Augusta

1^a Seletiva Interna – 2010/1

Sevidor BOCA:

`http://192.168.0.21:8001`
(acesso interno)

`http://200.19.107.16:8001`
(acesso externo)

Organizadores:

Alexandre Gonçalves Silva, Roberto Silvio Ubertino Rosso Jr., Claudio Cesar de Sá
{alexandre,rosso,claudio} at joinville dot udesc dot br

Lembretes:

- É permitido consultar livros, anotações ou qualquer outro material impresso durante a prova.
- A correção é automatizada, portanto, siga atentamente as exigências da tarefa quanto ao formato da entrada e saída de seu programa. Deve-se considerar entradas e saídas padrão.
- Procure resolver o problema de maneira eficiente. Se o tempo superar o limite pré-definido, a solução não é aceita. As soluções são testadas com outras entradas além das apresentadas como exemplo dos problemas.
- Teste seu programa antes de submetê-lo. A cada problema detectado (erro de compilação, erro em tempo de execução, solução incorreta, formatação imprecisa, tempo excedido ...), há penalização de 20 minutos. O tempo é critério de desempate entre duas ou mais equipes com a mesma quantidade de problemas resolvidos.
- Utilize o *clarification* para dúvidas da prova. Os juízes podem opcionalmente atendê-lo com respostas acessíveis a todos.

Augusta

1^a Seletiva Interna da UDESC

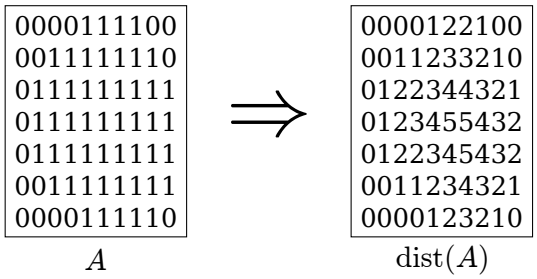
22 de maio de 2010

Conteúdo

1 Problema A: Transformada de distância

`Arquivo: dist.[c|cpp|java]`

Em uma matriz, a distância entre dois pontos p_0 e p_n pode ser definida como o comprimento n do menor caminho entre eles, dado por $p_0, p_1, \ldots, p_k, \ldots, p_n$, tal que p_k é vizinho de p_{k-1} para qualquer k $(1 \leq k \leq n)$ da sequência. Para simplificar, pontos p_k e p_{k-1} são vizinhos se posicionados: (a) na mesma linha e em colunas consecutivas; ou (b) na mesma coluna e em linhas consecutivas. A **transformada de distância** é uma operação que associa, a cada ponto de um objeto (igual a 1) em uma matriz, sua menor distância (conforme definido anteriormente) a pontos de fundo (iguais a 0). Exemplo de transformação da matriz A:



Este resultado pode ser utilizado em outras ferramentas como determinação de caminho mínimo entre dois pontos, diagrama de Voronoi, dimensão fractal, erosão e esqueleto morfológicos, entre outros algoritmos. Você está desenvolvendo uma técnica de casamento de formas, na qual a transformada de distância tem papel fundamental. O interesse está, em especial, nas coordenadas onde a distância dos pontos do objeto (valor 1) ao fundo (valor 0) é máxima.

Especificação da entrada

A entrada inicia com o número de casos de teste T $(1 \leq T \leq 100)$. Para cada teste, a primeira linha contém dois inteiros H e W com o número de linhas e de colunas, respectivamente, da matriz $(1 \leq H, W \leq 256)$. As H linhas seguintes referem-se à matriz binária.

Especificação da saída

Para cada caso de teste, deve-se produzir uma linha com o maior valor da transformada distância e a(s) coordenada(s), no formato $(linha, coluna)$, onde tal valor se encontra (as coordenadas são listadas na ordem de varredura "esquerda para direita e de cima para baixo"; o primeiro índice de linha e coluna é igual a zero).

Exemplo de entrada

```
1
5 7
0111000
1111100
1111100
1111100
0111000
```

Exemplo de saída

```
3 (1,2) (2,1) (2,2) (3,2)
```

2 Problema B: Decodifique os strings

`Arquivo: decod.[c|cpp|java]`

Bruce Force teve uma ideia interessante para codificar strings. Segue a descrição de como a codificação é feita:

Sendo $x_1, x_2, \ldots, x_n$ a sequência de caracteres da string a ser codificada.

1. Escolha um inteiro m e n números distintos $p_1, p_2, \ldots, p_n$ a partir do conjunto $\{1, 2, \ldots, n\}$ (uma permutação dos números de 1 a n).
2. Repita o passo seguinte m vezes.
3. Para $1 \leq i \leq n$, atribua x_{p_i} para y_i, e então, para $1 \leq i \leq n$, substitua x_i por y_i.

Por exemplo, quando queremos codificar a string "hello", e nós escolhemos o valor $m = 3$ e permutação $2, 3, 1, 5, 4$, os dados seriam codificados em três etapas:
"hello" $\rightarrow$ "elhol" $\rightarrow$ "lhelo" $\rightarrow$ "helol".

Bruce dá-lhe as strings codificadas, e os números m e $p_1, \ldots, p_n$ usado para codificar essas sequências. Ele afirma que porque ela usou um números altos de m para a codificação, você precisará de muito tempo para decodificar as sequências. Você pode desmentir esta afirmação através de uma rápida decodificação das strings?

Especificação da entrada

A entrada contém vários casos de teste. Cada caso de teste inicia com uma linha contendo dois números n e m ($1 \leq n \leq 80$, $1 \leq m \leq 10^9$). A linha a seguir consiste de n números diferentes $p_1, \ldots, p_n$ ($1 \leq p_i \leq n$). A terceira linha de cada caso de teste consiste de exatamente n caracteres, e representa a string codificada. O último caso de teste é seguido por uma linha contendo dois zeros.

Especificação da saída

Para cada caso de teste, imprima uma linha com a string decodificada.

Exemplo de entrada

```
5 3
2 3 1 5 4
helol
16 804289384
13 10 2 7 8 1 16 12 15 6 5 14 3 4 11 9
scssoet tcaede n
8 12
5 3 4 2 1 8 6 7
encoded?
0 0
```

Exemplo de saída

```
hello
second test case
encoded?
```

3 Problema C: Histograma acumulado

`Arquivo: ha.[c|cpp|java]`

Um histograma é um representação da distribuição de frequências de um conjunto de valores utilizado, por exemplo, em processamento de sinais. Basicamente, para cada valor, conta-se quantas vezes o mesmo aparece no conjunto. Por exemplo, para o conjunto

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

Tem-se que o `0` aparece uma única vez, o `1` aparece três vezes, o `2`, três vezes, o `3`, nenhuma vez, o `4`, uma vez, e o `5`, duas vezes. Deste modo, o histograma para este conjunto é

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

Para alguns tipos de processamentos, tais como equalização de histograma, utiliza-se a informação do histograma acumulado. Supondo h um histograma, os valores de seu histograma acumulado ha são calculados da seguinte forma:

$ha[0] = h[0]$
$ha[1] = h[1] + ha[0]$
$\vdots$
$ha[k] = h[k] + ha[k-1]$
$\vdots$
$ha[n] = h[n] + ha[n-1]$, sendo n igual ao maior valor do conjunto.

Portanto, para o exemplo anterior, o histograma acumulado é

```
1 4 7 7 8 10
```

Especificação da entrada

A entrada inicia com um número T ($1 \leq T \leq 100$), que representa o número de casos de teste. Então seguem T linhas, cada uma com um conjunto de valores inteiros positivos, separados por espaço, sendo cada valor v limitado em um amplo intervalo ($0 \leq v \leq 1000000000$). Ao final de cada linha, tem-se o valor -1 que não é parte do conjunto e apenas indica a finalização de um caso de teste.

Especificação da saída

Para cada caso de teste da entrada, apresente uma linha, na saída padrão, contendo $ha[n]$ para n igual ao maior valor do conjunto.

Exemplo de entrada

```
2
5 2 1 0 2 1 1 4 5 2 -1
0 10 5 9 8 8 5 -1
```

Exemplo de saída

```
10
7
```

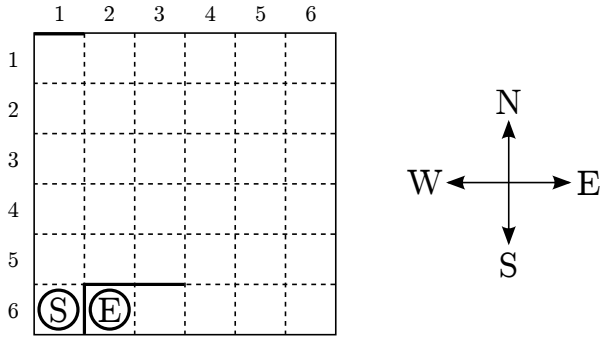
4 Problema D: Labirinto básico de paredes

`Arquivo: labirinto.[c|cpp|java]`

Neste problema você tem que resolver um labirinto muito simples, que consiste em:

1. Uma grade 6 por 6 quadrados unitários;
2. 3 paredes de comprimento entre 1 e 6 colocadas tanto na horizontal como na vertical para separar quadrados;
3. Uma marca de início e uma marca de fim

O labirinto tem este aspecto:



Você tem que encontrar o caminho mais curto entre a marca de início Ⓢ e a marca de fim Ⓔ. Só são permitidos movimentos entre quadrados adjacentes. Por adjacentes entendem-se os quadrados que compartilham uma aresta, e que não são separados por uma parede. Não é permitido fazer caminho fora da grade.

Especificação da entrada

A entrada consiste em vários casos de teste. Cada caso de teste consiste em cinco linhas: A primeira linha contém os números da coluna e da linha do quadrado com a marca de início. A segunda linha contém a os números da coluna e da linha do quadrado com a marca de fim. A terceira, quarta e quinta linhas especificam as localizações de três paredes. A especificação da parede pode ser dada pela posição da extrema esquerda, seguida pela posição da extrema direita (no caso de paredes horizontais) ou, pela posição do extremo superior seguida da posição do extremo inferior (no caso de paredes verticais). A posição de um ponto extremo de uma parede é dada pela distância medida a partir do lado esquerdo da grade, seguida da distância medida a partir do lado superior da grade.

Você pode assumir que as três paredes não intersectam umas as outras, mas elas podem se tocar em algum dos cantos da grade. E mais, sempre haverá um caminho válido da marca de início até a marca de fim. Note que o exemplo de entrada especifica o labirinto da figura acima.

O último caso de teste é seguido por uma linha com dois zeros.

Especificação da saída

Para cada caso de teste, imprima a descrição do caminho mais curto da marca de início até a marca de fim. A descrição deve especificar a direção de cada movimento ('N' para mover para cima, 'E' para direita, 'S' para baixo e 'W' para a esquerda).

Pode existir mais de um caminho mínimo, neste caso você pode imprimir qualquer um deles.

Exemplo de entrada

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

Exemplo de saída

```
NEEESWW
```

5 Problema E: Floresta

`Arquivo: floresta.[c|cpp|java]`

Bruce Force está em pé numa floresta. Ele fica imaginando qual o tronco de árvore mais distante, que não é bloqueado da sua vista pelos troncos de outras árvores. Bruce fez um mapa da floresta. O mapa mostra sua posição atual como sendo a origem do sistema de coordenadas cartesianas. A árvore i é mostrada no mapa como um círculo com centro (x_i, y_i) e raio r_i. Você pode assumir que o tronco de uma árvore é visível, se e somente se existe no mapa, um segmento de reta da origem $(0, 0)$ até um ponto na borda do círculo que representa o tronco de árvore, onde o segmento de reta não intersecta ou toca nenhum outro círculo.

Especificação da entrada

A entrada contém vários casos de teste. A primeira linha de cada caso de teste contém um número n ($1 \leq n \leq 1000$) onde n especifica quantas árvores existem no mapa. As linhas seguintes contêm em cada uma 3 inteiros x_i, y_i, r_i ($-10000 \leq x_i, y_i \leq 10000$, $1 \leq r_i \leq 1000$) onde (x_i, y_i) representa o centro do tronco de árvore i, e r_i é o raio do círculo. Você pode assumir que nenhum par de círculos das entradas se intersectam, isto é, para qualquer dois círculos, a distância entre seus centros é maior que a soma dos seus raios. E ainda, você pode assumir que nenhum círculo contém a origem.

O último caso de teste é seguido por uma linha contendo um zero.

Especificação da saída

Para cada caso de teste, imprima uma linha com a máxima distância euclidiana da origem até uma árvore visível. A distância até uma árvore deve ser medida usando o ponto da árvore mais próximo da origem, independente deste ponto ser ou não ser visível de fato.

Arredonde a resposta para 3 dígitos após o ponto decimal.

Exemplo de entrada

```
3
10 10 11
1 1 1
-20 -10 20
5
1 2 2
-2 1 1
2 -1 1
-1 -2 2
10000 -10000 1000
0
```

Exemplo de saída

```
3.142
1.236
```

Dica: No segundo caso de teste, as primeiras quatro árvores bloqueiam a vista de todas as árvores mais distantes do que estas quatro árvores.

6 Problema F: Raiz quadrada

`Arquivo: raiz.[c|cpp|java]`

Um estudante da área de tecnologia resolveu tornar seu código independente de algumas bibliotecas como, por exemplo, a matemática. No momento, ele está usando o Método de Newton para encontrar uma aproximação da raiz quadrada de um número a:

$$\begin{aligned} x_0 &= \frac{a}{2} \\ x_{i+1} &= \frac{1}{2}\left(x_i + \frac{a}{x_i}\right) \end{aligned}$$

para $i = 0, 1, 2, \ldots$

Especificação da entrada

A primeira linha da entrada contém um inteiro n. A linha seguinte contém n números reais.

Especificação da saída

Para cada número real, escreva o valor mínimo de i para que sua raiz quadrada x_{i+1} seja tal que $|x_{i+1} - x_i| \leq 0.001$. As saídas são separadas por um espaço em branco.

Exemplo de entrada

```
3
10.73 789.2132 5.487
```

Exemplo de saída

```
4 8 3
```

7 Problema G: Volte!

`Arquivo: volte.[c|cpp|java]`

Em algum lugar no meio de um deserto, vive uma pequena, completamente esquecida tribo N'Gubara. Todos os homens, mulheres e crianças da tribo vivem com alguns camelos no oásis N'Gubara (as pessoas de N'Gubara não têm grande criatividade quando se trata de nomes geográficos). Estas pobres pessoas só têm o bem em N'Gubara, alguns hectares de terra irrigada e a caverna N'Gubara. Sim, a caverna.

Você não se lembra exatamente como você veio parar em N'Gubara. Talvez o seu carro tenha quebrado quando atravessava o deserto. Talvez você tenha saltado de um avião. Talvez você tenha sido trazido aqui por marcianos. Mas isso não importa agora. Você sabe que está aqui, bem longe de qualquer civilização. E você quer desesperadamente voltar para casa.

A única possibilidade de chegar em casa é caminhando pelo deserto em direção à Chuville, a cidade mais próxima. Você tem que usar caminhos no deserto mostrados em seu mapa. Cada caminho conecta dois pontos de descanso. O oásis N'Gubara e Chuville também são considerados pontos de descanso. Você pode andar, mas você precisa de água. Para cada milha percorrida, você tem que beber uma unidade de água. Você é capaz de transportar, no máximo, C unidades de água por vez. Assim, você nunca pode andar mais de C milhas sem reabastecer seu suprimento de água. O caminho mais curto para Chuville é provavelmente muito maior que C milhas. Assim, parece que você vai ficar em N'Gubara para sempre, mas há um truque: no final de cada caminho, há uma ponto de descanso com um reservatório de água vazio. Você pode transportar água em um reservatório de N'Gubara ou de outros reservatórios onde guardou um pouco de água antes. Você pode então usar a água armazenada mais tarde (a água não evapora). Naturalmente, você só pode levar a água que estiver em qualquer reservatório. Você pode usar tanta água do oásis N'Gubara quanto for necessário mas, já que a água é muito valiosa no deserto, você prometeu que irá utilizar apenas a quantidade mínima necessária para o seu regresso a Chuville. Nessa tarefa, nós queremos que você calcule a quantidade mínima de água que você precisa.

Especificação da entrada

A primeira linha contém três inteiros N, M e C, onde N é o número do pontos de descanso, M é o número de caminhos e C é a sua capacidade de carga. M linhas seguem, cada uma descrevendo um caminho no mapa. Cada linha contém três números x, y e l, onde x e y são os pontos de descanso ligados por um caminho e l é o comprimento desse caminho em milhas. Os pontos de descanso são numerados de 1 a N, onde ponto de descanso 1 é o oásis N'Gubara e ponto de descanso N é Chuville.

Observação: Você pode assumir que cada ciclo no mapa passa pelo ponto de descanso N (ou seja, Chuville). Um ciclo é uma sequência de pontos de descanso distintos $r_1, r_2, \ldots, r_k$ ($k > 2$) tal que existe um caminho de r_1 para r_2, de r_2 para r_3, ..., de r_k para r_1.

Especificação da saída

Sua saída deve conter um único inteiro – a quantidade mínima de água necessária para que você volte de N'Gubara para Chuville. Se não houver possibilidade de chegar a Chuville com sua atual capacidade de carga C, você deve gerar como saída o número -1.

Exemplo de entrada

```
9 10 25
1 2 3
2 3 12
3 4 4
3 5 9
4 9 13
5 9 5
2 6 10
6 7 10
7 8 10
8 9 10
```

Exemplo de saída

```
65
```

Observação do exemplo: Você pode voltar a Chuville da seguinte forma: primeiro pegue 25 *unidades de água de N'Gubara, vá para o ponto de descanso* 2*, saia de lá* 19 *unidades no reservatório e volte para N'Gubara. Então você repete esta viagem, trazendo mais* 19 *unidades para o ponto de descanso* 2*. Por fim, você toma* 15 *unidades de N'Gubara para o ponto de descanso* 2*. Agora você tem* $19 + 19 + 12 = 50$ *unidades de água aqui. Você retorna ao ponto* 3 *e na volta, deixa ali* 1 *unidade de água. Agora você pega toda a água que deixou a partir do ponto* 2*, vai ao ponto* 3*, leva uma unidade a partir do ponto* 3 *(agora você tem* $25-12+1 = 14$ *unidades) e vai a Chuville através do ponto de descanso* 5*.*

8 Problema H: Wine trading in Gergovia

`Arquivo: gergovia.[c|cpp|java]`

As you may know from the comic "Asterix and the Chieftain's Shield", Gergovia consists of one street, and every inhabitant of the city is a wine salesman. You wonder how this economy works? Simple enough: everyone buys wine from other inhabitants of the city. Every day each inhabitant decides how much wine he wants to buy or sell. Interestingly, demand and supply is always the same, so that each inhabitant gets what he wants.

There is one problem, however: Transporting wine from one house to another results in work. Since all wines are equally good, the inhabitants of Gergovia don't care which persons they are doing trade with, they are only interested in selling or buying a specific amount of wine. They are clever enough to figure out a way of trading so that the overall amount of work needed for transports is minimized.

In this problem you are asked to reconstruct the trading during one day in Gergovia. For simplicity we will assume that the houses are built along a straight line with equal distance between adjacent houses. Transporting one bottle of wine from one house to an adjacent house results in one unit of work.

Input specification

The input consists of several test cases. Each test case starts with the number of inhabitants n ($2 \leq n \leq 100000$). The following line contains n integers a_i ($-1000 \leq a_i \leq 1000$). If $a_i \geq 0$, it means that the inhabitant living in the i^{th} house wants to buy a_i bottles of wine, otherwise if $a_i < 0$, he wants to sell $-a_i$ bottles of wine. You may assume that the numbers a_i sum up to 0. The last test case is followed by a line containing 0.

Output specification

For each test case print the minimum amount of work units needed so that every inhabitant has his demand fulfilled. You may assume that this number fits into a signed 64-bit integer (in C/C++ you can use the data type "long long", in JAVA the data type "long").

Input example

```
5
5 -4 1 -3 1
6
-1000 -1000 -1000 1000 1000 1000
0
```

Output example

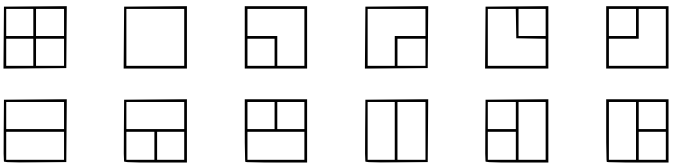
```
9
9000
```

9 Problema I: Delicious Cake

`Arquivo: cake.[c|cpp|java]`

Lenka likes to bake cakes since her childhood, when she has learned to bake from her mom. She soon became a cake expert able to bake chocolate cakes, apple pies, muffins, cookies, cheese cakes, tortes and many other cakes. Recently, she has started her studies of math at Comenius University in Bratislava. In the first year she is taking combinatorics class. Today she is studying for the final exam. Since the brain needs a lot of sugar to study math, she has baked, just for herself, her favorite, very delicious, strawberry cake. The cake, still hot, is lying on an $N \times M$ inch sheet pan. Hungrily waiting for the cake to cool off Lenka came up with an interesting combinatorial question: How many different possibilities to cut the cake are there so that every connected piece consists of some number of 1×1 inch unit squares?

The cake can be viewed as a grid consisting of $N \times M$ unit squares. We are allowed to cut the cake along the grid lines. As a result the cake splits into several connected pieces. (Two unit squares remain connected if they share a side which was not cut.) How many different ways are there to cut the cake? We consider two cuttings of the cake to be the same if the resulting connected pieces of both cuttings have the same shape and are at the same positions within the cake. In other words, we are only counting those cuttings where no cut leads between two unit squares that are in the same connected piece. The following picture ilustrates all the 12 different possible ways how to cut a 2×2 inch cake:



Note that cutting, for example, as on following picture  is the same as not cutting at all.

Input specification

The first line of the input file contains an integer T specifying the number of test cases. Each test case is preceded by a blank line. Each test case consists of a single line with two positive integers N and M – dimensions of the cake.

Output specification

For each test case output a line with a single positive integer – the number of different possibilities how to cut the cake.

Input example

```
2

1 2

2 2
```

Output example

```
2
12
```