Ejercicio 1: *Backtracking*

Se pide resolver el ejercicio 2 de la guía práctica 1 junto con las podas propuestas en el ejercicio. En este ejercicio vamos a utilizar los cuadrados mágicos de orden 3 y 4.

Definimos el orden lexicográfico entre dos matrices. Se dice que la matriz A es menor que la matriz B en el orden lexicográfico si y solo si se cumple que:

Existen dos índices i y j tal que la entrada de A en la fila i y columna j es menor que la entrada correspondiente en B, y para todo i' menor que i vale que los valores de las filas i' son iguales en todas las columnas, y para todo j' menor a j todos los valores en la fila i en la columna j' son iguales.

En otras palabras, la comparación se hace entrada por entrada en orden lexicográfico de izquierda a derecha y de arriba hacia abajo, y se detiene en la primera entrada en la que las matrices difieren. Si en esa posición la matriz A tiene un valor menor que la matriz B, entonces A es menor que B en el orden lexicográfico.

Dados los valores n y k. Queremos encontrar el k-ésimo cuadrado mágico de orden n si se tuvieran todos los cuadrados mágicos ordenados lexicográficamente.

Descripción de una instancia

Dos números en una línea, n y k. El valor de n es el orden del cuadrado mágico y k es la posición en el orden lexicográfico.

Descripción de la salida

Si existe solución, el programa debe devolver n líneas, cada una con n números, representando el cuadrado mágico. Si no existe solución, imprimir -1.

Ejemplo de entrada y salida

Se presenta un ejemplo de entrada y su correspondiente salida:

Entrada de ejemplo	Salida esperada de ejemplo
3 2	2 9 4 7 5 3 6 1 8

Entrada de ejemplo	Salida esperada de ejemplo
3 9	-1

Se pide:

1. Diseñar un algoritmo para resolver este problema que utilice podas para conseguir rápido una solución. Este algoritmo debe correr en el juez en un límite de tiempo estipulado.

Ejercicio 2: Programación dinámica

Se pide resolver una variante del ejercicio de 11 la guía práctica 1. Además de las operaciones del enunciado se agrega la operación - (resta). También, en vez de obtener el número w como resultado, se busca que luego de realizar las operaciones se obtenga un número w tal que el resto al dividir por un número m sea r.

Descripción de una instancia

El input contiene múltiples casos de test. La primera linea contiene el entero c, la cantidad de casos. Para cada caso de test hay dos líneas. La primera línea contiene tres enteros n, r y m. Donde n es el tamaño del vector v y r es el resto al dividir el resultado w por m. La segunda línea contiene n enteros con los valores del vector v.

Descripción de la salida

Para cada caso de test imprimir "Si" si se puede obtener el número r y "No" si no es posible.

Ejemplo de entrada y salida Se presenta un ejemplo de entrada y su correspondiente salida.

Entrada de ejemplo	Salida esperada de ejemplo
3 3 1 100000 4 4 7 4 10 100000 7 10 10 50 3 10 100000 1 1 1	Si Si No

Se pide:

1. Diseñar un algoritmo eficiente para resolver este problema. Este algoritmo debe correr en el juez en el límite de tiempo estipulado.

Ejercicio 3: Algoritmo goloso

Resolver el problema de selección de actividades presentado en el ejercicio 17 de la guía práctica 1 inciso *d)*. Vamos a considerar que una actividad puede comenzar en el instante que termina la anterior.

Descripción de una instancia

La primer línea contiene un número entero N. La siguientes N lineas contienen 2 enteros s_i y t_i respetando las restricciones en el enunciado de la guía.

Descripción de la salida

La primer línea debe contener la máxima cantidad de tareas x. La segunda línea debe contener x enteros con la secuencia $A_1 \dots A_x$.

Ejemplo de entrada y salida Se presenta un ejemplo de entrada y su correspondiente salida.

Entrada de ejemplo	**Salida esperada de ejemplo**
5 6 7 1 4 0 3 4 10 3 6	3 3 5 1

Se pide:

1. Una implementación del inciso *d)* que resuelva el problema de selección de actividades. Este algoritmo debe correr en el juez en el límite de tiempo estipulado.
2. Presentar un informe de hasta **5 páginas** que debe contener:
   - Una presentación y descripción del problema.
   - Una explicación del algoritmo desarrollado.
   - Justificar la correctitud del algoritmo utilizando la demostración del inciso *d)* (se deben incluir la demostración).
   - Determinar la complejidad computacional del algoritmo presentado y justificar.
   - Mostrar empíricamente con un conjunto de instancias que el algoritmo tiene la complejidad justificada.
   - Existe un conjunto de instancias que sean más fáciles de resolver? Existe un conjunto de instancias que sean más difíciles de resolver? En otras palabras, si fijamos el tamaño de instancia, podemos construir los s_i y t_i de tal forma que podamos decir algo del comportamiento del algoritmo?