24. Torres de colores.

(Examen febrero 2017, sesión 2, ejercicio 3) Laura quiere construir una torre con piezas de colores. En su juego de construcciones hay piezas azules, rojas y verdes, de cada una de las cuales tiene un determinado número disponible, respectivamente a, r y v. Quiere construir una torre que contenga $n >= 2$ piezas en total cada una encima de la anterior. No le gusta el color verde, asi que nunca coloca dos piezas verdes juntas, ni permite que el número de piezas verdes supere al de piezas azules en ningún momento mientras se va construyendo la torre. Además, como el color rojo es su favorito, las torres que construye siempre tienen en la parte inferior una pieza roja, y en la torre final el número de piezas rojas debe ser mayor que la suma de las piezas azules y verdes.

Implementar un algoritmo que muestre todas las formas posibles que tiene de construir una torre de la altura deseada cumpliendo con las restricciones mencionadas.

Entrada

La entrada que espera el corrector automático consta de una serie de casos de prueba y acabará cuando se introduzca una línea con cuatro ceros. Cada caso de prueba se escribe en una línea y consta de 4 enteros separados por blancos. El primero es mayor que uno y representa la altura de la torre. Los tres siguientes son mayores o iguales que cero y representan los cubos de color azul, rojo y verde respectivamente.

Salida

Para cada caso de prueba se escriben todas las posibles torres, una en cada línea ordenadas por orden lexicográfico y separando cada par de colores por un espacio. Cada caso termina con una línea en blanco. Si no se puede construir la torre con los números de bloques dados se escribirá *SIN SOLUCION*.

Entrada de ejemplo

```
4 4 4 4
5 2 2 2
5 3 3 1
2 1 2 1
0 0 0 0
```

Salida de ejemplo

```
rojo azul rojo rojo
rojo rojo azul rojo
rojo rojo rojo azul
rojo rojo rojo rojo

SIN SOLUCION

rojo azul azul rojo rojo
rojo azul rojo azul rojo
rojo azul rojo rojo azul
rojo azul rojo rojo verde
rojo azul rojo verde rojo
rojo azul verde rojo rojo
rojo rojo azul azul rojo
rojo rojo azul rojo azul
rojo rojo azul rojo verde
rojo rojo azul verde rojo
rojo rojo rojo azul azul
rojo rojo rojo azul verde

rojo rojo
```

25. Luces de Navidad

Mi tio se dedica a la fabricación de luces de Navidad. Cuando era pequeño me llevaba con él a la fábrica, donde se podían ver rollos y rollos de luces de colores, para adornar los árboles y las fachadas de las casas. Durante muchos años ha detectado que unas tiras se venden mas que otras. Para analizar el motivo, ha realizado un estudio de mercado con él que han detectado que a la gente no le importa realmente el color concreto de cada bombilla, sino que el aspecto total tenga bastante colorido. Por ello ha decidido hacer las tiras de luces de forma que no haya mas de dos luces seguidas del mismo color y que se cumpla que en cualquier punto de la tira la suma de las luces de un color no supere en mas de una unidad la suma de las luces de todos los demás colores. Además se debe tener en cuenta que las tiras de luces no deben consumir mas de una cierta cantidad de energía para cumplir con la legislación sobre el medio ambiente.

Ahora quiere saber cuantas posibles tiras puede hacer de una determinada longitud dado un número máximo de bombillas de cada tipo y un consuma máximo para la tira. Para poder seleccionar las tiras adecuadas cuenta con el consumo de cada tipo de bombilla.

Requisitos de implementación.

El problema se debe implementar empleando la técnica de vuelta atrás.

Para facilitar la implementación del programa se ha añadido al final del enunciado la salida del caso de ejemplo mostrando las posibles tiras para cada caso.

Entrada

La entrada consta de una serie de casos de prueba. Cada caso de prueba consta de 2 líneas. En la primera se indica la longitud de la línea de luces a fabricar; el número de colores diferentes que se van a utilizar, y el consumo máximo soportado por la tira. En la siguiente se indica el consumo de cada tipo de bombilla.

La longitud y el número de colores son números enteros mayores que uno. El consumo máximo es un entero mayor que cero. El consumo de cada bombilla es un valor entero mayor que cero.

Salida

Para cada caso de prueba se escribe en una línea el número de combinaciones posibles para los valores de entrada.

Entrada de ejemplo

```
4 2 6
2 1
4 2 5
2 1
5 3 7
2 1 3
5 3 8
2 1 3
5 3 17
5 4 3
6 4 27
6 5 4 5
```

Salida de ejemplo

```
4
0
4
16
4
84
```

A continuación se muestran las combinaciones de cada caso de prueba del ejemplo para ayudar a la depuración. El primer valor es el tipo de la primera bombilla de la tira, el segundo valor es el tipo de la segunda bombilla de la tira, el tercero es el tipo de la tercera bombilla etc.

Salida de ejemplo

```
0 1 0 1
0 1 1 0
1 0 0 1
1 0 1 0

SIN SOLUCION

0 1 0 1 1
0 1 1 0 1
1 0 0 1 1
1 0 1 0 1

0 1 0 1 0
0 1 0 1 1
0 1 1 0 0
0 1 1 0 1
0 1 1 2 1
0 1 2 1 1
1 0 0 1 0
1 0 0 1 1
1 0 1 0 0
1 0 1 0 1
1 0 1 2 1
1 0 2 1 1
1 2 0 1 1
1 2 1 0 1
2 1 0 1 1
2 1 1 0 1

1 2 1 2 2
1 2 2 1 2
2 1 1 2 2
2 1 2 1 2
```

Salida de ejemplo

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

Autor: Isabel Pita.

26. Papa Noel reparte juguetes

Cada año son más los niños que le piden regalos a Papa Noel. Esta noche debe repartir los regalos y todavía no tiene preparado lo que le dará a cada niño. Para poder llegar a tiempo los elfos han diseñado un programa informático, que asignará a cada niño 2 juguetes entre todos los disponibles. Como quieren que los niños queden contentos han elaborado una lista con la satisfacción que le produce a cada niño cada uno de los juguetes que tienen en la fábrica. El objetivo es que todos los niños puedan llegar a un cierto grado de satisfacción, aunque no reciban los juguetes que más les gustan. Papa Noel también ha pedido que los juguetes sean variados, para que ningún niño reciba dos cosas demasiado parecidas, para ello los elfos han clasificado los juguetes por tipos y no entregarán a un niño dos juguetes del mismo tipo.

El jefe elfo de informática ha puesto a su equipo a trabajar en un programa que obtenga todas las posibles asignaciones de juguetes que cumplen los requisitos pedidos por Papa Noel. Para evitar las asignaciones repetidas (por ejemplo dar a un niño el juguete A y el juguete B es la misma asignación que darle el juguete B y el juguete A) los dos juguetes de cada niño se obtendrán ordenados, el primer juguete será el de menor identificador y el segundo el de mayor identificador.

Requisitos de implementación.

El problema se debe implementar empleando la técnica de vuelta atrás.

En la plantilla propuesta, la solución se almacena en un vector donde las componentes pares tienen el primer juguete asignado a un niño y las componentes impares el segundo juguete. En las etapas pares se asigna el primer juguete de un niño y en las etapas impares el segundo juguete.

Entrada

La entrada consta de una serie de casos de prueba. Cada caso de prueba consta de $n+3$ líneas. En la primera se indica el número juguetes diferentes que se fabrican, m, el número de niños a los que se les reparten juguetes, n, y la mínima satisfacción pedida para los niños. En la siguiente línea se indica el número de unidades disponibles de cada juguete. En la tercera línea se indica el tipo que tiene cada juguete y en las n últimas líneas la satisfacción de cada uno de los n niños con cada uno de los m juguetes. Después de cada caso hay una línea en blanco para facilitar la identificación de los casos en el ejemplo.

El número de juguetes diferentes, el número de niños y la satisfaccón mínima son enteros mayores que cero. El número de unidades disponibles es un entero mayor o igual que cero. El tipo de cada juguete es una cadena de caracteres, y la satisfacción es un número entero que puede ser negativo si el niño aborrece el juguete.

Salida

Para cada caso de prueba se escriben todas las posibles asignaciones que se pueden realizar, una en cada línea. Si no existe ninguna asignación posible se escribirá SIN SOLUCION. Después de cada caso se escribe una línea en blanco.

Entrada de ejemplo

```
4 3 10
2 1 1 2
Tipo1 Tipo1 Tipo2 Tipo3
8 9 3 1
6 4 5 3
2 2 9 9

4 3 8
2 1 1 2
Tipo1 Tipo1 Tipo2 Tipo3
8 9 3 1
6 4 5 3
2 2 9 9

4 2 10
1 1 1 1
Tipo1 Tipo1 Tipo1 Tipo2
10 12 12 15
9 10 20 7

3 5 10
4 3 5
Tipo1 Tipo1 Tipo2
5 7 4
8 6 5
4 8 2
9 3 4
8 5 5
```

Salida de ejemplo

```
1 3 0 2 0 3

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

SIN SOLUCION

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

Autor: Isabel Pita

Problema número XXXXXX

Los funcionarios del Ministerio

Tiempo máximo: X s Memoria máxima: X KiB

http://www.aceptaelreto.com/problem/statement.php?id=XXXXX

El Ministro de Desinformación y Decencia se ha propuesto hacer trabajar en firme a sus funcionarios, para lo que se ha sacado de la manga una serie de trabajos (tantos como funcionarios). A pesar de su ineficacia, todos los funcionarios son capaces de hacer cualquier trabajo, aunque unos tardan más que otros. En el Ministerio todos se conocen bien, por lo que se sabe cuánto tardará cada funcionario en realizar cada uno de los trabajos. Para justificar su puesto, Su Excelencia el Sr. Ministro desea conocer la asignación óptima de trabajos a funcionarios de modo que la suma total de tiempos sea *mínima*.

Entrada

La entrada consta de una serie de casos de prueba. Cada uno comienza con una línea con el número N de funcionarios y trabajos ($1 \leq N \leq 20$). A continuación aparecerán N líneas, una por funcionario, con N números (entre 1 de 10.000) que indican lo que tarda ese funcionario en realizar cada uno de los N trabajos.

La entrada terminará con un caso sin funcionarios, que no debe procesarse.

Salida

Para cada caso de prueba se escribirá una línea con la suma total de tiempos que tardarán los funcionarios en realizar los trabajos asignados según la asignación óptima.

Entrada de ejemplo

```
3
10 20 30
40 20 10
60 10 20
3
10 15 20
30 40 50
60 80 99
0
```

Salida de ejemplo

```
30
120
```

Autor: Alberto Verdejo.

Revisores: Pedro Pablo Gómez Martín y Marco Antonio Gómez Martín.

Compra de la semana

Alonso Rodríguez tiene que hacer la compra de la semana. Ha hecho una lista de n productos que quiere comprar. En su barrio hay m supermercados en cada uno de los cuales se dispone de todos esos productos. Pero como es un comprador compulsivo no quiere comprar más de tres productos en cada uno de los supermercados ya que así pasa más tiempo comprando (se puede suponer que $n \leq 3m$). Dispone de una lista de precios (en céntimos) de los productos en cada uno de los supermercados. Se pide diseñar un algoritmo que permita a Alonso decidir cómo hacer la compra de forma que compre todo lo que necesita y que el coste total sea mínimo.

Entrada

La entrada comienza por una línea indicando el número de casos de prueba que deberán procesarse. Para cada caso de prueba la primera línea tiene dos números, el primero es el número de supermercados y el segundo el número de productos. Se garantiza que $0 \leq$ número de productos $\leq 3 *$ número de supermercados y que número de supermercados ≤ 20. A continuación aparecen tantas líneas como supermercados y en cada una de las líneas el precio de todos los productos en ese supermercado. En todos los supermercados se ofrecen todos los productos.

Salida

Por cada caso de prueba aparecerá una línea independiente con el coste de la mejor solución encontrada o bien el mensaje "Sin solucion factible" en el caso de que no haya ninguna.

Entrada de ejemplo

```
2
6 10
1820  510  370  1000  460  324  505  640  2030  409
2000  430  450  1110  606  290  530  670  2104  501
1760  502  395  1200  550  199  525  702  1830  550
2130  640  560  1307  735  450  600  720  2150  575
1143  455  505  1140  500  400  350  550  2030  399
1200  475  403  1002  560  350  502  640  2009  460
4 1
4020
3560
5540
3540
```

Salida de ejemplo

```
6743
3540
```

29. Necesito otro abrigo

Cuando volvía del trabajo he visto un abrigo precioso en un escaparate. Realmente necesito un abrigo para este invierno y este era precioso. Sin embargo, mi compañera ha asegurado que tengo suficientes y que ni siquiera tengo necesidad de llevar dos días seguidos el mismo, impidiéndome entrar en la tienda. Lo que ella no tiene en cuenta es que no todos los abrigos se pueden llevar todos los días. Cuando llueve no puedo llevar el de ante, y si la lluvia es fuerte solo sirven las gabardinas.

Después de mucho discutir me ha prometido que si puedo demostrar que no tengo forma de combinar los abrigos sin repetir dos días seguidos alguno me acompañará a comprarlo. Me he puesto manos a la obra y he conseguido una estimación de las precipitaciones de cada día del invierno. Ahora me toca decidir qué abrigo puedo llevar cada día para no mojarme y no tener que repetir prenda dos días consecutivos.

Además tengo una serie de manías que no me permiten ponerme cualquier cosa con tal de no mojarme. Para empezar, no me gusta utilizar un abrigo muchos más días que otro porque se desgasta y se pone feo. Por ello, no consideraré aquellas combinaciones en las cuales el abrigo que más haya utilizado supere en dos días o más a un tercio de los días que van transcurridos, es decir, la combinación de abrigos será válida si:

$$\forall i : 1 <= i <= \textit{número de días de la temporada} \Rightarrow \textit{dias}(a, i) \leq 2 + i/3.$$

Siendo a el abrigo que más he usado hasta el día i-ésimo y *dias(a,i)* el número de días que he utilizado el abrigo a en el periodo $[1..i]$.

Además el abrigo que utilizo el último día del invierno debe ser diferente al que utilicé el primer día.

Requisitos de implementación.

El problema se debe resolver utilizando la técnica de vuelta atrás. Se valorará que se eviten las llamadas recursivas que no producirán solución válida.

Poner comentarios e indicar el coste de la función *esValida* que se haya implementado. El coste debe ser lo menor posible.

Entrada

La entrada consta de una serie de casos de prueba. Cada caso de prueba tiene 3 líneas. En la primera se muestra el número de días n sobre el que se hace el estudio, seguido del número de abrigos a que tengo. En la segunda línea se muestran n valores correspondientes a la estimación de la precipitación de cada día. En la tercera línea se muestran a valores que indican la cantidad de precipitación que puede soportar cada uno de los abrigos. La entrada termina con una línea con cero días y cero abrigos.

El número de días es un entero, 1 <= n <= 12 y el número de abrigos es un entero 1 <= a <= 5. La estimación de precipitación es un entero 0 <= p <= 300.

Salida

Para cada caso de prueba se muestra en un línea el número de combinaciones posibles que tengo para ponerme los abrigos. Si no existe ninguna combinación se escribirá *Lo puedes comprar*.

Entrada de ejemplo

```
4 2
50 100 50 100
50 100
5 3
200 100 30 50 100
300 50 100
3 2
100 180 50
200 100
10 2
40 30 100 100 100 100 80 120 100 200
300 300
0 0
```

Salida de ejemplo

```
1
2
Lo puedes comprar
Lo puedes comprar
```

Combinaciones posibles

```
0 1 0 1

0 2 0 1 2
0 2 1 0 2

Lo puedes comprar
Lo puedes comprar
```

31. Buscando actores para la película

Se está preparando el rodaje de una nueva película. El guión ya está listo y ha llegado el momento de seleccionar a los actores que interpretarán cada uno de los papeles. Se ha convocado a todos los actores disponibles a un *casting* y el director está asignando una puntuación a cada uno de ellos para cada papel. Por otra parte el productor ha elaborado una lista con lo que cobraría cada actor por participar en la película.

a) El director ha encargado que se obtenga el reparto de la película, de forma que se maximice la afinidad en su conjunto de los actores con los personajes, siempre dentro de un determinado presupuesto. La afinidad de cada actor con cada personaje se mide con la puntuación asignada en el casting y lo que se busca es maximizar la suma de las puntuaciones. Debe tenerse en cuenta también, que para simplificar la grabación, un actor sólo puede interpretar un papel y que no se admitirá un reparto con actores cuya puntuación en el papel asignado sea menor que una cierta constante.

b) Mejorar la implementación anterior cortando las llamadas recursivas que no producirán una puntuación mejor que la encontrada.

Exámen febrero 2017 (sesión 1)

Entrada

La entrada consta de una serie de casos de prueba. Cada caso de prueba consta de $n+2$ líneas. En la primera se indica el número de papeles de la obra, $n > 0$, seguido del número de actores, $k >= n$, del presupuesto total con el que se cuenta, p > 0 y de la puntuación mínima requerida para actuar. En la segunda, lo que cobra cada actor por participar en la película. En las n siguientes se indica para cada papel la puntuación otorgada por el director a cada uno de los actores. Si un actor no puede interpretar un papel su puntuación es cero. La entrada termina con una línea con tres ceros.

La cantidad que cobran cada actor es un número entero positivo. La puntuación de cada actor con cada papel es un entero mayor o igual que cero.

Salida

Para cada caso de prueba se escribe en primer lugar la puntuación total obtenida con el reparto elegido seguido del presupuesto necesario para este reparto. A continuación tantas líneas como papeles tenga la película. En cada línea se indica un papel y el actor seleccionado para interpretarlo, ambos datos separados por un caracter blanco. Los casos de prueba se escriben uno a continuación del otro.

Si no existe un reparto posible, se escribirá una única línea con puntuación cero y presupuesto cero.

Entrada de ejemplo

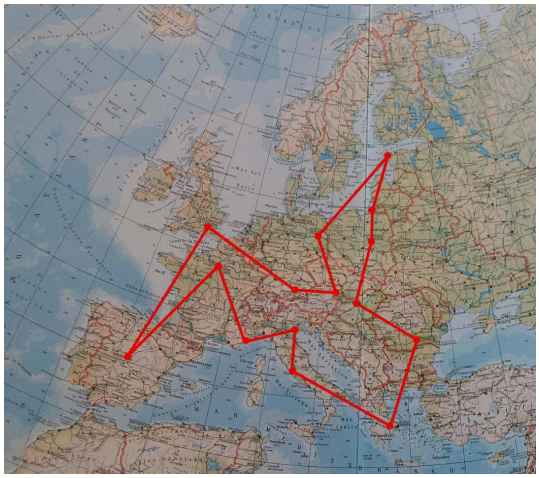
```
2 3 200 4
100 100 100
10 5 0
0 5 10
3 5 100 40
50 20 40 30 40
100 0 70 50 60
0 50 40 30 60
0 60 50 70 40
0 0 0 0
```

Salida de ejemplo

```
20 200
0 0
1 2
220 100
0 0
1 1
2 3
```

Autor: Isabel Pita

32. Planificando las vacaciones



Estas vacaciones quiero hacer un circuito por Europa visitando las principales ciudades. He seleccionado unas cuantas que todavía no conozco y tengo intención de visitar todas ellas. Como no me gusta viajar de noche, me desplazaré de una ciudad a otra durante la tarde, dormiré en un buen hotel de la ciudad y aprovecharé las mañanas para hacer turismo. Cada noche dormiré en una ciudad distinta, asi que sólo tengo un día para visitarla.

He preparado una ruta, que pasa por todas las cuidades sin repetir ninguna de ellas. Tengo el precio del billete más barato para ir de una ciudad a otra, así como el precio del hotel que más me gusta en cada ciudad. Pero no me llega con el dinero que he ahorrado este invierno y no quiero renunciar a ir a ninguna ciudad ni bajar la calidad de los hoteles que he elegido. He pedido consejo a mis amigos, y uno de ellos me ha indicado que si no me importa cambiar el orden en que visito las ciudades es posible que encuentre una ruta más barata, al utilizar trayectos de menor coste, o las ofertas de precios de los hoteles.

Ha prometido ayudarme si le paso el precio de los billetes entre cada ciudad, el precio de los hoteles en temporada alta y en temporada baja y cuantos días pasan desde que salgo hasta que comienza la temporada baja en los hoteles europeos. Le he recordado que debe incluir el precio de volver a casa desde la última ciudad, no vaya a ser que me quede sin fondos antes de tiempo.

Por ejemplo, en la figura del enunciado, si salgo de Madrid y voy a Paris, donde paso una noche, A continuación bajo a Niza, y luego Venecia y Roma... si la temporada baja comienza el día 3, esto significa que utilizaré el hotel de Paris (dia 1) y Niza (dia 2) con precio de temporada alta y Venecia (dia 3) será la primera ciudad donde tenga el precio de temporada baja.

Requisitos de implementación.

El problema se debe resolver utilizando la técnica de vuelta atrás. Se debe podar el árbol de ejecución impidiendo realizar llamadas recursivas que no darán soluciones en base a la solución ya construida, pero no es necesario realizar estimaciones sobre los días que todavía no han sido tratados.

Explicar el algoritmo: como es la solución que se va construyendo y como la vamos a ir construyendo. Poner comentarios en el código.

Entrada

La entrada consta de una serie de casos de prueba. Cada caso de prueba consta de varias líneas. En la primera se muestra el número de ciudades n que voy a visitar (entre las que no se incluye mi ciudad de origen), el día del viaje en que comienza la temporada baja y el presupuesto que tengo. En la línea siguiente se dan los precios de los hoteles de las n ciudades en temporada alta y el la siguiente línea el precio de los hoteles en la temporada baja. Por último en las $n+1$ líneas siguientes se da el precio de los billetes entre cada par de ciudades, incluyendo la mi ciudad de origen, que se considera la ciudad 0. El precio de ir de la ciudad A a la B será el mismo que el de ir de la ciudad B a la A. La entrada de datos termina con un caso con tres ceros.

El número de ciudades es un entero, 2 <= n <= 20 y el coste de los billetes y de los hoteles 10 <= p <= 500.

Salida

Para cada caso de prueba se muestra en un línea el precio de la ruta más barata. Si no existe ninguna ruta posible con el presupuesto que tengo se escribirá *No puedes ir*.

Entrada de ejemplo

```
1 1 100
150
80
0 10
10 0
4 3 200
1 1 1 1
1 1 1 1
0 30 20 10 50
30 0 15 40 60
20 15 0 16 40
10 40 16 0 35
50 60 40 35 0
4 3 350
50 100 150 70
40 50 70 60
0 30 20 10 50
30 0 15 40 60
20 15 0 16 40
10 40 16 0 35
50 60 40 35 0
0 0 0
```

Salida de ejemplo

```
100
134
No puedes ir
```

Una combinación de ciudades visitadas con el precio dado

```
42
0 3 0 2 3 4
48
0 1 0 1 0 1 0 1
50
0 1 1 2 0 2
```

33. Adornando la casa por Navidad

Este año mis padres han decidido renovar los adornos que tradicionalmente ponemos en la casa por Navidad. Quieren comprar algunas figuras para el jardin delantero y para colgar en la fachada; un enorme árbol del que cuelguen bolas, luces y muñecos; y varias guirnaldas para las paredes y techos. Han seleccionado en un catálogo todos aquellos objetos que les gustaría poner. Ahora quieren maximizar la superficie cubierta por los adornos, sin sobrepasar el presupuesto que tienen. Debemos tener en cuenta que los adornos no se pueden partir, ya que correríamos el riesgo de provocar un cortocircuito eléctrico.

Â

Requisitos de implementación.

El objetivo del problema es practicar la técnica de vuelta atrás. El árbol de exploración debe ser binario.

El problema puede resolverse de forma más eficiente utilizando la técnica de programación dinámica al ser los datos de la superficie ocupada por los objetos números enteros. Si estos datos fuesen números reales la técnica de programación dinámica no se podría utilizar y deberíamos resolverlo por vuelta atrás.

Entrada

La entrada consta de una serie de casos de prueba. Cada caso de prueba consta de 3 líneas. En la primera se indica el número de objetos que se pueden comprar y el presupuesto con que se cuenta. En la siguiente se indican los costes de cada uno de los objetos, y en la tercera la superficie que ocupa cada uno.

Salida

Para cada caso de prueba se escribe en una línea la máxima superficie que pueden cubrir.

Entrada de ejemplo

```
2 10
1 10
1 4
4 10
4 3 5 2
2 4 5 1
4 10
4 3 5 3
6 4 5 1
8 15
7 5 10 8 6 5 7 6
6 4 8 9 8 6 5 8
8 20
7 5 10 8 6 5 7 6
6 4 8 4 5 6 5 8
```

Salida de ejemplo

```
4
10
11
17
20
```

Salida del ejemplo mostrando un vector solución. En la primera línea se muestra una asignación óptima y en la línea siguiente el coste de esta solución. Esta forma de escribir los datos NO es la utilizada en el juez, solo se da para ayudar a depurar el programa.

Salida de ejemplo

```
1
4

1 2 3
10

0 1 3
11

3 4
17

0 5 7
20
```

Autor: Isabel Pita.

34. Las tareas de la clase

Ha comenzado el curso y la profesora tiene que asignar los diversos cargos y tareas de la clase: quien será el delegado, quien anotará los deberes en la agenda, quien borrará la pizarra entre clases o quien se encarga de recoger y repartir los trabajos. Para ello ha solicitado a los alumnos que le indiquen sus preferencias. Darán una puntuación entre 1 y 10 a cada tarea. 10 indica que les gustaría mucho realizarla y 1 que no quieren hacerla. Si hay varias tareas que les gustan parecido, pueden darles la misma puntuación.

Ahora ha llegado el momento de hacer la asignación. La profesora va a elegir 2 alumnos para cada puesto, ya que a veces alguno de ellos falta y no quiere que la tarea quede sin responsable. Para intentar que participen el mayor número posible de alumnos y evitar que unos pocos acaparen todos los puestos, ningún alumno ocupará más de t puestos. Como quiere que los alumnos estén lo más contentos posible, intentará asignar los puestos según las preferencias que han formulado. Busca la asignación tal que la suma de las preferencias expresadas por los alumnos seleccionados para los puestos sea máxima.

Requisitos de implementación.

El problema se debe resolver utilizando la técnica de vuelta atrás. Se debe podar el árbol de ejecución impidiendo realizar llamadas recursivas que no darán soluciones en base a la solución ya construida, pero no es necesario realizar estimaciones sobre las tareas que todavía no han sido tratadas.

Explicar el algoritmo: como es la solución que se va construyendo y como la vamos a ir construyendo. Poner comentarios en el código.

Entrada

La entrada consta de una serie de casos de prueba. Cada caso de prueba consta de varias líneas. En la primera se muestra el número de tareas n que hay que realizar, el número de alumnos de la clase a y el número máximo de tareas que puede realizar un alumno t. En las a líneas siguientes se muestran n valores que representan las preferencias del alumno para cada tarea.

El número de tareas es un entero, 1 <= n <= 6, el número de alumnos es un entero, 2 <= n <= 8 y el número máximo de tareas que puede realizar un alumno 1 <= t <= 6. Se garantiza que siempre existe una asignación posible, es decir $a*t > 2*n$

Salida

Para cada caso de prueba se muestra en una línea la suma de las preferencias de los alumnos que realizarán las tareas.

Entrada de ejemplo

```
3 5 2
5 10 2
1 8 4
3 10 3
4 7 7
1 7 6
4 2 4
5 5 5 5
7 7 7 7
3 3 2
10 10 8
8 9 1
1 7 8
0 0 0
```

Salida de ejemplo

```
42
48
50
```

Una asignación óptima

```
42
0 3 0 2 3 4
48
0 1 0 1 0 1 0 1
50
0 1 1 2 0 2
```

35. Escuchando música

La mejor forma de que un viaje se haga corto es ir escuchando música. Ultimamente presto mucha atención a como mi reproductor elige las canciones, ya que he comprobado que mi buen humor a lo largo del día depende en gran medida de la música que encucho cuando voy y vuelvo del trabajo. Para mejorar la selección, he puntuado todas las canciones que tengo del 1 al 100 y he modificado mi reproductor para que reproduzca las canciones que maximizan mis gustos. Después de probarlo un par de veces sigo sin estar satisfecho del todo. Es cierto que las canciones que reproduce me gustan, pero de vez en cuando la última canción se queda a medias. LLego al destino y no ha acabado. Esto me produce una gran frustación. A partir de ahora quiero que las canciones acaben justo cuando llego.

Para cada canción que tengo, el reproductor decidirá si la reproduce en el trayecto de ida, en el de vuelta o si no la reproduce, basándose en la puntuación que he dado a las canciones y en que deben ajustarse al tiempo del trayecto de ida y al del trayecto de vuelta.

Requisitos de implementación.

El problema se debe resolver utilizando la técnica de vuelta atrás. El árbol de ejecución que se recorre debe ser ternario, con una complejidad máxima del orden de 3 elevado a n siendo n el número de canciones. No se aceptarán soluciones cuyo esquema no sea este.

Se debe podar el árbol de ejecución impidiendo realizar llamadas recursivas que no darán soluciones en base a la solución ya construida, pero no es necesario realizar estimaciones sobre las canciones que todavía no han sido tratadas.

Explicar el algoritmo: como es la solución que se va construyendo y como la vamos a ir construyendo. Poner comentarios en el código.

Entrada

La entrada consta de una serie de casos de prueba. Cada caso de prueba consta de varias líneas. En la primera se muestra el número de canciones n que hay en el reproductor, el tiempo que dura el viaje de ida *t1* y el tiempo que dura el viaje de vuelta *t2*. En las n líneas siguientes se muestra la duración de cada canción seguida de la satisfacción que me produce escucharla.

El número de canciones es un valor, $1 <= n <= 50$ y el tiempo de cada trayecto $1 <= t1, t2 <= 100$

Salida

Para cada caso de prueba se muestra en una línea la satisfacción que obtengo al final del día con la música escuchada en los dos viajes o el mensaje *Imposible* si no es posible formar una solución con las canciones dadas que cumpla los requisitos.

Entrada de ejemplo

```
4 20 15
15 7
10 10
5 6
10 8
6 20 15
5 7
3 10
10 8
5 6
8 8
7 6
2 10 10
10 7
15 4
0 0 0
```

Salida de ejemplo

```
25
35
Imposible
```

Una asignación óptima

```
25
Canciones Ida: 1, 3
Canciones Vuelta: 0
35
Canciones Ida: 3,4,5
Canciones Vuelta: 0,2
Imposible
```