

EVALUACIÓN	Obligatorio	GRUPO	Todos	FECHA	15/9
MATERIA	Estructura de Datos y Algoritmos 2				
CARRERA	Ingeniería en Sistemas				
CONDICIONES	<ul style="list-style-type: none">- Puntos: Máximo: 30 Mínimo: 1- Fecha máxima de entrega: 03/12 <p>LA ENTREGA SE REALIZA EN FORMA ONLINE EN ARCHIVO NO MAYOR A 40MB EN FORMATO ZIP, RAR O PDF.</p> <p>IMPORTANTE:</p> <ul style="list-style-type: none">- Inscribirse- Formar grupos de hasta dos personas.- Subir el trabajo a Gestión antes de la hora indicada, ver hoja al final del documento: "RECORDATORIO"				

Obligatorio

El obligatorio de Estructuras de Datos y Algoritmos 2, para el semestre 2 de 2020, está compuesto por un conjunto de **12** problemas a resolver.

El desarrollo del obligatorio:

- Se deberá realizar en grupos de **2 estudiantes**.
- Se podrá utilizar tanto C++ (C++11) como Java (jdk8).
Para asegurar que utilizan C++11 deberán tener instalado el compilador C++ de GNU e invocar la siguiente orden: `g++ -std=c++11 ...`
- No se puede usar ninguna librería, clase, estructura, función o variable no definida por el estudiante, a excepción de `<iostream>`, `<string>` y `<cstring>` para C++. y `System.in`, `System.out` y `String` para Java.
- Si se quiere usar algo que no sea lo listado anteriormente, consultar previamente al profesor.
- Se deberá utilizar el formato de “miSolucion < input > output” trabajado durante el curso.
- La cátedra proveerá un conjunto de casos de prueba, pares de entrada y salida esperada. Se deberá comparar la salida de la solución contra una salida esperada.
- La solución de cada ejercicio deberá estar contenida en un único archivo.

La entrega:

- Se deberá entregar un zip, únicamente con el código fuente de cada ejercicio, en una estructura de archivos como la siguiente:

```
|—— ejercicio1.cpp
|—— ejercicio2.cpp
...
|—— ejercicio12.cpp
```

La corrección:

- La corrección implica la verificación contra la salida esperada, así también como la corrección del código fuente para verificar que se cumplan los requerimientos solicitados (órdenes de tiempo de ejecución, usos de estructuras o algoritmos en particular, etc.).
- Se verificarán TODOS los casos de prueba. Si el programa no termina para un caso de prueba, puede implicar la pérdida de puntos.
- Si el código fuente se encuentra en un estado que dificulta la comprensión, podrá perder puntos.

- Se utilizará MOSS para detectar copias.

La defensa:

- Luego de la entrega, se realizará una defensa de autoría a cada estudiante de manera individual.

1. SORTING Nombre de archivo: ejercicio1.cpp/Ejercicio1.java

Se desea ordenar un conjunto de números enteros, no acotados. Se solicita que implemente un algoritmo basado en la estrategia de *ordenación por inserción en árboles* en $O(N \log N)$ en el *peor caso*. Para lograr el objetivo, se deberá utilizar un AVL.

Formato de entrada

N
n_1
n_2
...
n_N

La primera línea indica la cantidad de elementos a ordenar. Las siguientes N líneas son los elementos a ordenar.

Formato de salida

La salida contendrá N líneas, siendo estos los elementos ordenados de menor a mayor.

2. DICCIONARIO Nom. de arch.: ejercicio2.cpp/Ejercicio2.java

Se desea construir un programa que opere eficientemente buscando palabras de un diccionario utilizando la técnica de hashing.

Formato de entrada

```
N
palabra1
palabra2
...
palabraN
M
palabra_test1
palabra_test2
...
palabra_testM
```

La primera línea, indica cuántas palabras contendrá el diccionario: $N \leq 10^6$. Cada una de las siguientes N líneas son las palabras del diccionario. Las palabras tendrán no más de 20 caracteres (recordar el carácter de finalización)

Luego viene otro número, $M \leq 10^6$, que indica cuántas palabras se desean buscar o determinar si existen en el diccionario. Es decir, primero se cargan N palabras y luego se consultan M palabras. Claramente, para que el ejercicio sea productivo, algunas de las M palabras a buscar estarán en el diccionario y otras no.

Formato de salida

La salida contendrá M líneas indicando en cada línea i si palabra_test_i pertenece al diccionario (1) o no (0).

3. INTERCALAR Nom. de arch.: ejercicio3.cpp/Ejercicio3.java

Se desea construir un programa que se encargue de intercalar un conjunto de K listas ordenadas, de diferente tamaño. Se debe resolver en $O(P \log_2 K)$, siendo P la cantidad total de elementos.

El algoritmo esperado se describe en:

https://en.wikipedia.org/wiki/K-way_merge_algorithm

Formato de entrada

```
K
N1
Elemento 1 de L1
Elemento 2 de L1
...
Elemento N1 de L1
N2
Elemento 1 de L2
Elemento 2 de L2
...
Elemento N2 de L2
...
```

La primera línea K indica cuántas listas contendrá el archivo de entrada.

La siguiente línea, N_1 nos indica el tamaño de la lista L_1 . Luego, las siguientes N_1 líneas, contienen los elementos de dicha lista. Esto se repite K veces, una vez por cada lista.

Formato de salida

La salida contendrá P líneas, cada una con los elementos de la lista resultante de intercalar las K listas ordenadamente.

Formato de entrada de entrada de ejercicios de grafos

Todos los ejercicios de grafos tendrán la misma codificación para los grafos. Es decir, una parte del formato de entrada, la que corresponde a la información del grafo será siempre igual. A continuación se describe:

```
V
E
v1 w1 [c1]
v2 w2 [c2]
...
vi wi [ci]
...
vE wE [cE]
```

Cada grafo comienza con la cantidad de vértices, V . Los vértices siempre serán números, a menos que se especifique lo contrario. Por ejemplo, si $V=3$, entonces los vértices serán: $\{1, 2, 3\}$ (siempre serán numerados a partir de 1).

La siguiente línea corresponde a la cantidad de aristas, E . Las siguientes E líneas en el formato $v \ w \ c$ corresponden a las aristas (v,w) con costo c si el grafo es *ponderado*, o en el formato $v \ w$, correspondiente a la arista (v,w) si *no es ponderado*. Es decir, c es opcional ($[c]$).

El grafo será dirigido o no, dependiendo el problema en particular. En caso de ser *no dirigido* solo solo se pasará un sentido de la arista, es decir, (v,w) pero no (w,v) (queda implícito). Por ejemplo:

```
2
1
1 2
```

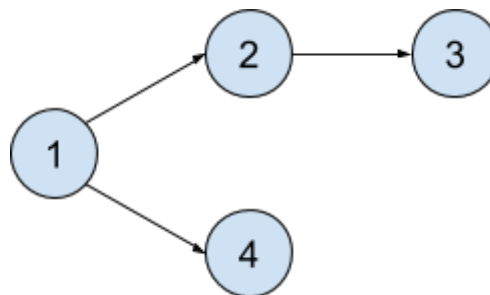
Representa al grafo completo de dos vértices y aristas: $\{(1,2), (2,1)\}$.

4. ORDEN TOP.

Nom. de archivo: ejercicio4.cpp/Ejercicio4.java

Implementar un algoritmo que dado un grafo *dirigido, no ponderado, débilmente conexo, disperso y acíclico* devuelva el orden topológico.

Ante la posibilidad de varios órdenes topológicos, desarrollar la estrategia que se describe a continuación. Supongamos el siguiente grafo:



Existen tres órdenes, a saber: $\langle 1,2,3,4 \rangle$, $\langle 1,4,2,3 \rangle$ y $\langle 1,2,4,3 \rangle$. Nuestro algoritmo debe retornar el último. La estrategia consiste en las siguientes dos condiciones:

1. Ante la posibilidad de dos o más órdenes, siempre retornar el que contenga más cantidad de nodos “consecutivos en el mismo nivel”. Con esto último queremos decir que tanto $\langle 1,2,4,3 \rangle$ como $\langle 1,4,2,3 \rangle$ mantienen a 2 y 4 consecutivos, mientras que $\langle 1,2,3,4 \rangle$ no. Decimos que 2 y 4 pertenecen al mismo nivel porque se encuentra a la misma cantidad de saltos desde el primer vértice en el orden¹.
2. En caso de que existan varios órdenes *luego de aplicar la condición anterior*, elegir el que sitúe a los nodos consecutivos en orden ascendente. En el ejemplo, el orden final será $\langle 1,2,4,3 \rangle$.

El algoritmo debe ejecutarse en $O(E + V \log V)$ tiempo de ejecución.

Formado de entrada

(Ver [Formato de entrada](#))

¹ Notar que si el grafo tuviera varios vértices iniciales, i.e, de grado de incidencia cero, esto no afectará la determinación de la condición o restricción.

Formato de salida

La salida contendrá V líneas, donde cada línea en la posición i representa al vértice v_i que se encuentra en la posición i del orden topológico.

5. CAMINO + CORTO Nom. de arch.: ejercicio5.cpp/Ejercicio5.java

Dado un grafo *dirigido, ponderado (sin costos negativos) y denso* implementar un algoritmo que devuelve el *costo total* de los caminos más cortos entre todo par de vértices.

Formato de entrada

(Ver [Formato de entrada](#))

Formato de salida

La salida contendrá $V \times V$ líneas, donde cada línea contendrá el costo total C_{ij} de ir desde v_i hasta j , o -1 si no existe tal camino o $i=j$.

Los costos C_{ij} deben aparecer ordenados en el archivo de salida según el orden lexicográfico del par (i,j) , con $1 \leq i \leq N$ y $1 \leq j \leq V$.

$C_{1,1}$ si hay un camino de v_1 a 1; -1 si no o $v_1=1$
 $C_{1,2}$ si hay un camino de v_1 a 2; -1 si no o $v_1=2$
...
 $C_{1,i}$ si hay un camino de v_1 a i ; -1 si no o $v_i=i$
...
 $C_{1,v}$ si hay un camino de v_1 a V ; -1 si no o $v_i=V$
...
 $C_{2,1}$ si hay un camino de v_2 a 1; -1 si no o $v_i=1$
...
 $C_{k,v}$ si hay un camino de v_k a V ; -1 si no o $v_k=V$
...
 $C_{v,v}$ si hay un camino de v_N a V ; -1 si no o $v_N=V$

6. ÁRBOL CUBR. MÍN. Nom. de arch.: ejercicio6.cpp/Ejercicio6.java

Implementar un algoritmo que dado un grafo *no dirigido y ponderado* devuelva el *costo total* del árbol de cubrimiento mínimo.

El algoritmo debe ejecutarse en $O(E \log V)$ tiempo de ejecución. No se exige un algoritmo en particular.

Formato de entrada

(Ver [Formato de entrada](#))

Formato de salida

El archivo de salida debe contener un solo número *sin EOL (fin de línea)* indicando el costo total (la suma del costo de cada arista) del árbol en caso de ser un grafo conexo, -1 en caso contrario (grafo no conexo).

7. MIN PASES Nom. de archivo: ejercicio7.cpp/Ejercicio7.java

Dada una matriz de enteros de tamaño $M \times N$, en la cual cada celda puede contener un valor positivo, negativo o cero, determinar la mínima cantidad de pases necesarios para convertir todo valor negativo a positivo.

En cada pase, las celdas con valores positivos podrán convertir los valores de sus celdas adyacentes, de ser necesario, de valores negativos a positivos.

Es decir, para una celda con posición (i, j) , si la misma tiene valor positivo, podrá convertir los valores de las celdas $(i + 1, j)$, $(i - 1, j)$, $(i, j + 1)$, $(i, j - 1)$.

Las celdas que contienen 0 se considerarán vacías y deberán ser ignoradas. En la matriz recibida siempre se podrá llegar de cualquier celda con número distinto a 0 a cualquier otra con número distinto a 0; el camino no podrá estar completamente bloqueado. A su vez, siempre existirá al menos una celda con número positivo.

Ejemplo:

Matriz parámetro:

-1	-9	0	-17	0
-5	-3	-2	9	-7
2	0	0	-6	0
0	-4	-3	5	4

Ignorando las celdas con valor 0, tenemos la siguiente matriz:

-1	-9		-17	
-5	-3	-2	9	-7
2			-6	
	-4	-3	5	4

En el primer pase, solo los números 2, 9 y 5 son positivos; por lo tanto solo podremos cambiar a positivo las celdas adyacentes a estos:

-1	-9		17	
5	-3	2	9	7
2			6	
	-4	3	5	4

En el segundo pase, el 5, 2 y el 3 tienen celdas adyacentes con valores negativos para cambiar:

1	-9		17	
5	3	2	9	7
2			6	
	4	3	5	4

Finalmente, en el tercer pase, 1 y 3 pueden convertir el valor de -9 a positivo por ser adyacentes al mismo:

1	9		17	
5	3	2	9	7
2			6	
	4	3	5	4

Por tanto, para la matriz recibida, el mínimo número de pases necesarios para convertir todo valor negativo a positivo es 3.

Formato de entrada

```
M
N
K11 K12 ... K1N
K21 K22 ... K2N
...
KM1 KM2 ... KMN
```

La primera línea M indica la cantidad de filas de la matriz. La segunda línea N indica la cantidad de columnas. Las siguientes M líneas corresponden a las filas de la matriz. Por cada fila, habrá N números que corresponden a los elementos pertenecientes a cada columna de dicha fila, separados por un espacio.

Formato de salida

La salida contendrá un solo número sin EOL que corresponderá a la mínima cantidad de pases necesarios para convertir todos los valores negativos a positivos.

8. PT. DE ART. Nom. de archivo: ejercicio8.cpp/Ejercicio8.java

Implementar un algoritmo que, dado un grafo *no dirigido, no ponderado, conexo y disperso*, encuentre todos los puntos de articulación del mismo.

Un punto de articulación es un vértice que, al removerlo (junto con todas las aristas que pasan por él), desconecta el grafo.

Se deberá resolver en $O(V*(E+V))$

Formato de entrada

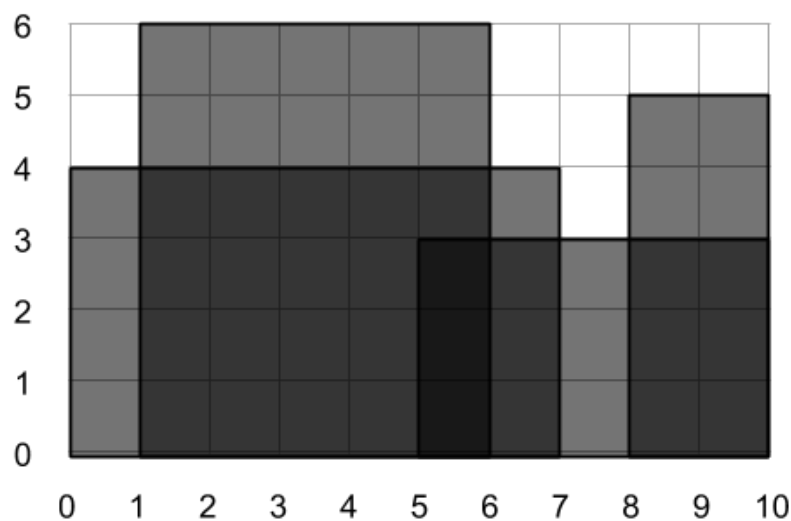
(Ver [Formato de entrada](#))

Formato de salida

La salida contendrá M líneas, donde cada línea contendrá el número del vértice que es un punto de articulación, ordenado de menor a mayor

9. SILUETAS Nom. de archivo: ejercicio9.cpp/Ejercicio9.java

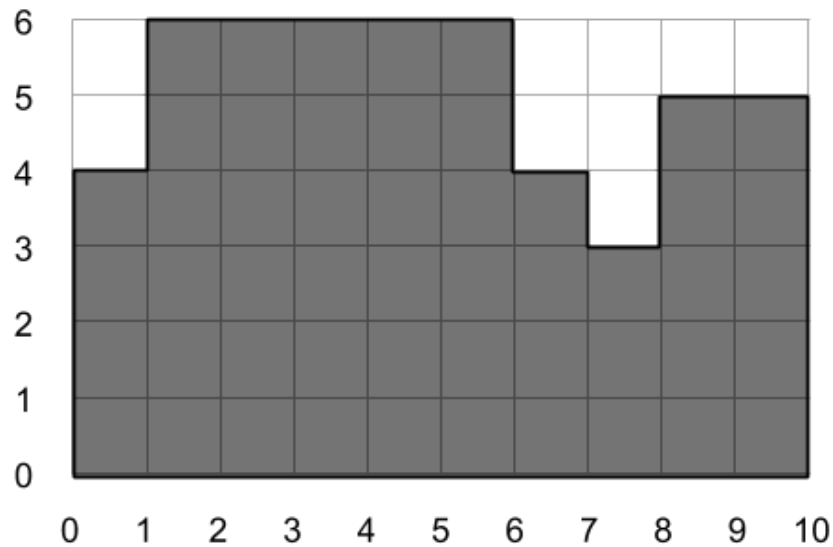
Dada una vista horizontal de una ciudad que consta únicamente de edificios rectangulares, le piden diseñar un algoritmo que dada la ubicación y altura exacta de los edificios de la ciudad, halle la “silueta” o línea de corte de todos los edificios con el cielo, es decir, la silueta de todos los edificios juntos, pero eliminando las intersecciones ocultas entre los mismos. Por ejemplo, sean los siguientes cuatro edificios:



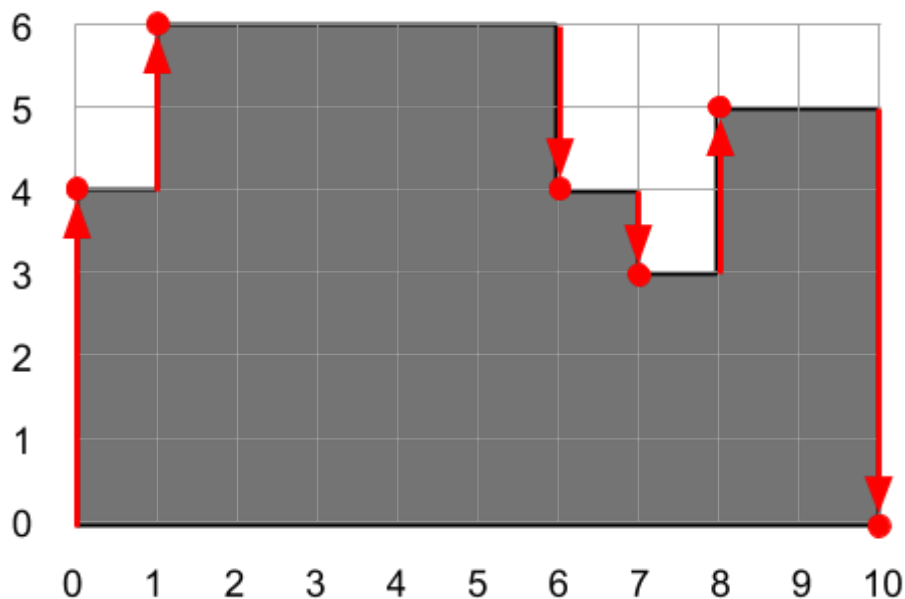
Los mismos se representarán mediante la secuencia de tripletas $[(0,7,4), (5,10,3), (1,6,6), (8,10,5)]$, donde cada componente de la tripleta (I,F,H) representa: I la coordenada en el eje horizontal donde comienza el edificio, F la coordenada donde termina y H la altura del mismo.

Se debe resolver en tiempo $O(N \log N)$ usando la técnica “Dividir para conquistar” o “Divide and conquer”, siendo N la cantidad de edificios.

La solución al problema anterior deberá ser la siguiente:



La solución retornada debe ser una secuencia de *tuplas*. Cada tupla representa una “tira” o vector en el plano dada por el punto (x,h) de destino. El punto de origen quedará determinado por la altura de la tira anterior. Para entenderlo mejor, veamos qué debería retornar el ejemplo anterior. La solución es la siguiente: $[(0,4), (1,6), (6,4), (7,3), (8,5), (10,0)]$.



Inicialmente, convendremos que la silueta comienza con $h_0=0$. La primera tira de la secuencia es $(0,4)$. Esto quiere decir que el vector comienza desde $(0,0)$ y va hasta $(0,4)$. En la segunda tira, tenemos $(1,6)$. Como la altura actual de la silueta $h_1=4$ (punto anterior), ahora la nueva tira va desde $(1,4)$ hasta $(1,6)$. Y así, sucesivamente.

Formato de entrada

N
$I_1 \quad F_1 \quad H_1$
$I_2 \quad F_2 \quad H_2$
...
$I_N \quad F_N \quad H_N$

La primera línea, N, indica la cantidad de edificios que se recibirán. Las siguientes N líneas contiene tres números $I_i \quad F_i \quad H_i$ representando a la tripleta (I_i, F_i, H_i) que describe al edificio i, es decir, siendo I_i su coordenada inicial, F_i la final y H_i su altura.

Formato de salida

La salida contendrá una secuencia de tuplas describiendo la silueta final, cada una separada por EOL.

La secuencia devuelta no puede contener tiras consecutivas con la misma altura, es decir, no puede existir ninguna ocurrencia $[..., (x_i, h), (x_{i+1}, h), ...]$.

10. LA CONSTR.

Nom. de arch.: ejercicio10.cpp/Ejercicio10.java

En un edificio en construcción se desea subir y bajar materiales de un piso a otro. Los materiales son situados en un sistema electrónico de poleas *instalado en un piso*, el cual es controlado por un *operador*, situado en el mismo. En un piso dado puede haber o bien un operador, o bien una *cuadrilla de obreros* que necesitan los materiales, pero *no ambos*. Debido a la alta demanda de materiales por parte de los obreros, cada uno de estos sistemas *solo puede atender a una cuadrilla*.

Dado un edificio de N pisos, con cierta instalación de poleas y organización de obreros, y sabiendo que cada sistema solo puede trasladar materiales K pisos arriba o abajo, se desea saber la máxima cantidad de pisos con cuadrillas que se pueden atender.

Los edificios serán representados con una secuencia de letras O y C, las cuales representan a los operadores y cuadrillas, respectivamente.

Nota: La solución debe ser greedy.

Formato de entrada

Se tiene un archivo el cual contendrá todos los casos de prueba. La primera línea tendrá un número representando la cantidad de casos de prueba, llamémosle P. Las siguientes 3P líneas contendrán a los casos de prueba. Cada caso de prueba consta de tres líneas, en donde: la primera representa el K mencionado previamente en la letra, la segunda N, y finalmente la tercera representa al edificio.

```
P
K1
N1
X1 X2 X3 X4 ... XK1
K2
N2
X1 X2 X3 X4 ... XK2
...
Kp
Np
X1 X2 X3 X4 ... XNp
```

Con $X_i=O$ ó $X_i=C$.

Por ejemplo, el siguiente archivo representa un solo caso, el cual es un edificio de cinco pisos con cierta organización y con máquinas que pueden transportar materiales solo un piso (hacia arriba o abajo).

```
1
1
5
0 C C O C
```

Formato de salida

La salida tendrá P líneas, cada una correspondiendo a cada caso de prueba.

Siguiendo el ejemplo anterior, la misma deberá ser:

```
2
```

Es decir, para el primer y único caso corresponde la primera y también única línea con “2”, indicando que en el edificio anterior la máxima cantidad de piso atendidos es 2.

11. SUMA SIN RESTA Nom. de arch.: ejercicio11.cpp/Ejercicio11.java

Dado un array numérico de largo N , se desea retornar un nuevo array que dada una posición x cualquiera, indique el valor de la suma de todos los elementos del array excepto el de la posición x . Por ejemplo, dado el array $[4, 56, 23, 2, 13, 8]$, se debe retornar $[102, 50, 83, 104, 93, 98]$.

Esto debería de ser muy sencillo, sin embargo tiene 2 restricciones importantes: No se puede restar (ni sumar alternando signos), y la operación debe ser resuelta en tiempo y espacio $O(N)$.

Formato de entrada

```
N
X1
X2
X3
X4
...
XN
```

La primera línea indica el largo del vector y las siguientes N líneas son los elementos.

Formato de salida

La salida contendrá N líneas, correspondiente a cada elemento del vector resultante.

12. LABERINTO

Nom. de archivo: ejercicio12.cpp/Ejercicio12.java

Es el primer día de retorno a clases, y resulta que se ha olvidado de la disposición de la estructura de la facultad. Si visualizamos la facultad como una matriz $M \times N$, con $1 \leq x \leq M$ y $1 \leq y \leq N$, podemos distinguir en los casilleros una letra que puede corresponder a paredes (P), bedelía (B) y pasillo/salón (C). Para saber cuál es el salón, primero debe pasar por bedelías para consultar cuál le corresponde, luego tiene que dirigirse al salón evitando paredes y otros salones.

Luego de subir una serie de pisos se encuentra exhausto, y para caminar lo menos posible, desea buscar el camino óptimo (más corto) para llegar al salón destino habiendo pasado primero por bedelías. Ten en cuenta que sólo se puede mover hacia arriba, abajo, izquierda o derecha (no se permiten movimientos diagonales).

Por ejemplo, dado el siguiente piso representado en una matriz 5×5 con (1, 4) de origen y (5, 4) de destino, la solución se encuentra coloreada en :

P	P	C	P	C
C	C	C	P	P
C	P	C	C	C
C	C	B	P	C
P	C	C	C	C

Formato de entrada

```

M N
L1,1 L1,1 L2,1 ... LM,1
L1,2 L1,2 L2,2 ... LM,2
L1,3 L1,3 L2,3 ... LM,3
...
L1,N L1,N L2,N ... LM,N
P
xi,1 yi,1 xf,1 yf,1
xi,2 yi,2 xf,2 yf,2
...
xi,P yi,P xf,P yf,P

```

Los primeros dos números indican las dimensiones de la matriz. Los siguientes $M \times N$ números representan a la matriz como se indica en la tabla. Luego sigue P indicando la

cantidad de casos de prueba. Por último, siguen 4P números representando las ordenadas y coordenadas de los puntos iniciales y finales, respectivamente, en el orden indicado.

Formato de salida

s_1
 s_2
 s_3
...
 s_p

El archivo de salida deberá contener la cantidad de casilleros que movimientos que tuvo que hacer para llegar al destino (origen y destino inclusive) o 0 si no existe.

RECORDATORIO: IMPORTANTE PARA LA ENTREGA

□ **Obligatorios** (Cap.IV.1, Doc. 220)

La entrega de los obligatorios será en formato digital online, a excepción de algunas materias que se entregarán en Bedelía y en ese caso recibirá información específica en el dictado de la misma.

Los principales aspectos a destacar sobre la **entrega online de obligatorios** son:

1. La entrega se realizará desde gestion.ort.edu.uy
2. Previo a la conformación de grupos cada estudiante deberá estar inscripto a la evaluación.
Sugerimos realizarlo con anticipación.
3. **Uno de los integrantes del grupo de obligatorio será el administrador del mismo** y es quien formará el equipo y subirá la entrega
4. Cada equipo debe entregar **un único archivo en formato zip o rar** (los documentos de texto deben ser pdf, y deben ir dentro del zip o rar)
5. El archivo a subir debe tener **un tamaño máximo de 40mb**
6. Les sugerimos **realicen una 'prueba de subida' al menos un día antes**, donde conformarán el '**grupo de obligatorio**'.
7. La **hora tope para subir el archivo será las 21:00** del día fijado para la entrega.
8. La entrega se podrá realizar desde cualquier lugar (ej. hogar del estudiante, laboratorios de la Universidad, etc)
9. Aquellos de ustedes que presenten alguna dificultad con su inscripción o tengan inconvenientes técnicos, por favor pasar por la oficina del Coordinador o por Coordinación adjunta **antes de las 20:00hs.** del día de la entrega

Si tuvieras una situación particular de fuerza mayor, debes dirigirte con suficiente antelación al plazo de entrega, al Coordinador de Cursos o Secretario Docente.