Guía Grafos

- Subidos junto a este documento están una implementación base de grafo y tests de unidad básicos para cada ejercicio (pueden escribir tests propios para obtener mayor confianza de haber llegado al resultado correcto).

Ejercicio 1

Agregar a la implementación de grafos adjunta los métodos:

- `void removeNode(String label)` dado el *label* de un nodo lo elimina del grafo
- `void removeEdge(String label1, String label2)` dados los *label* de dos nodos, elimina el eje entre sí.

Ejercicio 2

Agregar a la implementación de grafos adjunta los métodos:

- `int inDegree(String label)` devuelve el grado de entrada de un nodo.
- `int outDegree(String label)` devuelve el grado de salida de un nodo.

Para un digrafo, el grado de entrada de uno de sus nodos es la cantidad de ejes que "apuntan" hacia ese nodo y el de salida la cantidad de ejes que tienen su origen en el nodo.

Ejercicio 3

Agregar a la implementación de grafos adjunta el método:

```
int connectedComponents()
```

que devuelve la cantidad de componentes conexas del grafo.

Ejercicio 4

Dado un grafo simple (no dirigido, sin peso) y 2 nodos que le pertenecen, devolver una lista con todos los posibles caminos entre ellos sin usar ciclos.

Ejercicio 5

a) Dada una matriz de booleanos que representa un sector del océano donde *true* significa que hay tierra y *false* agua, contar la cantidad de islas que hay. Para que dos celdas con tierra se consideren parte de la misma isla, deben tener un camino entre si

moviéndose solo vertical y horizontalmente por la matriz y pasando solo por celdas con tierra.

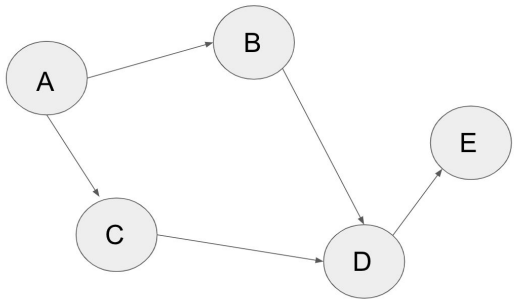
b) Dada una matriz de igual características a la del punto *a*, devolver el tamaño de la isla más grande. El tamaño está dado por la cantidad de celdas que la componen.

Ejercicio 6

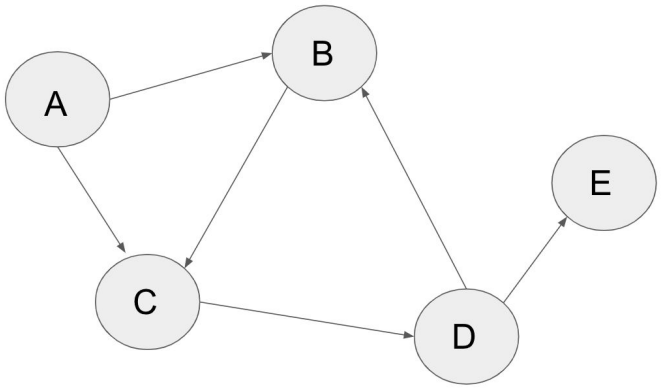
Dado un número N, la cantidad total de materias y las correlatividades entre ellas, determinar si es posible completarlas todas (devolver *true* si es posible y *false* si no lo es).

Se recibe un número N, la cantidad total de materias, y un listado de pares de números [*a, b*] que indica que para poder cursar *b* se debe haber aprobado *a*. Las materias están identificadas por un número que va de 1 a N por lo que *a* y *b* son números de 1 a N.

Por ejemplo, para el siguiente grafo se debe devolver *true* ya que se pueden cursar todas las materias sin problemas:



Para el siguiente grafo sin embargo se debe devolver *false* ya que para poder cursar B se debe haber aprobado D y para cursar D se debe haber aprobado C pero para cursar C se debe haber aprobado B, por lo que tenemos una situación imposible.



Ejercicio 7

Implementar DFS usando una matriz de adyacencia.

Ejercicio 8

Un sistema tiene N servidores, numerados de 1 a N. Se tiene una serie de mediciones de la cantidad de tiempo que tarda una señal en llegar desde el servidor *i* hasta el servidor *j* (no funciona en sentido contrario).

Si se envía una señal desde el servidor *k*, ¿cuanto tiempo tarda la señal en llegar hasta todos los servidores?

Ejercicio 9 (difícil)

Un país tiene N ciudades numeradas de 1 a N. La ciudad 1 es la capital del país. El sistema de transporte de ese país consiste en:

- Rutas entre la ciudad *i* y la ciudad *j* que tienen una distancia *d.*
- Trenes entre la capital (ciudad 1) y la ciudad *k* con distancia *t.*

El ministro de transporte quiere cerrar algunos trenes para ahorrar dinero, pero no quiere alargar el camino más corto entre la capital y ninguna otra ciudad. **¿Cuantos trenes puede cerrar sin hacerlo?**

Nota: Pueden haber varios caminos entre 2 ciudades y se garantiza que hay forma de llegar a todas las ciudades partiendo desde la capital.

Por ejemplo:

- 3 ciudades (1 es la capital)
- Caminos:
    - 1 2 3 *(desde ciudad 1 a ciudad 2 con distancia 3)*
    - 2 3 2 *(desde 2 a 3 con distancia 2)*
- Trenes:
    - 2 4 *(desde ciudad 1 hasta la 2 con distancia 4)*
    - 3 4 *(desde ciudad 1 hasta la 3 con distancia 4)*

En este caso se puede cerrar un único tren. El camino más rápido desde la ciudad 1 hasta la 2 es usando el primer camino y se llega con costo 3, por lo que el primer tren es innecesario. Para llegar a 3 sin embargo si se necesita el segundo tren con costo 4 ya que el camino sin trenes tiene costo 5.

Otros ejercicios con Dijkstra:
https://codeforces.com/contest/715/problem/B
https://codeforces.com/contest/59/problem/E
https://codeforces.com/contest/567/problem/E
https://www.hackerrank.com/challenges/synchronous-shopping/problem