이러한 과정을 반복하면서 변의 길이가 최대인 조각을 구하면 된다.

```
#include <stdio.h>
#include <time.h>

#define MAXN 700004
FILE *In = fopen( "input.txt", "r" );
FILE *Out = fopen("output.txt", "w" );

int Lx, Ly;
int N;
int Data[ MAXN + 2][ 2];
int AnsNum = 0, AnsMax = 0;
int ttt = clock();

void input()
{
	int i;
	fscanf( In, "%d %d\n", &Lx, &Ly );
	fscanf( In, "%d\n", &N );

	for( i =  1; i <= N; i ++ ){
		fscanf( In, "%d %d\n", &Data[ i][ 0], &Data[ i][ 1] );
	}
	Data[ N+1][ 0] = Data[ 1][ 0];
	Data[ N+1][ 1] = Data[ 1][ 1];

	fclose( In );
}
int Abs( int a )
{
	if( a < 0 ) return a*(-1);
	return a;
}
int length( int x1, int y1, int x2, int y2 )
{
	int sum = 0;
	while( 1 ){
		if( y1 == 0 && x1 != Lx ){
			if( y2 == 0 && x2 > x1 ){
				sum += (x2-x1);
				return sum;
			}
			else {
				sum += ( Lx - x1 );
				x1 = Lx;
			}
		}
		else if( x1 == Lx && y1 != Ly){
			if( x2 == Lx && y2 > y1 ){
				sum += (y2-y1);
				return sum;
			}
```

```
                        else {
                                sum += ( Ly - y1 );
                                y1 = Ly;
                        }
                }
                else if( y1 == Ly && x1 != 0 ){
                        if( y2 == Ly && x2 < x1 ){
                                sum += ( x1 - x2);
                                return sum;
                        }
                        else {
                                sum += x1;
                                x1 = 0;
                        }
                }
                else if( x1 == 0 && y1 != 0 ){
                        if( x2 == 0 && y2 < y1 ){
                                sum += ( y1 - y2 );
                                return sum;
                        }
                        else {
                                sum += y1;
                                y1 = 0;
                        }
                }
        }
        return 0;
}
void process()
{
        int i, sum = 0, s, tmp;
        bool chk = 0, chk2 = 1;

        for( s = 1; s <= N; s ++ ){
                if(     ( Data[ s][ 0] == Data[ s+1][ 0] && ( Data[ s][ 0] == 0 || Data[ s][ 0] == Lx ) )
                        || ( Data[ s][ 1] == Data[ s+1][ 1] && ( Data[ s][ 1] == 0 || Data[ s][ 1] ==
Ly ) ) ){
                        chk = 1;
                        break;
                }
        }

        if( chk ){
                tmp = s+1; if( tmp > N ) tmp = 1;

                for( i = 1; i <= N; i ++ ){
                        s ++;
                        if( s > N ) s = 1;

                        if(     ( Data[ s][ 0] == Data[ s+1][ 0] && ( Data[ s][ 0] == 0 || Data[ s][ 0]
== Lx ) )
```

```
                                        || ( Data[ s][ 1] == Data[ s+1][ 1] && ( Data[ s][ 1] == 0 || Data[
s][ 1] == Ly ) ) ){

                                        if( chk2 == 0 ){
                                                AnsNum ++;
                                                /*ret = length( Data[ tmp][ 0], Data[ tmp][ 1], Data[ s][
0], Data[ s][ 1] );
                                                if( AnsNum == 1 ){
                                                        firstcutting1 = sum;
                                                        firstcutting2 = ret;
                                                }
                                                sum += ret;*/

                                                sum += length( Data[ tmp][ 0], Data[ tmp][ 1], Data[ s][
0], Data[ s][ 1] );

                                                if( AnsMax < sum )
                                                        AnsMax = sum;

                                                sum = 0;
                                                chk2 = 1;
                                        }
                                        tmp = s+1; if( tmp > N ) tmp = 1;
                                }
                                else {
                                        chk2 = 0;
                                        sum += Abs( Data[ s][ 0] - Data[ s+1][ 0] + Data[ s][ 1] - Data[
s+1][ 1] );
                                }
                        }
                } else {
                        AnsNum = 1;
                        AnsMax = Lx + Lx + Ly + Ly;
                        for( s = 1; s <= N; s ++ ){
                                AnsMax += Abs( Data[ s][ 0] - Data[ s+1][ 0] + Data[ s][ 1] - Data[ s+1][ 1]
);
                        }

                }
}
void output()
{
        fprintf( Out, "%d %d\n", AnsNum, AnsMax );
}
int main()
{
        input();
        process();
        output();

        printf( "%lf", (double)( clock() - ttt ) / 1000 );
```

```
	return 0;
}
```

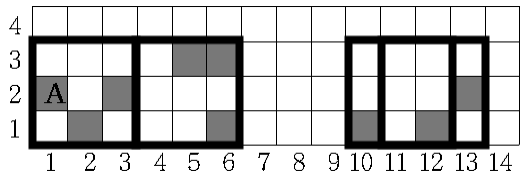
모자이크

수찬이는 선생님을 도와서 교실 벽면을 장식할 모자이크 그림을 그리기로 하였다. 이를 위하여 직사각형 모양의 큰 도화지를 준비하여 교실 벽에 붙이고 1cm 간격으로 가로선과 세로선을 그려서 정사각형 모양의 칸을 만들고, 각 칸마다 같은 색의 물감으로 색칠을 하였다. 그런데 잘못 칠해진 칸이 있음을 발견하게 되었다.

수찬이는 도화지와 색깔이 같은 색종이를 사서 잘못 칠해진 칸에 색종이를 붙이고 다시 그리는 것이 좋겠다고 생각하고 선생님께 상의를 드렸다. 선생님께서는 정해진 장수의 색종이를 사용하여 아래와 같은 조건을 따르면서 잘못 칠해진 칸을 모두 가리되, 가장 작은 색종이의 크기를 구하는 새로운 문제를 내셨다.

(조건 1) 사용되는 색종이는 모두 크기가 같고 정사각형 모양이다.
(조건 2) 색종이 크기는 한 변의 길이로 나타내며, 원하는 크기의 색종이는 모두 구할 수 있다.
(조건 3) 모든 색종이는 반드시 도화지의 밑변에 맞추어 붙인다. 이 때 색종이를 겹쳐서 붙일 수 있다.

도화지 위의 행은 다음 그림과 같이 맨 아래에서 위쪽으로 1번부터 순서대로 번호가 매겨져 있고, 열은 왼쪽에서 오른쪽으로 1번부터 번호가 매겨져 있다. 이 그림은 도화지에 가로선과 세로선을 그어서 4개의 행과 14개의 열, 그리고 56개의 칸으로 나눈 모양을 보여준다. 잘못 칠해진 칸은 회색으로 표시되어 있다.



도화지 위의 칸은 행 번호와 열 번호로 나타낸다. 예를 들어 위 그림에서 가장 왼쪽에 있는 잘못 칠해진 칸 A의 위치는 (2, 1)이다. 위 그림과 같이 도화지에서 잘못 칠해진 칸이 9개 주어지고 색종이 4장을 사용한다면 가장 작은 색종이의 크기는 3cm이다.

도화지의 행의 개수와 열의 개수, 그리고 도화지에 잘못 칠해진 칸들의 위치가 주어질 때, 주어진 장수의 색종이를 사용하여 앞의 세 가지 조건에 따라 모든 잘못 칠해진 칸을 가릴 수 있는 가장 작은 색종이의 크기를 구하는 프로그램을 작성하시오.

실행 파일의 이름은 CC.EXE로 하고, 실행시간은 1초를 넘을 수 없다. 부분점수는 없다.

입력 형식

입력 파일의 이름은 INPUT.TXT이다. 첫째 줄에는 도화지 위의 행의 개수와 열의 개수를 나타내는 자연수가 빈칸을 사이에 두고 주어진다. 행의 개수와 열의 개수는 모두 1000000 이하이다. 둘째 줄에는 사용할 색종이의 장수를 나타내는 자연수가 주어진다. 사용할 색종이는 100장 이하이다. 셋째 줄에는 도화지에 잘못 칠해진 칸의 개수를 나타내는 자연수가 주어진다. 잘못 칠해진 칸은

1000개 이하이다. 넷째 줄부터 마지막 줄까지 잘못 칠해진 칸의 위치가 한 줄에 하나씩 주어진다. 잘못 칠해진 칸의 위치는 빈칸을 사이에 두고 행 번호가 주어진 다음 열 번호가 주어진다.

출력 형식

출력 파일의 이름은 OUTPUT.TXT이다. 첫째 줄에 주어진 장수의 색종이를 사용하여 잘못 칠해진 칸을 모두 가릴 수 있는 가장 작은 색종이의 크기가 몇 cm인지를 나타내는 자연수를 출력한다.

입력과 출력의 예

입력(INPUT.TXT)

```
4 14
4
9
1 2
2 1
2 3
1 6
3 5
1 10
3 6
1 12
2 13
```

출력(OUTPUT.TXT)

```
3
```

풀이

동적 계획법을 이용하여 해결할 수 있다. 가려야 할 칸의 개수를 n, 색종이의 개수를 k라고 하자. 먼저 구현상의 편의를 위해서는 점들이 정렬된 형태로 있는 것이 편리하다. 따라서 입력으로 주어지는 모든 잘못 친해진 칸들을 열을 기준으로 정렬하도록 한다.

이제 D[i][j]를 i번 칸까지 j개의 색종이를 사용하여 가리려고 할 때 필요한 색종이의 최소 크기라고 정의하자. 만일 p(<=i)번 칸부터 i번 칸을 한 개의 색종이로 가린다고 가정하면, D[i][j]는 D[p-1][j-1]과, cost(p, i)중의 큰 값이다. 여기서 cost(p, i)란 p번 칸부터 i번 칸까지 한 개의 색종이를 사용하여 가릴 때 필요한 색종이의 길이인데, p번 칸부터 i번 칸까지의 모든 칸에 대해 즉 x좌표의 최대 최소 차와, y좌표의 최대 최소 차 중 큰 값이 된다. 또한 D[p-1][j-1]은 앞의 칸에 대한 정보인데, p번 칸부터가 하나의 색종이로 가려졌으므로 그 앞부분은 p-1번 칸까지가 된다. 또한 이 부분을 하나의 색종이로 가렸으므로, 그 앞에는 j-1개의 색종이로 가려야 한다.

D[i][j]를 구하기 위해서는 이러한 과정을 1부터 i까지의 모든 p에 대해 수행해야 한다. 마찬가지로, 가능한 i, j의 조합에 대해서도 이러한 과정을 수행해야 한다. 이러한 과정을 전체적으로 수행하여 D 배열을 다 채운 후에, 우리가 구하고자 하는 답은 D[n][k]이 된다.

```
#include <cstdio>
#include <cstdlib>
#include <algorithm>
using namespace std;

#define max_n 1002
#define max_k 102
#define max_len 1000000
#define in_file "input.txt"
#define out_file "output.txt"
const int MAX=999999999;

struct point
{
	int x,y;
};


bool comp(const point &a,const point &b)
{
	if (a.x<b.x) return true; else return false;
}


point p[max_n];
int sol,minx,miny,maxx,maxy,n,k,l,cost,i,j,min_cost,ga,se;
int d[max_n][max_k];


int imax(int a,int b)
{
	if (a>b) return a; else return b;
}

int imin(int a,int b)
{
	if (a>b) return b; else return a;
}

void main(void)
{
	freopen(in_file,"r",stdin);
	freopen(out_file,"w",stdout);

	scanf("%d %d",&se,&ga);
	scanf("%d %d",&k,&n);

	for(i=1;i<=n;i++)
	{
		scanf("%d %d",&p[i].y,&p[i].x);
	}
```

```
	sort(&p[1],&p[n+1],comp);

	fill(d[0],d[n+1],MAX);

	for(i=0;i<=k;i++)
		d[0][i] = 0;

	for(j=1;j<=k;j++)
	{
		for(i=1;i<=n;i++)
		{
			if (j==1 && i==4)
			{
				j=j;
			}
			//get d[i][j]
			minx=p[i].x-1;
			maxx=p[i].x;

			miny=0;
			maxy=p[i].y;
			min_cost = MAX;
			for(l=i;l>=1;l--)
			{
				minx = p[l].x-1;
				maxy = imax(maxy , p[l].y);
				cost = imax( d[l-1][j-1] , imax(maxx-minx , maxy-miny));
				min_cost = imin(min_cost,cost);
			}
			d[i][j] = imin(d[i][j-1], min_cost);

		}
	}
	printf("%d\n",d[n][k]);
}
```

절사평균

체조나 다이빙 등의 경기에서 일부 심판이 자기가 좋아하는 선수에게 높은 점수를, 싫어하는 선수에게 낮은 점수를 주는 경우가 종종 있었다. 따라서 심판들이 주는 점수의 평균점수를 선수에게 주게 되면 공정하지 않은 경우가 생길 수 있다. 이를 방지하기 위하여 절사평균이나 보정평균을 사용한다. 예를 들어 심사위원 일곱 명이 다음과 같이 점수를 주었다고 하자.

9.3, 9.5, 9.6, 9.8, 9.1, 5.0, 9.3

전체의 합이 61.6이 되므로 평균은 8.8이 된다. 이 평균점수는 한 심판이 다른 심판에 비하여 아주 낮은 점수인 5.0을 주어서 나온 결과로, 선수는 매우 불공정하다고 느낄 것이다.

위의 점수를 작은데서 큰 순서로 정렬하면 5.0, 9.1, 9.3, 9.3, 9.5, 9.6, 9.8 이 된다.

이때 **절사평균(7, 2)**는 정렬된 전체 점수 일곱 개 중 양쪽 끝에서 두 개씩을 제외하고 난 9.3, 9.3, 9.5의 평균인 9.37이 된다(소수점이하 셋째 자리에서 반올림). 또 **보정평균(7, 2)**는 정렬된 전체 점수 일곱 개 중 양쪽 끝에서 각각 두 개를, 남은 점수 중 가장 가까운 것으로 교체한 9.3, 9.3, 9.3, 9.3, 9.5, 9.5, 9.5의 평균으로 9.39가 된다(소수점이하 셋째 자리에서 반올림).

*N*개의 점수와 양쪽에서 제외하는 개수 *K* 값이 주어졌을 때 **절사평균(*N*, *K*)**와 **보정평균(*N*, *K*)**를 계산하는 프로그램을 작성하시오.

실행 파일의 이름은 AA.EXE로 하고, 실행시간은 0.5초를 넘을 수 없다. 부분점수가 주어질 수 있다.

입력 형식

입력 파일의 이름은 INPUT.TXT이다. 첫째 줄에 전체 점수의 개수 *N*과 제외되는 점수의 개수 *K*가 빈칸을 사이에 두고 주어진다. *N*은 3 이상 100,000 이하의 자연수이다. *K*는 0 이상 (*N*/2)-1 이하로 주어진다. 그 다음 *N*줄에는 각 심판의 점수가 한 줄에 하나씩 주어진다. 점수는 0 이상 10 이하의 실수로 소수점이하 첫째 자리까지 주어진다.

출력 형식

출력 파일의 이름은 OUTPUT.TXT이다. 첫째 줄에 **절사평균(*N*, *K*)**를, 둘째 줄에 **보정평균(*N*, *K*)**를 각각 소수점이하 셋째 자리에서 반올림하여 둘째 자리까지 출력한다. 예를 들어 결과값이 9.667인 경우 9.67로, 5인 경우 5.00으로, 5.5인 경우에는 5.50으로 출력한다.

입력과 출력의 예

입력(INPUT.TXT)

```
7 2
9.3
9.5
9.6
9.8
9.1
5.0
9.3
```

출력(OUTPUT.TXT)

```
9.37
9.39
```

풀이

주어진 조건대로 절사평균과 보정평균을 계산하면 되는 문제이다. 주어진 점수들을 차례로 정렬하고, 양 끝 K개의 점수를 제거한 뒤 평균을 계산하고, 제거한 점수들을 남은 점수들 중 가장 가까운 것으로 교체한 뒤 평균을 계산해서 두 개의 평균을 출력하면 된다. 가능한 수의 종류가 100종류뿐이므로 가짓수를 세는 O(N) 정렬을 사용할 수도 있다. 혹은, 다양한 O($n\log n$) 정렬을 이용하는 것도 가능하다.

문제에 주어진 조건에 따라서 처리하면 되는 문제지만, 프로그래밍을 할 때에는 몇 가지 주의해야 할 점이 있다. 무엇보다도, N이 10만까지 가능하기 때문에 점수의 합을 구하는 과정에서 실수 오차가 발생할 수 있다. 이를 해결하기 위해서는 다양한 방법이 있는데, 가장 간단한 방법은 주어진 점수를 정수 형태로 표현하는 것이다. 입력으로 주어진 수들에 10을 곱하면 정수 형태로 표현할 수 있고, 이러한 정수 형태로 평균을 계산한 후, 나중에 답을 출력할 때 다시 10으로 나눠서 실수 형태로 변환하는 방법이 가능하다.

```
#include<cstdio>
#include<vector>
#include<algorithm>

using namespace std;

vector<double> a, b, c;
int n, m;

double average(vector<double> a)
{
	double sum = 0;

	for(size_t l1 = 0; l1 < a.size(); l1++)
	{
		sum += a[l1];
	}

	return sum / (double)a.size();
}

int main(void)
{
	int l1;

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

	scanf("%d %d",&n,&m);
	a.resize(n);
	for(l1=0;l1<n;l1++)
	{
		scanf("%lf",&a[l1]);
	}

	sort(a.begin(), a.end());
	for(l1 = m; l1 < n - m; l1++) b.push_back(a[l1]);
	for(l1 = m; l1 < n - m; l1++) c.push_back(a[l1]);
	for(l1=0;l1<m;l1++){ c.push_back(a[m]); c.push_back(a[n-m-1]); }

	printf("%.2lf\n%.2lf\n", average(b), average(c));

	return 0;
}
```

월드컵

월드컵 조별 최종 예선에서는 6개국으로 구성된 각 조별로 동일한 조에 소속된 국가들과 한 번씩, 각 국가별로 총 5번의 경기를 치른다. 조별리그가 끝난 후, 기자가 보내온 각 나라의 승, 무승부, 패의 수가 가능한 결과인지를 판별하려고 한다. 다음은 가능한 결과와 가능하지 않은 결과의 예이다.

<예제 1> 가능한 결과

나라	승	무	패
A	5	0	0
B	3	0	2
C	2	0	3
D	0	0	5
E	4	0	1
F	1	0	4

<예제 2> 가능한 결과

나라	승	무	패
A	4	1	0
B	3	0	2
C	4	1	0
D	1	1	3
E	0	0	5
F	1	1	3

<예제 3> 불가능한 결과

나라	승	무	패
A	5	0	0
B	4	0	1
C	2	2	1
D	2	0	3
E	1	0	4
F	0	0	5

<예제 4> 불가능한 결과

나라	승	무	패
A	5	0	0
B	3	1	1
C	2	1	2
D	2	0	3
E	0	0	5
F	1	0	4

네 가지의 결과가 주어질 때 각각의 결과에 대하여 가능하면 1, 불가능하면 0을 출력하는 프로그램을 작성하시오.

실행 파일의 이름은 BB.EXE로 하고, 실행시간은 0.5초를 넘을 수 없다. 부분점수는 없다.

입력 형식

입력 파일의 이름은 INPUT.TXT이다. 첫째 줄부터 넷째 줄까지 각 줄마다 6개국의 결과가 나라별로 승, 무승부, 패의 순서로 빈칸을 하나 사이에 두고 18개의 숫자로 주어진다.

출력 형식

출력 파일의 이름은 OUTPUT.TXT이다. 입력에서 주어진 네 가지 결과에 대하여 가능한 결과는 1, 불가능한 결과는 0을 빈칸을 하나 사이에 두고 출력한다.

입력과 출력의 예

입력(INPUT.TXT)

```
5 0 0 3 0 2 2 0 3 0 0 5 4 0 1 1 0 4
4 1 0 3 0 2 4 1 0 1 1 3 0 0 5 1 1 3
5 0 0 4 0 1 2 2 1 2 0 3 1 0 4 0 0 5
5 0 0 3 1 1 2 1 2 2 0 3 0 0 5 1 0 4
```

출력(OUTPUT.TXT)

```
1 1 0 0
```

풀이

조별 리그가 끝난 후 그 결과가 가능한 결과인지를 판단하는 문제이다. 기본적으로 승, 무승부, 패의 결과의 합을 따져보면 많은 경우를 확인할 수 있다. 그러나 이러한 경우로는 충분하지 않으므로, 모든 경우를 따져보는 방법의 처리가 필요하다.

이를 처리하기 위해서, 실제로 주어진 결과를 만족시키는 각 팀별 게임 결과의 할당이 가능한지를 퇴각 검색(Backtracking)을 이용하여 판단해 보면 된다. 이 때 가능한 경우의 수를 따져보면 그렇게 많지 않는데, 이는 팀의 수가 6개밖에 되지 않기 때문이다. 따라서 모든 가능한 결과를 생성하여 주어진 표와 일치하는지 확인하는 식으로 처리할 수 있다.

이 때 불필요한 경우에 대해서 탐색을 수행하지 않으면 보다 효율적으로 처리할 수 있다. 예를 들어, 중간에 한 팀의 승, 무승부, 패의 수가 입력으로 주어진 것보다 많아지는 경우에는 더 이상 탐색을 진행하지 않아도 된다.

```
#include <fstream>

using namespace std;

#define n 6   // n denotes the nubmer of teams
#define m 15  // m denotes the number of games = n choose 2

int win[n], lose[n], draw[n];
int genWin[n], genLose[n], genDraw[n];
int gen[m], p1[m], p2[m];
bool feasibility;

void recur( int cnt )
{
	if (cnt == m)
	{
		feasibility = true;
		return;
	}
	int n1 = p1[cnt];
	int n2 = p2[cnt];

	genWin[n1]++; genLose[n2]++;
	if (genWin[n1]<=win[n1] && genLose[n2]<=lose[n2])
		recur(cnt+1);
	genWin[n1]--; genLose[n2]--;

	genDraw[n1]++; genDraw[n2]++;
	if (genDraw[n1]<=draw[n1] && genDraw[n2]<=draw[n2])
		recur(cnt+1);
	genDraw[n1]--; genDraw[n2]--;

	genLose[n1]++; genWin[n2]++;
	if (genLose[n1]<=lose[n1] && genWin[n2]<=win[n2])
		recur(cnt+1);
	genLose[n1]--; genWin[n2]--;
}

void process()
{
	int i, j, cnt = 0;
	feasibility = false;
	for (i=0; i<n; i++)
	{
		genWin[i] = 0;
		genLose[i] = 0;
		genDraw[i] = 0;
		if (win[i]+lose[i]+draw[i] != n-1) return;
	}
	for (i=0; i<n; i++)
		for (j=i+1; j<n; j++)
```

```
                {
                        p1[cnt] = i;
                        p2[cnt] = j;
                        cnt++;
                }
        recur(0);
}

int main()
{
        ifstream in("input.txt");
        ofstream out("output.txt");
        for (int loop = 0; loop<4; loop++)
        {
                for (int i=0; i<n; i++)
                        in >> win[i] >> draw[i] >> lose[i];
                process();
                if (feasibility) out << "1 ";
                else out << "0 ";
        }
        in.close();
        out.close();
        return 0;
}
```

타일 밟기

정보올림피아드 경시장으로 가는 길에 자연수 값이 적힌 타일이 한 줄로 깔려 있다. 철수는 타일에 적힌 자연수 값은 모두 다르고 시작에서부터 증가하는 순서로 깔려있음을 확인하였다.

철수는 임의의 한 타일에서 시작하여 몇 개의 타일을 밟아서 경시장으로 가려고 한다. 단, 타일들에 적힌 숫자가 일정한 자연수만큼 증가하도록 밟으려고 한다.

그러나 철수는 시작하는 위치와 증가하는 값에 따라 연속적으로 밟을 수 있는 타일의 개수가 달라지는 것을 알 수 있었다.

철수는 경시장으로 가면서 위와 같은 방법으로 연속적으로 밟을 수 있는 타일에 적힌 수들의 합 중에서 최대값은 얼마나 될까 계산하여 보기로 하였다. 연속적으로 밟을 수 있는 타일의 개수는 적어도 3개 이상이어야 한다.

예를 들어 타일에 적힌 자연수들의 순서가 다음과 같이 주어졌다고 하자.

1, 2, 6, 7, 11, 12, 13, 15, 17, 20, 23

이 경우 철수가 연속적으로 밟을 수 있는 타일들의 모든 가능한 순서를 열거하면 다음의 표와 같다. 표에 열거한 것 외에 타일을 연속적으로 3개 이상 밟을 수 있는 경우는 없다.

증가하는 값	밟은 타일의 순서	합
1	11, 12, 13	36
2	11, 13, 15, 17	56
3	17, 20, 23	60
4	7, 11, 15	33
5	2, 7, 12, 17	38
6	1, 7, 13	21
	11, 17, 23	51
7	6, 13, 20	39
8	7, 15, 23	45
9	2, 11, 20	33
11	1, 12, 23	36

따라서 이 예에 대해 철수가 경시장으로 가면서 시작하는 타일을 포함하여 연속적으로 밟을 수 있는 타일에 적힌 수들의 합 중에서 최대값은 60이다.

타일에 적힌 수들이 증가하는 순서로 주어질 때, 철수가 연속적으로 밟을 수 있는 타일이 3개 이상 있는 경우, 그렇게 연속적으로 밟을 수 있는 타일에 적힌 수들의 합 중에서 최대값을 출력하는 프로그램을 작성하시오. 연속해서 타일을 3개 이상 밟을 수 있는 경우가 존재하지 않으면 0을 출력한다.

실행 파일의 이름은 CC.EXE로 하고, 실행시간은 0.3초를 넘을 수 없다. 부분점수는 없다.

입력 형식

입력 파일의 이름은 INPUT.TXT이다. 첫째 줄에는 타일의 개수 N이 주어진다. N은 3 이상 3,000 이하이다. 둘째 줄에는 N개의 타일에 적힌 자연수들이 증가하는 순서로 빈칸을 사이에 두고 주어진다. 타일에 적힌 자연수들은 각각 1,000,000 이하이다.

출력 형식

출력 파일의 이름은 OUTPUT.TXT이다. 철수가 연속적으로 3개 이상 밟을 수 있는 타일이 존재하면, 그렇게 연속적으로 밟은 타일에 적힌 수들의 합 중에서 최대값을 첫째 줄에 출력하시오. 연속해서 타일을 3개 이상 밟을 수 있는 경우가 존재하지 않으면 0을 출력한다.

단, 결과 값이 2,000,000,000 이하인 입력 데이터만이 주어질 것이다.

입력과 출력의 예

입력(INPUT.TXT)

```
11
1 2 6 7 11 12 13 15 17 20 23
```

출력(OUTPUT.TXT)

```
60
```

풀이

이 문제는 여러 자연수들이 주어졌을 때, 그 자연수들로 만들 수 있는 등차수열들 중, 그 합이 최대가 되는 경우를 찾는 문제이다. 등차수열은 몇 개의 자연수를 나열한 수열의 일종으로, 인접한 두 수의 차이가 항상 일정한 수열이다. 이는 문제의 조건에서 타일을 밟을 때 일정한 자연수만큼 증가하도록 밟는다는 조건에 해당한다. 이 문제를 풀기 위해 여러 가지 방법을 생각해 볼 수 있으나, 제한 시간이 매우 빠듯한 문제이므로 O(n^2) 동적 계획법으로 해결해야 제한 시간 내에 답을 구할 수 있다.

문제에서 주어진 원래 수열을 S[1], S[2], .., S[n]이라고 두자. T[i][j]를 i번째 수와 j번째 수를 첫 두 항으로 갖는 등차수열의 최대 합으로 정의한다. 다음으로, 모든 i와 j에 대해서 실제로 S[i], S[j], S[k]가 등차수열이 되는 k의 위치를 미리 구해 놓는다. 문제의 조건에서 모든 자연수가 다르다고 하였으므로, 이러한 k가 존재한다면 항상 하나뿐이다. 따라서 먼저 수열의 차이 S[j]-S[i]를 계산하고, S[j]+(S[j]-S[i])와 값이 일치하는 k가 존재하는지 확인한다. 만일 이러한 k가 존재하는 경우라면, 이제 T[i][j]는 S[i]+S[j]+S[k]이거나 (항이 정확히 세 개인 경우) S[i] + T[j][k]일 것이므로 (항이 세 개보다 많은 경우), 둘 중에 큰 값을 저장하면 된다. 전자의 경우에는 단순히 세 개의 수로 등차수열을 만드는 것이고, 후자의 경우에는 j번째 수와 k번째 수를 첫 두 항으로 갖는 등차수열을 고려하는 경우이다.

이러한 방식으로 모든 T[i][j]값을 계산하고, 이들 중 가장 큰 값이 우리가 구하고자 하는 답이 된다. 계산을 할 때에는 순서에 주의해야 하는데, T[i][j]를 계산하기 위해서 고려하는 k가 i, j보다 크다는 것을 알 수 있다. 따라서 T 배열을 채울 때에는 역순으로 채워야 한다.

```
#include <fstream>
#include <iostream>
#define _max 3000
#define _range 1000001

using namespace std;

void readData(int &n, int* seq);
void writeData(int solution);
void procData(int n, int* seq, int& solution);

int table[_max][_max];
int map[_range];

int main()
{
	int n, sequence[_max];
	int solution;

	readData(n, sequence);
	procData(n, sequence, solution);
	writeData(solution);

	return 0;
}

void readData(int &n, int* seq)
{
	ifstream infile("input.txt");
	infile >> n;
	for(int i = 0; i < n; i++)
	{
		infile >> seq[i];
	}
	infile.close();
}

void writeData(int solution)
{
	ofstream outfile("output.txt");
	outfile << solution << endl;
	outfile.close();
}

void procData(int n, int* seq, int& solution)
{
	int i, j, k;
	int comm_diff;

	for(i = 0; i < n; i++) map[seq[i]] = i + 1;
	solution = 0;
```

```
	for(i = 0; i < n; i++)
	{
		for(j = 0; j < i; j++)
		{
			comm_diff = seq[i] - seq[j];
			if(seq[j] - comm_diff >= 0 && map[ seq[j] - comm_diff ] > 0)
			{
				k = map[ seq[j] - comm_diff ] - 1;
				if(table[j][k] > 0) {
					table[i][j] = seq[i] + table[j][k];
				}
				else
				{
					table[i][j] = seq[i] + seq[j] + seq[k];
				}
			}
			if(table[i][j] > solution)
			{
				solution = table[i][j];
			}
		}
	}
}
```

대표 자연수

정보초등학교의 연아는 여러 개의 자연수가 주어졌을 때, 이를 대표할 수 있는 대표 자연수에 대하여 연구하였다. 그 결과 어떤 자연수가 다음과 같은 성질을 가지면 대표 자연수로 적당할 것이라고 판단하였다.

"대표 자연수는 주어진 모든 자연수들에 대하여 그 차이를 계산하여 그 차이들 전체의 합을 최소로 하는 자연수이다."

예를 들어 주어진 자연수들이 [4, 3, 2, 2, 9, 10]이라 하자. 이 때 대표 자연수는 3 혹은 4가 된다. 왜냐하면 (4와 3의 차이) + (3과 3의 차이) + (2와 3의 차이) + (2와 3의 차이) + (9와 3의 차이) + (10과 3의 차이) = 1+0+1+1+6+7 = 16이고, (4와 4의 차이) + (3과 4의 차이) + (2와 4의 차이) + (2와 4의 차이) + (9와 4의 차이) + (10과 4의 차이) = 0+1+2+2+5+6 = 16으로 같으며, 이 두 경우가 차이들의 합을 최소로 하기 때문이다. 비교를 위하여 평균값인 5의 경우를 생각하여 보면, (4와 5의 차이) + (3과 5의 차이) + (2와 5의 차이) + (2와 5의 차이) + (9와 5의 차이) + (10과 5의 차이) = 1+2+3+3+4+5 = 18로 위의 두 경우보다 차이들의 합이 더 커짐을 볼 수 있다.

연아를 도와서 위의 성질을 만족하는 대표 자연수를 구하는 프로그램을 작성하시오.

실행 파일의 이름은 AA.EXE로 하고 실행시간은 1초를 넘을 수 없다. 부분점수는 없다.

입력 형식

입력 파일의 이름은 INPUT.TXT이다. 첫째 줄에는 자연수의 개수 N이 입력된다. N은 1 이상 20,000 이하이다. 둘째 줄에는 N개의 자연수가 빈칸을 사이에 두고 입력되며, 이 수들은 모두 1 이상 10,000 이하이다.

출력 형식

출력 파일의 이름은 OUTPUT.TXT이다. 첫째 줄에 대표 자연수를 출력한다. 대표 자연수가 두 개 이상일 경우 그 중 제일 작은 것을 출력한다.

입력과 출력의 예

입력(INPUT.TXT)

```
6
4 3 2 2 9 10
```

출력(OUTPUT.TXT)

```
3
```

풀이

N개의 자연수들이 주어지면 그 중간값을 구하는 문제이다. 주어진 자연수를 정렬해서 저장하고, N이 홀수와 짝수인 경우를 따로 나눠서 처리하면 되는 문제이다. 주의할 점은 처리해야 할 수의 개수가 매우 많다는 것으로, 이에 대한 고려를 하면서 프로그래밍을 해야 하는 문제이다.

먼저 N개의 자연수들을 오름차순으로 정렬해서 유지한다. 다음으로는 N에 따라서 경우가 나뉘게 되는데, 먼저 N이 홀수인 경우는 (N+1)/2번째 자연수가 답이 된다. 다음으로 N이 짝수인 경우는 N/2번째 자연수를 찾으면 된다.

제한 시간 내에 답을 구하기 위해서는 O($N\log N$) 정렬을 사용하거나, 주어지는 자연수의 범위가 10,000 이하인 것을 이용하여 각 자연수의 수를 세는 카운팅 (Counting) 정렬을 이용해서 O(N)에 해결할 수도 있다. 후자의 경우에는 카운팅 한 결과로부터 원래 배열을 다시 복원하는 방법을 사용해도 되지만, 카운팅 결과 자체로도 답을 구할 수 있다. 즉, 같은 수를 한 번에 묶어서 처리하면서 중간값에 해당하는 수가 무엇이 될지를 처리하는 방식도 가능하다.

```
#include <stdio.h>
#include <math.h>

int num[10001] = {0, };

int main()
{
	int size;
	int i;
	int index, answer;

	FILE *fp = fopen("input.txt", "r");

	fscanf(fp, "%d", &size);
	for (i = 0; i < size; ++i)
	{
		int a;
		fscanf(fp, "%d", &a);
		++num[a];
	}

	fclose(fp);

	index = 0;
	for (i = 1; i <= 10000; ++i)
	{
		index += num[i];
		if (index >= (size + 1) / 2)
		{
			answer = i;
			break;
		}
	}

	fp = fopen("output.txt", "w");
	fprintf(fp, "%d\n", answer);
	fclose(fp);

	return 0;
}
```

루빅의 사각형

4×4 격자판에 1에서 16까지 정수 번호가 매겨진 16개 타일이 임의로 놓여져 있다. 타일을 움직여 그림 1과 같이 타일을 놓이게 하려고 한다.

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

그림 1

타일을 움직이는 방법은 하나의 행(가로줄)을 오른쪽으로 원하는 칸 수만큼 순환적으로 움직이거나, 하나의 열(세로줄)을 원하는 칸 수만큼 아래쪽으로 순환적으로 움직이는 것이다. 그림 2는 그림 1의 2번째 행을 오른쪽으로 2칸 움직인 것이다. 그림 1의 2번째 행의 오른쪽 끝에 있는 7번 타일과 8번 타일이 오른쪽 경계를 넘어가서 왼쪽 끝으로 옮겨갔다.

1	2	3	4
7	8	5	6
9	10	11	12
13	14	15	16

그림 2

그림 3은 그림 2의 3번째 열을 아래쪽으로 1칸 움직인 것이다. 그림 2의 3번째 열의 가장 아래에 있는 15번 타일이 가장 위쪽으로 옮겨갔다.

1	2	15	4
7	8	3	6
9	10	5	12
13	14	11	16

그림 3

그림 3과 같이 타일이 놓여진 격자판이 주어졌다면 3번째 열을 3칸 움직인 다음, 2번째 행을 2칸 움직이면 그림 1과 같이 타일이 놓이게 된다. 따라서 2번 움직이면 된다.

1에서 16까지 번호가 매겨진 타일이 임의로 놓여져 있을 때 그림 1과 같이 타일이 놓일 수 있도록 타일을 움직이는 순서를 출력하는 프로그램을 작성하시오. 여기서 움직이는 횟수는 최소로 하여야 한다.

실행 파일의 이름은 BB.EXE로 하고, 실행시간은 1초를 넘을 수 없다. 부분점수는 없다.

입력 형식

입력 파일의 이름은 INPUT.TXT이다. 4×4 격자판에 놓여진 타일 번호가 행단위로 4개 줄에 주어진다. 타일 번호는 1부터 16까지의 정수이다. 각 줄에는 해당하는 행에 놓여지는 4개 타일의 번호가 빈칸을 사이에 두고 순서대로 주어진다.

출력 형식

출력 파일의 이름은 OUTPUT.TXT이다. 첫 번째 줄에는 움직이는 횟수를, 두 번째 줄부터는 한 줄에 하나씩 타일

을 움직이는 방법을 순서대로 출력한다. 이 때, 격자판의 i번째 행을 k칸 움직였다면 정수 1과 i와 k를 빈칸을 사이에 두고 한 줄에 출력한다. 그리고 격자판의 i번째 열을 k칸 움직였다면 정수 2와 i와 k를 빈칸을 사이에 두고 한 줄에 출력한다. 여기서 i는 1 이상 4 이하, k는 1 이상 3 이하의 정수이다.

입력과 출력의 예

입력(INPUT.TXT)

```
1 2 15 4
7 8 3 6
9 10 5 12
13 14 11 16
```

출력(OUTPUT.TXT)

```
2
2 3 3
1 2 2
```

풀이

상태 공간을 탐색하는 문제이다. 모든 상태 공간의 수는 16!개로 매우 많으나, 최대 7번의 이동으로 올바른 상태로 이동할 수 있는 경우만이 입력으로 주어짐을 이용하면 약 24^7개로 줄어든다는 것을 알 수 있다. 여기에 적절한 가지치기를 추가한 퇴각 검색(Backtracking)을 이용하면 빠른 시간 내에 모든 데이터에 대해 답을 구할 수 있다.

가지치기에는 여러 가지 방법이 있을 수 있지만, 모범 답안에서는 현재 상태에서 올바른 위치에 존재하는 수의 개수를 이용한 방법을 사용하였다. 예를 들어, 지금까지 알려진 최적해보다 2번 적게 이동을 수행한 상태에서, 알려진 최적해보다 더 좋은 해를 찾기 위해서는 최소한 하나의 열이나 하나의 행을 제외한 나머지 수들이 모두 올바른 위치에 있어야 한다. 이를 조금 더 발전시키면, 마찬가지로 알려진 최적해보다 3번 또는 4번 적게 이동을 수행한 상태에 대해서도 적절한 가지치기가 가능하다는 것을 알 수 있다. 모범 답안에 각각의 경우가 구현되어 있다.

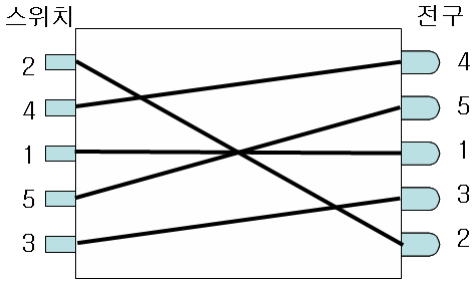
```
#include <stdio.h>

int grid[4][4];
int answer;
int answer_moves[7][3];

int correct_cells(int current_grid[4][4])
{
	int n = 1;
	int cnt = 0;

	for (int i = 0 ; i < 4 ; i++)
		for (int j = 0 ; j < 4 ; j++)
		{
			if (current_grid[i][j] == n)
				cnt++;
			n++;
		}

	return cnt;
}

void y_rotate(int current_grid[4][4], int y)
{
	int temp[4];
	int i;

	for (i = 0 ; i < 4 ; i++)
		temp[i] = current_grid[y][i];
	for (i = 0 ; i < 4 ; i++)
	{
		if (i + 1 < 4)
			current_grid[y][i + 1] = temp[i];
		else
			current_grid[y][i - 3] = temp[i];
	}
}

void x_rotate(int current_grid[4][4], int x)
{
	int temp[4];
	int i;

	for (i = 0 ; i < 4 ; i++)
		temp[i] = current_grid[i][x];
	for (i = 0 ; i < 4 ; i++)
	{
		if (i + 1 < 4)
			current_grid[i + 1][x] = temp[i];
		else
			current_grid[i - 3][x] = temp[i];
```

```
		}
}

void recur(int n, int current[4][4], int moves[7][3])
{
		int ccells = correct_cells(current);

		if (ccells == 16)
		{
				answer = n;
				for (int i = 0 ; i < n ; i++)
				{
						answer_moves[i][0] = moves[i][0];
						answer_moves[i][1] = moves[i][1];
						answer_moves[i][2] = moves[i][2];
				}

				return;
		}

		if (n + 1 >= answer)
				return;
		if (ccells != 12 && n + 2 >= answer)
				return;
		if (ccells != 12 && ccells != 8 && ccells != 9 && n + 3 >= answer)
				return;
		if (ccells < 4 && n + 4 >= answer)
				return;

		// garo
		{
				for (int i = 0 ; i < 4 ; i++)
				{
						for (int j = 0 ; j < 3 ; j++)
						{
								moves[n][0] = 1;
								moves[n][1] = i + 1;
								moves[n][2] = j + 1;
								y_rotate(current, i);

								recur(n + 1, current, moves);
						}
						y_rotate(current, i);
				}
		}
		// sero
		{
				for (int i = 0 ; i < 4 ; i++)
				{
						for (int j = 0 ; j < 3 ; j++)
						{
```

```
                                        moves[n][0] = 2;
                                        moves[n][1] = i + 1;
                                        moves[n][2] = j + 1;
                                        x_rotate(current, i);

                                        recur(n + 1, current, moves);
                                }
                                x_rotate(current, i);
                        }
                }
        }

int main()
{
        {
                FILE* fp = fopen("input.txt", "r");
                for (int i = 0 ; i < 4 ; i++)
                        for (int j = 0 ; j < 4 ; j++)
                                fscanf(fp, "%d", &grid[i][j]);
                fclose(fp);
        }

        {
                int moves[7][3];

                answer = 8;
                recur(0, grid, moves);
        }

        {
                FILE* fp = fopen("output.txt", "w");
                fprintf(fp, "%d\n", answer);
                for (int i = 0 ; i < answer ; i++)
                        fprintf(fp, "%d %d %d\n", answer_moves[i][0], answer_moves[i][1],
answer_moves[i][2]);
                fclose(fp);
        }

        return 0;
}
```

전구

N개의 스위치와 N개의 전구를 가진 하나의 스위칭 박스가 있다. 이 박스의 왼편에는 스위치가 있고, 오른편에는 전구가 달려있다. 모든 스위치와 전구들은 1에서부터 N까지의 번호를 가지며 같은 번호의 스위치와 전구는 전선으로 서로 연결되어 있다.



하나의 스위치를 누르면 그 스위치와 연결된 전구에 불이 들어오게 된다. 두 개 이상의 스위치를 같이 누르는 경우, 전선이 서로 만나면 만난 전선에 연결된 전구들의 불은 켜지지 않는다.

위 그림에서 1번과 4번의 스위치를 같이 누르면 1번과 4번의 전구에는 불이 켜지지만, 1번과 2번의 스위치를 같이 누르면 1번과 2번 전구의 불은 켜지지 않는다. 1번과 3번 그리고 5번 스위치를 같이 누르면 전선이 만나는 1번과 5번 전구는 켜지지 않지만 3번 전구는 켜지게 된다.

여러분이 할 일은 가장 많은 전구가 켜지도록 스위치를 누르는 것이다. 위 그림에서는 3번과 4번 그리고 5번 스위치를 누르는 경우와 1번과 3번 그리고 4번을 누르는 경우에 세 개의 전구가 켜지게 되고, 이 두 가지 경우가 가장 많은 전구가 켜지는 경우이다.

스위치의 번호순서와 전구의 번호순서가 주어질 때, 어떤 스위치를 누르면 가장 많은 전구가 켜지는지를 알아내는 프로그램을 작성하시오.

실행 파일의 이름은 CC.EXE로 하고, 실행시간은 1초를 넘을 수 없다. 부분점수는 없다.

입력 형식

입력 파일의 이름은 INPUT.TXT이다. 입력의 첫 번째 줄에는 스위치의 수(전구의 수)를 나타내는 정수 N (1≤N≤10,000)이 주어진다. 두 번째 줄에는 N개의 스위치 번호들이 위에서부터 순서대로 빈칸을 사이에 두고 주어진다. 세 번째 줄에는 N개의 전구 번호들이 위에서부터 순서대로 빈칸을 사이에 두고 주어진다.

출력 형식

출력 파일의 이름은 OUTPUT.TXT이다. 첫 번째 줄에는 가장 많은 전구가 켜지게 하는 스위치의 수를 출력한다. 두 번째 줄에는 눌러야 하는 스위치의 번호를 오름차순(번호가 커지는 순서)으로 빈칸을 사이에 두고 하나의 줄에 출력한다. 단, 두 번째 줄에 출력할 수 있는 답이 두 가지 이상일 때에는 그 중 한 가지만 출력한다.

입력과 출력의 예

입력(INPUT.TXT)

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

출력(OUTPUT.TXT)

```
3
3 4 5
```

풀이

전선들이 겹치지 않는다는 것은 다시 생각해 보면 스위치의 번호가 증가하면 대응되는 전구 또한 증가하는 순서여야 한다는 것과 같은 이야기이다. 따라서 이 문제는 결국 스위치의 번호가 증가하는 순서대로 전구 번호를 정렬한 뒤, 전구 번호 수열에서 가장 긴 증가하는 부분수열을 구하는 문제가 된다.

N의 크기가 최대 10,000이므로, $O(N^2)$이 아닌 $O(N\log N)$ 동적 계획법을 이용해 해결할 수 있다. 수열을 S[1], ..., S[N]이라고 놓고, D[k][i]를 1번 원소부터 k번째 원소까지 고려했을 때, 길이가 i가 되는 증가수열의 마지막 원소가 될 수 있는 값들 중에서 최소값으로 정의하자.

이제 k를 하나씩 증가시켜 가며 D[k][i]들을 구해 나간다. 다음의 두 가지 경우가 있는데, 먼저 D[k-1][i-1]<S[k-1]이면 S[k]를 지금까지 구한 증가수열에 붙일 수 있다는 것이므로 D[k][i]는 D[k-1][i]와 S[k]중 작은 값으로 두면 된다. 그 외의 경우에서 D[k][i]는 결국 D[k-1][i]와 같다.

이 때, D[k-1][i-1]<S[k-1]인 가장 큰 i를 찾아야 하는데, 이를 이분 탐색을 이용해 찾으면 1차원 배열만을 가지고 있으면서 배열을 각 단계마다 $O(\log N)$ 시간복잡도에 갱신해 나갈 수 있다. 마지막 단계까지 수행한 뒤 배열의 길이와 배열에 저장된 값들이 우리가 구하고자 하는 답이 된다.

```
#include<cstdio>
#include<algorithm>
#define N 10000
using namespace std;

int n, cnt;
int L[ N], R[ N], Index[ N + 1];
int D[ N], Trace[ N], TurnOn[ N];

int main ( void) {

	int i, a;

	FILE *fin = fopen ( "input.txt", "rt");
	fscanf( fin, "%d", &n);
	for ( i = 0; i < n; i++) {
		fscanf( fin, "%d", &L[ i]);
		Index[ L[ i]] = i;
	}
	for ( i = 0; i < n; i++) {
		fscanf( fin, "%d", &a);
		R[ i] = Index[ a];
	}
	fclose ( fin);

	cnt = 0;

	for ( i = 0; i < n; i++) {
		if ( cnt == 0) {
			Trace[ i] = -1;
			D[ cnt++] = i;
		} else if ( R[ D[ cnt - 1]] < R[ i]) {
			Trace[ i] = D[ cnt - 1];
			D[ cnt++] = i;
		} else {
			int s = 0, e = cnt - 1, mid;
			while ( s < e) {
				mid = ( s + e) / 2;
				if ( R[ D[ mid]] < R[ i])
					s = mid + 1;
				else
					e = mid;
			}

			D[ s] = i;
			if ( s == 0)
				Trace[ i] = -1;
			else
				Trace[ i] = D[ s - 1];
		}
	}
```

```
        a = D[ cnt - 1];
        for ( i = 0; i < cnt; i++) {
                TurnOn[ i] = L[ R[ a]];
                a = Trace[ a];
        }

        sort( TurnOn, TurnOn + cnt);


        FILE *fout = fopen ( "output.txt", "wt");
        fprintf( fout, "%d\n", cnt);
        for ( i = 0; i < cnt; i++)
                fprintf( fout, "%d ", TurnOn[ i]);
        fprintf( fout, "\n");
        fclose( fout);

        return 0;
}
```

용액

KOI 부설 과학연구소에서는 많은 종류의 산성 용액과 알칼리성 용액을 보유하고 있다. 각 용액에는 그 용액의 특성을 나타내는 하나의 정수가 주어져있다. 산성 용액의 특성값은 1부터 1,000,000,000까지의 양의 정수로 나타내고, 알칼리성 용액의 특성값은 -1부터 -1,000,000,000까지의 음의 정수로 나타낸다.

같은 양의 두 용액을 혼합한 용액의 특성값은 혼합에 사용된 각 용액의 특성값의 합으로 정의한다. 이 연구소에서는 같은 양의 두 용액을 혼합하여 특성값이 0에 가장 가까운 용액을 만들려고 한다.

예를 들어, 주어진 용액들의 특성값이 [-99, -2, -1, 4, 98]인 경우에는 특성값이 -99인 용액과 특성값이 98인 용액을 혼합하면 특성값이 -1인 용액을 만들 수 있고, 이 용액의 특성값이 0에 가장 가까운 용액이다. 참고로, 두 종류의 알칼리성 용액만으로나 혹은 두 종류의 산성 용액만으로 특성값이 0에 가장 가까운 혼합 용액을 만드는 경우도 존재할 수 있다.

산성 용액과 알칼리성 용액의 특성값이 정렬된 순서로 주어졌을 때, 이 중 두 개의 서로 다른 용액을 혼합하여 특성값이 0에 가장 가까운 용액을 만들어내는 두 용액을 찾는 프로그램을 작성하시오.

실행 파일의 이름은 AA.EXE로 하고 실행시간은 1초를 넘을 수 없다. 부분점수는 없다.

입력 형식

입력 파일의 이름은 INPUT.TXT이다. 첫째 줄에는 전체 용액의 수 N이 입력된다. N은 2 이상 100,000 이하의 정수이다. 둘째 줄에는 용액의 특성값을 나타내는 N개의 정수가 빈칸을 사이에 두고 오름차순으로 입력되며, 이 수들은 모두 -1,000,000,000 이상 1,000,000,000 이하이다. N개의 용액들의 특성값은 모두 서로 다르고, 산성 용액만으로나 알칼리성 용액만으로 입력이 주어지는 경우도 있을 수 있다.

출력 형식

출력 파일의 이름은 OUTPUT.TXT이다. 첫째 줄에 특성값이 0에 가장 가까운 용액을 만들어내는 두 용액의 특성값을 출력한다. 출력해야하는 두 용액은 특성값의 오름차순으로 출력한다. 특성값이 0에 가장 가까운 용액을 만들어내는 경우가 두 개 이상일 경우에는 그 중 아무것이나 하나를 출력한다.

입력과 출력의 예1

입력(INPUT.TXT)

```
5
-99 -2 -1 4 98
```

출력(OUTPUT.TXT)

```
-99 98
```

입력과 출력의 예2

입력(INPUT.TXT)

```
4
-100 -2 -1 103
```

출력(OUTPUT.TXT)

```
-2 -1
```

풀이

이 문제는 정수로 구성된 배열 A[1..N]이 주어졌을 때, A[i]+A[j]의 절댓값이 0에 가장 가까운 서로 다른 두 i, j를 찾는 문제이다. 만일 i가 고정되어 있다면 A[j]가 -A[i]에 가장 가까운 경우가 최적임을 알 수 있고, 따라서 최대한 이에 가까운 경우를 찾아주면 그 경우가 답이 된다.

이를 해결하기 위해서는 다양한 방법을 이용할 수 있는데, 이진 탐색을 이용하여 이를 $O(\log N)$에 해결할 수 있다. 즉, 배열을 정렬된 상태로 유지하면서 -A[i]를 검색한 후, 탐색이 종료되는 위치를 이용하여 -A[i]에 제일 가까운 값을 찾는 방법이다. 하지만 좀 더 살펴보면 효율적인 방법을 생각할 수 있는데, i < i′인 경우, A[i′]와 최적의 대응을 이루는 A[j′]에 대해서 j′ <= j가 만족된다는 점을 알 수 있다. 따라서 매번 i에 대해서 새로이 j를 탐색하지 말고, 이전 i에 대해서 탐색한 결과를 계속 유지해 나가는 방법을 이용하면 된다. 이 경우 결국 j는 전체적으로 N만큼의 반복문을 수행하게 되므로, 전체 $O(N)$ 시간에 문제를 풀 수 있다.

```
#include<stdio.h>

#define Swap(aa,bb) {cc=aa;aa=bb;bb=cc;}

const int MAXN = 100000;

int a[MAXN], b[MAXN], an, bn, n, cc;
int Opt, Opt_x, Opt_y, Diff, Diff_x, Diff_y;

void Sort_a(int l, int r)
{
  if(l < r)
  {
    int i = l-1, j = r+1, mid = a[(l+r)>>1];
    while(1)
    {
      while(a[++i] < mid);
      while(a[--j] > mid);
      if(i >= j) break;
      Swap(a[i], a[j]);
    }
    Sort_a(l, i-1);
    Sort_a(j+1, r);
  }
}

void Sort_b(int l, int r)
{
  if(l < r)
  {
    int i = l-1, j = r+1, mid = b[(l+r)>>1];
    while(1)
    {
      while(b[++i] < mid);
      while(b[--j] > mid);
      if(i >= j) break;
      Swap(b[i], b[j]);
    }
    Sort_b(l, i-1);
    Sort_b(j+1, r);
  }
}

int main(void)
{
  int l1, l2, l3, t1;

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

  scanf("%d",&n);
  for(l1=0;l1<n;l1++)
  {
    scanf("%d",&t1);
    if(t1 > 0) a[an++] = t1;
    else b[bn++] = t1;
  }

  Sort_a(0, an-1);
  Sort_b(0, bn-1);

  Opt = 0x7fffffff;

  if(an > 1)
  {
    Diff = a[0] + a[1];
    if(Diff < Opt)
    {
```

```
      Opt = Diff;
      Opt_x = a[0];
      Opt_y = a[1];
    }
  }
  if(bn > 1)
  {
    Diff = - b[bn-1] - b[bn-2];
    if(Diff < Opt)
    {
      Opt = Diff;
      Opt_x = b[bn-2];
      Opt_y = b[bn-1];
    }
  }

  if(an > 0 && bn > 0)
  {
    l2 = bn - 1;
    for(l1=0;l1<an;l1++)
    {
      while(1)
      {
        if(l2 < 0) break;
        if(a[l1] + b[l2] <= 0) break;
        l2--;
      }

      for(l3=l2-1;l3<=l2+1;l3++)
      {
        if(l3 < 0 || l3 >= bn) continue;

        Diff = a[l1] + b[l3];
        if(Diff < 0) Diff = -Diff;

        if(Diff < Opt)
        {
          Opt = Diff;
          Opt_x = b[l3];
          Opt_y = a[l1];
        }
      }
    }
  }

  printf("%d %d\n", Opt_x, Opt_y);

  return 0;
}
```

안전 영역

재난방재청에서는 많은 비가 내리는 장마철에 대비해서 다음과 같은 일을 계획하고 있다. 먼저 어떤 지역의 높이 정보를 파악한다. 그 다음에 그 지역에 많은 비가 내렸을 때 물에 잠기지 않는 안전한 영역이 최대로 몇 개가 만들어지는 지를 조사하려고 한다. 이 때, 문제를 간단하게 하기 위하여, 장마철에 내리는 비의 양에 따라 일정한 높이 이하의 모든 지점은 물에 잠긴다고 가정한다.

어떤 지역의 높이 정보는 행과 열의 크기가 각각 N인 2차원 배열 형태로 주어지며 배열의 각 원소는 해당 지점의 높이를 표시하는 자연수이다. 예를 들어, 다음은 N=5인 지역의 높이 정보이다.

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

이제 위와 같은 지역에 많은 비가 내려서 높이가 4 이하인 모든 지점이 물에 잠겼다고 하자. 이 경우에 물에 잠기는 지점을 회색으로 표시하면 다음과 같다.

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

물에 잠기지 않는 안전한 영역이라 함은 물에 잠기지 않는 지점들이 위, 아래, 오른쪽 혹은 왼쪽으로 인접해 있으며 그 크기가 최대인 영역을 말한다. 위의 경우에서 물에 잠기지 않는 안전한 영역은 5개가 된다(꼭지점으로만 붙어 있는 두 지점은 인접하지 않는다고 취급한다).

또한 위와 같은 지역에서 높이가 6이하인 지점을 모두 잠기게 만드는 많은 비가 내리면 물에 잠기지 않는 안전한 영역은 아래 그림에서와 같이 네 개가 됨을 확인할 수 있다.

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

이와 같이 장마철에 내리는 비의 양에 따라서 물에 잠기지 않는 안전한 영역의 개수는 다르게 된다. 위의 예와 같은 지역에서 내리는 비의 양에 따른 모든 경우를 다 조사해 보면 물에 잠기지 않는 안전한 영역의 개수 중에서 최대인 경우는 5임을 알 수 있다.

어떤 지역의 높이 정보가 주어졌을 때, 장마철에 물에 잠기지 않는 안전한 영역의 최대 개수를 계산하는 프로그램을 작성하시오.

실행 파일의 이름은 BB.EXE로 하고 실행시간은 1초를 넘을 수 없다. 부분 점수는 없다.

입력 형식

입력 파일의 이름은 INPUT.TXT이다. 첫째 줄에는 어떤 지역을 나타내는 2차원 배열의 행과 열의 개수를 나타내는 수 N이 입력된다. N은 2 이상 100 이하의 정수이다. 둘째 줄부터 N 개의 각 줄에는 2차원 배열의 첫 번째 행부터 N번째 행까지 순서대로 한 행씩 높이 정보가 입력된다. 각 줄에는 각 행의 첫 번째 열부터 N번째 열까지 N 개의 높이 정보를 나타내는 자연수가 빈칸을 사이에 두고 입력된다. 높이는 1이상 100 이하의 정수이다.

출력 형식

출력 파일의 이름은 OUTPUT.TXT이다. 첫째 줄에 장마철에 물에 잠기지 않는 안전한 영역의 최대 개수를 출력한다.

입력과 출력의 예1

입력(INPUT.TXT)

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

출력(OUTPUT.TXT)

```
5
```

입력과 출력의 예2

입력(INPUT.TXT)

```
7
9 9 9 9 9 9 9
9 2 1 2 1 2 9
9 1 8 7 8 1 9
9 2 7 9 7 2 9
9 1 8 7 8 1 9
9 2 1 2 1 2 9
9 9 9 9 9 9 9
```

출력(OUTPUT.TXT)

```
6
```

풀이

이 문제는 주어진 $N \times N$ 배열에서 특정 값 이상인 영역의 개수를 세고, 이러한 영역의 수가 가장 많아지는 경우를 찾는 문제이다. 배열의 크기와 배열에 저장된 값의 범위가 모두 작으므로, 모든 경우에 대해서 영역의 개수를 센 후 그들 중 최대값을 찾으면 된다.

이를 처리하기 위해서는 다양한 방법이 가능한데, 모범 풀이에서는 다음과 같은 방법을 이용하였다. 먼저, 비가 어떤 높이까지 올라온다고 가정했을 때, 그 높이보다 낮은 부분을 0으로 바꾼 배열을 하나 만든다. 그 후, 만들어진 배열에서 너비우선탐색(BFS)을 이용해서 플러드 필(Flood-Fill)을 하는데, 아직 방문되지 않았으면서 0이 아닌 부분인 부분을 찾을 때마다 플러드 필을 수행한다. 이 과정을 수행하면 안전 영역의 수를 계산할 수 있는데, 매번 이러한 부분을 찾을 때마다 변수를 누적시켜나가는 방법을 이용하면 된다.

이제 이러한 과정을 가능한 모든 높이에 대해서 수행한다. 높이의 제한이 100 이하므로, $O(N^2)$ 알고리즘을 100회 수행하여 풀 수 있다.

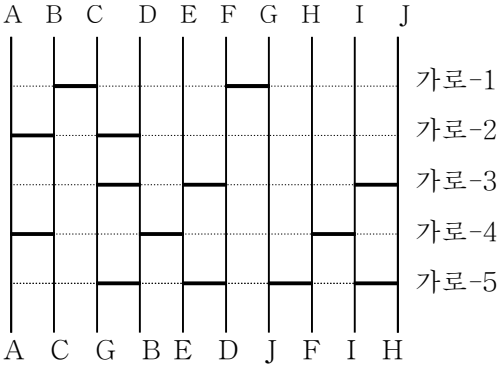
```
#include <stdio.h>

#define MaxN 500
#define NSQR 250000

static int n, m, Map[ MaxN + 2 ][ MaxN + 2 ], h, Max, qx[ NSQR + 1 ], qy[ NSQR + 1 ];
static int dx[ 5 ] = { 0, 0, 1, 0, -1 }, dy[ 5 ] = { 0, 1, 0, -1, 0 };
static bool Check[ MaxN + 2 ][ MaxN + 2 ];

void Input( void )
{
  int i, j;

  FILE *In = fopen( "input.txt" , "r" );
  fscanf( In , "%d" , &n );
  for(i = 1; i <= n; i++)
  {
    for(j = 1; j <= n; j++)
    {
      fscanf( In , "%d" , &Map[ i ][ j ] );
      if(Map[ i ][ j ] > h)
        h = Map[ i ][ j ];
    }
  }
  fclose( In );
}

void BFS( int Limit , int y , int x )
{
  int i, Head, Tail;

  Head = 0, Tail = 1;
  qy[ 1 ] = y; qx[ 1 ] = x;
  while( Head != Tail )
  {
    Head++;
    for(i = 1; i <= 4; i++)
    {
      y = qy[ Head ] + dy[ i ];
      x = qx[ Head ] + dx[ i ];
      if(!Check[ y ][ x ])
      {
        if(Map[ y ][ x ] >= Limit)
        {
          qy[ ++Tail ] = y;
          qx[ Tail ] = x;
          Check[ y ][ x ] = true;
        }
      }
    }
  }
```

```
}

void Process( void )
{
  int i, j, Loop, t;

  for(Loop = 1; Loop <= h; Loop++)
  {
    for(i = 1; i <= n; i++)
      for(j = 1; j <= n; j++)
        Check[ i ][ j ] = false;
    for(t = 0, i = 1; i <= n; i++)
      for(j = 1; j <= n; j++)
        if(!Check[ i ][ j ] && Map[ i ][ j ] >= Loop)
          BFS( Loop , i , j ), t++;
    if(t > Max)
      Max = t;
  }
}

void Output( void )
{
  FILE *Out = fopen( "output.txt" , "w" );
  fprintf( Out , "%d" , Max );
  fclose( Out );
}

int main( void )
{
  Input();
  Process();
  Output();

  return 0;
}
```

사다리 타기

k명의 참가자들이 사다리 타기를 통하여 어떤 순서를 결정한다. 참가자들은 알파벳 대문자 첫 k개로 표현되며, 사다리 타기를 시작할 때의 순서는 아래 그림과 같이 항상 알파벳 순서대로이다.

k=10 인 예를 들어 보자. 10명의 A, B, C, D, E, F, G, H, I, J 참가자들이 사다리 타기를 준비한다. 아래 그림은 10개의 세로 줄과 5개의 가로 줄을 가지고 있는 사다리의 한 예를 보여주고 있다.



이 사다리에서 점선은 가로 막대가 없음을, 굵은 가로 실선은 옆으로 건너갈 수 있는 가로 막대가 있음을 나타내고 있다.

따라서 위에 제시된 사다리를 타면 그 최종 도달된 순서는 왼쪽으로부터 A, C, G, B, E, D, J, F, I, H 가 된다.

사다리 타기는 세로 막대를 타고 내려오는 중에 가로막대를 만나면 그 쪽으로 옮겨 가면서 끝까지 내려가는 과정이다. 따라서 사다리 타기의 규칙 특성상 아래 그림과 같이 두 가로 막대가 직접 연결될 수는 없으므로 이 상황은 이 문제에서 고려할 필요가 없다.

우리는 하나의 가로 줄이 감추어진 사다리를 받아서 그 줄의 각 칸에 가로 막대를 적절히 넣어서 참가자들의 최종 순서가 원하는 순서대로 나오도록 만들려고 한다.

입력에서 사다리의 전체 모양은 각 줄에 있는 가로 막대의 유무로 표현된다. 각 줄에서 가로 막대가 없는 경우에는 '*' (별)문자, 있을 경우에는 '-' (빼기) 문자로 표시된다. 그리고 감추어진 특정 가로 줄은 길이 k-1인 '?' (물음표) 문자열로 표시되어 있다.

프로그램의 이름은 CC.EXE로 하고 실행시간은 1초를 넘을 수 없다. 부분 점수는 없다.

입력 형식

입력 파일의 이름은 INPUT.TXT이다. 입력 파일의 첫 줄에는 참가한 사람의 수 k가 나온다($3 \le k \le 26$). 그 다음 줄에는 가로 막대가 놓일 전체 가로 줄의 수를 나타내는 n이 나온다($3 \le n \le 1{,}000$). 그리고 세 번째 줄에는 사다리를 타고 난 후 결정된 참가자들의 최종 순서가 길이 k인 대문자 문자열로 들어온다.

k와 n, 그리고 도착순서 문자열이 나타난 다음, 이어지는 n개의 줄에는 앞서 설명한 바와 같이 '*' 와 '-' 문자로 이루어진 길이 k-1인 문자열이 주어진다. 그 중 감추어진 가로 줄은 길이가 k-1인 '?' 문자열로 표시되어 있다.

출력 형식

출력 파일의 이름은 OUTPUT.TXT이다. 여러

분은 입력 파일 세 번째 줄에서 지정한 도착순서가 해당 사다리에서 만들어질 수 있도록 감추어진 가로 줄을 구성해야 한다.

여러분은 감추어진 가로 줄의 상태를 재구성하여 이를 '*' (별) 문자와 '-' (빼기) 문자로 구성된 길이 k-1인 문자열로 만들어, 출력 파일 OUTPUT.TXT의 첫 줄에 출력해야 한다.

그런데 어떤 경우에는 그 감추어진 가로 줄을 어떻게 구성해도 원하는 순서를 얻을 수 없는 경우도 있다. 이 경우에는 'x' (소문자 엑스)로 구성된 길이 k-1인 문자열을 출력해야 한다. 이 경우는 아래 두 번째 입출력의 예에서 확인해 볼 수 있다.

입력과 출력의 예 1

입력(INPUT.TXT)

```
10
5
ACGBEDJFIH
*-***-***
-*-******
??????????
-**-***-*
**-*-*-*-
```

출력(OUTPUT.TXT)

```
**-*-***-
```

입력과 출력의 예 2

```
11
5
CGBEDJFKIHA
*-***-****
-*-******-
??????????
-**-***-*-
**-*-*-*-*
```

출력(OUTPUT.TXT)

```
xxxxxxxxxx
```

풀이

이 문제는 사다리 타기에서 한 줄이 비어있을 경우, 원하는 결과를 얻기 위해서 빈 줄을 어떻게 채워야 하는지를 찾는 문제이다. 이를 위해서는 주어진 사다리에 대해서 사다리 타기를 수행할 수 있어야 하고, 그 결과를 이용하여 빈 줄을 채우는 방법을 고안해야 한다.

먼저 ?가 없는 상황에서 각각의 사람이 어디로 내려가는지 구하는 방법을 알아보자. 단순히 눈으로 구하듯이 한 사람씩 내려오는 방법도 있겠지만, 살펴보면 가로막대를 만날 때마다 인접한 두 사람의 위치가 바뀐다는 것을 알 수 있다. 즉, 위에서부터 내려오면서 가로막대가 있으면 현재 상황에서 가로막대의 양 끝에 있는 두 사람의 위치를 서로 바꿔주면 된다.

이 원리를 이용하면 한 줄이 비어있을 때도 쉽게 구할 수 있다. 먼저 위의 방식을 위에서부터 빈 줄까지 내려오면서 수행하고, 반대로 제일 아래에서부터 빈 줄까지 올라가면서 수행한다. 그러면 두 개의 배열을 얻을 수 있는데, 이제 이 두 개의 결과가 사다리에 의해 만들어질 수 있는 결과인지를 살펴보면 된다. 즉, 두 줄 사이에 사다리를 놓아 보면서, 두 상태가 언제 같아지는지를 확인하고, 이러한 경우가 있다면 이를 답으로 출력하면 된다.

```
#include <stdio.h>

#define N 30
#define Maxm 1011
#define swap(a,b) { temp = a; a=b; b=temp; }

FILE *fin = fopen("input.txt", "r");
FILE *fout = fopen("output.txt", "w");

static int downward[N], upward[N];
static char map[Maxm][N], final[N], answer[N];

int n, m, blank, temp;

void init() {
  int i, j;
  fscanf(fin, "%d %d", &n, &m);
  fscanf(fin, "%s", final);
  for(i = 0; i < m; i++) {
    fscanf(fin, "%s", map[i]);
    if(map[i][0] == '?') blank = i;
  }
  for(j = 0; j < n; j++) {
    downward[j] = j + 1;
    upward[j] = final[j] - 'A' + 1;
  }
}

void process() {
  int i, j;
  for(i = 0; i < blank; i++) {
    for(j = 0; j < n - 1; j++) {
      if(map[i][j] == '-') {
        swap(downward[j], downward[j + 1]);
      }
    }
  }
  for(i = m-1; i > blank; i--) {
    for(j = 0; j < n - 1; j++) {
      if(map[i][j] == '-') {
        swap(upward[j], upward[j + 1]);
      }
    }
  }
}

int output() {
  int i;
  for(i = 0; i < n - 1; i++) {
    if(upward[i] != downward[i]) {
      answer[i] = '-';
```

```
        swap(downward[i], downward[i + 1]);
      }
      else answer[i] = '*';
      if(upward[i] != downward[i]) return -1;
    }
    return 1;
}

int main() {
    int i, judge;
    init();
    process();
    judge = output();
    if(judge == -1) {
      for(i = 0; i < n - 1; i++) {
        fprintf(fout, "x");
      }
      fprintf(fout, "\n");
    }
    else {
      fprintf(fout, "%s\n", answer);
    }
    return 0;
}
```

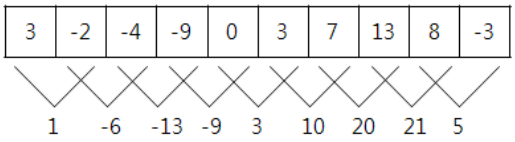
수열

매일 아침 9시에 학교에서 측정한 온도가 어떤 정수의 수열로 주어졌을 때, 연속적인 며칠 동안의 온도의 합이 가장 큰 값을 알아보고자 한다.

예를 들어, 아래와 같이 10일 간의 온도가 주어졌을 때,

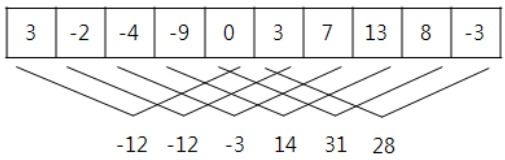
3 -2 -4 -9 0 3 7 13 8 -3

모든 연속적인 이틀간의 온도의 합은 아래와 같다.



이때, 온도의 합이 가장 큰 값은 21이다.

또 다른 예로 위와 같은 온도가 주어졌을 때, 모든 연속적인 5일 간의 온도의 합은 아래와 같으며,



이때, 온도의 합이 가장 큰 값은 31이다.

매일 측정한 온도가 정수의 수열로 주어졌을 때, 연속적인 며칠 동안의 온도의 합이 가장 큰 값을 계산하는 프로그램을 작성하시오.

실행 파일의 이름은 AA.EXE로 하고 실행시간은 1초를 넘을 수 없다. 부분 점수는 없다.

입력 형식

입력 파일의 이름은 INPUT.TXT이다. 첫째 줄에는 두 개의 정수 N과 K가 한 개의 공백을 사이에 두고 순서대로 주어진다. 첫 번째 정수 N은 온도를 측정한 전체 날짜의 수이다. N은 2 이상 100,000 이하이다. 두 번째 정수 K는 합을 구하기 위한 연속적인 날짜의 수이다. K는 1과 N 사이의 정수이다. 둘째 줄에는 매일 측정한 온도를 나타내는 N개의 정수가 빈칸을 사이에 두고 주어진다. 이 수들은 모두 -100 이상 100 이하이다.

출력 형식

출력 파일의 이름은 OUTPUT.TXT이다. 첫째 줄에는 입력되는 온도의 수열에서 연속적인 K일의 온도의 합이 최대가 되는 값을 출력한다.

입력과 출력의 예1

입력(INPUT.TXT)

```
10 2
3 -2 -4 -9 0 3 7 13 8 -3
```

출력(OUTPUT.TXT)

```
21
```

입력과 출력의 예2

입력(INPUT.TXT)

```
10 5
3 -2 -4 -9 0 3 7 13 8 -3
```

출력(OUTPUT.TXT)

```
31
```

풀이

우선 일반적으로 생각해 볼 수 있는 답안은 모든 가능한 연속된 k개의 수열의 합을 계산해보는 것이다. 수열 $A_0 \cdots A_n$에 대해 $A_0 \cdots A_{k-1}, A_1 \cdots A_k, A_2 \cdots A_{k+1}, \cdots$을 계산해보는 방법으로 이럴 경우 시간복잡도는 $O(kN)$이 될 것이다. 그러나 보다 효율적인 알고리즘을 찾기 위하여, $A_1 \cdots A_k$의 합을 보다 간단히 구하는 방법을 고려한다. $A_1 \cdots A_k$ 의 합= $A_0 \cdots A_{k-1}$의 합$- A_0 + A_k$이므로 $A_0 \cdots A_{k-1}$의 합을 알고 있는 상황에서는 $A_1 \cdots A_k$의 합을 $O(1)$만에 해를 계산해낼 수 있다. 이후로도 계속해서 앞의 아이디어를 적용할 수 있으므로 전체 시간복잡도 $O(N)$에 문제를 해결할 수 있다.

초등부의 가장 쉬운 문제로 할당된 문제인 만큼 $O(kN)$의 시간복잡도로 문제를 해결한 학생도 절반 이상의 부분점수를 받을 수 있도록 데이터를 만들었다.

초등부 1번 문제(AA) 모범 풀이

```
#include <cstdio>
#include <algorithm>
#include <iostream>
#include <fstream>
#include <cstdlib>
#include <ctime>

using namespace std;

int n,k;
int data[100005];

int main(void)
{
	int i;
	ifstream fin("input.txt");
	fin >> n >> k;

	if(k == 0) return 0;

	int sum = 0;
	int max_sum = -99999999;
	for(i=0;i<n;i++)
	{
		fin >> data[i];

		if(data[i] > 100 || data[i] < -100)
			return -1;
		sum += data[i];
		if(i >= k-1)
		{
			if(sum > max_sum)
				max_sum = sum;
			sum -= data[i-k+1];
		}
	}

	ofstream fout("output.txt");
	fout << max_sum << endl;
	return 0;
}
```

초등부 1번 문제(AA) 모범 풀이(BASIC)

```
Attribute VB_Name = "Module1"
Sub main()
    Open "input.txt" For Input As #1
    Open "output.txt" For Output As #2

    Dim arr(100005) As Long
    Dim n As Long
    Dim k As Long

    Input #1, n
    Input #1, k

    Dim sum As Long
    sum = 0
    Dim max_sum As Long
    max_sum = -99999999

    For i = 0 To n - 1
        Input #1, arr(i)

        sum = sum + arr(i)

        If i >= k - 1 Then
            If sum > max_sum Then max_sum = sum
            sum = sum - arr(i - k + 1)
        End If

    Next i

    Print #2, LTrim(Str(max_sum))
End Sub
```

공약수

어떤 두 자연수에 공통인 약수들 중에서 가장 큰 수를 최대공약수라고 하고, 두 자연수의 공통인 배수들 중에서 가장 작은 수를 최소공배수라고 한다.

예를 들어, 두 자연수 12와 90의 최대공약수는 6이며, 최소공배수는 180이다.

이와 반대로 두 개의 자연수 A, B가 주어졌을 때, A를 최대공약수로, B를 최소공배수로 하는 두 개의 자연수를 구할 수 있다. 그러나, 이러한 두 개의 자연수 쌍은 여러 개 있을 수 있으며, 또한 없을 수도 있다.

예를 들어, 최대공약수가 6이며 최소공배수가 180인 두 정수는 위의 예에서와 같이 12와 90일 수도 있으며, 30과 36, 18과 60, 혹은 6과 180일 수도 있다. 그러나, 최대공약수가 6이며 최소공배수가 20인 두 자연수는 있을 수 없다.

두 개의 자연수가 주어졌을 때, 이 두 수를 최대공약수와 최소공배수로 하는 두 개의 자연수를 구하는 프로그램을 작성하시오.

실행 파일의 이름은 BB.EXE로 하고 실행 시간은 1초를 넘을 수 없다. 부분 점수는 없다.

입력 형식

입력 파일의 이름은 INPUT.TXT이다. 첫째 줄에 두 개의 자연수가 빈칸을 사이에 두고 주어진다. 첫 번째 수는 어떤 두 개의 자연수의 최대공약수이고, 두 번째 수는 그 자연수들의 최소공배수이다. 입력되는 두 자연수는 2 이상 100,000,000 이하이다.

출력 형식

출력 파일의 이름은 OUTPUT.TXT이다. 첫째 줄에는 입력되는 두 자연수를 최대공약수와 최소공배수로 하는 두 개의 자연수를 크기가 작은 수부터 하나의 공백을 사이에 두고 출력한다. 입력되는 두 자연수를 최대공약수와 최소공배수로 하는 두 개의 자연수 쌍이 여러 개 있는 경우에는 두 자연수의 합이 최소가 되는 두 수를 출력한다.

입력과 출력의 예1

입력(INPUT.TXT)

```
6 180
```

출력(OUTPUT.TXT)

```
30 36
```

입력과 출력의 예2

입력(INPUT.TXT)

```
2 86486400
```

출력(OUTPUT.TXT)

```
11648 14850
```

풀이

$1 \leqq X, Y \leqq L$ 범위에 대해 두 숫자의 최대공약수가 G이고, 최소공배수가 L인지 확인하는 방법은 시간복잡도 $O(L^2)$이 되므로 문제에서 원하는 해를 제 시간 내에 얻을 수 없다. 그러나 X, Y의 검색범위는 몇 가지 최적화를 통해 크게 개선할 수 있다.

우선 임의의 두 수 X, Y의 최대공약수 G, 최소공배수 L에 대해 다음 공식이 성립함을 알 수 있다.

$X = GX'$ (X는 G의 배수이므로, 이 식이 성립하는 X'가 존재함)

$Y = GY'$ (위와 마찬가지로 이 식이 성립하는 Y'가 존재함)

$\gcd(X', Y') = 1$ (만약 X'와 Y'의 최대공약수가 1보다 큰 수 k라면, $X = GX' = kGX''$, $Y = GY' = kGY''$가 각각 성립하게 되고, X, Y의 최대공약수 G보다 더 큰 공약수 kG가 존재한다는 뜻이 되므로 모순이 된다. 그러므로 X'와 Y'의 최대공약수는 1이 될 수밖에 없다)

$L = GX'Y'$

(최소공배수 L은 X와 Y의 배수이므로 어떤 a,b에 대해 L=GX'a=GY'b가 성립해야 하는데, L/G=X'a=Y'b가 성립해야 하고 X'와 Y'는 서로 소이므로 X'와 Y'의 최소공배수는 X'Y'가 된다. 즉, $\frac{L}{G} = X'Y'$이고, $L = GX'Y'$가 성립하게 된다)

그러므로 X가 G의 배수라는 점을 이용하면 X의 검색범위를 $1, G, 2G, \ldots$으로 줄일 수 있고, 이후 $\frac{L}{X} = Y'$라는 점을 이용하여 X로부터 Y를 계산해낼 수 있다. 결국 모든 가능한 $\frac{L}{G}$의 약수들을 X의 범위로 찾아보면 되는데, 한 번 더 생각해보면 $\sqrt{\frac{L}{G}}$보다 큰 약수는 계산해 볼 필요가 없음을 알 수 있다.($\sqrt{\frac{L}{G}}$를 넘는 약수 d에 대해, $\frac{\sqrt{\frac{L}{G}}}{d}$ 또한 $\sqrt{\frac{L}{G}}$의 약수이면서 $\sqrt{\frac{L}{G}}$보다 작은 약수이므로) 그러므로 시간복잡도 $O(\sqrt{\frac{L}{G}})$로 문제의 해를 구할 수 있다.

초등부 2번 문제(BB) 모범 풀이

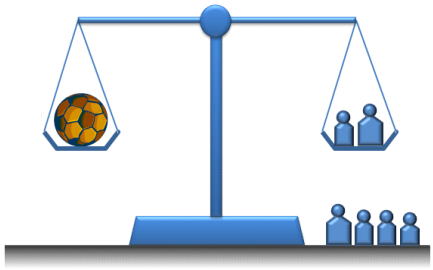
```
#include <stdio.h>
#include <cstdio>
#include <cmath>
#include <ctime>
#include <cstdlib>

int gcd(int a,int b)
{
	if(a<b) return gcd(b,a);
	if(b==0) return a;
	return gcd(b, a%b);
}

int main(void)
{
	int G,L;
	FILE *in = fopen("input.txt","r");
	fscanf(in , "%d %d" , &G , &L);
	fclose(in);

	if(L == 0)
	{
		printf("0 %d₩n",G);
		return 0;
	}

	L /= G;

	int i;
	int mindiff = L;
	int ret = 1;
	for(i=1;i*i<=L;i++)
	{
		if( (L%i) == 0 )
		{
			int j = L/i;
			if( gcd(i,j) != 1) continue;

			if( abs(i-j) < mindiff )
			{
				mindiff = abs(i-j);
				ret = i;
			}
		}
	}

	FILE *out = fopen("output.txt", "w");
	fprintf(out, "%d %d₩n", ret*G, L/ret*G);

	return 0;
}
```

초등부 2번 문제(BB) 모범 풀이(BASIC)

```
Attribute VB_Name = "Module1"
Sub main()
    Open "input.txt" For Input As #1
    Open "output.txt" For Output As #2

    Dim G As Long
    Dim L As Long

    Input #1, G
    Input #1, L

    L = L / G

    Dim minsum, mind1, mind2 As Long
    minsum = L + 1
    mind1 = G
    mind2 = L * G

    For i = 1 To L
        If i * i > L Then GoTo br
        If (L Mod i) = 0 Then
            Dim j, kk As Long
            j = L / i
            kk = i
            While True
                If j > kk Then
                    Dim tmp As Long
                    tmp = j
                    j = kk
                    kk = tmp
                End If
                If j = 0 Then GoTo bb
                kk = kk Mod j
            Wend
bb:
            If kk > 1 Then GoTo ebr
            Dim sum As Long
            sum = i + L / i

            If sum < minsum Then
                minsum = sum
                mind1 = i * G
                mind2 = L / i * G
            End If
        End If
ebr:
    Next i
br:

    Print #2, LTrim(Str(mind1)) + " " + LTrim(Str(mind2))
End Sub
```

저울

하나의 양팔 저울을 이용하여 물건의 무게를 측정하려고 한다. 이 저울의 양 팔의 끝에는 물건이나 추를 올려놓는 접시가 달려 있고, 양팔의 길이는 같다. 또한, 저울의 한쪽에는 저울추들만 놓을 수 있고, 다른 쪽에는 무게를 측정하려는 물건만 올려놓을 수 있다.



무게가 양의 정수인 N개의 저울추가 주어질 때, 이 추들을 사용하여 측정할 수 없는 양의 정수 무게 중 최소값을 구하는 프로그램을 작성하시오.

예를 들어, 무게가 각각 3, 1, 6, 2, 7, 30, 1인 7개의 저울추가 주어졌을 때, 이 추들로 측정할 수 없는 양의 정수 무게 중 최소값은 21이다.

실행 파일의 이름은 CC.EXE로 하고 실행 시간은 1초를 넘을 수 없다. 부분 점수는 없다.

입력 형식

입력 파일의 이름은 INPUT.TXT이다. 첫째 줄에는 저울추의 개수를 나타내는 양의 정수 N이 주어진다. N은 1 이상 1,000 이하이다. 둘째 줄에는 저울추의 무게를 나타내는 N개의 양의 정수가 빈칸을 사이에 두고 주어진다. 각 추의 무게는 1이상 1,000,000 이하이다.

출력 형식

출력 파일의 이름은 OUTPUT.TXT이다. 첫째 줄에 주어진 추들로 측정할 수 없는 양의 정수 무게 중 최소값을 출력한다.

제약 조건

- $1 \leq N \leq 10$인 경우는 전체 테스트 데이터의 30%이다.

입력과 출력의 예

입력(INPUT.TXT)

```
7
3 1 6 2 7 30 1
```

출력(OUTPUT.TXT)

```
21
```

풀이

이 문제는 주어진 추를 이용해 만들 수 없는 최소 무게를 구하는 문제이다.

추를 무게가 작은 순서대로 사용해서 만들 수 있는 무게를 생각해 보자. 무게 1을 만들려면 가장 가벼운 추의 무게가 1이어야 한다. 그 다음 무게 2를 만들려면 무게가 1인 추가 하나 더 있거나 무게가 2인 추가 있어야 한다. 만약 무게가 2인 추가 있다면 무게를 3 까지 만들 수 있다.

위와 같은 방식으로 생각해보면 N개의 추를 사용해 1 이상 M 이하의 무게를 만들 수 있을 때 $N+1$번째로 가벼운 추의 무게 w_{N+1}가 $M+1$보다 작거나 같으면 $N+1$개의 추를 사용해 $M+1$은 물론, 1 이상 $M+w_{N+1}$ 이하의 무게를 모두 만들 수 있다.

위를 모든 추에 대해 반복하여 주어진 추를 무게가 작은 순서대로 정렬하면 만들 수 없는 최소 무게를 간단히 확인할 수 있다. 주어진 추들을 정렬하는 것은 $O(N^2)$ 혹은 $O(N\log N)$의 시간복잡도로 해결할 수 있고, 앞에서부터 무게의 합과 추의 무게를 비교하면서 답을 구하는 것은 $O(N)$의 시간복잡도를 가진다.

초등부 3번 문제(CC) 모범 풀이

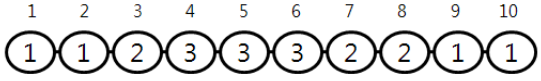
```
#include <cstdio>
#include <cassert>
#include <algorithm>
using namespace std;

int weight[1000];

int getSolution(int N) {
	int ret = 0;
	int i;
	for (i = 0; i < N; i++) {
		if (ret + 1 < weight[i])
			return ret + 1;
		ret += weight[i];
	}
	return ret + 1;
}

int main(void) {
	FILE *fin = fopen("input.txt", "rt");
	int N;
	fscanf(fin, "%d", &N);
	assert(1 <= N && N <= 1000);

	int i;
	int bef = 1, sum = 0;
	for (i = 0; i < N; i++) {
		fscanf(fin, "%d", &weight[i]);
		assert(1 <= weight[i] && weight[i] <= 1000000);
		bef = weight[i];
		sum += weight[i];
		assert(sum <= 1000000000);
	}
	fclose(fin);

	sort(weight, weight + N);

	FILE *fout = fopen("output.txt", "wt");
	fprintf(fout, "%d", getSolution(N));
	fclose(fout);

	return 0;
}
```

초등부 3번 문제(CC) 모범 풀이(BASIC)

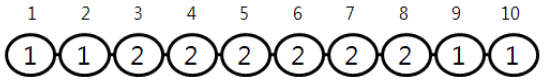
```
Attribute VB_Name = "Module1"
Sub main()

    Dim weight(1000) As Long
    Dim N, ans, tmp As Long

    Open "input.txt" For Input As #1
    Open "output.txt" For Output As #2

    Input #1, N
    For i = 1 To N
        Input #1, weight(i)
    Next i

    For i = 1 To N
        For j = i + 1 To N
            If weight(i) > weight(j) Then
                tmp = weight(i)
                weight(i) = weight(j)
                weight(j) = tmp
            End If
        Next j
    Next i


    ans = 0

    For i = 1 To N
        If ans + 1 < weight(i) Then
            Exit For
        End If
        ans = ans + weight(i)
    Next i

    Print #2, LTrim(Str(ans + 1))
End Sub
```

전구

최대 K가지의 서로 다른 색을 표현할 수 있는 전구들이 있다. 이 전구 N개를 다음의 그림과 같이 한 줄로 배치하여 서로 연결한다. (동그라미 안의 숫자는 전구의 색을 의미한다)



각 전구는 스위치가 있어서 전구의 색을 임의의 색으로 바꿀 수 있다. 하나의 전구 색을 바꾸는 경우에는, 색이 바뀌는 전구에 인접한 전구가 같은 색이면, 이 전구의 색도 같이 바뀌게 되며 인접한 전구가 다른 색이 나올 때까지 계속 바뀌게 된다. 예들 들어, 위의 그림에서 4번 전구의 색을 2번 색으로 바꾸면, 5번 전구가 4번 전구와 같은 색이었으므로 2번 색으로 바뀌고, 6번 전구도 5번 전구와 같은 색이었으므로 2번 색으로 바뀌게 된다. 즉, 4번 전구의 색을 2번 색으로 바꾸면, 연결된 같은 색의 모든 전구인 4, 5, 6번의 전구가 2번 색으로 바뀌게 되어 아래의 그림과 같이 된다.



전구의 수 N과 N개의 전등에 대한 초기 색이 주어질 때, 모든 전구의 색이 하나로 같아질 때까지 최소 몇 번 전구의 색을 바꾸어야 하는지를 구하는 프로그램을 작성하시오. 단, 전구의 각 색은 1부터 K까지의 정수로 나타낸다.

실행 파일의 이름은 DD.EXE로 하고 실행시간은 1초를 넘을 수 없다. 부분 점수는 없다.

입력 형식

입력 파일의 이름은 INPUT.TXT이다. 입력의 첫 번째 줄에는 전구의 수를 나타내는 양의 정수 N과 전구가 표현할 수 있는 색의 수 K가 주어진다. 단, N은 1이상 200이하의 정수이며, K는 1이상 20이하의 정수이다. 두 번째 줄에는 N개 전구의 색이 전구번호의 순서대로 하나의 정수로 하나의 빈칸을 사이에 두고 주어진다.

출력 형식

출력 파일의 이름은 OUTPUT.TXT이다. 첫째 줄에 모든 전구의 색이 하나로 같아질 때까지 전구의 색을 바꾸는 횟수의 최소값을 하나의 정수로 출력한다.

제약 조건

- $1 \le N \le 10$인 경우는 전체 테스트 데이터의 10%이다.
- $K = 2$인 경우는 전체 테스트 데이터의 30%이다.
- $1 \le N \le 100$이고, $1 \le K \le 10$인 경우는 전체 테스트 데이터의 80%이다.

입력과 출력의 예

입력(INPUT.TXT)

```
10 3
1 1 2 3 3 3 2 2 1 1
```

출력(OUTPUT.TXT)

```
2
```

풀이

간단한 동적 프로그래밍 문제이다.

D(i, j, c) = i~j번째 전구를 모두 c색깔로 맞추는데 필요한 연산 횟수라고 정의하였을 때 D(i, j, c)를 계산하는 방법을 생각해보면

color(i)=c이면 D(i, i, c) = 0 이고.

color(i)≠c이면 D(i, i, c) = 1이다.

D(i, j, c)= min(min(D(i, k, c)+ D(k+ 1, j, c) for $i \le k < j$), min(D(i, j, c')+ 1 for $c' \ne c$))

가 된다. (i, j사이를 적당히 갈라서 각각 c로 만드는 방법과 전체적으로 다른 한 색깔로 다 같이 맞춘 뒤 c로 만들어주는 방법이 있다)

이를 구현하면 $O(kn^3)$의 시간복잡도가 생기고 공간은 $O(kn^2)$만큼 필요하다.

초등부 4번 문제(DD) 모범 풀이

```
#include<stdio.h>
#define MAX_N 220
#define MAX_Col 22
#define INF 2100000000

int n,c,col[MAX_N];
int d[MAX_N+1][MAX_N][MAX_Col+1];

void input(){
	freopen("input.txt","r",stdin);
	scanf("%d%d",&n,&c);
	int i;
	for(i=0;i<n;i++){
		scanf("%d",&col[i]);
	}
}

void process(){
	int l,i,j,k,cp,val,min,ans=INF;
	for(i=0;i<n;i++){
		for(j=1;j<=c;j++) d[1][i][j]=1;
		d[1][i][col[i]]=0;
	}
	for(l=2;l<=n;l++){
		for(i=0;i<n-l+1;i++){
			min=INF;
			for(cp=1;cp<=c;cp++){
				d[l][i][cp]=INF;
				for(k=1;k<l;k++){
					j=i+k;
					val=d[k][i][cp]+d[l-k][j][cp];
					if(val<d[l][i][cp]) d[l][i][cp]=val;
				}
				if(d[l][i][cp]<min) min=d[l][i][cp];
			}
			for(cp=1;cp<=c;cp++){
				if(d[l][i][cp]>min) d[l][i][cp]=min+1;
				if(l==n && d[l][i][cp]<ans) ans=d[l][i][cp];
			}
		}
	}
	if(n==1) ans=0;
	freopen("output.txt","w",stdout);
	printf("%d₩n",ans);
}

int main(){
	input();
	process();
	return 0;
}
```

초등부 4번 문제(DD) 모범 풀이(BASIC)

```
Attribute VB_Name = "Module1"
Sub main()

    Dim n, c, col(220), d(220, 220, 22) As Integer
    Dim l, i, j, k, cp, val, min, ans As Integer

    Open "input.txt" For Input As #1
    Open "output.txt" For Output As #2

    Input #1, n
    Input #1, c

    For i = 1 To n
        Input #1, col(i)
    Next i

    For i = 1 To n
        For j = 1 To c
            d(1, i, j) = 1
        Next j
        d(1, i, col(i)) = 0
    Next i

    ans = n + 1

    For l = 2 To n
        For i = 1 To n - l + 1
            min = n + 1
            For cp = 1 To c
                d(l, i, cp) = n + 1
                For k = 1 To l - 1
                    j = i + k
                    val = d(k, i, cp) + d(l - k, j, cp)
                    If val < d(l, i, cp) Then d(l, i, cp) = val
                Next k
                If d(l, i, cp) < min Then min = d(l, i, cp)
            Next cp
            For cp = 1 To c
                If d(l, i, cp) > min Then d(l, i, cp) = min + 1
                If l = n And d(l, i, cp) < ans Then ans = d(l, i, cp)
            Next cp
        Next i
    Next l

    If n = 1 Then ans = 0

    Print #2, LTrim(Str(ans))
End Sub
```