**Recopilación Ejercicios de ADA**

Enunciados algoritmos voraces.

* El dragoon.

Érase una vez un feroz dragón de múltiples cabezas que estaba aterrorizando a las tranquilas gentes de un lejano país. Los habitantes de los poblados se estaban organizando para acabar con el dragón. Parece que el único método que funciona es cortar todas las cabezas a la vez. Si no se consigue cortarlas todas simultáneamente, se vuelven a regenerar y el dragón mata a los atacantes.

Para acabar con el dragón tienen que atacarle tantas personas como cabezas tiene el dragón.

Además, como las cabezas del dragón están a distintas alturas, los atacantes deben alcanzar alturas distintas con sus espadas para que puedan cortar cada cabeza.

Una persona podrá cortar cualquier cabeza del dragón que se encuentre a una altura inferior o igual a la máxima altura alcanzada con la espada.

Un poblado te ha pedido que hagas un programa de forma que puedan saber si van a conseguir derrotar al dragón o no.

Te pasan la siguiente información:

- N: número de cabezas del dragón

- P: número de personas que van a atacar al dragón

- ci: altura de cada cabeza del dragón (habrá N valores ci).

- ai: altura máxima alcanzada con la espada por cada una de las personas que van a por el dragón (habrá P valores ai).

Tienes que decirles si conseguirán acabar o no con el dragón.

**Por ejemplo**:

N: 3 P: 4 ci = [1.6, 2.0, 1.8] ai = [2.1, 1.6, 2.0, 1.55] 🡪 Pueden matar al dragón

N: 4 P: 4 ci = [1.6, 2.0, 1.8, 1.9] ai = [2.1, 1.6, 2.0, 1.55] 🡪 No matan al dragón

* Knacksap Problem
* Travelling salesman
* Carpanta en el bufé.

Algunos restaurantes ofrecen bufés libres donde puedes comer todo lo que quieras, una buena solución para estudiantes hambrientos. A Carpanta le gustan este tipo de restaurantes, pero está siempre preocupado por la mejor forma de llenar su plato de comida.

Al entrar, valora cada uno de los n alimentos que se ofrecen. Su objetivo es maximizar el valor del plato, llenándolo con los alimentos más valiosos, teniendo en cuenta el área limitada del plato y la disponibilidad de los alimentos.

Afortunadamente, todos los alimentos del menú se pueden dividir fácilmente y Carpanta puede coger una fracción de cada alimento. ¿Le puedes ayudar a llenar su plato?

**Entrada**

La entrada del programa está formada por n+2 líneas:

La primera línea contiene el número n de alimentos disponibles en el menú (1 <= n <= 1000).

La segunda línea consiste en el área a del plato de Carpanta, un entero en mm2 (0 <= a <= 100000).

Cada una de las siguientes n líneas contiene la información de un alimento del menú. Son dos enteros separados por un espacio: el primero es el valor vi del alimento i por mm2 según la valoración de Carpanta; el segundo es el área ai en mm2 que el alimento i ocuparía entero en el plato. (0 <= vi <=100; 0 <= ai <= 100000000).

**Salida**

Una única línea con un entero: el valor máximo que Carpanta puede conseguir en su plato.

**Ejemplo de entrada**

5

1000

50 230

80 12

10 1000000

25 450

2 50

**Ejemplo de salida**

26790

* El camionero presuroso.

Un camionero conduce desde Bilbao a Málaga siguiendo una ruta prefijada y llevando un camión que le permite, con el tanque de gasolina lleno, recorrer r kilómetros sin parar. El camionero dispone de un mapa de carreteras que le indica las distancias entre las gasolineras que hay en su ruta. Como va con prisa, el camionero desea pararse a repostar el menor número de veces posible.

Deseamos diseñar un algoritmo ávido para determinar en qué gasolineras tiene que parar.

Disponemos de un vector con la información que tiene el camionero sobre las distancias entre las n gasolineras, de forma que el i-ésimo elemento del vector indica los kilómetros que hay entre las gasolineras i-1 e i, considerando las gasolineras ordenadas según el orden de aparición en el trayecto del camionero. Para que el problema tenga solución, suponemos que ningún valor del vector es mayor que r.

**Entrada**

La entrada estará formada inicialmente por el entero r. Después a continuación en otra línea diferente una secuencia de números que acaba en 0 y también cada uno en una línea diferente, donde se va indicando a cuántos kilómetros queda la siguiente gasolinera.

**Salida**

La salida estará formada por un vector de ceros y unos, donde cero es que el camionero no va a parar y 1 que sí tiene que repostar, los elementos de este vector no irán separados entre sí.

**Ejemplo de entrada**

150

75

150

25

40

80

130

20

0

**Ejemplo de salida**

1100100

* Alí\_Babá.

Cuando Alí-Babá consigue entrar en la Cueva de los Cuarenta Ladrones encuentra una gran cantidad de objetos muy valiosos. A pesar de su pobreza, Alí-Babá conoce muy bien el peso y valor de cada uno de los objetos de la cueva. Debido a los peligros que tiene que afrontar en su camino de vuelta, sólo puede llevar consigo aquellas riquezas que quepan en su pequeña mochila, que soporta un peso máximo conocido.

**a)** Suponiendo que los objetos son diferentes y fraccionables, determinar qué estrategia debe seguir Alí-Babá para elegir los objetos y maximizar el valor total de lo que pueda llevar en su mochila.

**b)** ¿Sigue siendo óptima la estrategia utilizada en el apartado anterior para elegir los objetos, en el caso de que no sean fraccionables?

**c)** Supongamos ahora que en la cueva hay una cantidad determinada de riquezas agrupadas por clases (ahora, los objetos pueden repetirse). Cada clase es fraccionable y tiene cierto peso y valor conocidos. Implementa un algoritmo para determinar qué cantidad de riquezas de cada clase debe coger Alí-Babá para maximizar el valor total de lo que puede llevar en su mochila.

* Temporada de conciertos.

La Orquesta de Extremadura quiere que les ayudes con la planificación de la temporada de conciertos del próximo año.

El director quiere tocar una serie de obras a lo largo de todos los conciertos de la temporada. La duración máxima de los conciertos está limitada, así que no sabe si podrá organizar las obras para cada concierto de forma que no superen esa duración máxima.

**Entrada**

La entrada contiene un único caso de prueba.

La primera línea contiene el número n de conciertos que se van a dar en el año (1 <= n <= 15).

La segunda línea tiene la duración máxima dm en minutos de un concierto (un número entero). Todos los conciertos tienen la misma duración máxima.

La tercera línea contiene el número nob de obras que quiere tocar la orquesta (1 <= nob <= 100).

A continuación, en una línea, separadas por espacios, nob enteros con la duración de cada una de las obras. (La duración de cada obra estará entre 1 y dm)

**Salida**

Se escribirá SI o NO si se puede o no asignar cada obra a un concierto y no se sobrepasa la duración máxima de ningún concierto con las obras que se le han asignado.

**Ejemplo de entrada 1**

3

100

7

40 80 30 10 20 50 50

**Ejemplo de salida 1**

SI

**Ejemplo de entrada 2**

3

100

4

80 80 90 30

**Ejemplo de salida 2**

NO

Enunciados backtrackings

* Sudoku.

El Sudoku es un rompecabezas matemático de colocación que se popularizó en Japón en 1986 y se dio a conocer en el ámbito internacional en 2005.

El objetivo es rellenar una cuadrícula de 9x9 celdas dividida en sub-cuadrículas de 3x3 con las cifras del 1 al 9 partiendo de algunos números ya dispuestos en algunas de las celdas. Las condiciones de colocación son:

No se debe repetir ninguna cifra en una misma fila.

No se debe repetir ninguna cifra en una misma columna.

No se debe repetir ninguna cifra en una misma sub-cuadrícula.

**Entrada**

Matriz de 9x9 donde un 0 indicará posición de libre colocación de cifra de 1 a 9, y cualquier valor distinto de 0 indicará un número ya dispuesto que no deberá modificarse.

**Salida**

Si el problema tiene solución, mostrará una matriz de 9x9 con la solución del sudoku. En caso contrario, deberá mostrar el mensaje: NO SOLUCION (no utilizar acentos).

**Ejemplo de entrada**

5 3 0 0 7 0 0 0 0

6 0 0 1 9 5 0 0 0

0 9 8 0 0 0 0 6 0

8 0 0 0 6 0 0 0 3

4 0 0 8 0 3 0 0 1

7 0 0 0 2 0 0 0 6

0 6 0 0 0 0 2 8 0

0 0 0 4 1 9 0 0 5

0 0 0 0 8 0 0 7 9

**Ejemplo de salida**

5 3 4 6 7 8 9 1 2

6 7 2 1 9 5 3 4 8

1 9 8 3 4 2 5 6 7

8 5 9 7 6 1 4 2 3

4 2 6 8 5 3 7 9 1

7 1 3 9 2 4 8 5 6

9 6 1 5 3 7 2 8 4

2 8 7 4 1 9 6 3 5

3 4 5 2 8 6 1 7 9

* Anfitrión perfecto.

Un amigo va a celebrar una cena (a la que te ha invitado), y quiere que no haya problemas entre los asistentes, ya que algunos no se llevan demasiado bien. Ayúdale a resolver este problema mediante un programa.

El problema consiste en sentar alrededor de una mesa redonda a las N personas que asisten (Cada asistente está identificado por un número de 0 a N-1). Tenemos una matriz afinidad que contiene un valor entre 0 y 10 que indica lo bien que se llevan las personas i y j (0: no se soportan; 10: se llevan muy bien). (Se supone que la relación de afinidad es simétrica.)

El anfitrión quiere que diseñes un programa que le diga cómo debe distribuir a los n asistentes alrededor de la mesa de forma que se maximice el bienestar de todos ellos. Este bienestar se calcula sumando la afinidad que tiene cada invitado con las personas que están sentadas justo a su lado (derecha e izquierda).

**Entrada**

Primera línea un número N entre 1 y 12. A continuación vendrá dada la matriz de afinidad, de tamaño NxN.

**Salida**

Primera línea un vector de tamaño N con el orden de asientos en la mesa circular. En la línea siguiente, valor de bienestar de la solución encontrada.

**Ejemplo de entrada**

6

0 1 4 10 7 8

1 0 0 5 6 3

4 0 0 1 10 2

10 5 1 0 8 7

7 6 10 8 0 3

8 3 2 7 3 0

**Ejemplo de Salida**

0 3 1 4 2 5

82

* El polígono mágico.

Un polígono regular es un polígono que tiene n lados, todos iguales y también los n ángulos iguales. Además, diremos que es un polígono mágico si podemos asignar a cada vértice un entero entre 1 y n (sin repeticiones) de forma que la suma de dos números en vértices adyacentes sea un número primo.

Escribe un módulo que escriba todas las asignaciones posibles de los números naturales 1, 2, 3, ...n a los vértices, de modo que formen un polígono mágico. Para evitar generar la misma solución en sus diferentes traslaciones, fija a 1 el valor del primer vértice que vayas a tratar. De esta forma se evita generar soluciones repetidas.

**La entrada**: consiste simplemente en un dígito indicando el número de lados del polígono.

**La salida**: debes escribir cada solución en una línea. Cada valor debe ir separado del anterior por un espacio en blanco.

**Ejemplo de entrada**: 6

**Ejemplo de salida**

1 4 3 2 5 6

1 6 5 2 3 4

* Subconjuntos de suma dada.

Sea W una secuencia de enteros no negativos y M un número entero positivo. El problema consiste en diseñar un algoritmo para encontrar todos los posibles subconjuntos de W cuya suma sea exactamente M.

**Entrada**

La entrada estará formada inicialmente por el entero M. A continuación, debes leer una secuencia de números enteros positivos que acaba en 0, cada uno en una línea diferente. Estarán ordenados de menor a mayor y no hay valores repetidos.

**Salida**

La salida estará formada por vectores (subconjuntos) donde cada elemento (número) irá separado por un espacio en blanco y cada vector estará en una línea distinta.

**Ejemplo de entrada**

7

1

2

3

4

6

8

0

**Ejemplo de salida**

1 2 4

1 6

3 4

* El ministerio.

El ministro de Desinformación se ha propuesto hacer trabajar en firme a sus n funcionarios, para lo que se ha sacado de la manga n trabajos. Todos los funcionarios son capaces de hacer cualquier trabajo, aunque unos lo hacen de forma más eficiente que otros. La información al respecto se recoge en una matriz bidimensional cuadrada de n x n elementos, E, donde E[i,j] representa la eficiencia con la que el funcionario i realiza el trabajo j.

El Ministro desea conocer la asignación óptima de estos n trabajos a los n funcionarios (una tarea para cada uno) de tal modo que la suma total de eficacias sea máxima.

**Entrada**

La entrada estará formada inicialmente por el número n, en la siguiente línea aparecerá la primera fila de la matriz E separada por espacios en blanco, y así sucesivamente para el resto de las filas de la matriz hasta completarla.

**Salida**

La salida estará formada por el mejor vector de todos los posibles, con sus elementos separados por un espacio en blanco y a continuación separado por un espacio en blanco de dicho vector su eficiencia total.

**Ejemplo de entrada**

4

1 5 1 5

1 1 2 2

1 3 1 2

1 1 4 1

**Ejemplo de salida**

Indica que la forma de asignar las tareas a cada funcionario es 3, 0, 1 y 2 respectivamente. Y la mejor eficiencia global que se puede obtener es 13.

3 0 1 2 13

* Subconjuntos de números binarios.

Aunque ya sabemos que no es necesario “tanta complejidad” para generar números binarios, para que seas más hábil en el uso de los esquemas algorítmicos, se plantea este problema. Desarrollar un algoritmo que se ajuste a un esquema de backtracking y que genere de forma ordenada y sistemática todos los números binarios de n dígitos y que contengan una cantidad par de 0´s.

**ENTRADA**

La entrada estará formada únicamente por un natural n mayor que 1 y menor que 10.

**SALIDA**

La salida estará formada por los números binarios desde el 0 en adelante, si cumplen la restricción de tener una cantidad par de 0´s. Cada uno de ellos se escribe en una línea.

**Ejemplo de entrada**

4

**Ejemplo de salida**

0000

0011

0101

0110

1001

1010

1100

1111