Algorítmica: práctica 3

Recubrimiento de un grafo no dirigido

Grupo 2

Sofía Almeida Bruno Antonio Coín Castro María Victoria Granados Pozo
Miguel Lentisco Ballesteros José María Martín Luque

21 de mayo de 2017

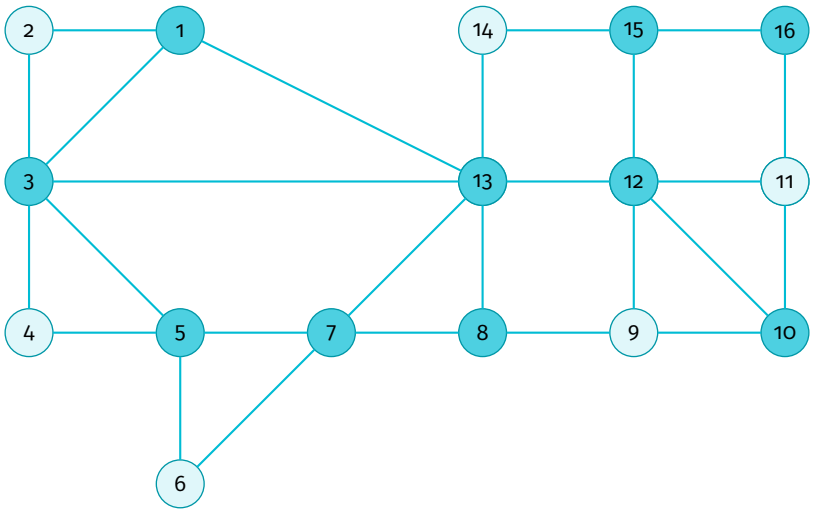
Análisis del problema

Dado un grafo no dirigido G = (V, E), un conjunto U ⊆ V es un recubrimiento de G si cada arista de E incide en, al menos, un vértice o nodo de U. Nuestro problema consiste en encontrar un recubrimiento minimal del grafo G, esto es, un recubrimiento con el menor número posible de nodos.

Para resolver el problema, empleamos un algoritmo *greedy* (voraz). La solución que proporcionamos es el conjunto de nodos que forman el recubrimiento junto con el coste (número de nodos).

Para representar nuestro problema, hemos usado una clase `Problema` y una clase `Solucion`. En `Problema` se encapsula la matriz de adyacencia del grafo (simétrica) junto con su tamaño, mientras que en `Solucion` se encapsula un vector de nodos que forman el recubrimiento, junto a su tamaño. Se adjunta el código fuente de ambas clases.

Figura 1: Ejemplo de grafo. Los nodos más oscuros forman un recubrimiento



Diseño de la solución

Empleando un enfoque *greedy* para la resolución del problema, la idea es ir seleccionando para el recubrimiento los nodos del grafo que tienen el mayor número de incidencias. El número de incidencias de un nodo (número de aristas que tienen a ese nodo como extremo) se conoce también como *grado del nodo*.

En primer lugar, vamos a mostrar las componentes Greedy de este problema:

- *Lista de candidatos:* los nodos del grafo.
- *Lista de candidatos utilizados:* los nodos ya considerados para el recubrimiento.
- *Función solución:* que no haya ninguna arista sin considerar.
- *Criterio de factibilidad:* que el nodo esté en la lista de candidatos (no es necesario ningún criterio adicional).

- *Función objetivo:* recubrimiento de coste mínimo.
- *Función de selección:* nodo en el que inciden más aristas.

Una vez que hemos visto las componentes, pasemos a ver un esqueleto o *pseudocódigo* del algoritmo:

N ≡ número de nodos del grafo
M ≡ número de nodos del recubrimiento
L ≡ matriz de adyacencia
T[1, . . . , M] ≡ vector de nodos del recubrimiento
V[1, . . . , N] ≡ vector de incidencias

---

function RecubrimientoGrafoGreedy (L[1, . . . , N][1, . . . , N])
  T = ∅ ▷ Recubrimiento
  for i = 1, . . . , N **do**
    $V[i] \leftarrow \sum_{j=1}^{N} L[i][j]$ ▷ Número de incidencias del nodo i
  end for
  p ← Nodo con más incidencias (V[p] ≥ V[i] ∀i) ▷ Función de selección

  while V[p] > 0 **do** ▷ Función solución
    T ← T ∪ {p}
    V[p] ← 0
    for j = 1, . . . , N **do**
      if Están conectados p y j **and** V[j] > 0 **then**
        V[j] ← V[j] − 1
      end if
    end for
    p ← Nodo con más incidencias (V[p] ≥ V[i] ∀i) ▷ Función de selección
  end while

  return T
end function

Como se puede observar, algunas de las componentes *greedy* listadas anteriormente se han omitido o modificado en la implementación, pues o bien no son necesarias, o bien es más sencillo implementarlas de otra manera.

- La *lista de candidatos* y la *lista de candidatos utilizados* están representadas como un vector de incidencias V, donde si el nodo i está utilizado, entonces V[i] = 0.
- El *criterio de factibilidad*, como ya dijimos, no es necesario, pues todos los nodos de la *lista de candidatos* son factibles. Implícitamente, estamos considerando como factibles los nodos i tales que V[i] > 0.

- La *función solución* es que el vector de incidencias V sea distinto del vector 0, o lo que es lo mismo, que el máximo de las componentes de V no sea 0. Esto significaría que ya se han considerado todas las aristas que inciden en cada uno de los nodos.
- La *función de selección* consiste en seleccionar, en cada paso, la componente máxima del vector V, es decir, el nodo con mayor grado.

En el diseño del algoritmo nos aprovechamos del vector V para identificar **cada nodo del grafo con cada posición de** V. Es decir, V[i] representa el número de incidencias del nodo i.

Algoritmo Greedy

Veamos ahora el algoritmo implementado en el lenguaje C++. En primer lugar, hemos separado la implementación de la *función de selección*, para que se aprecie con claridad:

Código fuente 1: Función de selección

```
/**
 * Función de selección. Devuelve el nodo con mayor número
 * de incidencias de la lista de candidatos, cuyo tamaño es N.
 */
int FSeleccion(int* LC, int N) {
  int pos_max = 0;
  for (int i = 1; i < N; i++) {
    if (LC[i] > LC[pos_max])
      pos_max = i;
  }
  return pos_max;
}
```

Ahora pasemos a la función principal. El algoritmo comienza construyendo un vector de incidencias, que contiene en cada posición i el número de aristas que inciden en el nodo i–ésimo. Después, seleccionamos el nodo con un número mayor de incidencias, lo añadimos a la solución y eliminamos las aristas que inciden en dicho nodo. A efectos prácticos, esto consiste en poner a 0 el número de incidencias del nodo en V, y en decrementar en 1 el número de incidencias de los nodos conectados con él.

El algoritmo continúa repitiendo el paso anterior hasta que no quedan aristas sin considerar (notemos que no volverá a seleccionar nunca un nodo ya considerado). Cuando finalice el algoritmo, tendremos como solución un vector con los nodos utilizados en el recubrimiento, y el coste será el número de nodos del mismo.

Código fuente 2: Algoritmo Greedy

```
Solucion RecubrimientoGrafoGreedy(const Problema& p) {

  Solucion sol; // Solución a devolver
  int num_nodos = p.getNumNodos();  // Número de candidatos sin utilizar
  int incidencias[num_nodos];  // Vector de incidencias de cada nodo (LC)
  int pos_max;
```

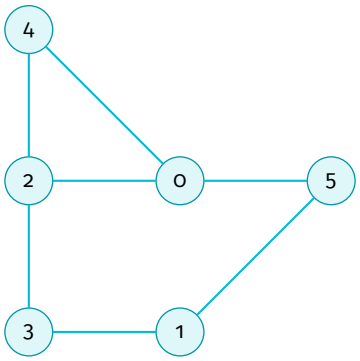
```
7
8   // Inicializar la lista de candidatos (LC)
9   for (int i = 0; i < num_nodos; ++i)
10    incidencias[i] = p.getNumIncidencias(i);
11
12  // Nodo con más incidencias
13  pos_max = FSeleccion(incidencias, num_nodos);
14
15  while (incidencias[pos_max] > 0) { // Mientras el vector de incidencias no sea {0,0,...,0}
16    // Añadir el nodo con más incidencias a la solución (siempre es factible)
17    sol.addNodo(pos_max);
18
19    // Eliminar el nodo de la LC
20    incidencias[pos_max] = 0;
21
22    // Decrementar número de incidencias de nodos conectados con el seleccionado
23    for (int j = 0; j < num_nodos; ++j) {
24      if (p.estanConectados(pos_max,j) && incidencias[j] > 0)
25  --incidencias[j];
26    }
27
28    // Seleccionar nodo con más incidencias
29    pos_max = FSeleccion(incidencias, num_nodos);
30  }
31
32  return sol;
33 }
```

La solución `sol` utilizada es una instacia de la clase `Solucion`, que contiene un vector con los nodos y su tamaño. Esta función `RecubrimientoGrafoGreedy` recibe como parámetro un objeto `p` de la clase `Problema` que encapsula la matriz de adyacencia del grafo en cuestión, donde se ven reflejadas indirectamente las incidencias de cada nodo.

Ejemplo paso a paso

Vamos a ver con un grafo concreto cómo funciona el algoritmo. Consideremos el siguiente grafo.

Figura 2: Ejemplo de grafo sin recubrir



Inicialización:
N = num_nodos = 6

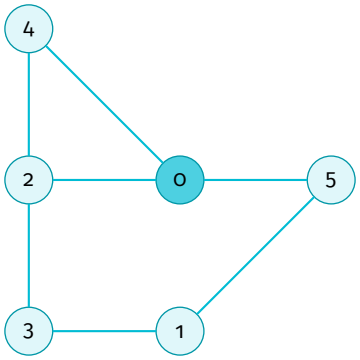
M = `sol.coste` = 0

$$L = \texttt{p.matriz\_adyacencia} = \begin{pmatrix} 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$V = \texttt{incidencias} = \begin{pmatrix} 3 \\ 2 \\ 3 \\ 2 \\ 2 \\ 2 \end{pmatrix}$$

p = `pos_max` = 0

Bucle `while`:
`incidencias[0]` = 3 >0, entra en el bucle
Añadimos el nodo 0 al vector solución. T = `sol.nodos` = {0}. M = `sol.coste` = 1

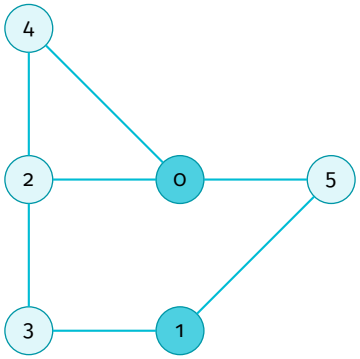


Actualizamos el vector de incidencias, eliminando a este nodo y las incidencias correspondientes. Como las aristas adyacentes al nodo 0 inciden también en los nodos 2, 4 y 5, restamos uno en dichas posiciones del vector de incidencias:

$$V = \texttt{incidencias} = \begin{pmatrix} 3 \\ 2 \\ 3 \\ 2 \\ 2 \\ 2 \end{pmatrix} \Longrightarrow V = \texttt{incidencias} = \begin{pmatrix} 0 \\ 2 \\ 2 \\ 2 \\ 1 \\ 1 \end{pmatrix}$$

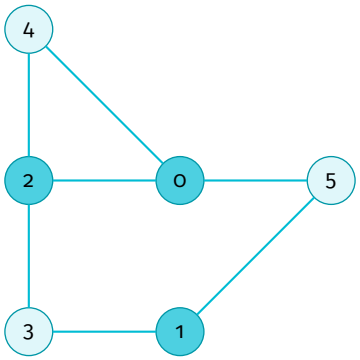
Restablecemos el valor de p = `pos_max` = 1
Como `incidencias[1]` = 2 >0, realiza otra iteración de forma similar:

T = `sol.nodos` = {0, 1}, M = `sol.coste` = 2, las aristas que inciden en el nodo 1 lo hacen también en los nodos 3 y 5.



$$V = \texttt{incidencias} = \begin{pmatrix} 0 \\ 2 \\ 2 \\ 2 \\ 1 \\ 1 \end{pmatrix} \Longrightarrow V = \texttt{incidencias} = \begin{pmatrix} 0 \\ 0 \\ 2 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

Volvemos a actualizar el valor de p = `pos_max` = 2.
`incidencias[1]` = 2 >0, por tanto, repetimos el proceso una vez más: T = `sol.nodos` = {0, 1, 2}. M = `sol.coste` = 3.



Eliminando las incidencias de las aristas adyacentes a este nodo el vector de incidencias sufre el siguiente cambio:

$$V = \texttt{incidencias} = \begin{pmatrix} 0 \\ 0 \\ 2 \\ 1 \\ 1 \\ 0 \end{pmatrix} \Longrightarrow V = \texttt{incidencias} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

p = `pos_max` = 0, por tanto, `incidencias[1]` = 0 == 0, no entra en bucle while. Como último paso devolvemos la solución: T = `sol.nodos` = {0, 1, 2}, M = `sol.coste` = 3.

Eficiencia teórica

La eficiencia del algoritmo viene dada por el bucle *while*. Este bucle se ejecutará hasta que todos los valores del vector de incidencias sean 0, que en el peor de los casos es el número total de nodos menos 1 (N−1).

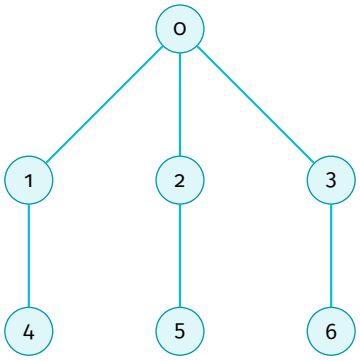
En el interior de este bucle, lo que determina la eficiencia es la *función de selección*, que como tiene que recorrer el vector V es de orden N. Por tanto la eficiencia de este algoritmo es $N(N-1) = N^2 - N$. Notemos que el bucle *for* dentro del *while* tiene el mismo orden de eficiencia que la función de selección.

Es decir, si llamamos V al conjunto de nodos del grafo, la eficiencia del algoritmo es $O(|V|^2)$.

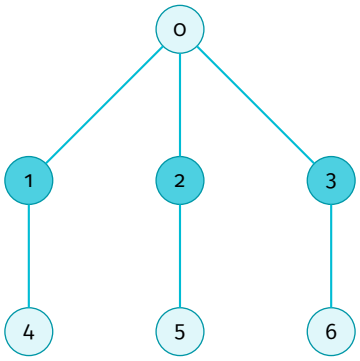
Optimalidad

Hemos notado que nuestro algoritmo no es óptimo, es decir, no siempre nos proporciona un recubrimiento minimal. Sin embargo, siempre es posible encontrar una solución, aunque sea la trivial (a saber, un recubrimiento formado por N − 1 nodos).

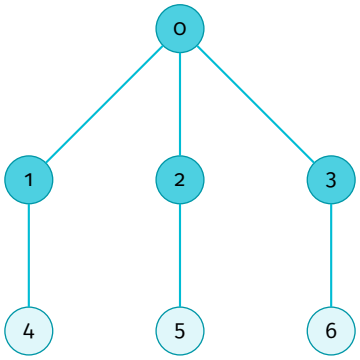
Veamos con un contraejemplo que nuestro algoritmo puede proporcionarnos un recubrimiento que no es óptimo. Consideremos el siguiente grafo:



Es obvio que el recubrimiento minimal de este grafo sería el siguiente:



Sin embargo, como nuestro algoritmo selecciona en primer lugar los nodos de mayor grado, seleccionaría al comienzo el nodo 0, lo que ya condiciona a que el recubrimiento obtenido sea el siguiente, que no es minimal:



Problema real de recubrimiento

Idea 1: los nodos son ciudades, los nodos del recubrimiento son ciudades que tienen un hospital, y los nodos que no están en el recubrimiento son ciudades sin hospital. De esta forma, para ir a un hospital habría que desplazarse, como máximo, una ciudad.
Idea 2: los nodos forman una especie de red de comunicación. Cada nodo es capaz de detectar fallos en sus nodos adyacentes. Con un recubrimiento minimal, podemos detectar fallos en todos los nodos con el mínimo coste.

Compilación y ejecución del proyecto

Para compilar este proyecto simplemente hay que situarse en la carpeta `practica_greedy` y ejecutar el comando make.

Una vez compilado, si estamos situados en la misma carpeta ejecutamos `./bin/main` pasando como parámetro cualquiera de los archivos `.dat` de la carpeta *datos*.