[1.5] 대표문제. 평탄화

대표문제. 평탄화

한쪽 벽면에 다음과 같이 노란색 상자들이 쌓여있었다. 높은 곳의 상자를 낮은 곳에 옮기는 방식으로 최고점과 최저점의 간격을 줄이는 작업을 평탄화라고 한다. 평탄화를 모두 수행하고 나면, 가장 높은 곳과 가장 낮은 곳의 차이가 최대 1이내가 된다. 평탄화 작업을 위해서 상자를 옮기는 작업 횟수에 제한이 걸려있을 때, 제한된 횟수만큼 옮기는 작업을 한 이후, 최고점과 최저점의 차이를 리턴하는 함수를 구현하시오.

가장 높은 곳에 있는 상자를 가장 낮은 곳으로 옮기는 작업을 덤프라고 정의한다.

생각해 봅시다

어떻게 풀 수 있을까?!

[1.5] 대표문제. 평탄화 해결하기

입력

첫 번째 줄에는 덤프횟수가 주어진다. 그리고 바로 다음 줄에 테스트 케이스가 주어진다. 총 10개의 테스트케이스가 주어진다.

입력 예시

834
42 68 35 1 70 25 82 28 62 92 96 43 28 37 92 5 3 54 93 83 22 17 19 96...
617
16 40 59 5 31 23 25 73 71 30 78 74 98 13 87 91 62 37 56 68 56 75 32 53...

출력

각 테스트케이스의 최고점과 최저점의 높이 차를 출력한다.

출력 예시

13
32

해설

위의 예시에서 제1회 덤프를 수행한 이후 화면은 다음과 같다.

A부분의 상자를 가장 낮은 B부분에 덤프하였으며, A대신 A'부분의 상자를 사용해도 무방하다. 다음은 제2회 덤프를 수행한 이후의 화면이다. A'부분의 상자를 옮겨서, C 부분에 덤프하였다, 이때 C 대신 C'부분에 덤프해도 무방하다.

2회의 덤프 후, 최고점과 최저점의 차이는 8 – 2 = 6 이 되었다. (최초덤프 이전에는 9 – 1 = 8 이었다.) 덤프횟수가 2회로 제한된다면, 이 예시문제의 정답은 6이 된다.

덤프 횟수 또는 최고점과 최하점의 차이가 1이하일때까지 아래의 과정을 반복하면 본 문제의 정답을 구할 수 있다..

1. 상자 높이 중 최고 점을 찾는다.
2. 상자 높이 중 최하 점을 찾는다.
3. 찾아진 최고점에서 1 감소하고 찾아진 최하점에서 1 증가 한다. 최고점과 최하점의 차이를 계산하여 반환한다.

코드 제공

```
1  dump(data[100], dumpCount)
2  {
3    int max ← 0; // 최고높이
4    int min ← 0; // 최저높이
5    int maxIndex, minIndex; // 최고,최저높이의 인덱스 값을 저장
6
7    while(dumpCount != 0) // 카운트가 0이 되면 loop를 나옴
8    {
9      max ← data[0]; // 순차검색을 위한 초기화
10     min ← data[0];
```

```
11     maxIndex ← 0;
12     minIndex ← 0;
13
14     for i from 1 to 99
15     {
16       if(max < data[i]) // 최고 높이의 상자 찾기
17       {
18         max ← data[i];
19         maxIndex ← i;
20       }
21       if(min > data[i]) // 최저 높이의 상자 찾기
22       {
23         min ← data[i];
24         minIndex ← i;
25       }
26     }
27
28     data[maxIndex] ← data[maxIndex] - 1; // 최고 높이 감소
29     data[minIndex] ← data[maxIndex] + 1; // 최저 높이 증가
30
31     dumpCount ← dumpCount - 1; // 카운트 감소
32   }
33
34   /* 최종 덤프 수행 후, 최고점과 최저점의 높이차 반환*/
35   max ← data[0]; // 순차검색을 위한 초기화
36   min ← data[0];
37
38   for i from 1 to 99
39   {
40     if(max < data[i]) // 최고 높이의 상자 찾기
41     {
42       max ← data[i];
43     }
44
45     if(min > data[i]) // 최저 높이의 상자 찾기
46     {
47       min ← data[i];
48     }
49   }
50
51   return max-min;
52 }
53
54 main()
55 {
56   int dumpCount;
57   int data[100];
58
59   for i from 0 to 9
```

```
60   {
61     read dumpCount;
62
63     for j from 0 to 99
64     {
65       read data[j];
66     }
67
68     print '#', i+1, ' ',  dump(data, dumpCount);
69
70   }
71
72   return 0;
73 }
```

14 ~ 26	상자들의 높이 중 최고점과 최저점을 찾는다.
28 ~ 29	최고 높이에 있는 상자를 최저 높이를 가지는 위치로 이동한다.