# 색종이 만들기

아래 <그림 1>과 같이 여러 개의 정사각형 칸들로 이루어진 정사각형 모양의 종이가 주어져 있고, 각 정사각형 칸들은 하얀색으로 칠해져 있거나 파란색으로 칠해져 있다. 주어진 종이를 일정한 규칙에 따라 잘라서 다양한 크기를 가진 정사각형 모양의 하얀색 또한 파란색 색종이를 만들려고 한다.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 |
| 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |

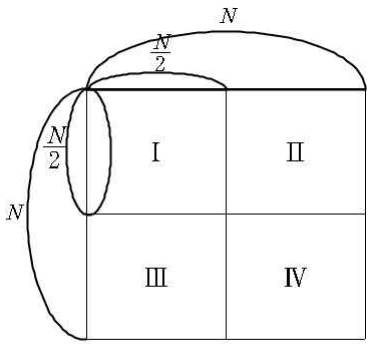
하얀색
파란색

<그림 1> 8×8 종이

전체 종이의 크기가 $N \times N$ ( $N = 2^k$, $k$는 1 이상 7 이하의 자연수) 이라면 종이를 자르는 규칙은 다음과 같다.

전체 종이가 모두 같은 색으로 칠해져 있지 않으면 가로와 세로로 중간 부분을 잘라서 <그림 2>의 Ⅰ, Ⅱ, Ⅲ, Ⅳ와 같이 똑같은 크기의 네 개의 $\frac{N}{2} \times \frac{N}{2}$ 색종이로 나눈다. 나누어진 종이 Ⅰ, Ⅱ, Ⅲ, Ⅳ 각각에 대해서도 앞에서와 마찬가지로 모두 같은 색으로 칠해져 있지 않으면 같은 방법으로 똑같은 크기의 네 개의 색종이로 나눈다. 이와 같은 과정을 잘라진 종이가 모두 하얀색 또는 모두 파란색으로 칠해져 있거나, 하나의 정사각형 칸이 되어 더 이상 자를 수 없을 때까지 반복한다.

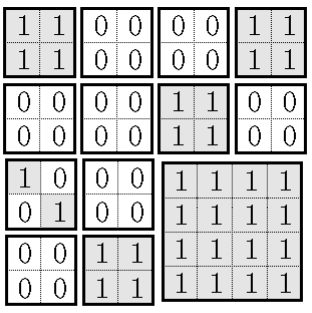
위와 같은 규칙에 따라 잘랐을 때 <그림 3>은 <그림 1>의 종이를 처음 나눈 후의 상태를, <그림 4>는 두 번째 나눈 후의 상태를, <그림 5>는 최종적으로 만들어진 다양한 크기의 9장의 하얀색 색종이와 7장의 파란색 색종이를 보여주고 있다.



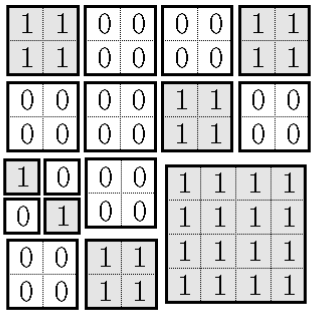

<그림 2>

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 |
| 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |

<그림 3> 처음 나눈 후의 상태



<그림 4> 두 번째 나눈 후의 상태



<그림 5> 최종적으로 나누어진 색종이들

입력으로 주어진 종이의 한 변의 길이 $N$과 각 정사각형칸의 색 (하얀색 또는 파란색)이 주어질 때 잘라진 하얀색 색종이와 파란색 색종이의 개수를 구하는 프로그램을 작성하시오.

실행 파일의 이름은 COLOR.EXE로 하고, 프로그램의 실행시간은 5초를 초과할 수 없다. 부분 점수는 없다.

## 입력 형식

입력 파일명은 INPUT.TXT로 한다. 입력 파일의 첫째 줄에는 전체 종이의 한 변의 길이 $N$이 주어져 있다. $N$은 2, 4, 8, 16, 32, 64, 128 중 하나이다. 색종이의 각 가로줄의 정사각형칸들의 색이 윗줄부터 차례로 입력 파일의 둘째 줄부터 마지막 줄까지 주어진다. 하얀색으로 칠해진 칸은 0, 파란색으로 칠해진 칸은 1로 주어지며 각 숫자 사이에는 빈 칸이 하나씩 있다.

## 출력 형식

출력 파일명은 OUTPUT.TXT로 한다. 첫째 줄에는 잘라진 하얀색 색종이의 개수를 출력하고 둘째 줄에는 파란색 색종이의 개수를 출력한다.

## 입력과 출력의 예

입력(INPUT.TXT)

```
8
1 1 0 0 0 0 1 1
1 1 0 0 0 0 1 1
0 0 0 0 1 1 0 0
0 0 0 0 1 1 0 0
1 0 0 0 1 1 1 1
0 1 0 0 1 1 1 1
0 0 1 1 1 1 1 1
0 0 1 1 1 1 1 1
```

출력(OUTPUT.TXT)

```
9
7
```

**풀이**

이 문제는 종이에 칠해진 색깔에 대한 정보가 주어졌을 때, 같은 색깔을 갖는 정사각형 모양의 색종이를 규칙적으로 만들어 나가는 문제이다. 문제에서 색종이를 나누는 방법에 대한 설명이 주어지므로, 이 방법을 그대로 수행하면서 더 이상 나눌 필요가 없을 때까지 진행하면 된다.

이러한 자료는 쿼드트리 형태로 표현되는데, 이는 네 개의 자식 노드를 가진 트리이다. 각 색종이를 규칙대로 4등분한 후, 각각을 자식 노드로 생각하면 된다. 이러한 과정을 재귀적으로 수행해 나가는데, 만일 주어진 색종이가 모두 같은 색으로 칠해져 있으면 중단하고, 그렇지 않다면 네 개의 작은 색종이로 쪼갠다. 이제 각각의 색종이에 대해서 같은 문제를 반복해서 풀어 나가면 된다.

이러한 과정을 전체적으로 반복하면서, 중단할 때마다 해당 색종이가 무슨 색이었는지를 확인한다. 그 때마다 해당 색깔의 색종이의 개수를 하나씩 증가시킨 후, 최종적으로 저장된 값을 답으로 출력하면 된다.

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

#define MAX 128

int a[MAX][MAX];

int sumblue=0,sumwhite=0;

void ansfinder(int x, int y, int step)
{
	int blue=0,white=0,i,j;
	for (i=x;i<x+step;i++)
		for (j=y;j<y+step;j++)
			if (a[i][j]==1) blue++; else white++;
	if (blue==0) { sumwhite++; return }
	if (white==0) {sumblue++; return }
	ansfinder(x, y, step/2);
	ansfinder(x, y+step/2, step/2);
	ansfinder(x+step/2, y, step/2);
	ansfinder(x+step/2, y+step/2, step/2);
}

void main()
{
	int n,i,j;

	FILE *in = fopen("input.txt","r");
	fscanf(in,"%d",&n);
	for (i=0;i<n;i++)
		for (j=0;j<n;j++)
			fscanf(in,"%d",&a[i][j]);
	fclose(in);

	ansfinder(0,0,n);

	FILE *out = fopen("output.txt","w");
	fprintf(out,"%d\n",sumwhite);
	fprintf(out,"%d\n",sumblue);
	fclose(out);
}
```

# 줄 세우기

KOI 어린이집에는 $N$명의 아이들이 있다. 오늘은 소풍을 가는 날이다. 선생님은 1번부터 $N$번까지 번호가 적혀있는 번호표를 아이들의 가슴에 붙여주었다. 선생님은 아이들을 효과적으로 보호하기 위해 목적지까지 번호순서대로 일렬로 서서 걸어가도록 하였다. 이동 도중에 보니 아이들의 번호순서가 바뀌었다. 그래서 선생님은 다시 번호 순서대로 줄을 세우기 위해서 아이들의 위치를 옮기려고 한다. 그리고 아이들이 혼란스러워하지 않도록 하기 위해 위치를 옮기는 아이들의 수를 최소로 하려고 한다.

예를 들어, 7명의 아이들이 다음과 같은 순서대로 줄을 서 있다고 하자.

3 7 5 2 6 1 4

아이들을 순서대로 줄을 세우기 위해, 먼저 4번 아이를 7번 아이의 뒤로 옮겨보자. 그러면 다음과 같은 순서가 된다.

3 7 4 5 2 6 1

이제, 7번 아이를 맨 뒤로 옮긴다.

3 4 5 2 6 1 7

다음, 1번 아이를 맨 앞으로 옮긴다.

1 3 4 5 2 6 7

마지막으로 2번 아이를 1번 아이의 뒤로 옮기면 번호 순서대로 배치된다.

1 2 3 4 5 6 7

위의 방법으로는 모두 4명의 아이를 옮겨 번호 순서대로 줄을 세운다. 위의 예에서 3명의 아이만을 옮겨서는 순서대로 배치할 수가 없다. 따라서, 4명을 옮기는 것이 가장 적은 수의 아이를 옮기는 것이다.

$N$명의 아이들이 임의의 순서로 줄을 서 있을 때, 번호 순서대로 배치하기 위해 옮겨지는 아이의 최소 수를 구하는 프로그램을 작성하시오.

실행파일의 이름은 LINE.EXE로 하고, 프로그램의 실행시간은 5초를 초과할 수 없다. 부분 점수는 없다.

## 입력 형식

입력 파일명은 INPUT.TXT로 한다. 입력 파일의 첫째 줄에는 아이들의 수 $N$이 주어진다. 둘째 줄부터는 1부터 $N$까지의 숫자가 한 줄에 하나씩 주어진다. $N$은 2이상 200이하의 정수이다.

## 출력 형식

출력 파일명은 OUTPUT.TXT로 한다. 출력 파일의 첫째 줄에는 번호 순서대로 줄을 세우는데 옮겨지는 아이들의 최소 수를 출력한다.

## 입력과 출력의 예

입력(INPUT.TXT)

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

출력(OUTPUT.TXT)

```
4
```

**풀이**

이 문제는 아이들을 조건대로 줄 세우기 위해서 최소로 옮길 아이들을 구하는 문제이다. 주어진 문제는 옮길 아이들의 수를 구하는 것이지만, 잘 생각해보면 반대로 옮기지 않을 아이들의 수를 구하는 문제로도 풀 수 있다. 이 문제는 동적 계획법으로 해결할 수 있는 문제이다.

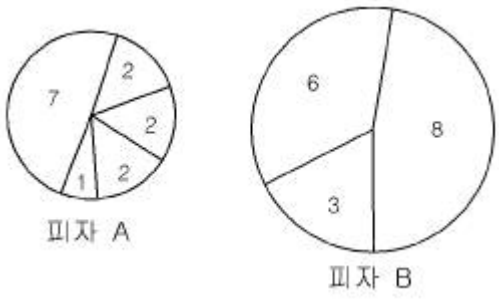
먼저 확인할 수 있는 것은 옮겨지는 아이들을 최소로 한다는 것이 곧 옮겨지지 않는 아이들을 최대로 한다는 것과 같다는 것이다. 이 때, 옮겨지지 않는 아이들의 번호는 증가하는 순서여야 하므로, 결국 이 문제는 증가하는 가장 긴 부분수열을 찾아 그 길이를 전체 학생 수에서 뺀 값을 구하면 된다.

길이가 N인 수열 A[1], ..., A[N]이 있을 때 D[i]를 i번으로 끝나는 가장 긴 증가수열의 길이로 정의하면, D[i]는 A[j]<A[i]인 1<=j<i들에 대해 D[j]+1가 가장 큰 값이다. 이는 j로 끝나는 길이 D[j]의 증가수열을 구성한 후, 그 뒤에 A[i]를 덧붙이는 것을 의미한다. 이제 D[1], ..., D[N]중 가장 큰 값을 찾아 이를 N에서 뺀 값을 출력하면 그것이 원래 문제의 답이 된다.

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

const int MAX = 210; // spare -_-

int n;
int lineup[MAX];

void get_input()
{
	int i;
	FILE *in = fopen("input.txt","r");
	fscanf(in,"%d",&n);
	for (i=0;i<n;i++)
		fscanf(in,"%d",&lineup[i]);
	fclose(in);
}

int find_MIS()
{
	int i,j,max=0;
	int partmax[MAX];
	for (i=0;i<n;i++) {
		partmax[i] = 1;
		for (j=0;j<i;j++)
			if (lineup[j]<lineup[i])
				if (partmax[j]+1>partmax[i])
					partmax[i] = partmax[j]+1;
		if (partmax[i]>max)
			max = partmax[i];
	}
	return max;
}

void print_out(int ans)
{
	FILE *out = fopen("output.txt","w");
	fprintf(out,"%d\n",ans);
	fclose(out);
}

void main()
{
	get_input();
	int incr = find_MIS();
	print_out(n-incr);
}
```

## 피자 판매

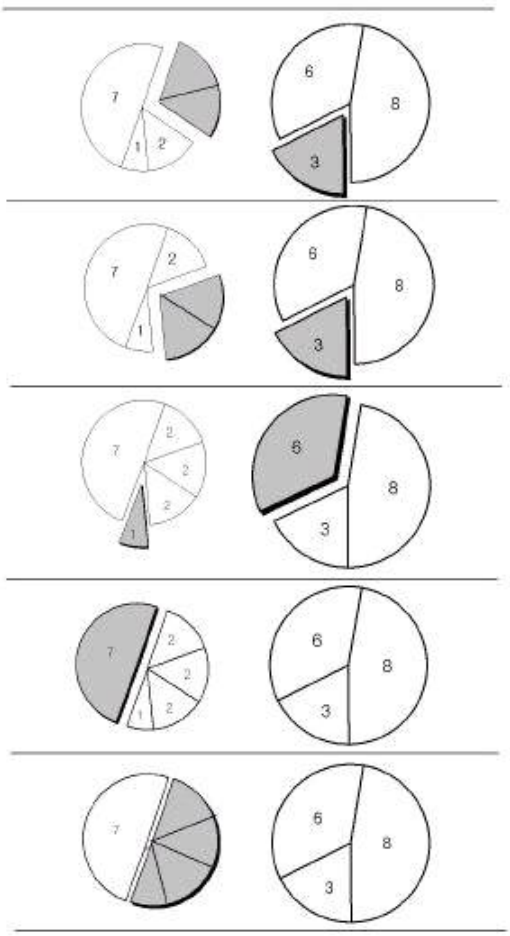
고객이 두 종류의 피자 A와 B를 취급하는 피자가게에서 피자를 주문하고자 한다. <그림 1>과 같이 각 종류의 피자는 다양한 크기의 여러 개의 피자조각으로 나누어져 있다. 각 조각에 쓰여진 숫자는 피자조각의 크기를 나타낸다.




고객이 원하는 피자의 크기를 이야기하면, 피자가게에서는 한 종류의 피자를 2조각 이상 판매할 때는 반드시 연속된 조각들을 잘라서 판매한다. 이때 판매한 피자조각의 크기 합이 주문한 크기가 되어야 한다. 판매한 피자조각은 모두 A 종류이거나, 모두 B종류이거나, 또는 A와 B 종류가 혼합될 수 있다. 예를 들어서, <그림 1>과 같이 잘라진 피자가 있을 때, 손님이 전체 크기가 7 인 피자를 주문하면, 피자 가게에서는 <그림 2>와 같이 5 가지 방법으로 피자를 판매할 수 있다.

피자가게에서 손님이 원하는 크기의 피자를 판매하는 모든 방법의 가지 수를 계산하는 프로그램을 작성하시오.

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



<그림 2>

## 입력 형식

입력 파일명은 INPUT.TXT로 한다. 입력 파일의 첫 번째 줄에는 손님이 구매하고자 하는 피자크기를 나타내는 2,000,000 이하의 자연수가 주어진다. 두 번째 줄에는 A, B 피자의 피자조각의 개수를 나타내는 정수 $m, n$ 이 차례로 주어진다 ( $3 \le m, n \le 1000$). 세 번째 줄부터 차례로 $m$ 개의 줄에는 피자 A의 미리 잘라진 피자조각의 크기를 나타내는 정수가 주어진다. 그 다음 $n$ 개의 줄에는 차례로 피자 B의 미리 잘라진 피자조각의 크기를 나타내는 정수가 주어진다. 각 종류의 피자조각의 크기는 시

계방향으로 차례로 주어지며, 각 피자 조각의 크기는 1000 이하의 자연수이다.

## 출력 형식

출력 파일명은 OUTPUT.TXT로 한다. 출력 파일의 첫째 줄에는 피자를 판매하는 방법의 가지 수를 나타내는 정수를 출력한다. 피자를 판매하는 방법이 없는 경우에는 숫자 0을 출력한다.

## 입력과 출력의 예

입력(INPUT.TXT)

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

출력(OUTPUT.TXT)

```
5
```

**풀이**

이 문제는 피자 조각에 대한 정보가 주어졌을 때, 피자를 판매하는 방법의 경우의 수를 구하는 문제이다. 쉽게 생각할 수 있는 방법은 가능한 모든 경우를 따져본 후, 그들 중 실제로 판매가 가능한 경우의 개수를 세는 방법이다. 이 때 이를 효율적으로 잘 프로그래밍하면 문제를 해결할 수 있다.

피자 조각의 개수를 n이라고 하면, 피자 A와 B 각각에 대해 연속적인 합이 될 수 있는 모든 가짓수를 $O(n^2)$에 계산할 수 있다. 이제 이를 각각 정렬하는데, 하나는 오름차순으로 정렬하고 나머지 하나는 내림차순으로 정렬한다. 이제 각각 순서대로 비교하면 합이 입력에서 주어진 값과 같은 경우를 $O(n^2)$에 찾을 수 있다. 정렬이 필요하므로 전체 시간 복잡도는 $O(n^2 \log n)$이 된다.

이렇게 하면 시간 내에 답을 내는 풀이를 어렵지 않게 만들어낼 수 있지만, 당시 대회 환경에서는 $n^2$의 크기를 갖는 배열을 잡을 수 없었다. 따라서 두 개의 크기 n인 힙을 이용해서 위의 과정을 재현해 냈는데, A에 대해서는 최소 힙, B에 대해서는 최대 힙을 만들어서, A에 대해서는 각 위치별로 인접한 한 개의 피자만을 고려했을 때의 합을 각각 가지고 있고, B에 대해서는 마찬가지로 각 위치별로 인접한 n-1개의 피자만들 고려했을 때의 합을 각각 가지고 있도록 한다. 처리를 마친 뒤 힙에서 원소를 제거하면서, 구간을 하나 더 보도록 늘려서 다시 힙에 삽입하고, B에 대해서는 마찬가지로 구간을 하나 줄여서 다시 힙에 삽입하면 위와 동일한 작업을 수행해 낼 수 있다. 크기 n인 힙에 대해 삽입과 삭제 연산을 $O(n^2)$번 수행하므로 시간복잡도는 마찬가지로 $O(n^2 \log n)$이 된다.

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

#define INF 99999999

// The O(N^2 log N) algorithm.

const int MAX = 1000;
int n,m;
long weight;
long pa[MAX],pb[MAX];

void get_input();
long go_go_main();
void print_out(long sum);

void main()
{
        long sum=0;

        get_input();

        sum = go_go_main();

        print_out(sum);
}

void get_input()
{
        int i;
        FILE *in = fopen("input.txt","r");
        fscanf(in,"%ld",&weight);
        fscanf(in,"%d %d",&n,&m);
        for (i=0;i<n;i++)
                fscanf(in,"%ld",&pa[i]);
        for (i=0;i<m;i++)
                fscanf(in,"%ld",&pb[i]);
        fclose(in);
}

long sumA[MAX]; int placeA[MAX],heapA[MAX];
long sumB[MAX]; int placeB[MAX],heapB[MAX];

void init_A()
{
        int i,j,temp;
        sumA[0] = 0;
        for (i=1;i<n;i++)
                sumA[i] = pa[i];
        placeA[0] = 0;
        for (i=1;i<n;i++)
```

```
                placeA[i] = 1;
        for (i=0;i<n;i++)
                heapA[i] = i;
        for (i=0;i<n;i++)
                for (j=i+1;j<n;j++)
                        if (sumA[heapA[i]]>sumA[heapA[j]]) {
                                temp = heapA[i];
                                heapA[i] = heapA[j];
                                heapA[j] = temp;
                        }
}

void init_B()
{
        int i,j,temp;
        sumB[0] = 0;
        for (i=0;i<m;i++)
                sumB[0] += pb[i];
        for (i=1;i<m;i++)
                sumB[i] = sumB[0] - pb[i];
        placeB[0] = m;
        for (i=1;i<m;i++)
                placeB[i] = m-1;
        for (i=0;i<m;i++)
                heapB[i] = i;
        for (i=0;i<m;i++)
                for (j=i+1;j<m;j++)
                        if (sumB[heapB[i]]<sumB[heapB[j]]) {
                                temp = heapB[i];
                                heapB[i] = heapB[j];
                                heapB[j] = temp;
                        }
}

long out_and_in_A()
{
        long retnum;

        int hp = heapA[0];
        retnum = sumA[hp];
        if (((placeA[hp]==n-1) && (hp!=0))
                || ((placeA[hp]==n) && (hp==0)))
                sumA[hp] = +INF;
        else
                sumA[hp] += pa[(hp+placeA[hp])%n];
        placeA[hp]++;
        heapA[0] = heapA[n-1];

        int pm;
        int temp,chal;
```

```
	pm = 0;
	while (1) {
		// no challenger
		if (pm*2+1>=n-1) break

		if (pm*2+1==n-2)
			chal = pm*2+1; // 1 challenger
		else if (sumA[heapA[pm*2+1]]>sumA[heapA[pm*2+2]])
			chal = pm*2+2; // 2 challenger and the latter defeats
		else chal = pm*2+1; // 2 challenger and the former defeats

		if (sumA[heapA[pm]]>sumA[heapA[chal]]) {
			temp = heapA[pm];
			heapA[pm] = heapA[chal];
			heapA[chal] = temp;
		}
		else
			break

		pm = chal;
	}

	heapA[n-1] = hp;
	pm = n-1;
	while (1) {
		// no challenger
		if (pm==0) break

		// 1 challenger
		chal = (pm-1)/2;

		if (sumA[heapA[pm]]<sumA[heapA[chal]]) {
			temp = heapA[pm];
			heapA[pm] = heapA[chal];
			heapA[chal] = temp;
		}
		else
			break

		pm = chal;
	}
	return retnum;
}

long out_and_in_B()
{
	long retnum;

	int hp = heapB[0];
	retnum = sumB[hp];
	if (((placeB[hp]==1) && (hp!=0))
```

```
            || ((placeB[hp]==0) && (hp==0)))
            sumB[hp] = -INF;
else
            sumB[hp] -= pb[(hp+placeB[hp])%m];
placeB[hp]--;
heapB[0] = heapB[m-1];

int pm;
int temp,chal;

pm = 0;
while (1) {
            // no challenger
            if (pm*2+1>=m-1) break

            if (pm*2+1==m-2)
                        chal = pm*2+1; // 1 challenger
            else if (sumB[heapB[pm*2+1]]<sumB[heapB[pm*2+2]])
                        chal = pm*2+2; // 2 challenger and the latter defeats
            else chal = pm*2+1; // 2 challenger and the former defeats

            if (sumB[heapB[pm]]<sumB[heapB[chal]]) {
                        temp = heapB[pm];
                        heapB[pm] = heapB[chal];
                        heapB[chal] = temp;
            }
            else
                        break

            pm = chal;
}

heapB[m-1] = hp;
pm = m-1;
while (1) {
            // no challenger
            if (pm==0) break

            // 1 challenger
            chal = (pm-1)/2;

            if (sumB[heapB[pm]]>sumB[heapB[chal]]) {
                        temp = heapB[pm];
                        heapB[pm] = heapB[chal];
                        heapB[chal] = temp;
            }
            else
                        break

            pm = chal;
}
```

```
        return retnum;
}

long exA, exB;
int nuA, nuB;

void extract_A()
{
        nuA = 0;
        exA = sumA[heapA[0]];
        if (exA==+INF) return
        while (sumA[heapA[0]]==exA) {
                out_and_in_A();
                nuA++;
        }
}

void extract_B()
{
        nuB = 0;
        exB = sumB[heapB[0]];
        if (exB==-INF) return
        while (sumB[heapB[0]]==exB) {
                out_and_in_B();
                nuB++;
        }
}

long go_go_main()
{
        long ret_num = 0;

        init_A();
        init_B();

        extract_A();
        extract_B();

        while (1) {
                if (exA+exB==weight)
                        ret_num+=((long)nuA)*((long)nuB);

                if (exA+exB>weight)
                        extract_B();
                else extract_A();
                if (exA==+INF || exB==-INF)
                        return ret_num;
        }
}

void print_out(long sum)
```

```
{
        FILE *out = fopen("output.txt","w");
        fprintf(out,"%ld\n",sum);
        fclose(out);
}
```