Mario Casas Pérez, Ángela Maria Garrido Ruiz, Rocío Barragán Moreno, y Marina Ruiz Palomino 2º D - GII

Práctica 3: Algoritmos Greedy

1. Formalizar, si es posible, la descripción de funcionamiento del método anterior como un algoritmo Greedy. Para ello:

- **Compruebe si se puede resolver mediante Greedy.**

Los algoritmos Greedy se caracterizan por:

- Construir la solución por pasos
- Seleccionar un movimiento adecuado según un criterio de selección
- No considerar movimientos ya realizados
- Tener una función objetivo

Vamos a ver si podemos obtener estas características en nuestro problema a resolver.

a) Demostración de la construcción por pasos → Partimos de un nodo aleatorio del grafo (v), del cual seleccionamos una arista de las que forman el grado del vértice, para ir a otro nodo (w). Una vez estamos en w, lo tomamos como el nodo v, es decir, el actual, borramos la arista por la que hemos pasado y continuamos con el proceso.
b) Criterio de selección → Seleccionamos aquella arista que nos lleva al nodo que tenga mayor grado de entre los posibles para conservar la conectividad del grafo.
c) Consideración de movimientos realizados → No consideramos movimientos anteriores ya que vamos eliminando las aristas por las que pasamos y actualizando el nodo actual.
d) Función objetivo → El objetivo es pasar por todas las aristas del grafo sin repetirlas.

Como podemos ver nuestro problema se puede resolver mediante el método Greedy.

- **Diseñe las componentes Greedy del algoritmo.**

Las componentes de un algoritmo Greedy son:

1. Lista de candidatos → Las aristas adyacentes que salen de los nodos de un grafo.
2. Listas de candidatos ya utilizados → Las aristas que eliminamos de la lista de candidatos cuando pasamos por ellas.
3. Función solución → Aristas por las que se pasa para formar el camino de Euler
4. Criterio de factibilidad → La arista seleccionada mantiene la característica del grafo de que sea conexo.

5. Función de selección →Seleccionamos aquella arista que nos lleva al nodo que tenga mayor grado de entre los posibles para conservar la conectividad del grafo.
6. Función objetivo → Pasar por todas las aristas del grafo sin repetirlas.

- **Adapte la plantilla de diseño Greedy a las componentes propuestas.**

S = ∅ ;   // Solución final
C = Conjunto de aristas candidatas ;
Mientras (C != ∅)
hacer:
    x = Selección del candidato de C ;
    C = C \ {x} ;   // Quito la arista del grafo
    Si es factible (S U {x}) entonces
        S = S U {x} ;
Fin - Mientras

Si S es solución
    Devolver S ;
Si no
    Devolver "No hay solución" ;

2. Implemente el algoritmo en una función C/C++.

Para implementar el algoritmo hemos desarrollado dos funciones auxiliares, Factible y SeleccionarArista

- Factible → devuelve un bool diciendo si quitando esa arista se sigue cumpliendo la condición de que sea conexo.

```
bool Factible(vector<Arista> ar) // funcion auxiliar que me dice que un grafo sigue siendo conexo quitando esa arista o no
{
    if (ar.size() == 0)
    {
        return true;
    }

    MatrizA nuevoG(ar); // creo un nuevo grafo con las conexiones que me pasan

    nuevoG.imprimirMatriz();

    if (nuevoG.tam() <= 1)
    {
        return true;
    }

    for (int i = 0; i < nuevoG.tam(); i++) // miro si alguna fila de mi matriz esta completa con 0s
    {
        vector<int> fila = nuevoG.obtenerFila(i); // obtengo una fila de la matriz
        bool f = false;
        for (int j = 0; j < fila.size() and f == false; j++) // recorro esa fila, si el bool se pone a true paro de recorrer ya que se que sera factible
        {
            if (fila[j] == 1) // si el valor es 1 pongo el bool a true
            {
                f = true;
            }
        }
        if (f == false)
        {
            return false;
        }
    }
    return true;
}
```

- SeleccionarArista → dada la nueva matriz tras quitar una arista selecciona la que me lleva al nodo con mayor grado de entre los que quedan

```
Arista SeleccionArista(vector<Arista> ar, Nodo v)
{
    int max_gr = 0;
    Arista resultado;
    // Miro con que nodos se conecta v y escojo el que mayor grado tenga
    for (int i = 0; i < ar.size(); i++)
    {
        if (ar[i].getEnlace().first.getNumNodo() == v.getNumNodo())
        {                                               // encuento la conexion v con el nodo n
            max_gr = ar[i].getEnlace().second.getGrado(); // obtengo el grado del nodo con el que se relaciona
            resultado = ar[i];                          // lo selecciono como la solucion provisional
        }
    }
    // Busco si hay mas conexiones con mi nodo v
    for (int j = 0; j < ar.size(); j++)
    {
        if (ar[j].getEnlace().first.getNumNodo() == v.getNumNodo() and max_gr < ar[j].getEnlace().second.getGrado())
        { // si el grado de la nueva coincidencia es mayor que el de la anterior
            max_gr = ar[j].getEnlace().second.getGrado();
            resultado = ar[j]; // cambio de arista seleccionada
        }
    }

    return resultado;
}
```

Con estos elementos hemos podido desarrollar la función central de nuestro problema que es la que se muestra a continuación y que se adapta a la plantilla descrita en el ejercicio anterior:

```
vector<Arista> caminosEulerGreedy(MatrizA g, Nodo v, vector<Arista> a)
{

    g.imprimirMatriz();
    vector<Arista> solucion;
    vector<Arista> candidatas = a;

    // Caso base -> el grafo solo contiene un nodo
    if (a.size() == 0)
    {                    // si solo tengo un nodo el vector de aristas sera de tamaño 0 ya que no tengo ninguna conexion
        return solucion; // devuelvo un camino vacío
    }

    while (candidatas.size() > 0) // mientas que tenga aristas por las que pasar y el nodo no sea la solucion
    {
        // selecciono la arista que me lleve desde v al nodo con mayor grado
        Arista x = SeleccionArista(candidatas, v);

        pair<Nodo, Nodo> n = x.getEnlace();
        // Quito la arista seleccionada de la lista de candidatos
        for (auto it = candidatas.begin(); it != candidatas.end();) // recorro el vector de candidatos
        {
            if ((*it).getEnlace().first.getNumNodo() == x.getEnlace().first.getNumNodo() and (*it).getEnlace().second.getNumNodo() == x.getEnlace().second.getNumNodo())
            {                         // cuando encuentro la que he seleccionado
                candidatas.erase(it); // borro lo que hay en esa posicion del vector
            }
            // Borro el nodo simetrico
            else if ((*it).getEnlace().second.getNumNodo() == x.getEnlace().first.getNumNodo() and (*it).getEnlace().first.getNumNodo() == x.getEnlace().second.getNumNodo())
            {                         // cuando encuentro la que he seleccionado
                candidatas.erase(it); // borro lo que hay en esa posicion del vector
            }
            else if (it != candidatas.end())
            {
                it++;
            }
        }
        // Miro si es factible la arista seleccionada, será factible si quitandola se mantiene que el grafo es conexo
        if (Factible(candidatas))
        {
            solucion.push_back(x);    // la añado al vector solucion
            v = x.getEnlace().second; // actucalizacion de la v
        }
    }

    // Si al finalizar mi conjunto de aristas solucion esta relleno lo devuelvo
    if (solucion.size() > 0){
        return solucion;
    }
}
```

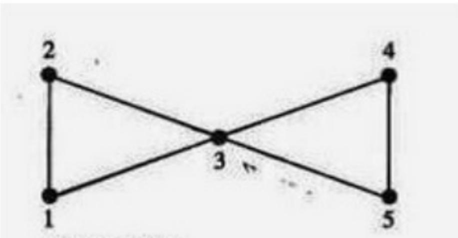
3. Proponga dos ejemplos de grafos de Euler que se puedan leer desde fichero, y ejecute el programa implementado con estos dos ejemplos.

La estructura de datos elegida para la representación de los grafos ha sido la matriz de adyacencia que está rellena de 0s y 1s. Los 1s representan los enlaces entre los nodos del grafo.

Basándonos en esos unos de la matriz hemos creado un vector<Aristas> que guardan las relaciones entre los nodos con un pair<Nodo,Nodo>

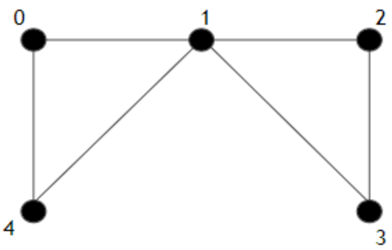
Aristas y Nodo son objetos de clases, de las que se puede ver la definición en nuestro archivo cpp.

A continuación se muestran los grafos elegidos y el fichero correspondiente. Además mostramos la salida tras su ejecución:




```
0 1 1 0 0
1 0 1 0 0
1 1 0 1 1
0 0 1 0 1
0 0 1 1 0
```

```
angela@angela:~/Escritorio/ALG/P3$ ./f3 matrizGrafoDosTriangulos.txt 5
Resultado:
0->2
2->4
4->3
3->2
2->1
1->0
```




```
0 1 0 0 1
1 0 1 1 1
0 1 0 1 0
0 1 1 0 0
1 1 0 0 0
```

```
angela@angela:~/Escritorio/ALG/P3$ ./p3 grafoEjemplo2.txt 5
Resultado:
0->4
4->1
1->3
3->2
2->1
1->0
```

4. Calcule la eficiencia en el caso peor del algoritmo.

Para sacar la eficiencia de nuestro algoritmo tenemos que hallar la eficiencia de las funciones auxiliares básicas:

- Factible

EFICIENCIA FACTIBLE

→ Primer bucle anidado

$i(n) = O(1)$
$g(n) = O(1)$
$h(n) = O(n)$
$f(n) = O(n)$
$a(n) = O(1)$

$O(1+1+n(1+n+1)) = O(2+n^2+2n) = O(n^2)$

→ Segundo bucle anidado

$i(n) = O(1)$
$g(n) = O(1)$
$h(n) = O(n)$
$f(n) = O(1)$
$a(n) = O(1)$

$O(1+1+n(1+1+1)) = O(2+3n) = O(n)$

Por tanto, la eficiencia total de la función es $\max(O(n^2), O(n))$

$\boxed{\text{EFICIENCIA TOTAL DEL MÉTODO FACTIBLE} \Rightarrow O(n^2)}$

- SeleccionarArista

EFICIENCIA SELECCIÓN ARISTA

→ Primer bucle

$i(n) = O(1)$
$g(n) = O(1)$
$h(n) = O(n)$
$f(n) = O(1)$
$a(n) = O(1)$

$O(1+1+n(1+1+1)) = O(2+3n) = O(n)$

Es importante destacar que todos los "get" que se encuentran en ambos bucles tienen eficiencia O(1)

→ Segundo bucle

$i(n) = O(1)$
$g(n) = O(1)$
$h(n) = O(n)$
$f(n) = O(1)$
$a(n) = O(1)$

$O(1+1+n(1+1+1)) = O(2+3n) = O(n)$

$\boxed{\text{EFICIENCIA TOTAL DE LA FUNCIÓN SELECCIÓN ARISTA} \Rightarrow O(n)}$

- imprimirMatriz → esta función tiene como objetivo ir viendo como va cambiando la matriz. Para habilitar esta función solo tenemos que poner a true el booleano de la clase MatrizA.

EFICIENCIA IMPRIME MATRIZ

→ Primer bucle anidado

$i(n) = O(1)$
$g(n) = O(1)$
$h(n) = O(n)$
$f(n) = O(n)$
$a(n) = O(1)$

$O(1+1+n(1+n+1)) =$
$O(2+n^2+2n) = O(n^2)$

→ Segundo bucle anidado

$i(n) = O(1)$
$g(n) = O(1)$
$h(n) = O(n)$
$f(n) = O(1)$
$a(n) = O(1)$

$O(1+1+n(1+1+1)) =$
$O(2+3n) = O(n)$

Por tanto, la eficiencia total de la función es $\max(O(n^2), O(n))$

$$\boxed{\text{EFICIENCIA TOTAL DEL MÉTODO IMPRIME MATRIZ} \Rightarrow O(n^2)}$$

Tras estos cálculos la eficiencia de la función caminoEulerGreedy es la siguiente:

EFICIENCIA CAMINOS EULER GREEDY

La función "erase()" de <vector> tiene eficiencia O(n) en el peor de los casos, pues un borrado toma un tiempo lineal.

→ Bucle for

$i(n) = O(1)$
$g(n) = O(1)$
$h(n) = O(n)$
$f(n) = O(n)$
$a(n) = O(1)$

$O(1+1+n(1+n+1)) =$
$O(2+n^2+2n) = O(n^2)$

→ Bucle while

$h(n) = O(n)$
$g(n) = O(1)$
$f(n) = O(n^2)$

$O(1+n(1+n^2)) =$
$O(1+n+n^3) = O(n^3)$

// Factible (candidatos) $\in O(n^2)$
// Condición if (Factible)
El método push_back tiene eficiencia O(1)

Por tanto, la eficiencia total de caminos EulerGreedy es $\max(O(n^2), O(n^2), O(n^3))$

$$\boxed{\text{EFICIENCIA TOTAL DE LA FUNCIÓN CAMINOS EULER GREEDY} \Rightarrow O(n^3)}$$