PRÁCTICA 3:
ALGORITMOS GREEDY

Algorítmica

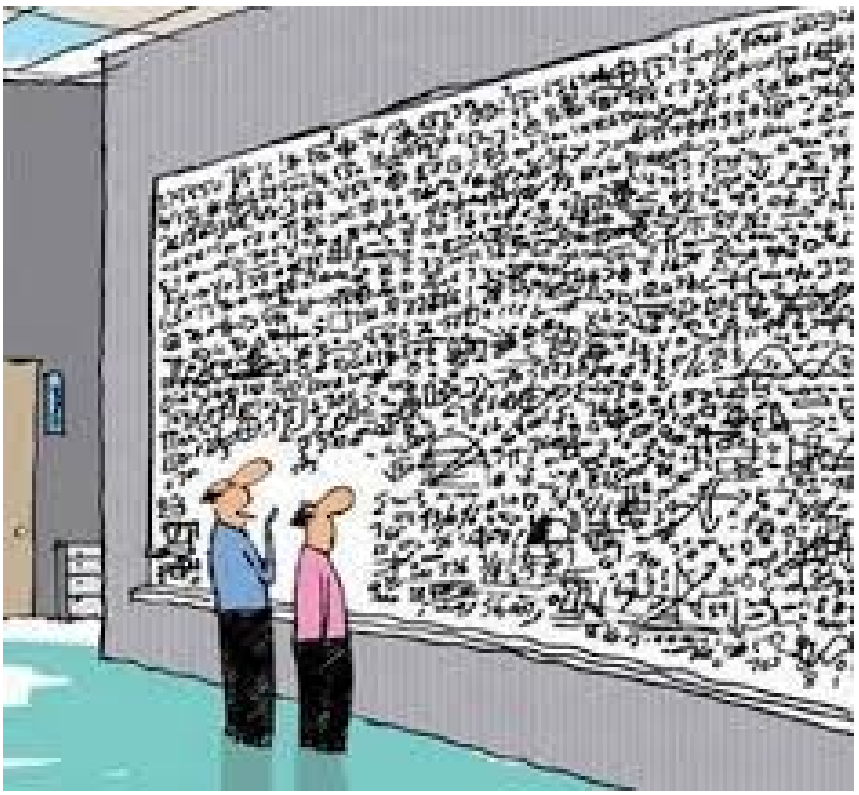
2º Grado en Ingeniería Informática

UNIVERSIDAD
DE GRANADA

JAVIER BUENAVENTURA MARTÍN JIMÉNEZ
EMILIO GUILLÉN ÁLVAREZ
ALBERTO RODRÍGUEZ FERNÁNDEZ

GUIÓN

- Descripción de cómo es un algoritmo greedy → Introducción.
- Cuáles son las condiciones para que se pueda hacer un algoritmo Greedy.
- Porque el problema del grafo de Euler se puede resolver por la metodología Greedy.
- Cuáles son los problemas de utilizar la metodología greedy, y cuando no va a dar una solución óptima.
- Pseudocódigo.
- Implementar el código.
- Dos ejemplos con enunciado, desarrollo y solución.
- Eficiencia del código.



1. Formalizar, si es posible, la descripción de funcionamiento del método anterior como un algoritmo Greedy. Para ello:
    A. Compruebe si se puede resolver mediante Greedy.
    B. Diseñe las componentes Greedy del algoritmo.
    C. Adapte la plantilla de diseño Greedy a las componentes propuestas.

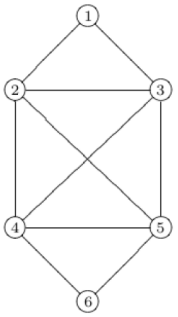
La idea básica en la que se apoya el uso de la técnica Greedy se basa en seleccionar en cada momento lo mejor de entre un conjunto de candidatos, sin tener en cuenta lo ya hecho, hasta obtener una solución para el problema, y así cuando el algoritmo que resuelve un problema está diseñado de acuerdo a la técnica Greedy, se dice que es un algoritmo Greedy. Generalmente se suelen utilizar para problemas de optimización.

Un algoritmo Greedy procede siempre de la siguiente manera: Se parte de un conjunto vacío: S = ∅. De la lista de candidatos, se elige el mejor (de acuerdo con la función de selección). Comprobamos si se puede llegar a una solución con el candidato seleccionado (función de factibilidad). Si no es así, lo eliminamos de la lista de candidatos posibles y nunca más lo consideraremos. Si aún no hemos llegado a una solución, seleccionamos otro candidato y repetimos el proceso hasta llegar a una solución [o quedarnos sin posibles candidatos].

Para poder resolver un problema usando el enfoque greedy, tendremos que considerar 6 elementos:

1. Conjunto de candidatos (elementos seleccionables).
2. Solución parcial (candidatos seleccionados).
3. Función de selección (determina el mejor candidato del conjunto de candidatos seleccionables).
4. Función de factibilidad (determina si es posible completar la solución parcial para alcanzar una solución del problema).
5. Criterio que define lo que es una solución (indica si la solución parcial obtenida resuelve el problema).
6. Función objetivo (valor de la solución alcanzada).

No obstante, hay problemas que sin tener todas las anteriores características, también pueden resolverse según un enfoque Greedy.
El problema que se nos plantea es resolver mediante un algoritmo Greedy un grafo concreto que se nos ofrece para encontrar la solución del Grafo de Euler, es decir, averiguar aplicando Greedy un camino que siga el planteamiento de este matemático para nuestro grafo.
Lo que Euler especificó fue que se realizara sobre un grafo convexo, donde desde cualquiera de los puntos existe la posibilidad de llegar a otro de los puntos. El nuestro cumple esta condición:



Para comprobar que podemos aplicar la técnica Greedy en el ejemplo anterior nos basamos en las 6 características o requisitos especificados anteriormente.
Entre el conjunto de candidatos disponibles tenemos las diferentes aristas que unen el conjunto de nodos; se parte de un nodo dado v del grafo G (por ejemplo, nodo 1). Si G contiene sólo un nodo v, el algoritmo termina (en este caso no termina porque G tiene más de un nodo). Si hay una única arista a que incide en v, entonces llamamos w al otro vértice que conecta la arista a, y la quitamos del grafo (en este caso si hay más de una arista que incide en v: nodo 1). Como hay más de un lado que incide en v, elegimos uno de estos (lo llamamos w) de modo que al quitarlo del grafo G, el grafo siga siendo conexo. Cogemos la arista que une v con w y la quitamos del grafo (se elige entre nodo 2 o nodo 3 y se elimina la arista que los conecta).
Cambiamos el nodo v por el nodo w y volvemos a analizar si existe una única arista que incida en un nodo; en caso contrario, aplicamos de nuevo el algoritmo del caso en el cual inciden varias aristas en un nodo, y repetimos el proceso hasta que terminemos de hacer el circuito de Euler.

Entre los principales problemas que podríamos encontrar al utilizar la metodología Greedy podríamos mencionar algunos tales como:

- Situación en la que el número de aristas que se conectan a un nodo no tienen grado par, que se soluciona poniendo una precondición de que el grafo tenga en cada nodo grado par. De lo contrario no funcionará.

- Soluciones subóptimas: Aunque la técnica Greedy toma decisiones locales óptimas en cada paso, no siempre garantiza que la solución global sea óptima. En algunos casos, tomar una decisión local óptima puede llevar a una solución global subóptima.

- Incapaz de encontrar una solución factible: En algunos casos, puede que no haya ninguna solución factible que satisfaga los criterios de la metodología Greedy. Por lo tanto, la aplicación de esta técnica puede fallar para estos problemas.

- Costosa en términos computacionales: En algunos problemas, el número de posibles decisiones puede ser enorme y el algoritmo Greedy puede tener que evaluar todas las opciones antes de tomar una decisión. Esto puede llevar a una complejidad computacional alta, lo que hace que la metodología Greedy no sea práctica para estos problemas.

- No ser aplicable en problemas que involucran restricciones: En algunos problemas, las decisiones que se toman en un paso pueden limitar las opciones disponibles en los pasos posteriores. En estos casos, la metodología Greedy puede no ser aplicable, ya que puede tomar decisiones que violen las restricciones posteriores.

La metodología Greedy no aportará una solución óptima cuando:

- Cuando el problema no cumple con la propiedad de optimalidad de subestructura: La optimalidad de subestructura es una propiedad que debe cumplir un problema para que la técnica Greedy pueda ser aplicada. Esta propiedad se refiere a que cualquier solución óptima al problema debe contener una o más soluciones óptimas a subproblemas del mismo. Si el problema no cumple esta propiedad, la metodología Greedy no puede garantizar que se alcanzará la solución óptima.

- Cuando la solución óptima no se puede construir de manera incremental: La metodología Greedy construye la solución de manera incremental, tomando decisiones óptimas en cada paso. En algunos casos, la solución óptima del problema no se puede construir de manera incremental, por lo que la metodología Greedy no es adecuada para resolver el problema.

- Cuando la elección de la solución óptima local no garantiza una solución óptima global: La metodología Greedy toma decisiones óptimas en cada paso, pero estas decisiones locales no siempre garantizan que se alcanzará una solución óptima global. En algunos casos, tomar la decisión óptima local en cada paso puede llevar a una solución subóptima global.

- Cuando el problema tiene restricciones: La metodología Greedy puede no ser adecuada para resolver problemas con restricciones, ya que las decisiones locales óptimas en cada paso pueden limitar las opciones disponibles en pasos posteriores y violar las restricciones.

Pseudocódigo:

- Diseñar una lista de candidatos para formar una solución: las distintas aristas.

- Identificar una lista de candidatos ya utilizados: las aristas a las cuales se puede acudir desde un mismo nodo.

- Diseñar una función solución para saber cuándo un conjunto de candidatos es solución al problema: empezar por nodo origen, pasar por las aristas una sola vez y volver al nodo origen.

- Diseñar una función de selección del candidato más prometedor para formar parte de la solución: se selecciona la arista correspondiente al nodo que más aristas tenga conectadas.

- Diseñar un criterio de factibilidad (cuándo un conjunto de candidatos puede ampliarse para formar la solución final): se tiene que volver al nodo origen por el que se empezó.

- Existe una función objetivo de minimización/maximización: pasar por todas las aristas una y solo una vez y volver al nodo origen.

```
Función S= Voraz(lista de candidatos (aristas) C )
S =Ø
Mientras (C != Ø) y (S no es solución) hacer:
        Utilizamos un método para Buscar (x); x = La arista que conecta al
        nodo que más aristas tenga conectadas y con el mayor valor.
        C = C - {x}
        Si es factible(S U {x}) entonces
                S = S U {x}
Fin-Mientras

Si S es solución entonces
        Devolver S
En otro caso
        Devolver “No hay solución”
```

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

Aparte de las correspondientes fotos, aportamos los códigos fuente en un fichero .ZIP.

```
int Buscar( int** v,int contador, int nodo){
    int  sol,cont, maximo=0;
    for (int i = 0; v[nodo][i] != -1; i++) {
        if(v[nodo][i] != -2 ) {
            int l;
            cont = 0;
            for ( l= 0; v[v[nodo][i]-1][l] != -1; l++) {
                if (v[v[nodo][i]-1][l] != -2 ){cont++;}
            }
            if (maximo <= cont) {
                maximo = cont;
                sol = v[nodo][i];
            }
        }
    }
    return sol;
}
```

```
int main(int argc, char *argv[]) {

    int ** v, *sol, *candidatos;
    int n, i, j, argumento, k=0;
    chrono::time_point<std::chrono::high_resolution_clock> t0, tf; // Para medir el tiempo de ejecuciÃ³n
    unsigned long  tejecucion; // tiempo de ejecucion del algoritmo en ms
    unsigned long int semilla;
    ifstream fentrada;
    ofstream fsalida;

    if (argc <= 2) {
        cerr<<"\nError: El programa se debe ejecutar de la siguiente forma.\n\n";
        cerr<<argv[0]<<" NombreFicheroEntrada NombreFicheroSalida\n\n";
        return 0;
    }
    cout << "Abirmos fichero entrada " << endl;
    // Abrimos fichero de entrada
    fentrada.open(argv[1]);
    if (!fentrada.is_open()) {
        cerr<<"Error: No se pudo abrir fichero para lectura "<<argv[1]<<"\n\n";
        return 0;
    }
    // Inicializamos generador del grafo
    char valorcogido;
    int contador=0, valornum;
    fentrada >> valorcogido;
    valornum = (int)valorcogido;
    //TamaÃ±o del vector con los datos y cada uno con un vector con sus conexiones
    v = new int* [valornum];
    for (i = 0; i < valornum ; i++) {
        v[i] = new int [valornum];
    }
```

```
fentrada >> valorcogido;
while (valorcogido != '#' ){
    for(i=0; valorcogido != '*' ;i++){
        //GUARDO NUMEROS DEL FICHERO AL VECTOR
        valornum = (int)valorcogido - '0';
        v[contador][i] = valornum;
        k++;
        fentrada >> valorcogido;

    }
    v[contador][i] = -1;
    contador++;
    fentrada >> valorcogido;
}
//Vector donde irÃ¡n las soluciones
int maximacantidadearistas = (contador * (contador+1))/2;
sol = new int [maximacantidadearistas];

fentrada.close();

cout << "Abirmos fichero salida " << endl;
// Abrimos fichero de salida
fsalida.open(argv[2]);
if (!fsalida.is_open()) {
    cerr<<"Error: No se pudo abrir fichero para escritura "<<argv[1]<<"\n\n";
    return 0;
}

// BUSQUEDAD DEL CAMINO

t0 = std::chrono::high_resolution_clock::now(); // Cogemos el tiempo en que comienza la ejecuciÃ³n del algoritmo

int pos=0;
int nodollegar,nodo;


//elegimos un punto para empezar
int nodoinicial=2  -1;
nodo=nodoinicial;

sol[pos]=nodo+1;
pos++;
```

```
bool caminodirecto=false;
//k Es la cantidad de conexiones que tienen las aristas (aristas * 2)
while (k!=0){
    //BUSCAMOS EL NODO MAYOR CON EL MAYOR GRADO DE ARISTAS CONECTADAS
    nodollegar = Buscar(v,contador,nodo);

    if (nodollegar == -1 ) {
        cerr<<"Error: NO SE PUEDE ENCONTRAR UN CAMINO  VÃLIDO"<<"\n\n";
        return 1;
    }
    fsalida<<nodollegar << "  " ;
    //BUSCA EL NODO PARA ELIMINARLO Y GUARDARLO EN LA SOLUCIÃ"N
    for (int f=0; v[nodo][f] != -1 && caminodirecto==false ; f++){

       if(v[nodo][f] == nodollegar){
            caminodirecto=true;
            v[nodo][f]=-2;
        }
    }
    sol[pos]=nodollegar;
    pos++;
    //ELIMINAMOS EL NODO EN EL REGISTRO DEL NUEVO
    int guardar3 = nodo+1;
    nodo = nodollegar-1;

    for (int f=0; v[nodo][f] != -1 ; f++){
        if(v[nodo][f] == guardar3){
            v[nodo][f]=-2;
        }
    }
    //ELIMINO LOS DOS PUNTOS DE CONEXIÃ"N DE K
    k= k -2;
    caminodirecto=false;

}
//INDICO EL FINAL DEL ARRAY CON -1
sol[pos]=-1;

tf= std::chrono::high_resolution_clock::now(); // Cogemos el tiempo en que finaliza la ejecuciÃ³n del algoritmo

tejecucion= std::chrono::duration_cast<std::chrono::microseconds>(tf - t0).count();

cerr << "\tTiempo de ejec. (us): " << tejecucion << " para tam. caso "<< n<<endl;
```

```
        // Guardamos tam. de caso y t_ejecucion a fichero de salida
//      fentrada << n << " " << tejecucion << "\n";

        // Liberamos memoria del vector
    for (int l = 0; sol[l] != -1; l++) {
        fsalida<<sol[l] << "  " << endl;
    }

    delete [] v;
    delete [] sol;


    // Cerramos fichero de salida
    fsalida.close();

    return 0;
}
```

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

EJEMPLO 1

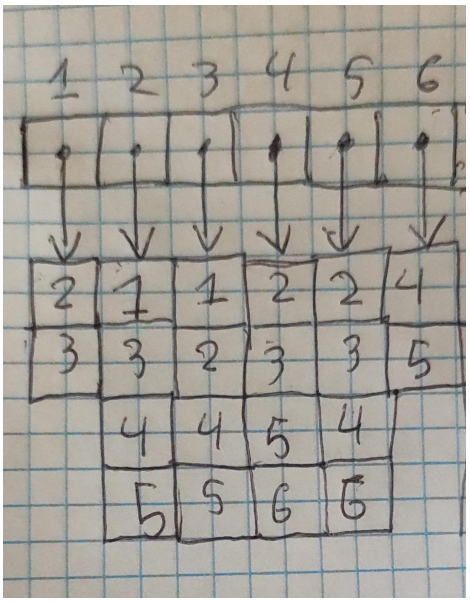
Tomamos como ejemplo el grafo de Euler que nos aporta el propio pdf, tomando como nodo origen el nodo 1. Podemos comprobar que se cumple, partiendo por ejemplo del nodo 1 y comprobando si pasando tanto por el nodo 2 como por el nodo 3, llegamos al nodo origen (nodo 1), pasando una y sola una vez por todas las aristas:

EJECUCIÓN DEL PROGRAMA

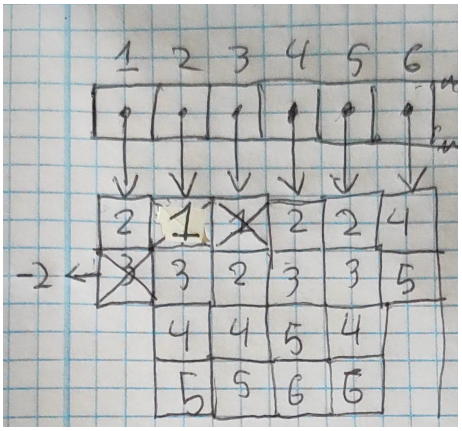
El fichero de entrada (tendrá este diseño: nº de nodos, nodos a los que está conectado el primer nodo, utilizamos el signo * para pasar el siguiente nodo, nodos a los que está conectado el segundo nodo,... e indica que no quedan más nodos con el signo #):

6 2 3 * 1 3 4 5 * 1 2 4 5 * 2 3 5 6 * 2 3 4 6 * 4 5 * #

Guardamos estos datos en un puntero de punteros a int:

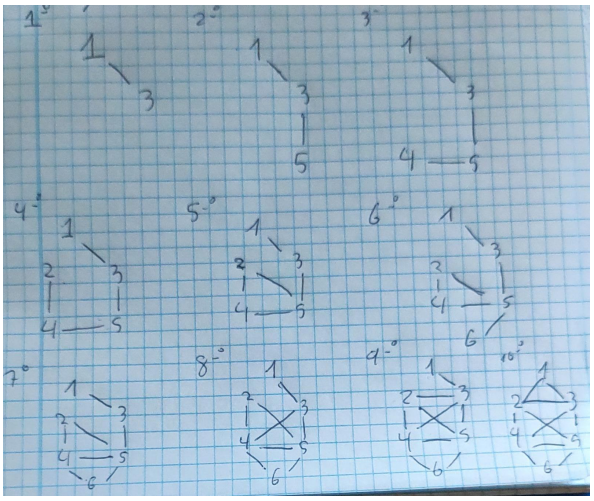


Partimos del nodo 1 (el código funciona partiendo desde cualquier nodo), y saltamos al nodo 3, ya que es el que tiene más aristas y mayor valor. Por lo tanto borramos el valor 3 en el nodo 1 y el valor 1 del nodo 3, sacando esas aristas de los posibles candidatos.



De esta forma llegamos a la siguiente solución:

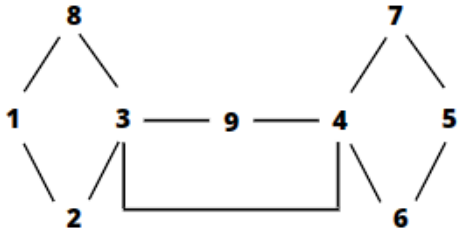
1 3 5 4 2 5 6 4 3 2 1



EJEMPLO 2

Ahora se plantea un contraejemplo en el cual no se puede aplicar Greedy, o en el caso de usar dicha técnica no siempre proporcionaría la solución correcta, siendo este grafo, perteneciente al tipo de Grafo de Euler.

Podemos comprobar que para un determinado camino como por ejemplo: 3-2-1-8-3-4-6-5-7-4-9-3 o 4-6-5-7-4-9-3-8-1-2-3-4, sí se cumpliría el camino de Euler ya que no repite ninguna arista. Sin embargo, la técnica Greedy podría elegir otro camino distinto que no fuese uno de estos, y funcionar o no, ya que la prioridad de este algoritmo en esta implementación es buscar el nodo adyacente de mayor valor y grado. Por ejemplo, si cogemos y empezamos por el vértice 1 y seguimos hasta llegar al 9, por aristas no visitadas, en este punto, ninguna de las opciones para seguir cumpliría el camino de Euler, ningún camino. Por lo tanto Greedy no sería la opción correcta y sería necesario recurrir a algoritmos más sofisticados como el de Fleury.

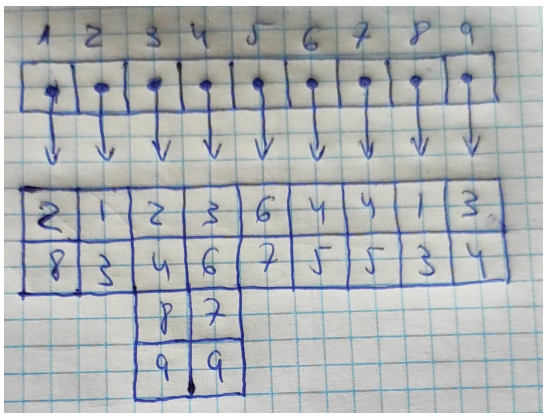


EJECUCIÓN DEL PROGRAMA

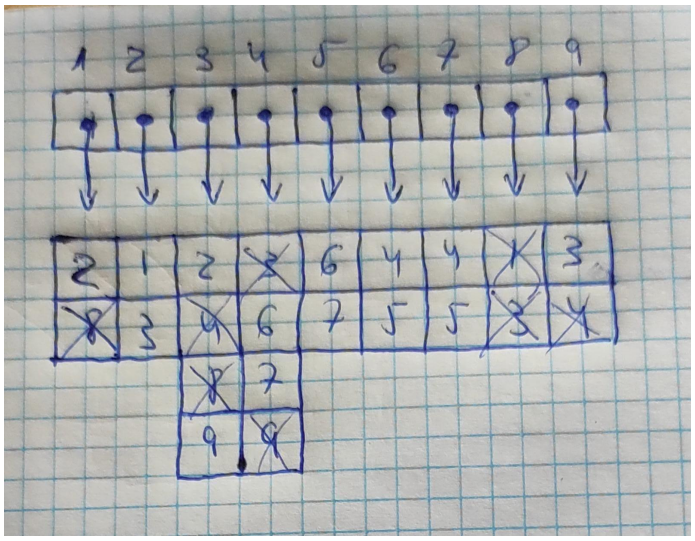
Metemos el fichero con los datos:

9 2 8 * 1 3 * 2 4 8 9 * 3 6 7 9 * 6 7 * 4 5 * 4 5 * 1 3 * 3 4 * #

Guardamos estos datos en un puntero de punteros a int:

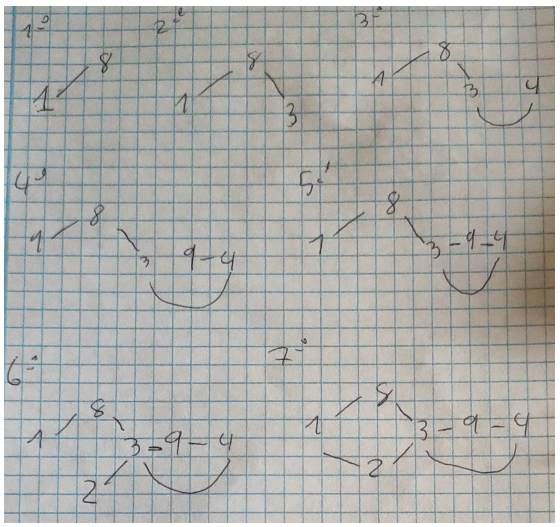


El algoritmo Greedy en este caso daría error puesto que vemos que llegaría al nodo 9 sin haber pasado por todas las aristas y sin tener opción a volver para recorrerlas.



De esta forma llegamos a la siguiente solución (QUE NO ES VÁLIDA):

1 8 3 4 9 3 2 1



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

El código original está en fichero.zip.

```
int Buscar( int** v,int contador, int nodo){
    int  sol,cont, maximo=0;
    for (int i = 0; v[nodo][i] != -1; i++) {
        if(v[nodo][i] != -2 ) {
            int l;
            cont = 0;
            for ( l= 0; v[v[nodo][i]-1][l] != -1; l++) {
                if (v[v[nodo][i]-1][l] != -2 ){cont++;}
            }
            if (maximo <= cont) {
                maximo = cont;
                sol = v[nodo][i];
            }
        }
    }
    return sol;
}
```

O(1)

O(1)

O(1)

O(n*m)

```
int pos=0;
int nodollegar,nodo;


//elegimos un punto para empezar
int nodoinicial=2  -1;
nodo=nodoinicial;

sol[pos]=nodo+1;
pos++;

bool caminodirecto=false;
//k Es la cantidad de conexiones que tienen las aristas (aristas * 2)
while (k!=0){
    //BUSCAMOS EL NODO MAYOR CON EL MAYOR GRADO DE ARISTAS CONECTADAS
    nodollegar = Buscar(v,contador,nodo);
    //BUSCA EL NODO PARA ELIMINARLO Y GUARDARLO EN LA SOLUCIÃ"N
    for (int f=0; v[nodo][f] != -1 && caminodirecto==false ; f++){

       if(v[nodo][f] == nodollegar){
            caminodirecto=true;
            v[nodo][f]=-2;
        }
    }
    sol[pos]=nodollegar;
    pos++;
    //ELIMINAMOS EL NODO EN EL REGISTRO DEL NUEVO
    int guardar3 = nodo+1;
    nodo = nodollegar-1;

    for (int f=0; v[nodo][f] != -1 ; f++){
        if(v[nodo][f] == guardar3){
            v[nodo][f]=-2;
        }
    }
    //ELIMINO LOS DOS PUNTOS DE CONEXIÃ"N DE K
    k= k -2;
    caminodirecto=false;
}
//INDICO EL FINAL DEL ARRAY CON -1
sol[pos]=-1;
```

O(1)

O(n*m)

O(n)

O(1)

O(n)

O(1)

O(n*m)