UNIVERSIDAD ORT
Uruguay

Facultad de Ingeniería
Bernard Wand-Polak
Cuareim 1451
11.100 Montevideo, Uruguay
Tel. 2902 15 05 Fax 2908 13 70
www.ort.edu.uy

EVALUACIÓN	Obligatorio	**GRUPO**	Todos	**FECHA**	15/9
MATERIA	Estructura de Datos y Algoritmos 2				
CARRERA	Ingeniería en Sistemas				
CONDICIONES	- Puntos: Máximo: 30 Mínimo: 1 - Fecha máxima de entrega: 03/12 LA ENTREGA SE REALIZA EN FORMA ONLINE EN ARCHIVO NO MAYOR A 40MB EN FORMATO ZIP, RAR O PDF. **IMPORTANTE:** - Inscribirse - Formar grupos de hasta dos personas. - Subir el trabajo a Gestión antes de la hora indicada, ver hoja al final del documento: “RECORDATORIO”				

Facultad de Ingeniería
Bernard Wand-Polak
Cuareim 1451
11.100 Montevideo, Uruguay
Tel. 2902 15 05 Fax 2908 13 70
www.ort.edu.uy

Obligatorio

El obligatorio de Estructuras de Datos y Algoritmos 2, para el semestre 2 de 2020, está compuesto por un conjunto de **13** 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
...
└─── ejercicio13.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.

Facultad de Ingeniería
Bernard Wand-Polak
Cuareim 1451
11.100 Montevideo, Uruguay
Tel. 2902 15 05 Fax 2908 13 70
www.ort.edu.uy

- Se utilizará MOSS para detectar copias.

La defensa:

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

UNIVERSIDAD ORT
Uruguay

Facultad de Ingeniería
Bernard Wand-Polak
Cuareim 1451
11.100 Montevideo, Uruguay
Tel. 2902 15 05 Fax 2908 13 70
www.ort.edu.uy

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
n1
n2
...
nN
```

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.

Facultad de Ingeniería
Bernard Wand-Polak
Cuareim 1451
11.100 Montevideo, Uruguay
Tel. 2902 15 05 Fax 2908 13 70
www.ort.edu.uy

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
palabra_1
palabra_2
...
palabra_N
M
palabra_test_1
palabra_test_2
...
palabra_test_M
```

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 producente, 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 $\texttt{palabra\_test}_i$ pertenece al diccionario (`1`) o no (`0`).

Facultad de Ingeniería
Bernard Wand-Polak
Cuareim 1451
11.100 Montevideo, Uruguay
Tel. 2902 15 05 Fax 2908 13 70
www.ort.edu.uy

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_i 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.

Facultad de Ingeniería
Bernard Wand-Polak
Cuareim 1451
11.100 Montevideo, Uruguay
Tel. 2902 15 05 Fax 2908 13 70
www.ort.edu.uy

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: $\{(1,2), (2,1)\}$.

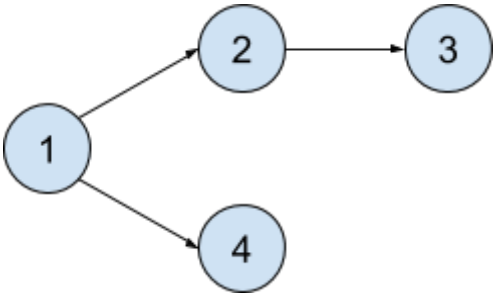
Facultad de Ingeniería
Bernard Wand-Polak
Cuareim 1451
11.100 Montevideo, Uruguay
Tel. 2902 15 05 Fax 2908 13 70
www.ort.edu.uy

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: <1,2,3,4>, <1,4,2,3> y <1,2,4,3>. 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 <1,2,4,3> como <1,4,2,3> mantienen a 2 y 4 consecutivos, mientras que <1,2,3,4> no. Decimos que 2 y 4 pertenecen la mismo nivel porque se encuentra a la misma cantidad de saltos desde el primer vértice en el orden[1].
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 <1,2,4,3>.

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.

Facultad de Ingeniería
Bernard Wand-Polak
Cuareim 1451
11.100 Montevideo, Uruguay
Tel. 2902 15 05 Fax 2908 13 70
www.ort.edu.uy

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.

Facultad de Ingeniería
Bernard Wand-Polak
Cuareim 1451
11.100 Montevideo, Uruguay
Tel. 2902 15 05 Fax 2908 13 70
www.ort.edu.uy

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 $V*V$ líneas, donde cada línea contendrá el costo total $C_{i,j}$ de ir desde v_i hasta j, o -1 si no existe tal camino o $i=j$.
Los costos $C_{i,j}$ 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 $v_N=V$

Facultad de Ingeniería
Bernard Wand-Polak
Cuareim 1451
11.100 Montevideo, Uruguay
Tel. 2902 15 05 Fax 2908 13 70
www.ort.edu.uy

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).

Facultad de Ingeniería
Bernard Wand-Polak
Cuareim 1451
11.100 Montevideo, Uruguay
Tel. 2902 15 05 Fax 2908 13 70
www.ort.edu.uy

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

Dada una matriz de enteros de tamaño MxN, 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

Facultad de Ingeniería
Bernard Wand-Polak
Cuareim 1451
11.100 Montevideo, Uruguay
Tel. 2902 15 05 Fax 2908 13 70
www.ort.edu.uy

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.

Facultad de Ingeniería
Bernard Wand-Polak
Cuareim 1451
11.100 Montevideo, Uruguay
Tel. 2902 15 05 Fax 2908 13 70
www.ort.edu.uy

Formato de entrada

M
N
K_{11} K_{12} ... K_{1N}
K_{21} K_{22} ... K_{2N}
...
K_{M1} K_{M2} ... K_{MN}

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.

Facultad de Ingeniería
Bernard Wand-Polak
Cuareim 1451
11.100 Montevideo, Uruguay
Tel. 2902 15 05 Fax 2908 13 70
www.ort.edu.uy

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

Facultad de Ingeniería
Bernard Wand-Polak
Cuareim 1451
11.100 Montevideo, Uruguay
Tel. 2902 15 05 Fax 2908 13 70
www.ort.edu.uy

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. S**ugerimos 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.