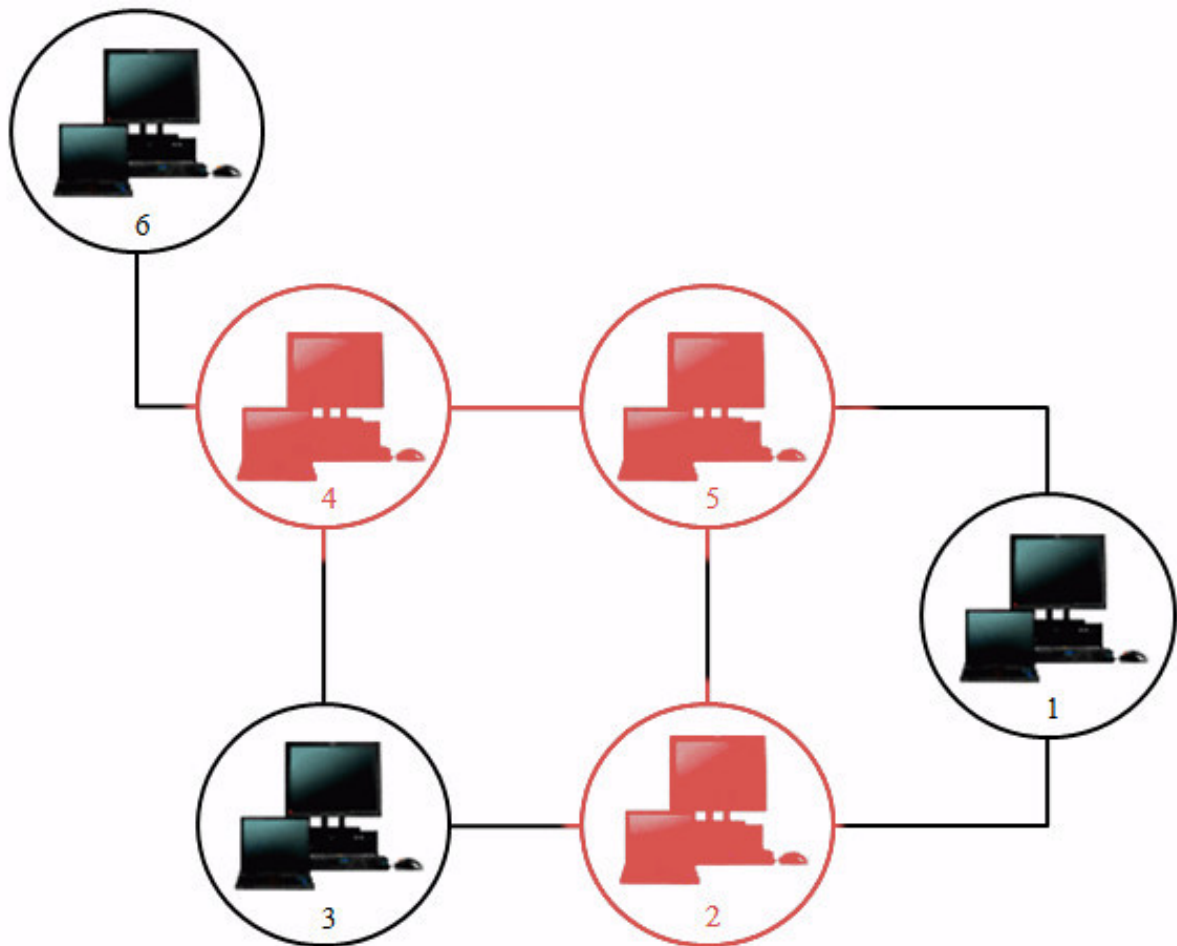


Recubrimiento de un grafo por vértices (Vertex Cover)

practica 3: algoritmos greedy



AUTORES:

**Pablo Moreno Megías, Diego Lerena García, Manuel Vallejo Felipe, Ángel Díaz de la Torre,
Francisco Navarro Morales, Marcel Kemp Muñoz y David Redondo Correa**

Algorítmica [PRACTICAS]
Segundo curso del Grado de Ingeniería Informática.
Universidad de Granada.
curso 2016-2017.

Índice

1. Análisis del problema.	1
2. Diseño de la solución.	1
3. Esqueleto del algoritmo Greedy.	1
4. Funcionamiento del algoritmo.	2
5. Ejemplo real de uso.	4
6. Cálculo del orden de eficiencia teórico del algoritmo.	6
7. Instrucciones sobre cómo compilar y ejecutar el código	7

1 Análisis del problema.

Consideremos un grafo no dirigido $G = (V, E)$. Un conjunto $U \subseteq V$ 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 $\forall (x, y) \in E$, bien $x \in U$ o bien $y \in U$.

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).

2 Diseño de la solución.

Conjunto de candidatos a seleccionar: El conjunto de vértices de los que se compone el problema

Conjunto de candidatos seleccionados: Conjunto de vértices que forman el recubrimiento del grafo.

Función solución: Todas las aristas están conectadas por al menos 2 vértices, es decir, no quedan aristas sueltas.

Función Factibilidad: El problema siempre es factible, sean cuales sean los vértices seleccionados.

Función Selección: Seleccionaremos el vértice que tenga mayor grado (al que lleguen más aristas).

Función Objetivo: Que la suma de vértices del conjunto seleccionado sea la menor posible.

3 Esqueleto del algoritmo Greedy.

Sea G un grafo y R el conjunto de vértices que forman la solución (inicialmente vacío)

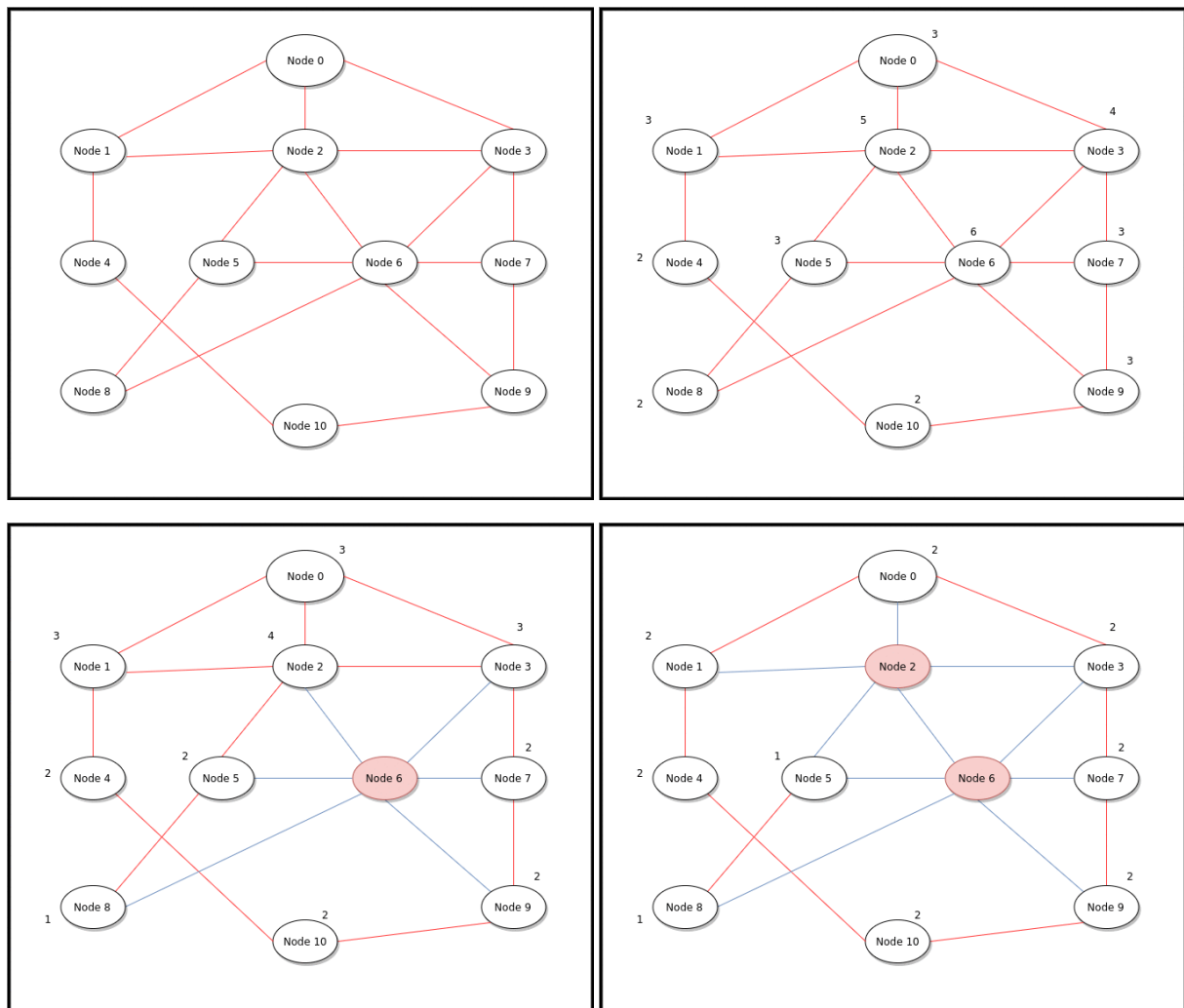
$R = \emptyset$

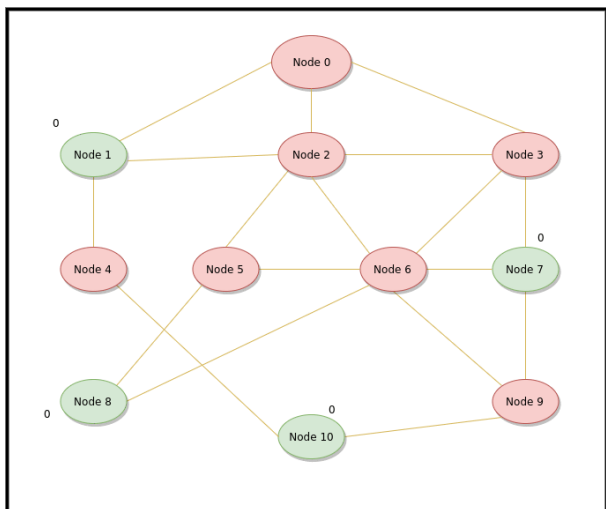
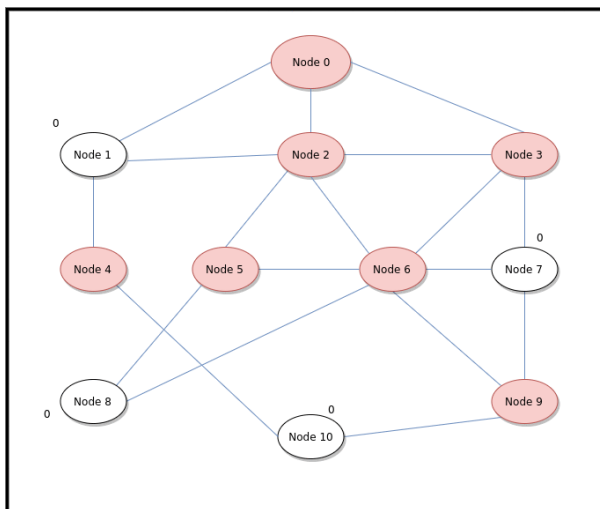
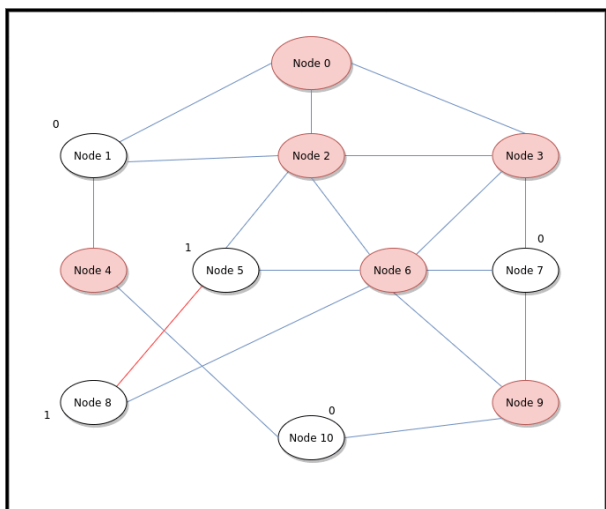
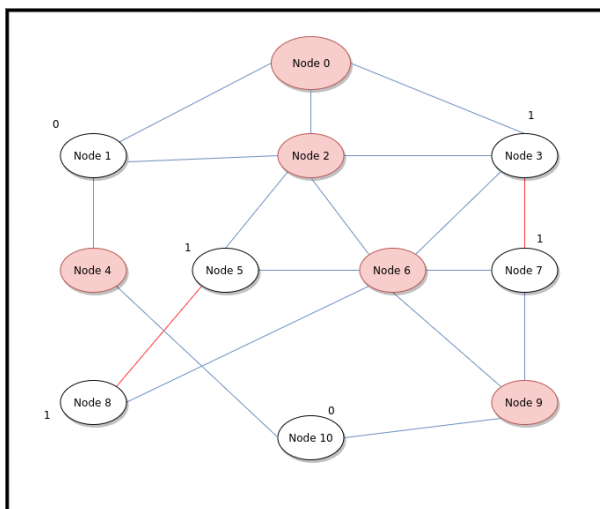
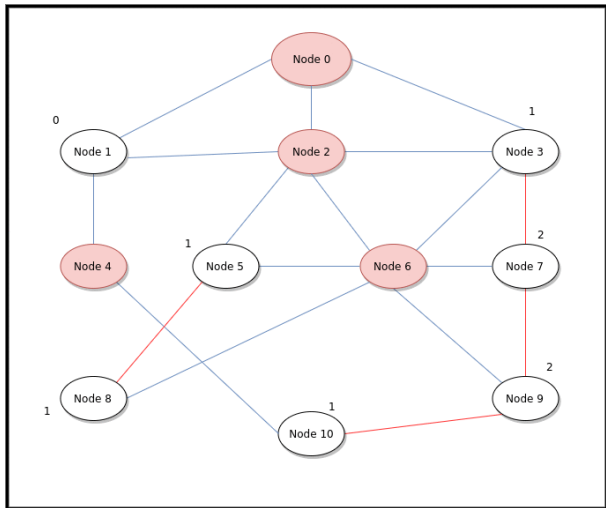
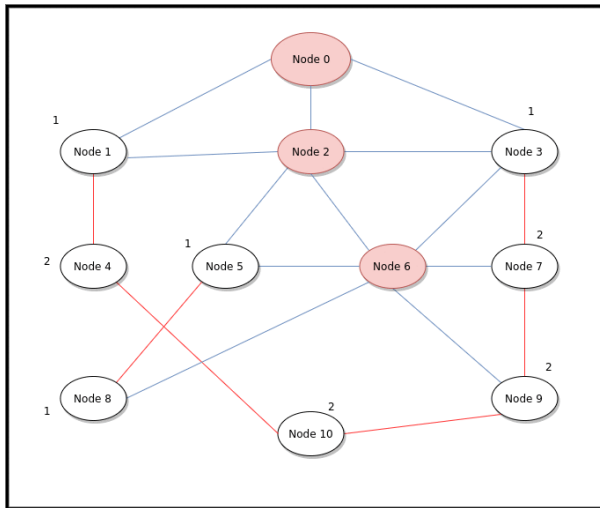
Sea un Grafo $G = (V, A)$, y C el conjunto recubrimiento solución inicialmente vacío:

Elegir en cada paso del algoritmo el vértice $v \in V$ de mayor grado. Añadir el vértice v al conjunto solución C . Borrar el vértice v del grafo G . Marcar las aristas incidentes del vértice seleccionado como cubiertas y reducir el grado de los vértices que conectaban en uno. Repetir este paso hasta que no queden aristas por cubrir.

4 Funcionamiento del algoritmo.

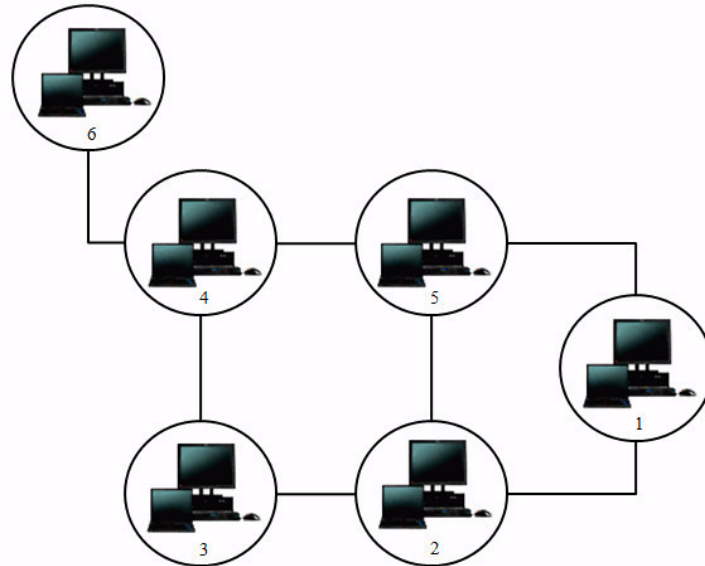
Se comienza calculando los grados de los vértices. A continuación se va seleccionando el nodo con mayor grado y se eliminan los vértices que lo conectan, reduciendo en uno el grado de los vértices conectados al vértice seleccionado. Se repite hasta que no quedan aristas sin eliminar.



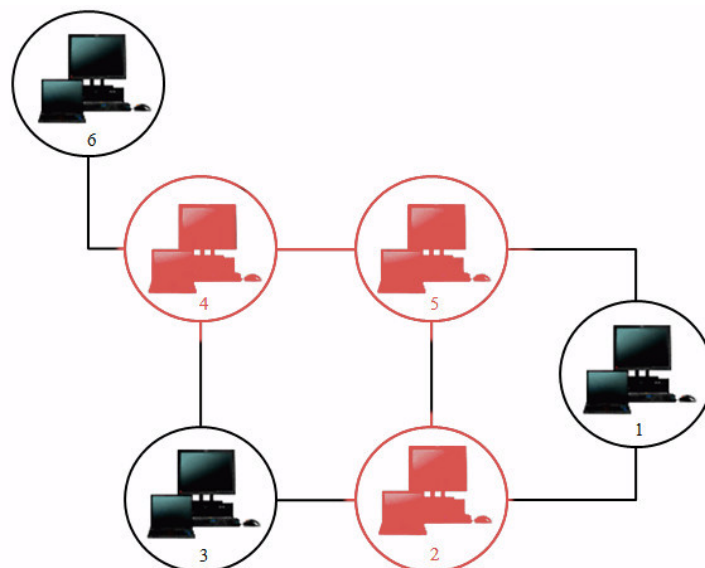


5 Ejemplo real de uso.

Un equipo de informáticos liderados por Eric Filiol en el 'and Cryptology Lab, ESAT' utilizó recientemente algoritmos para obtener la cobertura por vértices de un grafo para simular la propagación de 'gusanos' en grandes redes informáticas y diseñar estrategias para optimizar la protección de las redes contra ataques de estos virus en tiempo real.



La simulación se realizó en una gran red virtual y se descubrió que la distribución de los servidores tenía un gran impacto en la propagación de gusanos y que algunos servidores tenían una importancia mucho más relevante que otros. Así, la necesidad de encontrar en tiempo real qué servidores son más importantes para la seguridad de la red se hizo patente. Se descubrió que, viendo el problema como un grafo en el que los vértices son los servidores de ruta y las aristas las (posiblemente dinámicas) conexiones entre estos, los vértices que forman un recubrimiento minimal son aquellos de vital importancia para la seguridad de la red. Es decir, que resolviendo en cada momento el problema de recubrimiento del grafo dado por el estado de servidores y conexiones entre ellos se puede saber qué servidores son más vulnerables requieren mayor compromiso por parte de los encargados de la seguridad de la red.



La base del algoritmo son los métodos de selección (que consiste en calcular el máximo de los grados de los nodos en $O(n)$) y el de añadir un nodo a la solución (que además conlleva reducir los grados de los vértices adyacentes a este y se hace en $O(n)$). En el peor de los casos el bucle while tendría n iteraciones de complejidad n , luego la eficiencia del algoritmo es $O(n^2)$.

6 Cálculo del orden de eficiencia teórico del algoritmo.

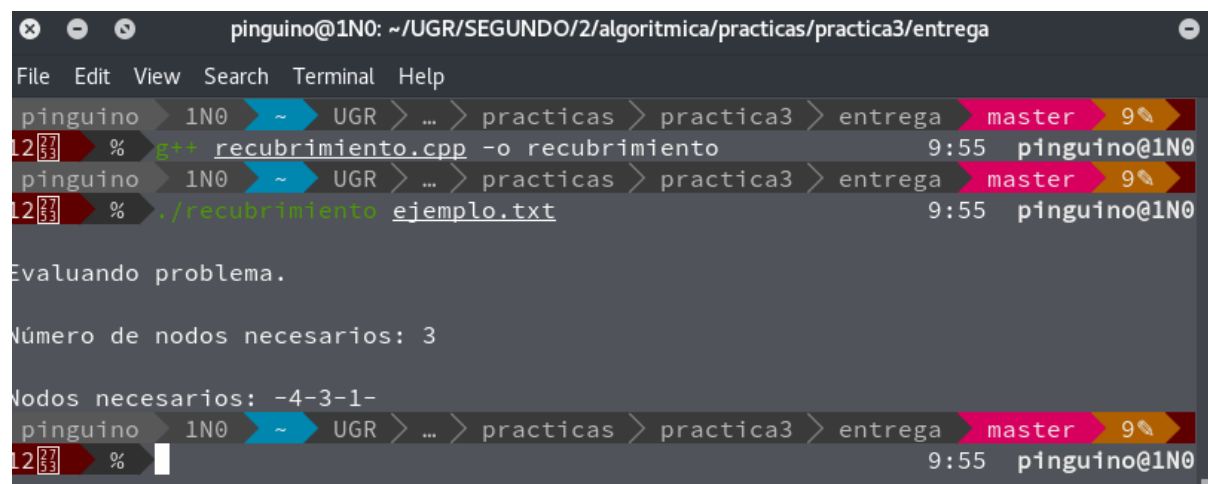
```
void Solucion::Evaluar(Problema p) {  
  
    // calcular los grados de los nodos  
  
    int * grados;           → O(1)  
    int nodo_seleccionado;   → O(1)  
    bool solucion = false;   → O(1)  
  
    grados = new int[p.N] ;   → O(1)  
  
    for(int i = 0 ; i < p.N ; i++) { //Solo para calcular grados  
        for(int j = i ; j < p.N) {  
            if(p.adyacencia[i][j] == 1){  
                grados[i]++ ;  
                grados[j]++ ;  
            }  
        }  
    }  
  
    while(recubrimiento.size() < p.N and !solucion) {           → for(i=0; i < p.N && !solucion;  
i++)                  →  $\sum_{n=1}^{n-1} n = O(n^2)$   
        nodo_seleccionado = getMaximo(grados, p.N);           → O(n)  
        solucion = añadirNodo(nodo_seleccionado);             → O(n)  
    }  
}  
  
bool Solucion::añadirNodo(int nodo, int * grados) {           → O(n)  
    bool solucion = true;                                       → O(1)  
    grados[nodo] = 0;                                           → O(1)  
    for(int i=0; i<p.N; i++) {                                     → O(n)  
        if(adyacencia[nodo][i] == 1) {                         → max(0(1),0) = O(1)  
            grados[i]--;                                         → O(1)  
            adyacencia[nodo][i] = 0;                             → O(1)  
        }  
    }  
    recubrimiento.push_back(nodo);                               → O(1)  
  
    for(int i=0; i<p.N && !solucion; i++) { //Solo para comprobar si ha llegado a la solución  
        if(grados[i] > 0) {  
            solucion = false;  
        }  
    }  
  
    return solucion;                                             → O(1)  
}
```

7 Instrucciones sobre cómo compilar y ejecutar el código

Hemos programado el algoritmo Greedy en un único archivo .cpp llamado `recubrimiento.cpp` que bastará con compilar con `g++`. El programa recibe como parámetro el nombre del fichero que contiene los datos del problema con el formato:

- Una primera línea con el número de nodos, N .
- N líneas con la matriz de adyacencia del grafo (en la posición i,j hay un 1 si los nodos son adyacentes o un 0 si no).

Un ejemplo de ejecución y salida con el problema que hemos entregado como prueba (que es el nodo del ejemplo de la seguridad de servidores):



```
pinguino@1N0: ~/UGR/SEGUNDO/2/algorithmica/practicas/practica3/entrega
File Edit View Search Terminal Help
pinguino 1N0 ~ UGR > ... > practicas > practica3 > entrega master 9
12 27 % g++ recubrimiento.cpp -o recubrimiento 9:55 pinguino@1N0
pinguino 1N0 ~ UGR > ... > practicas > practica3 > entrega master 9
12 27 % ./recubrimiento ejemplo.txt 9:55 pinguino@1N0

Evaluando problema.

Número de nodos necesarios: 3

Nodos necesarios: -4-3-1-
pinguino 1N0 ~ UGR > ... > practicas > practica3 > entrega master 9
12 27 % 9:55 pinguino@1N0
```