O **Longest Increase Subsequence (LIS)** problema é encontrar o maior subseqüência crescente de uma determinada seqüência.

Mais longa subseqüência comum

Uma maneira simples de encontrar o maior subseqüência crescente é a utilização do Mais longa subseqüência comum (LCS)(Programação Dinâmica) Algoritmo.

1. Faça uma classificado cópia da seqüência *A*, Denominado *B*. $O(n\log(n))$ tempo.
2. Utilização Mais longa subseqüência comum em com *A* e *B*. $O(n^2)$ tempo.

Programação Dinâmica

Há um solução direta utilizando Programação Dinâmica em $O(n^2)$ tempo. Embora este seja assintoticamente equivalente ao Mais longa subseqüência comum versão da solução, a constante é menor, porque há menos sobrecarga. Existe uma solução ainda melhor O (n lg n) baseado na solução $O(n^2)$.

```
// Returns the length of the longest increasing
subsequence.
// Note that this is looking for the longest
strictly increasing subsequence.

//    This can be easily modified for other
situations.
int lcs( int* a, int N ) {

   int *best, *prev, i, j, max = 0;
   best = (int*) malloc ( sizeof( int ) * N );

   prev = (int*) malloc ( sizeof( int ) * N );


   for ( i = 0; i < N; i++ ) best[i] = 1, prev[i] =
i;


   for ( i = 1; i < N; i++ )

      for ( j = 0; j < i; j++ )

         if ( a[i] > a[j] && best[i] < best[j] + 1 )
```

```
            best[i] = best[j] + 1, prev[i] = j;   //
prev[] is for backtracking the subsequence


    for ( i = 0; i < N; i++ )

        if ( max < best[i] )

            max = best[i];

    free( best );

    free( prev );

    return max;
}


// Sample usage.
int main(){

  int b[] = { 1, 3, 2, 4, 3, 5, 4, 6 };

  // the longest increasing subsequence = 13456?
  // the length would be 5, as well lcs(b,8) will
return.

  printf("%d\n", lcs( b, 8 ) );

}
```

Pirâmide

Joana quer ser artista plástica, mas enquanto estuda procura trabalhos temporários durante suas férias escolares. Joana conseguiu emprego como auxiliar de almoxarifado em uma grande transportadora. A transportadora recebeu um enorme carregamento de caixas de mesma altura mas com largura e profundidades diferentes. As caixas podem ser empilhadas indefinidamente mas não podem ser deitadas em outra posição (todas têm que ser armazenadas obedecendo à indicação "Este lado para cima"). Joana é a responsável por armazenar o carregamento de caixas, e, seguindo seu senso artístico,

quer construir com as caixas a pilha mais alta possível na forma de uma "pirâmide", ou seja, uma pilha construída de tal forma que uma caixa A é empilhada sobre uma outra caixa B somente se as dimensões de A (largura e a profundidade) não são maiores do que as dimensões de B (as caixas podem ser viradas de forma a trocar a profundidade com a largura). Você pode ajudá-la?

Tarefa

É dado um conjuto de caixas de mesma altura mas com largura e profundidades diferentes. Sua tarefa é escrever um programa que determine qual a pilha de caixas mais alta que Joana pode construir com as restrições acima.

Entrada

A entrada é composta de vários conjuntos de teste. A primeira linha de um conjunto de testes contém um número inteiro N que indica a quantidade de caixas do conjunto. As N linhas seguintes contêm, cada uma, a descrição de uma caixa. Uma caixa é descrita por dois inteiros X e Y ($1 \leq X \leq 15000$ e $1 \leq Y \leq 15000$) que representam os valores de cada lado da peça. O final da entrada é indicado por N = 0.

Exemplo de Entrada

```
3
100 100
1000 2000
2000 500
6
3 4
5 7
7 5
1 5
4 4
10 2
0
```

Saída

Para cada conjunto de teste da entrada seu programa deve produzir três linhas na saída. A primeira linha deve conter um identificador do conjunto de teste, no formato "Teste n", onde n é numerado a partir de 1. A segunda linha deve conter o número máximo de caixas que podem ser empilhadas na forma de uma pirâmide, conforme determinado pelo seu programa. A terceira linha deve ser deixada em branco. A grafia mostrada no Exemplo de Saída, abaixo, deve ser seguida rigorosamente.

Exemplo de Saída

```
Teste 1
3

Teste 2
4
```

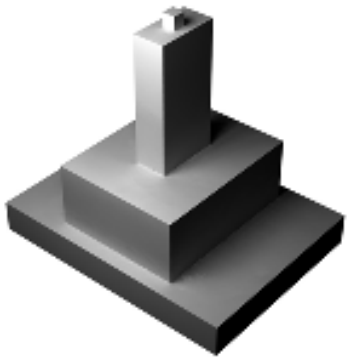
(esta saída corresponde ao exemplo de entrada acima)

Restrições

$0 \leq N \leq 5000$ (N = 0 apenas para indicar o final da entrada)
$1 \leq X \leq 15000$
$1 \leq Y \leq 15000$

SPOJ Problem Set (classical)

100. Tower of Babylon



Problem code: BABTWR

Apart from the Hanging Gardens the Babylonians (around 3000-539 b.c.) built the Tower of Babylon as well. The tower was meant to reach the sky, but the project failed because of a confusion of language imposed from much higher above.

For the 2638th anniversary a model of the tower will be rebuilt. *n* different types of blocks are available. Each one of them may be duplicated as many times as you like. Each type has a height *y*, a width *x* and a depth *z*. The blocks are to be stacked one upon eachother so that the resulting tower is as high as possible. Of course the blocks can be rotated as desired before stacking. However for reasons of stability a block can only be stacked upon another if *both* of its baselines are shorter.

Input

The number of types of blocks *n* is located in the first line of each test case. On the subsequent *n* lines the height y_i, the width x_i and the depth z_i of each type of blocks are given. There are never more than 30 different types available.

There are many test cases, which come one by one. Input terminates with n = 0.

Output

For each test case your program should output one line with the height of the highest possible tower.

Example

Sample input:

```
5
31 41 59
26 53 58
97 93 23
84 62 64
33 83 27
1
1 1 1
0
```

Sample output:

```
342
1
```

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