

<b>EVALUACIÓN</b>	Obligatorio	<b>GRUPO</b>	Todos	<b>FECHA</b>	30/8
<b>MATERIA</b>	Estructura de Datos y Algoritmos 2				
<b>CARRERA</b>	Ingeniería en Sistemas				
<b>CONDICIONES</b>	<p>- Puntos: Máximo: 30    Mínimo: 1</p> <p>LA ENTREGA SE REALIZA EN FORMA ONLINE EN ARCHIVO NO MAYOR A 40MB EN FORMATO ZIP, RAR O PDF.</p> <p><b>IMPORTANTE:</b></p> <ul style="list-style-type: none"><li>- Inscribirse</li><li>- Formar grupos de hasta dos personas.</li><li>- Subir el trabajo a Gestión antes de la hora indicada, ver hoja al final del documento: "RECORDATORIO"</li></ul>				

## Obligatorio

El obligatorio de Estructuras de Datos y Algoritmos 2, para el semestre 2 de 2022, está compuesto por un conjunto de **10** 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 ejecución trabajado durante el curso:  
`miSolucion < input > output`
- 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.
- **De no cumplir las condiciones anteriores el ejercicio se considera invalido, por lo tanto 0 puntos.**

La entrega:

- El formato de entrega se encuentra definido en [este enlace](#).

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.
- Los códigos dados en clase como algoritmos encontrados de otras fuentes serán usados como referencia. **Considerándose copia el uso de ellos.**
- **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. Emails

Nombre de archivo: ejercicio1.cpp/Ejercicio1.java

### Letra

Se tiene una base de datos de usuario identificados por e-mail pero se empezó a notar de que existen usuarios duplicados (comparten el mismo e-mail) por lo cual se quiere calcular cuantos usuarios reales existen.

### Input

La primera línea de la entrada contiene un número entero positivo  $N$  ( $2 \leq N \leq 10^5$ ), el número de usuarios actuales del sistema.

Las siguientes  $N$  líneas contienen los email actuales del sistema

### Salida

Imprime el único número entero: la cantidad real de usuarios (sin repetidos).

### Restricciones

- Utilizar una tabla de **hash cerrado**.
- Resolver en orden temporal:  $O(N)$  promedio, siendo  $N$  la cantidad de usuarios.

### Ejemplos de entrada y salida

#### Entrada

```
5
abc@abc.com
cba@cba.com
abc@abc.com
abc@abc.com
aaa@aaa.com
```

#### Salida

```
3
```

## 2. SORTING      Nombre de archivo: ejercicio2.cpp/Ejercicio2.java

### Letra

Se desea ordenar un conjunto de números enteros, no acotados. Se solicita que implemente un algoritmo basado en el uso de un heap, en particular mediante el uso de la técnica de “heapificación” (**heapify** en inglés).

### Input

La primera línea de la entrada contiene un número entero positivo  $N$  ( $2 \leq N \leq 10^5$ ), el largo del conjunto de números.

Las siguientes  $N$  líneas contienen los números del conjunto.

### Salida

Imprime  $N$  líneas con los números ordenados de menor a mayor

### Restricciones

- Utilizar la técnica **heapify**.
- Resolver en orden temporal:  $O(N \log N)$  promedio, siendo el tamaño del conjunto.
- Resolver en orden espacial auxiliar (sin contar el array de números):  $O(1)$

### Ejemplos de entrada y salida

#### Entrada

```
5
10
4
7
42
3
```

#### Salida

```
3
4
```

7 10 42
---------------

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

### 3. ACM-E

Nombre de archivo: ejercicio3.cpp/Ejercicio3.java

#### Letra

Dado un grafo conexo, no dirigido y disperso, se desea obtener un árbol de cubrimiento mínimo del mismo, el cual debe excluir a ciertos vértices indicados.

#### Input

Inicialmente, respeta el [Formato de Grafos](#). Luego, posee una línea con un entero indicando la cantidad de vértices a ignorar, seguido de dichos vértices.

#### Salida

Línea indicando la cantidad de aristas, seguido por una línea por cada arista.  
Por cada arista se imprime de forma que el vertice de menor número vaya primero.  
El orden de toda la lista esta dado por número de vertice.

#### Restricciones

Resolver en orden temporal:  $O(E \log V)$  promedio, siendo E la cantidad de aristas y V la cantidad de vértices.

#### Ejemplos de entrada y salida

##### Entrada

```
4
6
1 2 10
1 3 200
1 4 1891
2 3 40
2 4 50
3 4 60
1
2
```

##### Salida

```
2
```



1 3 200  
3 4 60

## 4. Triconexo

**Nombre de archivo:** ejercicio4.cpp/Ejercicio4.java

### Letra

Dado un grafo no dirigido, indicar si el mismo es triconexo. Un grafo es triconexo cuando todo par de vértices están conectados por al menos tres caminos disjuntos. Dicho de otra forma, un grafo es triconexo cuando luego de eliminar dos vértices cualesquiera, el grafo continúa siendo conexo.

### Input

[Formato de Grafos](#)

### Salida

1 en caso que el grafo sea triconexo. 0 en caso contrario.

### Restricciones

Resolver en orden temporal:  $O(V^2 * (V + E))$  peor caso, siendo E la cantidad de aristas y V la cantidad de vértices.

### Ejemplos de entrada y salida

#### Entrada

4  
6  
1 2  
1 3  
1 4  
2 3  
2 4  
3 4

Salida

1
---

## 5. Del más al menos

**Nombre de archivo:** ejercicio5.cpp/Ejercicio5.java

### Letra

Dado un grafo dirigido, retornar un listado con sus vértices ordenados decrecientemente por grado de incidencias. En el listado, deberá incluirse el número de vértice seguido por su cantidad de aristas incidentes. En caso de haber dos o más vértices que tengan igual grado de incidencia, deberán ordenarse decrecientemente por número.

### Input

[Formato de Grafos](#)

### Salida

V líneas de formato <vértice> <grado de incidencia> (sin los símbolos < y >).

### Restricciones

Resolver en orden temporal:  $O(V \log V)$  peor caso, siendo V la cantidad de vértices (sin tener en cuenta la lectura de datos).

### Ejemplos de entrada y salida

#### Entrada

```
4
5
1 2
1 3
2 1
4 2
4 3
```

#### Salida

```
3 2
2 2
1 1
```

4 0

## 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](http://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.