Prácticas Algorítmica:

Algoritmos Greedy:

El problema del recubrimiento de un grafo no dirigido

Jesús Sánchez de Lechina Tejada
Álvaro López Jiménez
Miguel Ángel Robles Urquiza
Antonio Martín Ruíz

Índice:

1. ANÁLISIS DEL PROBLEMA

Enunciado:

Consideremos un grafo no dirigido G = (V, E). Un conjunto U ⊆ V se dice que es un recubrimiento de G si cada arista en E incide en, al menos, un vértice o nodo de U. Es decir ∀(x, y) ∈ E, bien x ∈ U o y ∈ U . Un conjunto de nodos es un recubrimiento minimal de G si es un recubrimiento con el menor número posible de nodos.

Se pide diseñar e implementar un algoritmo greedy que solucione este problema. Como salida, se deberá proporcionar el conjunto de nodos que forman el recubrimiento junto con el coste (número de nodos).

Análisis:

Como podemos concluir del enunciado, en primer lugar, el programa recibirá como parámetro un grafo del cual tenemos que extraer un recubrimiento. En realidad, el grafo lo recibiremos como un archivo formado por su matriz de adyacencia y el tamaño de dicha matriz, por lo que será necesario que dicho archivo tenga un formato específico, en concreto:

La primera línea estará formada sólo por un número que nos indicará el tamaño de la matriz. En las sucesivas líneas, las filas de la matriz, es decir, los elementos de la matriz (que son enteros 0, si no hay adyacencia, 1 si la hay) separados por tabulaciones.

Por otro lado, para construir ese recubrimiento tendremos que coger ciertos nodos del grafo de tal forma que la unión de esos nodos junto con sus adyacencias, es decir, los nodos adyacentes a esos nodos formen el grafo completo.

Por último, se devolverá como solución un conjunto de nodos almacenados en un vector por ejemplo, que forman el recubrimiento y su coste, que será el tamaño de dicho vector.

2. Diseño y componentes de la solución

En este apartado, especificaremos las componentes de los algoritmos greedy y los definiremos en función de nuestro caso concreto, es decir, el ejercicio del recubrimiento.

- **Lista de candidatos:** Todos los nodos del grafo que recibimos como parámetro, que serán las filas o columnas de la matriz de adyacencia.
- **Lista de candidatos utilizados:** Nodos que vamos seleccionando en cada iteración del grafo original, y sus nodos adyacentes a él.
- **Función solución:** El algoritmo se detendrá cuando haya alcanzado la solución, que será cuando la lista de candidatos utilizados coincida con la lista de candidatos, es decir, cuando todos los nodos del grafo estén recubiertos por el recubrimiento.
- **Criterio de factibilidad:** A la hora de seleccionar un nodo, este será o no factible si dicho nodo no se encuentra en el recubrimiento, es decir, no ha sido utilizado.
- **Función de selección:** La decisión de qué nodo utilizar será tomada en función del número de adyacencias que este nodo tenga, es decir, tendrán prioridad aquellos nodos que tengan más nodos adyacentes.
- **Función objetivo:** El objetivo, como ya sabemos, es obtener un recubrimiento del grafo con el mínimo número de nodos o mínimo coste.

3. Esqueleto del algoritmo (pseudocódigo)

La manera de diseñar y desarrollar el algoritmo se ha basado en el siguiente pseudocódigo mediante el que se resuelve el problema:

```
vector<int> recubrimiento ( vector<vector<int> >  m) {
        S = solución = conjunto de nodos que conforman el recubrimiento
        N = lista de nodos
        NC = lista de nodos cubiertos por el recubrimiento
        LCU = lista de nodos utilizados

        Mientras ( NC != N )  {
                nodo = nodo con mayor nº de incidencias no utilizado = getNodoMaxInc(m,LCU)
                Si el nodo no está ya en el recubrimiento {
                        NC <- nodo
                        NC <- nodos adyacentes a nodo = adyacencias (nodo, NC, m)
                        S <- nodo
                }
        }
        devuelve S
}
```

donde hemos optado por utilizar una implementación basada en vectores de la STL. Así S,N,NC,LCU son vectores de enteros (nodos del grafo).

En este esqueleto podemos ver como el vector N es la lista de candidatos, es decir, todos los nodos del grafo que nos pasan. Por otro lado, el vector LCU es la lista de candidatos utilizada, la cual es pasada a la función que selecciona el nodo con mayor incidencias no utilizado, función que actualiza el estado de este vector. Dicha función será la que tenga el papel de función selección en el algoritmo greedy, y en su interior, se harán comprobaciones para saber si el nodo es válido según el criterio de factibilidad.
La función solución consistirá en una función que compruebe en cada iteración la condición del bucle “Mientras”.

4. Explicación del funcionamiento del algoritmo

El algoritmo, cuya implementación está realizada en C++ en el código fuente *recubrimiento.cpp* que añadiremos a la entrega, contiene 4 funciones más el método principal que solucionan el problema del recubrimiento. A continuación explicaremos el funcionamiento de cada función y la conexión entre las mismas.

En primer lugar, hemos desarrollado una función llamada *iguales* cuya función es, dado dos vectores, comprobar si tienen los mismos elementos, aunque el orden de los mismos no sea igual en ambos vectores. Por tanto esta función devolverá un valor de tipo boolean confirmando si tienen los mismos elementos o no. Dentro del algoritmo esta función será la encargada de comprobar si el vector que contiene los nodos del recubrimiento NC, (no confundir con el vector solución S) tiene los mismos elementos que el vector que contiene todos los nodos N.

Seguidamente tenemos una función *contains* cuya función es, dado un vector y un elemento del mismo tipo que los elementos del vector (es decir, de tipo int), comprobar que el vector contiene el elemento mencionado. Por tanto, esta función devolverá un valor de tipo boolean confirmando si el vector contiene a ese elemento o no. Dentro del algoritmo esta función será la encargada de comprobar el criterio de factibilidad, es decir, si el nodo con el que estamos trabajando en un instante dado ya está en la lista de candidatos utilizados LCU o no.

A continuación, tenemos una función *adyacencias* cuya función es, dado un nodo, el vector que contiene los nodos cubiertos por el recubrimiento NC, y la matriz de adyacencia, añadir al vector NC los nodos adyacentes al nodo que le pasamos como parámetro. Por tanto, esta función no tendrá un valor devuelto aunque sí modificará el vector NC que le pasamos. Dentro del algoritmo esta función será la encargada de actualizar el vector NC que se utilizará en la función solución. Merece la pena destacar que el nodo que le pasamos a esta función será el nodo que hemos elegido en función del criterio de factibilidad.

La siguiente función que hemos implementado es *getNodoMaxInc* cuya función es, dada la matriz de adyacencia del nodo y el vector o lista de candidatos utilizados LCU, selecciona el nodo del grafo con mayor número de incidencias siempre y cuando no se haya utilizado, es decir, no esté en LCU. Por tanto, esta función devolverá un valor de tipo int con la etiqueta del nodo que tenga más adyacencias y no se haya usado. Dentro del algoritmo greedy esta función realiza el papel de función selección, y será llamada mientras que la función solución no dictamine que ya se ha cumplido el objetivo que es que N = NC.

La última función que hemos implementado es *recubrimiento* cuya función es, dada la matriz de adyacencia del nodo, llevar a cabo el algoritmo que se encuentra reflejado en el apartado 3. Para ello, llamará cuando se requiera a las funciones que he descrito. Por tanto, esta función devolverá un vector solución que contiene los nodos que forman el recubrimiento del grafo.

5. Aplicaciones

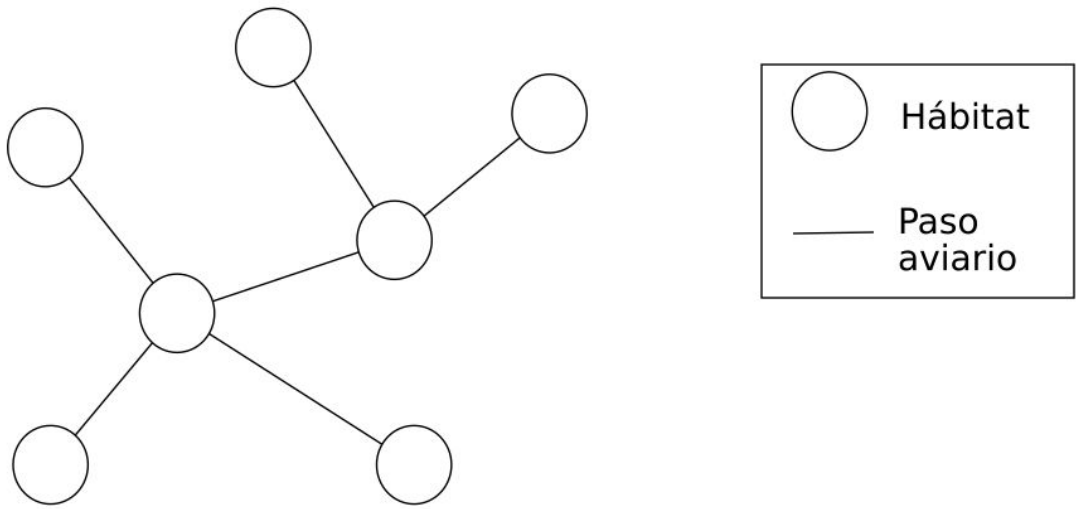
Existen varias aplicaciones del recubrimiento de grafos, las más comunes son algo más abstractas, aplicaciones en demostraciones de teoría de grafos.

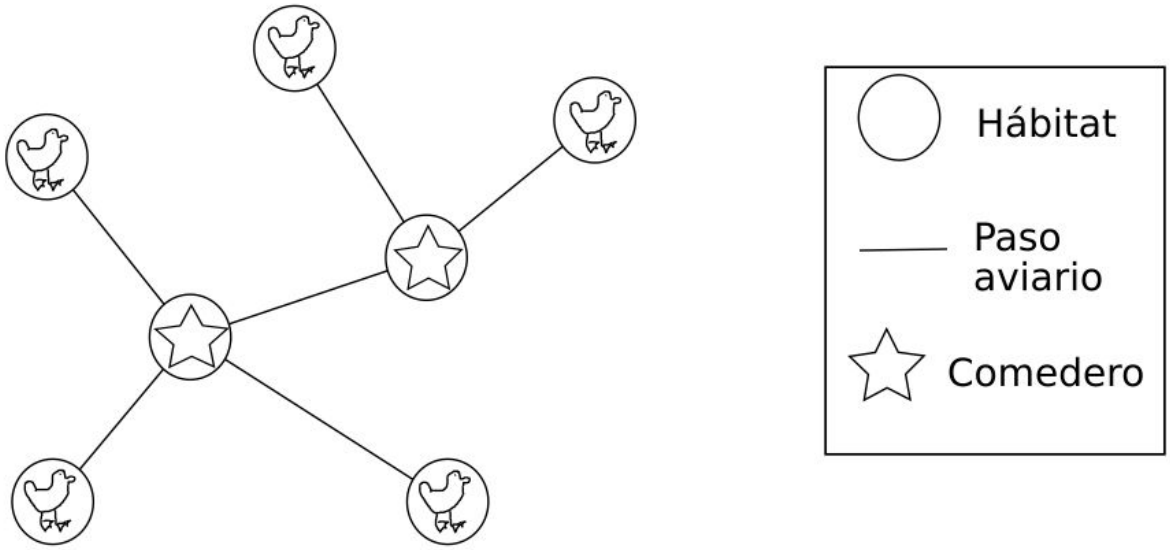
Pero podemos usar otras aplicaciones algo más inusuales, a expensas de no ser la alternativa más eficiente. Un ejemplo de esta es "*El problema de Félix Rodríguez de la Fuente*".

El amante de los animales Félix es el encargado de alimentar a una población de patos.

Estos se encuentran divididos en distintos hábitats que están unidos por pasos aviarios por los que pueden pasar. Pero estos animales en cautividad sólo pueden desplazarse a un hábitat adyacente al día y tienen que estar de vuelta a la noche para dormir en sus nidos.

Félix debe de asegurarse de establecer comederos para que todos los patitos estén bien alimentados.




Hábitat
Paso aviario
Comedero

6. Cálculo de la eficiencia teórica del algoritmo

El orden de eficiencia del algoritmo en el peor de los casos es: $O(n^3)$

En recubrimiento podemos ver un bucle for, pero este no resulta ser el más relevante desde el punto de vista computacional. Inmediatamente después nos encontramos un bucle while que, en el peor de los casos tiene que hacer n iteraciones, pero en cada iteración está llamando a otras funciones con bucles en su interior. Las llamadas más costosas tienen dos bucles for anidados. Por lo que este problema es de orden $O(n^3)$.

7. Compilación y ejecución del código

Para compilar el código tenemos varias opciones:

La más sencilla es ejecutar en la terminal el siguiente comando:

$> g++ -o recubrimiento recubrimiento.cpp

para lo cual debemos estar situados en el directorio donde se encuentre el código fuente.

Otra opción es ejecutar en la terminal el siguiente comando:

$> make

para lo cual, de nuevo, debemos estar situados en el directorio donde se encuentre el archivo de tipo Makefile y el código fuente.

Para ejecutar el código bastará con ejecutar en la terminal el siguiente comando:

$> ./recubrimiento recubrimiento.dat

donde recubrimiento.dat (archivos que también se adjuntarán en la entrega) son archivos que contienen matrices de adyacencia de grafos según el formato que hemos comentado con anterioridad.