PRÁCTICA 3
ALGORÍTMICA

Integrantes:

Raúl Granados López
Hossam El Amraoui Leghzali
Javier Montaño Rubio

1. ***Formalización del funcionamiento del algoritmo de Fleury como algoritmo Greedy***

Hemos entendido que este problema si se puede resolver mediante Greedy. Ya que se cumplen todos los pasos y requisitos para realizar el diseño de componentes.

Diseño de componentes Greedy

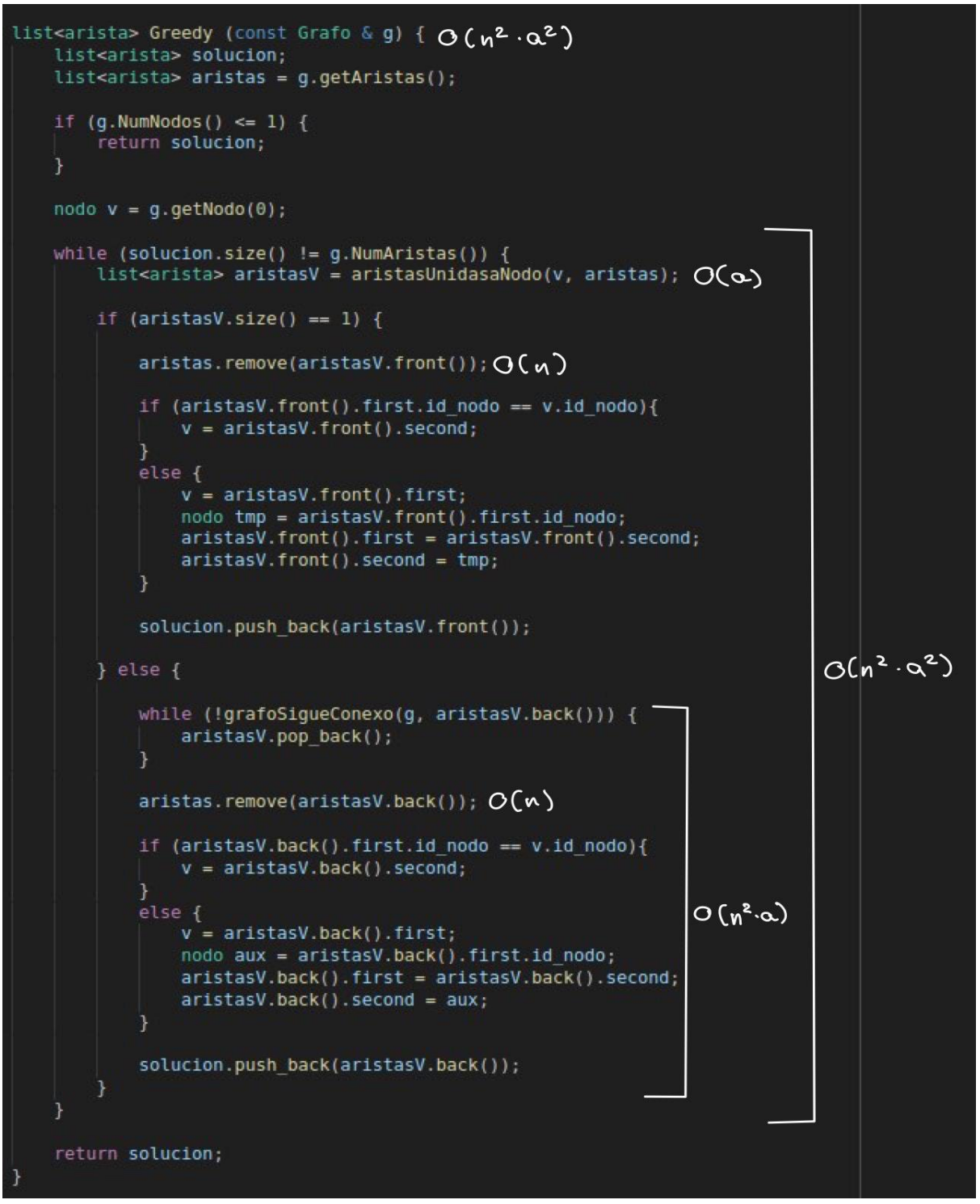
- **Lista de candidatos:** Las aristas del grafo.
- **Lista de candidatos utilizados:** Las aristas que se han ido seleccionando desde el grafo original.
- **Función solución:** Se encuentra un camino que pasa por todas las aristas y no repite ninguna.
- **Función de selección:** Si hay solo una arista se escoge esa. En caso contrario, se escoge cualquier arista que permita que el grafo siga siendo conexo.
- **Criterio de factibilidad:** Sigue habiendo aristas sin explorar.
- **F. Objetivo:** Minimizar el número de pasos por cada arista antes de pasar por todas las aristas del grafo.

Diseño del algoritmo Greedy

ALGORITMO T = Greedy (grafo G=(V,A))

```
T = Ø  // Solución a crear
Si (|V| == 1), hacer:
        Devolver T
Fin-Si
N = Número de aristas en A
v = nodo cualquiera de V
Mientras (|T| != N), hacer:
        Buscar en A todas las aristas que unen v con otros nodos
        Si solo existe una arista a=(b,v) donde b es un nodo unido a v mediante a, hacer:
                T = T U {a}
                A = A \ {a}
                v = b
        En caso contrario, hacer:
                Escoger una arista a=(b,v) donde b es un nodo unido a v mediante a y que mantiene el grafo conexo
                T = T U {a}
                A = A \ {a}
                v = b
        Fin-Si
Fin-Mientras
Devolver T
```

2. *Implementación y eficiencia del algoritmo en C++*



Esta es la función principal. Es, básicamente, una implementación del diseño mostrado en el primer ejercicio haciendo uso de las funciones que se van a explicar a continuación. En cuanto a la eficiencia, en las imágenes se va mostrando el análisis de esta, llegando a la conclusión de que la eficiencia total es de orden **O(n^2 * a^2)**, siendo **n** el número de nodos y **a** el número de aristas.

```
list<arista> aristasUnidasaNodo (nodo &n, list<arista> &aristas) { O(a)
    list<arista> solucion;

    for (arista i : aristas) { nº veces = nº aristas → O(a)
        if (i.first.id_nodo == n.id_nodo || i.second.id_nodo == n.id_nodo) {
            solucion.push_back(i);
        }
    }

    return solucion;
}
```

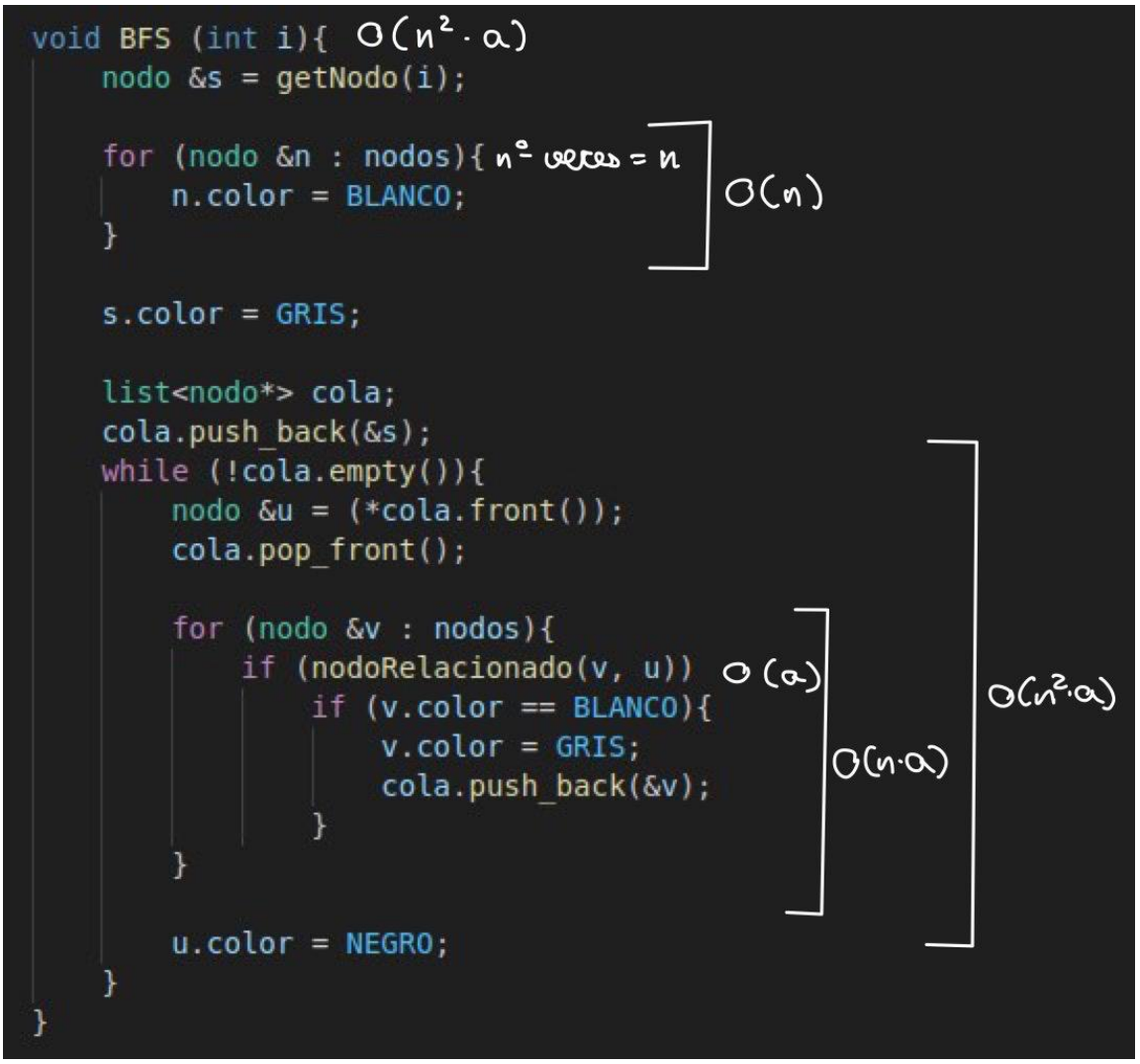
Esta función determina todas las aristas que están unidas a un determinado nodo comprobando si el nodo forma parte de todas y cada una de las aristas que se le pasan como argumento, añadiendo a la lista solución las aristas de las que el nodo sí forma parte.

```
bool grafoSigueConexo (const Grafo &g, arista &arist){  O(n² · a)
    Grafo aux;
    aux.setNodos(g.getNodos());
    aux.setAristas(g.getAristas());

    aux.getAristas().remove(arist); O(n)
    aux.BFS(0); O(n² · a)

    for (nodo &n : aux.getNodos()){ O(n)
        if (n.color != NEGRO) return false;
    }

    return true;
}
```

Esta función indica si un grafo pasado como argumento sigue siendo conexo si le quitamos una arista que también pasamos como argumento. Crea un grafo auxiliar igual al grafo del argumento y le quita la arista especificada. Luego le aplica la función de búsqueda en anchura que se explica al final de este ejercicio. Finalmente, comprueba el color de cada nodo. Si algún nodo no es de color negro (es decir, no ha sido explorado), significará que el grafo no es conexo ya que el nodo no ha podido ser explorado por no estar unido a los demás nodos mediante una arista. En caso contrario, significa que el grafo seguirá siendo conexo aún quitándole la arista.

```
bool nodoRelacionado (const nodo &n1, const nodo &n2){ O(a)
    for (arista a : aristas){ nº veces = a → O(a)
        if (a.first.id_nodo == n1.id_nodo && a.second.id_nodo == n2.id_nodo)
            return true;
        else if (a.first.id_nodo == n2.id_nodo && a.second.id_nodo == n1.id_nodo)
            return true;
    }
    return false;
}
```

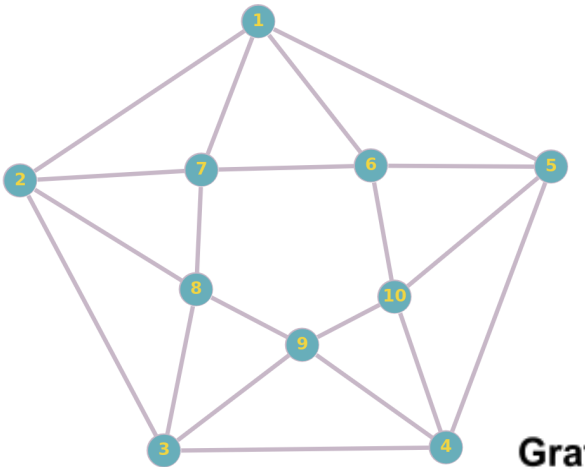
Esta función simplemente indica si un nodo está unido a otro revisando si ambos nodos forman parte de alguna de todas las aristas disponibles en el grafo.




```
void BFS (int i){
    nodo &s = getNodo(i);

    for (nodo &n : nodos){
        n.color = BLANCO;
    }

    s.color = GRIS;

    list<nodo*> cola;
    cola.push_back(&s);
    while (!cola.empty()){
        nodo &u = (*cola.front());
        cola.pop_front();

        for (nodo &v : nodos){
            if (nodoRelacionado(v, u))
                if (v.color == BLANCO){
                    v.color = GRIS;
                    cola.push_back(&v);
                }
        }

        u.color = NEGRO;
    }
}
```

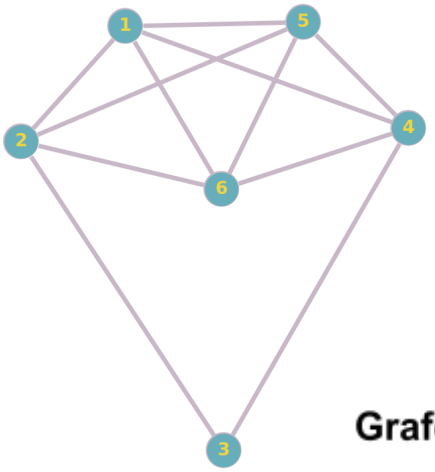
Esta función pertenece a la clase Grafo y se encarga de recorrer el grafo usando la búsqueda en anchura. Su modo de funcionamiento consiste en que le asigna un color a cada nodo dependiendo de si aún no ha sido explorado, está siendo explorado o ya ha sido explorado.
Al principio le asigna el color blanco a todos los nodos y luego le asigna el color gris a un nodo (en este caso aleatorio, elegimos el 0 por simplicidad) que se le pasa como argumento. Un nodo está siendo explorado, es decir, en color gris cuando se están comprobando sus hijos (en este caso, los nodos relacionados con este, que determinamos con la función explicada anteriormente). Añade el nodo anterior a una cola, explora sus hijos, los mete en la cola, saca el nodo que estaba siendo explorado de la cola y le asigna el color negro. El algoritmo va haciendo esto con todos los nodos que se van introduciendo en la cola hasta que no haya más nodos relacionados.

3. *Ejemplos de grafos utilizados y ejecución del programa*



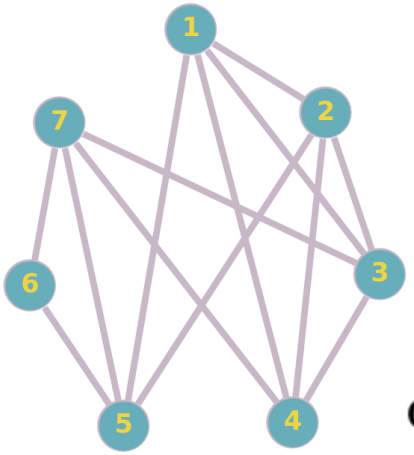
Grafo 1

```
hossam@DESKTOP-FOF3I96:~/ALG/Practicas-ALG/Practica3$ practica3 grafo1.txt
1 5
5 6
6 7
7 8
8 9
9 10
10 6
6 1
1 7
7 2
2 3
3 4
4 5
5 10
10 4
4 9
9 3
3 8
8 2
2 1
```



Grafo 2

```
hossam@DESKTOP-FOF3I96:~/ALG/Practicas-ALG/Practica3$ practica3 grafo2.txt
1 6
6 5
5 4
4 6
6 2
2 5
5 1
1 4
4 3
3 2
2 1
```



Grafo 3

```
hossam@DESKTOP-FOF3I96:~/ALG/Practicas-ALG/Practica3$ practica3 grafo3.txt
1 5
5 7
7 3
3 4
4 7
7 6
6 5
5 2
2 4
4 1
1 3
3 2
2 1
```