***Instrucciones***

A continuación, se presenta un problema que se tendrá que resolver usando técnicas de análisis y diseño de algoritmos. La solución tendrá que ser implementado en un lenguaje de programación (C#, Java, JavaScript) o pseudocódigo o la herramienta que Uds considere adecuada (papel, etc).

La solución propuesta, completa o parcial, deberá ser sustentada durante la etapa de entrevista (Hacer énfasis en la complejidad computacional).

Glosario

**Complejidad Computacional:** Es la rama de las ciencias de la computación que estudia, de manera teórica, la optimización de los recursos requeridos durante la ejecución de un algoritmo para resolver un problema en cuestión.

Ejm

O(n) si el costo es lineal

O(n2) si el costo es cuadrático - generalmente involucra tener dos bucles anidados

**Problema nro 1 - Cálculo de salarios**

Se está trabajando en el departamento de recursos humanos de una gran corporación. Cada empleado puede tener varios jefes directos y/o varios subordinados directos. Por supuesto, los subordinados también pueden tener sus propios subordinados, y los jefes pueden tener sus propios jefes.

Un empleado X es un jefe de otro empleado Y, si existe una secuencia de empleados A, B, ..., D, tal que X es el jefe de A, A es el jefe de B, y así sucesivamente, y D es el jefe de Y (por supuesto, si X es un jefe directo del empleado Y, X será un “jefe directo” del empleado Y).

Si A es un jefe de B, entonces B no puede ser un jefe de A. De acuerdo con la nueva política de la empresa, el salario de un empleado sin subordinados es 1. Si algún empleado tiene subordinados, entonces su salario es igual a la suma de los salarios de sus subordinados directos.

Se le dará un arreglo de string, donde el elemento ‘j-ésimo’ del ‘i-ésimo’ elemento es 'Y' si el empleado ‘i’ es un jefe directo del empleado ‘j’, y 'N' de otra manera. Devolver la suma de los salarios de todos los empleados ( para ‘j’ considerar la posición del carácter dentro de un string y para ‘i’ considerar la posición del string en el arreglo - Ver los ejemplos)

Otras consideraciones

Las matrices que se ingresan son cuadráticas

Las matrices solo contienen ‘Y’ o ‘N’

Carácter i del elemento i de la matriz, siempre será "N" para todos los i.

Si A es un jefe de B, entonces B no va a ser un jefe de A.

Ejemplo 1

{“N”}

La respuesta para este caso es 1, ya que se trata del caso base

Ejemplo 2

{"NNYN",

"NNYN",

"NNNN",

"NYYN"}

En este caso tenemos 4 empleados (4 empleados por las 4 filas)

Son jefes la fila 0, la fila 1 y la fila 3 (ya que por lo menos tienen una ‘Y’)

No son jefes la fila 2 (ya que solo contiene “N”)

Los jefes 0 y 1, tienen como subordinado a 2

El jefe 3, tiene como subordinado a 2 y a 1

Resolución

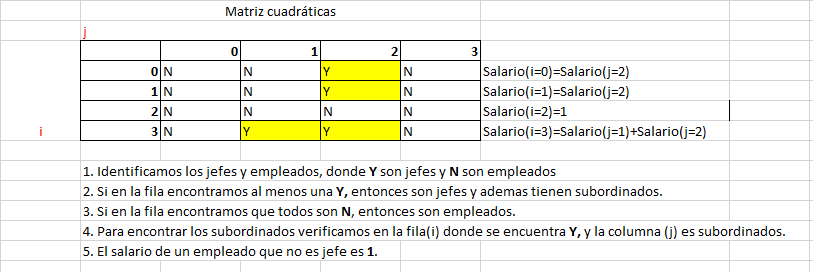
salario (0) = salario (2) = 1

salario (1) = salario (2) = 1

salario (2) = 1

salario (3) = salario (2) + salario (1) = 2

Rpta = 5



**public class** Filas {

**private** **int** i;

**private** **boolean** esjefe;

**private** List<Integer> subordinados;

**public** Filas(**int** i, List<Integer> subordinados) {

**this**.i = i;

**this**.subordinados = subordinados;

}

**public** Filas() {

}

**public** **boolean** isEsjefe() {

**return** esjefe;

}

**public** **void** setEsjefe(**boolean** esjefe) {

**this**.esjefe = esjefe;

}

**public** **int** getI() {

**return** i;

}

**public** **void** setI(**int** i) {

**this**.i = i;

}

**public** List<Integer> getSubordinados() {

**return** subordinados;

}

**public** **void** setSubordinados(List<Integer> subordinados) {

**this**.subordinados = subordinados;

}

@Override

**public** String toString() {

**return** "Filas [i=" + i + ", subordinados=" + subordinados + "]";

}

}

**public** **class** Main {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

String[][] jefesEmpleados = { { "N", "N", "Y", "N" }, { "N", "N", "Y", "N" }, { "N", "N", "N", "N" },

{ "N", "Y", "Y", "N" } };

**int** longitud = jefesEmpleados.length;

**for** (**int** i = 0; i < longitud; i++) {

Filas fila = **new** Filas();

fila.setI(i);

fila.setEsjefe(**false**);

List<Integer> subordinados = **new** ArrayList<>();

**for** (**int** j = 0; j < longitud; j++) {

**if** (jefesEmpleados[i][j].equals("Y")) {

fila.setEsjefe(**true**);

subordinados.add(j);

}

}

fila.setSubordinados(subordinados);

**if** (fila.isEsjefe()) {

System.***out***.print("Salario(i=" + fila.getI() + ")=");

**for** (**int** k = 0; k < subordinados.size(); k++) {

System.***out***.print("Salario(j=" + subordinados.get(k) + ")");

**if**(k<subordinados.size()-1)

System.***out***.print("+");

}

System.***out***.print("="+subordinados.size());

} **else** {

System.***out***.print("Salario(i=" + fila.getI() + ")=1");

}

System.***out***.println();

}

}

}

Fuente: **TOPCODER**

**Problema nro 2 – Buenos amigos**

Pocas cosas le gustan tanto a Leopoldo como compartir sus conocimientos y ayudar a la gente. Es por eso que se la pasa dando Training Camps de programación competitiva: viajando de acá para allá, encendiendo chispas de creatividad e ingenio.  
  
En el último curso que Leopoldo dictó, olvidó sus libros favoritos de programación competitiva. Desafortunadamente la agenda de Leo está muy llena (Leo se caracteriza por ser una persona muy ocupada) y no tiene tiempo de ir a buscarlos.  
  
La buena noticia es que Leo es una persona muy querida (y tiene muchos amigos). Al enterarse de su problema sus amigos y conocidos se ofrecieron gratamente a ayudarlo.  
  
Los libros de Leo han quedado a una distancia D de donde él actualmente se encuentra y es posible caminar a ellos en línea recta. Como todas las personas que están con él también tienen otras obligaciones, se han ofrecido a ayudarlo yendo desde donde él está hasta los libros y acercarlos lo más posible. Cada uno puede caminar a lo sumo una determinada distancia dependiendo de cuanto tiempo dispone (sumando lo que le demanda llegar hasta donde están los libros y la distancia que los acercan). Luego de haber caminado dicha distancia, dejan los libros y se van a atender sus otras responsabilidades.  
  
Leo puede pedirle ayuda a sus M mejores amigos, que están dispuestos a caminar una distancia D1 cada uno. Como Leo es muy popular y estimado, otras N personas se ofrecieron a caminar una distancia D2 cada una (D2 ≤ D1).  
  
Por ejemplo, si los libros estuvieran a distancia 8, Leo tuviera solo un mejor amigo dispuesto a caminar una distancia total de 15 y solo un compañero que se ofreciera a caminar una distancia total de 10 (D = 8, M = 1, D1= 15, N = 1, D2 = 10) debería pedirle ayuda a ambas personas.  
  
Con una no alcanzaría, pues 15 < 8 + 8. Pero con la ayuda de ambos sí: por ejemplo, podría ir el compañero y acercar los libros hasta una distancia 6 de Leo (yendo 8 de ida y 2 de vuelta). Luego ir su mejor amigo y caminar los 12 que faltan para traer los libros (6 + 6 ≤ 15).  
  
A Leopoldo le encanta optimizar y quiere molestar a la mínima cantidad de personas que le sea posible. ¿Podrían ayudarlo a saber el mínimo número de personas a las cuales debería pedir ayuda para poder traer sus libros de vuelta?

#### Especificación de entrada

La entrada consiste en una sola línea con los enteros D (1 ≤ D ≤ 1018), M (0 ≤ M ≤ 100), D1 (1 ≤ D1 ≤ 1018), N (0 ≤ N ≤ 2·109), D2 (1 ≤ D2 ≤ D1) (en dicho orden) representando respectivamente: la distancia a los libros, la cantidad de mejores amigos, la distancia que dichos M mejores amigos están dispuestos a caminar a lo sumo cada uno, la cantidad de otros compañeros que se ofrecen a ayudarlo y la distancia que dichos N compañeros se ofrecen a caminar a lo sumo cada uno.

#### Especificación de salida

Imprimir en la salida una línea conteniendo un entero que representa la mínima cantidad de personas a las que Leo deba pedir ayuda. Si Leo no puede traer los libros de vuelta con las personas disponibles la respuesta debería ser **-1**.

#### Ejemplo de entrada

8 1 15 1 10

#### Ejemplo de salida

2

**Otro ejemplo entrada 1**  
7 1 15 1 10  
  
**Otro ejemplo salida 1**  
1  
  
**Otro ejemplo entrada 2**  
11 1 15 1 10  
  
**Otro ejemplo salida 2**  
-1  
  
**Otro ejemplo entrada 3**  
10000000000 1 100000000000 1 10  
  
**Otro ejemplo salida 3**  
1

Primero, debemos calcular la distancia que pueden caminar los amigos y compañeros de Leo.

La distancia total D1 =números de amigos (M)\* distancia recurrida (D1).

La distancia total D2 =números de amigos (N)\* distancia recurrida (D2)

La máxima cantidad de personas M y N.

Si la máxima distancia es mayor o igual a la distancia de los libros (D) entonces devolvemos al cantidad de personas involucradas, esta condición se debe verificar en cada interacción.

Si llegamos al final de todas las iteraciones sin haber encontrado una solución, entonces devolvemos -1.

import java.util.Scanner;

public class Main {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int D = sc.nextInt();

int M = sc.nextInt();

int D1 = sc. nextInt ();

int N = sc.nextInt();

int D2 = sc. nextInt ();

int D1\_total = D1 \* M;

int D2\_total = D2 \* N;

for (int i = 1; i <= M + N; i++) {

int num\_M = Math.min(i, M); // número de amigos M necesarios

int num\_N = i - num\_M; // número de amigos N necesarios

int max\_distance = num\_M \* D1 + num\_N \* D2;

if (max\_distance >= D) {

System.out.println(i);

return;

}

}

System.out.println("-1");

}

}