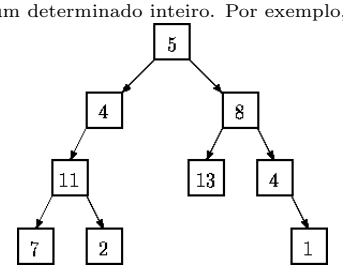
## Árvores-UVA:112

LISP é uma das primeiras linguagens de programação de alto nível e junto com FORTRAN é uma das mais antigas línguagens ainda em uso. Listas, que são as estruturas de dados fundamentais em LISP podem ser adaptadas para representar outras estruturas importantes como a árvore.

Este problema lida com a determinação se árvores binárias representadas por expressões LISP possuem uma certa propriedade.

Dada uma árvore binária de inteiros você deve escrever um programa para determinar se existe um caminho de raiz até uma árvore que totalize um determinado inteiro. Por exemplo, na árvore




há 4 caminhos, cujas somas são 27, 22, 26 e 18.

A árvore do desenho é representada pela expressão

```
(5 (4 (11 (7 () ()) (2 () ()) ) ()) (8 (13 () ()) (4 () (1 () ()))))
```

Note que nesta formulação todas as folhas da árvore tem a forma `(inteiro () () )`. Já que uma árvore vazia não tem caminhos raiz-até-folha qualquer pergunta sobre se um caminho existe com uma determinada soma deve ser – neste caso – respondida negativamente.

**Entrada**

Uma sequência de casos na forma de pares inteiro/árvore. Cada caso consiste de um inteiro seguido por uma árvore binária formatada através de uma expressão como descrito acima. Todas as expressões são válidas, mas as expressões podem se espalhar em várias linhas e conter um ou mais espaços. Haverá um ou mais de um caso na entrada que se terminará pelo EOF.

**Saída** Para cada caso, o programa deve escrever uma saída informando "yes" caso haja o caminho somando o valor solicitado ou "no" caso não haja.

```
Exemplo
22 (5(4(11(7()())(2()()))()) (8(13()())(4()(1()()))))     yes
20 (5(4(11(7()())(2()()))()) (8(13()())(4()(1()()))))     no
10 (3                                                     yes
     (2 (4 () () )
        (8 () () ) )
     (1 (6 () () )
        (4 () () ) ) )
5 ()                                                      no
```

## O poder da criptografia-UVA:113

O trabalho corrente em criptografia envolve (entre outras coisas) números primos muito grandes e a computação de potências de módulos destes primos. O trabalho nesta área resultou que pode-se usar aspectos da teoria dos números e outras questões da matemática que em outras eras tinham interesse apenas teórico.

Este problema envolve a computação eficiente de raizes inteiras de números. Dado um inteiro $n \geq 1$ e um inteiro $p \geq 1$ você deve escrever um programa que determine $\sqrt[n]{p}$ ou a n-ésima raiz de p. Neste problema, dados $n$ e $p$, $p$ sempre terá a forma $p = k^n$ para um inteiro $k$. É este o inteiro que o programa deve achar.

**Entrada** A entrada consiste de uma sequencia de pares de inteiros, n e p, com cada inteiro em uma linha. O valor de $n$ varia em $1 \leq n \leq 200$ e $p$ como $1 \leq p \leq 10^{101}$. Já $k$ está em $1 \leq k \leq 10^9$.

**Saída** Para cada par de inteiros $n$ e $p$, deve-se calcular $k$ tal que $k^n = p$. O valor $k$ deve ser impresso.

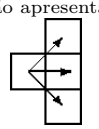
```
Exemplo
2
16                          4
3
27                          3
7
435718618402138220454 4     1234
```

## TSP Unidirecional-UVA:116

Problemas que requerem o cálculo do caminho mínimo aparecem em diferentes áreas da ciência da computação. Por exemplo, em problemas de projeto de VLSI deve-se minimizar o comprimento dos fios. O problema do vendedor viajante (Traveling Salesman Problem=TSP, em inglês) é achar se todas as cidades da rota do vendedor podem ser visitadas exatamente uma vez em um dado limite (tempo, disância, custo...). Este problema é um dos exemplos canônicos de problemas NP-completos. Suas soluções requerem um tempo enorme para serem geradas, mas são simples de verificar.

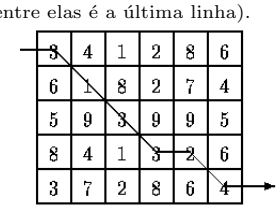
Este problema aqui lida com a busca do caminho mínimo através de uma grade de pontos que deve ser percorrida da esquerda para a direita.

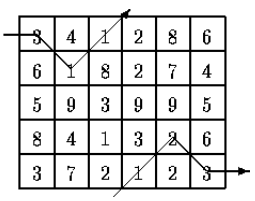
Dada uma matriz $m \times n$ de inteiros você deve escrever um programa que calcule o custo do caminho mínimo. Um caminho sempre começa na coluna 1 (a primeira coluna) e consiste de uma sequência de etapas terminando na coluna $n$ (a última coluna). Uma etapa consiste em passar da coluna $i$ para a coluna $i + 1$ em uma linha adjacente (na horizontal ou na diagonal). A primeira e a última linhas (linhas 1 e $m$) são consideradas adjacentes, isto é a matriz "se enrola" como se ela fosse um cilindro horizontal. Etapas válidas são apresentadas no desenho:



O custo de um caminho é a soma dos inteiros em cada uma das células da matriz que são visitadas.

Por exemplo, duas matrizes $5 \times 6$ são mostradas abaixo (a única diferença entre elas é a última linha).




O caminho mínimo é ilustrado para cada matriz. Note que o caminho da matriz da direita usa a propriedade da adjacência entre a primeira e a última linha.

**Entrada**

Uma sequência de especificações de matrizes. Cada especificação traz o número de linhas e de colinas seguidas pelos inteiros que compõe a matriz. Note que os inteiros podem ser negativos.

Para cada especificação, o número de linhas deve estar entre 1 e 10 inclusive e o número de colunas entre 1 e 100 inclusive. O custo do caminho não excederá inteiros representados usando 30 bits.

**Saída**

Duas linhas: a primeira o caminho mínimo, através da indicação das linhas que o compõe. A segunda é o custo do caminho mínimo.

Se mais do que um caminho tem o mesmo custo, o caminho escolhido deve ser o lexicograficamente menor.

```
5 6
3 4 1 2 8 6
6 1 8 2 7 4
5 9 3 9 9 5
8 4 1 3 2 6                 1 2 3 4 4 5
3 7 2 8 6 4                 16
5 6
3 4 1 2 8 6
6 1 8 2 7 4
5 9 3 9 9 5                 1 2 1 5 4 5
8 4 1 3 2 6                 11
3 7 2 1 2 3
2 2                         1 1
9 10 9 10                   19
```

## ☞ Para você fazer

**Árvore parentizada**

```
40 (16(12(5(3()())(9()(10()(11()()))))(14()(15()()))))(17()()))
```

**Criptografia**

```
6       4608273662721
```

**TSP Unidirecional**

```
13   10   14   19   16    7   19   19
13   18   13    2   12   16   14   19
13    6    5   18    4   14   19   18
13   10   16   18   17    1    5    8
17    4   17    5   14   12   12    4
12    8   19    1   19   15    7   17
```

| árvore | valor de $k$ | caminho e custo |
|---|---|---|
| | | |

1