Qualification Round 2010 Google Code Jam

Theme Park

Problem

Roller coasters are so much fun! It seems like everybody who visits the theme park wants to ride the roller coaster. Some people go alone; other people go in groups, and don't want to board the roller coaster unless they can all go together. And *everyone* who rides the roller coaster wants to ride again. A ride costs 1 Euro per person; your job is to figure out how much money the roller coaster will make today.

The roller coaster can hold **k** people at once. People queue for it in groups. Groups board the roller coaster, one at a time, until there are no more groups left or there is no room for the next group; then the roller coaster goes, whether it's full or not. Once the ride is over, all of its passengers re-queue in the same order. The roller coaster will run **R** times in a day.

For example, suppose **R**=4, **k**=6, and there are four groups of people with sizes: 1, 4, 2, 1. The first time the roller coaster goes, the first two groups [1, 4] will ride, leaving an empty seat (the group of 2 won't fit, and the group of 1 can't go ahead of them). Then they'll go to the back of the queue, which now looks like 2, 1, 1, 4. The second time, the coaster will hold 4 people: [2, 1, 1]. Now the queue looks like 4, 2, 1, 1. The third time, it will hold 6 people: [4, 2]. Now the queue looks like [1, 1, 4, 2]. Finally, it will hold 6 people: [1, 1, 4]. The roller coaster has made a total of 21 Euros!

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow, with each test case consisting of two lines. The first line contains three space-separated integers: **R**, **k** and **N**. The second line contains **N** space-separated integers $\mathbf{g_i}$, each of which is the size of a group that wants to ride. $\mathbf{g_0}$ is the size of the first group, $\mathbf{g_1}$ is the size of the second group, etc.

Output

For each test case, output one line containing "Case #x: y", where x is the case number (starting from 1) and y is the number of Euros made by the roller coaster.

Limits

$1 \leq \mathbf{T} \leq 50$.
$\mathbf{g_i} \leq \mathbf{k}$.

Small dataset

$1 \leq \mathbf{R} \leq 1000$.
$1 \leq \mathbf{k} \leq 100$.
$1 \leq \mathbf{N} \leq 10$.
$1 \leq \mathbf{g_i} \leq 10$.

Large dataset

$1 \le \mathbf{R} \le 10^8$.
$1 \le \mathbf{k} \le 10^9$.
$1 \le \mathbf{N} \le 1000$.
$1 \le \mathbf{g_i} \le 10^7$.

Sample

Input	Output
3 4 6 4 1 4 2 1 100 10 1 1 5 5 10 2 4 2 3 4 2 1 2 1 3	Case #1: 21 Case #2: 100 Case #3: 20

Colheita de Caju

Nome do arquivo fonte: `caju.c`, `caju.cpp`, *ou* `caju.pas`

Conrado é gerente em uma das fazendas de plantação de caju da Sociedade de Beneficiamento de Caju (SBC), um grupo que cultiva caju em grandes propriedades para o mercado externo.

Os cajueiros são plantados dispostos em linhas e colunas, formando uma espécie de grade. Na fazenda administrada por Conrado existem L linhas de cajueiros, cada uma formada por C colunas. Nesta semana Conrado deve executar a colheita da produção de um subconjunto contínuo de cajueiros. Esse subconjunto é formado por M linhas e N colunas de cajueiros. Há uma semana, seus funcionários analisaram cada cajueiro da fazenda e estimaram a sua produtividade em número de cajus prontos para a colheita. Conrado agora precisa da sua ajuda para determinar qual a produtividade máxima estimada (em número de cajus) de uma área de $M \times N$ cajueiros.

Tarefa

Sua tarefa é escrever um programa que, dado um mapa da fazenda contendo o número de cajus prontos para colheita em cada cajueiro, encontre qual o número máximo de cajus que podem ser colhidos na fazenda em uma área de $M \times N$ cajueiros.

Entrada

A entrada contém um único conjunto de testes, que deve ser lido do *dispositivo de entrada padrão* (normalmente o teclado). A primeira linha da entrada contém quatro números inteiros, L, C, M e N. L e C representam, respectivamente, o número de linhas ($1 \leq L \leq 1000$) e de colunas ($1 \leq C \leq 1000$) de cajueiros existentes na fazenda. M e N representam, respectivamente, o número de linhas ($1 \leq M \leq L$) e de colunas ($1 \leq N \leq C$) de cajueiros a serem colhidos. As L linhas seguintes contêm C inteiros cada, representando número de cajus prontos para colheita no cajueiro localizado naquela linha e coluna.

Saída

Seu programa deve imprimir, na *saída padrão*, uma única linha que contém o número máximo estimado de cajus que podem ser colhidos em uma área contínua de $M \times N$. Esse número não será superior a 1000000

Entrada	**Entrada**	**Entrada**
3 3 1 1 1 2 3 1 3 3 1 10 1	4 4 2 1 1 2 3 4 5 6 7 8 1 10 5 2 1 5 9 10	5 5 2 2 1 1 1 3 1 1 2 1 1 1 1 1 1 2 1 1 1 2 1 1 1 3 1 1 3
Saída	**Saída**	**Saída**
10	16	7

Uniform Generator

Computer simulations often require random numbers. One way to generate pseudo-random numbers is via a function of the form

$$seed(x+1) = [seed(x) + STEP]\%MOD$$

where `` $\%$ '' is the modulus operator.

Such a function will generate pseudo-random numbers (*seed*) between 0 and *MOD*-1. One problem with functions of this form is that they will always generate the same pattern over and over. In order to minimize this effect, selecting the *STEP* and *MOD* values carefully can result in a uniform distribution of all values between (and including) 0 and *MOD*-1.

For example, if *STEP* = 3 and *MOD* = 5, the function will generate the series of pseudo-random numbers 0, 3, 1, 4, 2 in a repeating cycle. In this example, all of the numbers between and including 0 and *MOD*-1 will be generated every *MOD* iterations of the function. Note that by the nature of the function to generate the same *seed*(*x*+1) every time *seed*(*x*) occurs means that if a function will generate all the numbers between 0 and *MOD*-1, it will generate pseudo-random numbers uniformly with every *MOD* iterations.

If *STEP* = 15 and *MOD* = 20, the function generates the series 0, 15, 10, 5 (or any other repeating series if the initial seed is other than 0). This is a poor selection of *STEP* and *MOD* because no initial seed will generate all of the numbers from 0 and *MOD*-1.

Your program will determine if choices of *STEP* and *MOD* will generate a uniform distribution of pseudo-random numbers.

Input

Each line of input will contain a pair of integers for *STEP* and *MOD* in that order ($1 \leq STEP, MOD \leq 10\,000\,000$).

Output

For each line of input, your program should print the *STEP* value right- justified in columns 1 through 10, the *MOD* value right-justified in columns 11 through 20 and either ``Good Choice'' or ``Bad Choice'' left-justified starting in column 25. The ``Good Choice'' message should be printed when the selection of *STEP* and *MOD* will generate all the numbers between and including 0 and *MOD*-1 when MOD numbers are generated. Otherwise, your program should print the message ``Bad Choice''. After each output test set, your program should print exactly one blank line.

Sample Input

```
3 5
15 20
63923 99999
```

Sample Output

```
         3         5    Good Choice

        15        20    Bad Choice

     63923     99999    Good Choice
```

Theme Park
Colheita de Caju
Uniform Generator

No problema Uniform Generator, descobrimos que conhecer alguns resultados de teorias de números pode ser bastante útil para melhorar um algoritmo.
Neste problema, temos que descobrir se um número é uma boa escolha ou uma má escolha para uma função que vai gerar números pseudo randômicos.

seed(0) = 0
seed(x+1) = (seed(x) + STEP) % MOD

Temos que escolher os valores de STEP e MOD cuidadosamente para resultar em uma distribuição uniforme dos valores entre 0 e MOD-1.
Note que para alguns valores como STEP=3 e MOD=5 podemos gerar todos os valores 0,3,1,4,2,0,...
Mas para outros como STEP=15 MOD = 20 vamos gerar 0, 15, 10, 5,0
Para encontrar um STEP gerador em Z_MOD temos que mdc(STEP,MOD)=1

Colheita de Caju

É um problema bastante interessante que podemos transformar uma solução trivial ~ O(n^4) em uma solução O(n^2) usando técnica de programação dinâmica.

Theme Park

Um problema da rodada de qualificação do google code jam 2010.
Vc tem que decidir quanto uma montanha-russa vai faturar considerando que N grupos de pessoas estão em uma fila e você vai funcionar a montanha-russa R vezes e cada grupo entra ou não entra na montanha-russa.

O algoritmo trivial seria simular a cada iteração quais são os grupos que entrarão na montanha-russa O(NR)

Otimização 1

Criar um vetor SUM[a,b] = somatório do número de pessoas no intervalo entre grupo a e o grupo b + busca binária para descobrir que vai entrar na montanha-russa
O(Rlog N)

Otimização 2

Podemos contruir uma tabela de transição da seguinte maneira dado um grupo i podemos descobrir quantas pessoas vão entrar na montanha-russa e em qual grupo ele vai ficar.

Com essa idéia podemos desenvolver um algoritmo O(R)

Revendo as restrições do problema, podemos notar que R <= 10^8 e N<=10^3

Podemos notar que o tamanho do maior ciclo é N. Temos que rodar no máximo N para encontrar um ciclo.