Práctica 3 : ALGORITMOS VORACES (GREEDY)

Álvaro Fernández García
Alexandra-Diana Dumitru
Ana María Romero Delgado
Germán Castro Montes
Marino Fajardo Torres

Índice

Descripción del problema

En el ejercicio propuesto es el siguiente:

> Consideremos un grafo no dirigido G = (V, E). Un conjunto U se dice que es un recubrimiento de G si U ⊆ V y 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.

Por lo tanto, se deben realizar las siguientes tareas:

- Diseñar un algoritmo greedy para **intentar obtener un recubrimiento minimal** de G.
- Demostrar que este algoritmo es correcto, o dar un contraejemplo.
- Diseñar un algoritmo greedy que **obtenga un recubrimiento minimal** para el caso particular de grafos que sean árboles.

Algoritmo greedy para el recubrimiento minimal de un grafo

Para esta primera parte, se debe intentar obtener un recubrimiento minimal del grafo.

El algoritmo cuenta con los siguientes elementos:

- **Conjunto de candidatos (C):** vértices del grafo representados de la siguiente forma: *|identificador del vértice, grado del vértice, {conjunto de aristas del vértice}|*
  Este conjunto de vértices se encontrará ordenado de mayor grado a menor.
- **Conjunto de aristas (A):** todas aquellas aristas que forman parte del grafo.
- **Conjunto de seleccionados (S):** representa el conjunto de vértices seleccionados hasta el momento para el recubrimiento minimal.

- **Función solución:** El conjunto de aristas (A) se encuentra vacío, es decir, se han usado todas las aristas del grafo.
- **Función selección (FS):** Selecciona el vértice de mayor grado.
- **Función de factibilidad (FF):** comprueba que el vértice seleccionado tiene al menos una arista que todavía no ha sido recubierta.
- **Función objetivo:** Elegir el mínimo número de vértices posibles que constituyan un recubrimiento.

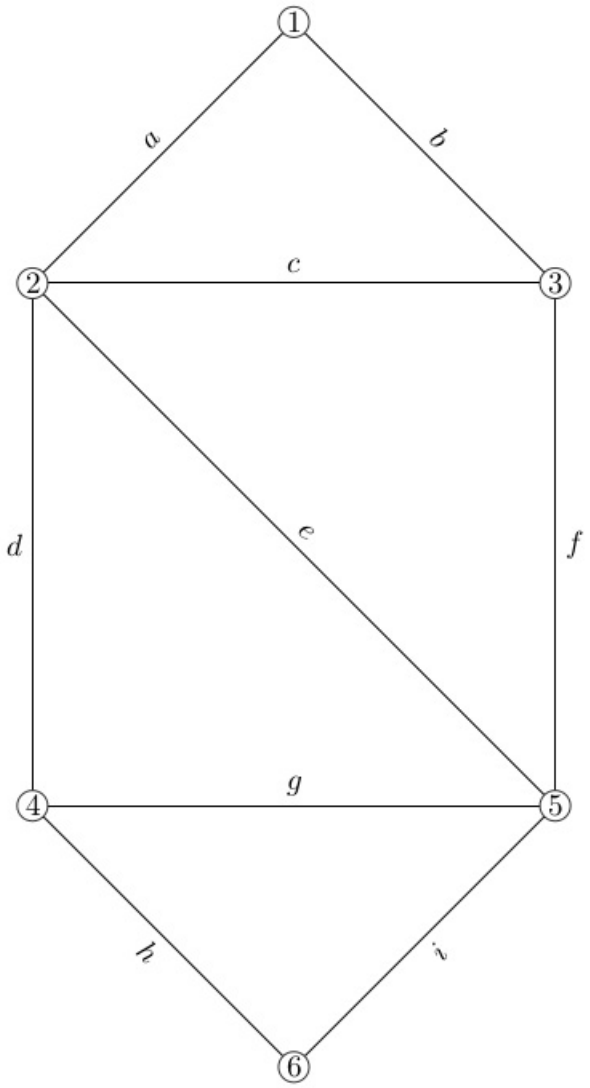
En definitiva, el funcionamiento básico del algoritmo reside en ir seleccionando del conjunto de candidatos (el cual estará ordenado de mayor a menor grado) un vértice. Si ese vértice tiene al menos una arista aun no cubierta (que aun aparece en el conjunto auxiliar de aristas (A)) se añade al conjunto de seleccionados y se eliminan sus aristas asociadas. El proceso continua hasta que todas las aristas están seleccionadas (el conjunto A está vacío).

A continuación se desarrolla, en pseudocódigo, el algoritmo necesario para resolver el problema.

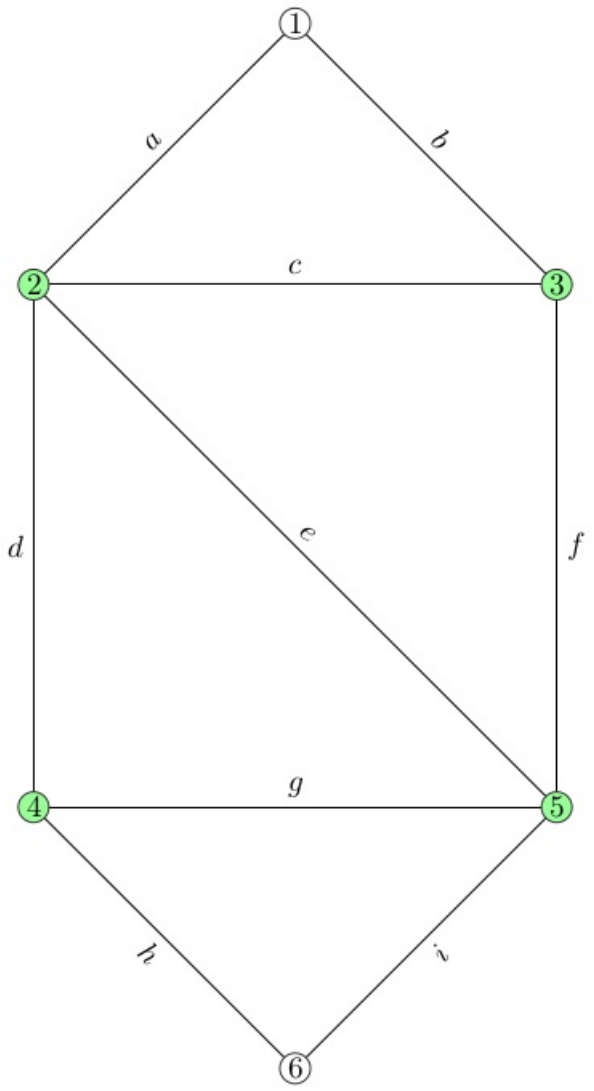
```
Recubrimiento minimal (C)
    S = Ø
    A = {aristas del grafo}

    mientras (A != Ø) hacer      //Función solución (el conjunto de aristas del grafo está vacío)
        x = FS(C)                //Seleccionar el de mayor grado.
        C = C - {x}              //Eliminarlo del conjunto de candidatos.

        si FF(x) entonces        //Si el vértice elegido es factible
            S = S U {x}          //Añadir el vértice al conjunto de seleccionados.
            A = A - {x.aristas}  //Eliminar las aristas cubiertas del conjunto A.
        fin si
    fin mientras

    devolver S
fin
```

Una vez dicho esto, se presentan a continuación una serie de grafos para ilustrar su funcionamiento:

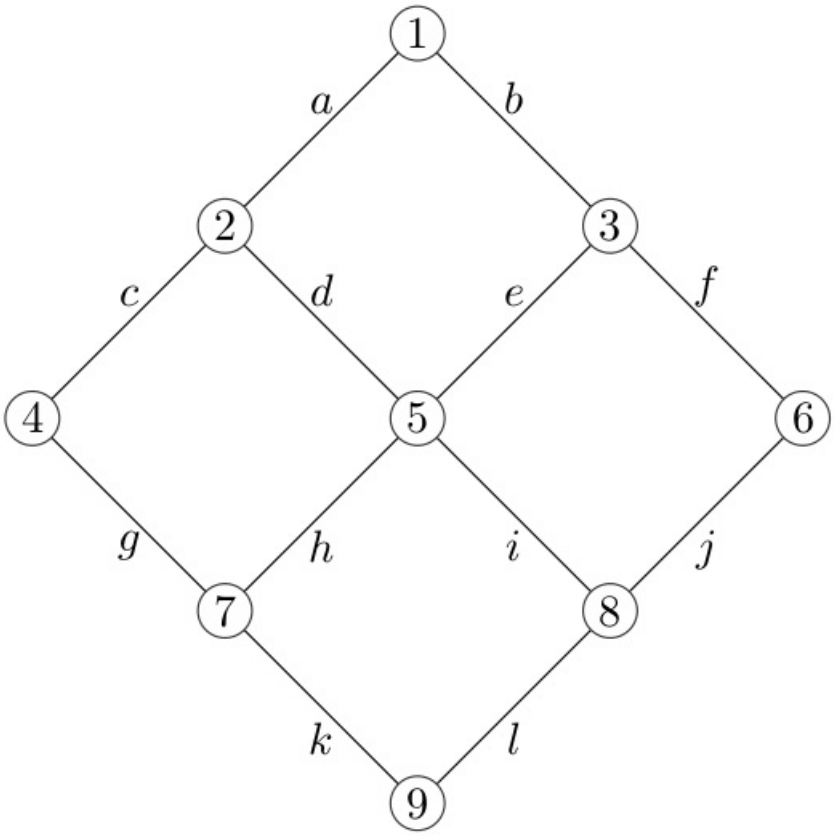
Primero se procede a resolver el siguiente grafo que presenta el siguiente conjunto de candidatos:
(2, 4, {a,c,d,e}),
(5, 4, {e,f,g,i}),
(3, 3, {b,c,f}) ,
(4, 3, {d,g,h}) ,
(1, 2, {a,b}),
(6, 2, {h,i}).



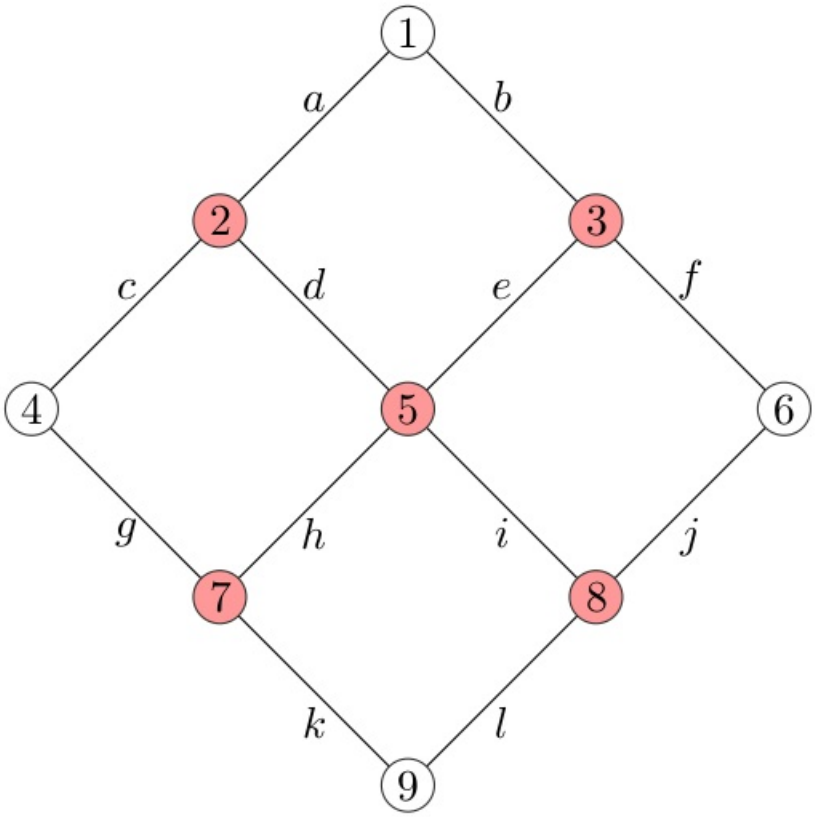
El algoritmo seleccionaría los 4 primeros, con los cuales quedan cubiertas todas las aristas. Además, en este caso, se trata de una solución óptima, es decir obtiene un recubrimiento minimal:



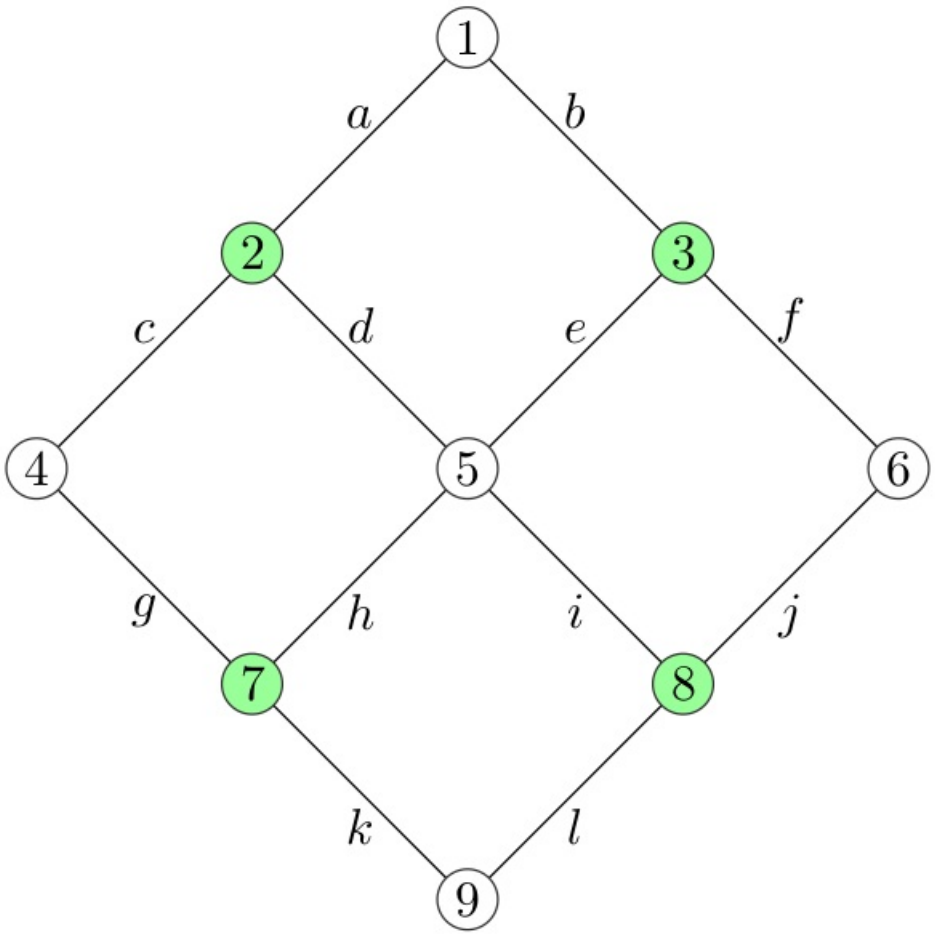
No obstante este algoritmo no siempre encuentra una solución óptima, y para demostrarlo basta con dar un contraejemplo.
El siguiente grafo presenta el siguiente conjunto de candidatos:
(5, 4, {d,e,h,i}),
(2, 3, {a,c,d}),
(3, 3, {b,e,f}),
(7, 3, {g,h,k}),
(8, 3, {i,j,l}),
(1, 2, {a,b}),
(4, 2, {c,g}),
(6, 2, {f,j}),
(9, 2, {k,l})



El algoritmo seleccionaría los 5 primeros vértices con los cuales quedan cubiertas todas las aristas:



Sin embargo, no se trata de una solución óptima, existe una solución con 4 vértices:



Por lo tanto, el anterior algoritmo no nos da una solución minimal en todos los casos, como se está observando.

Algoritmo greedy para el recubrimiento minimal de un árbol

Como se ha podido comprobar, el algoritmo anterior no produce una solución minimal del problema. En el caso anterior, se trataban todo tipo de grafos.

En este apartado se tratará con el caso particular de grafos que son árboles. Para ello se tomará una mecánica distinta a la anterior.

Se procederá a desarrollar un algoritmo para grafos de tipo árbol que siempre encontrará el recubrimiento minimal.

Este algoritmo contará con los siguientes elementos:

- **Conjunto de candidatos (C):** Vértices del árbol. Se encuentran ordenados por nivel de derecha a izquierda. Se parte del nivel más profundo.

- **Conjunto de seleccionados (S):** representa el conjunto de vértices seleccionados hasta el momento para el recubrimiento minimal.
- **Función solución:** El conjunto de candidatos (C) se encuentra vacío.
- **Función selección (FS):** Selecciona el siguiente vértice de la lista de candidatos.
- **Función de factibilidad (FF):** El nodo tiene hijos, es decir, no es una hoja.

Se requiere la matriz de adyacencia (MA) para saber si tiene o no hijos.

Se requiere una función auxiliar que pode el árbol en caso de que el nodo sea factible (los cambios se aplican sobre la MA)

- **Función objetivo:** Elegir el mínimo número de vértices posibles que constituyan un recubrimiento.

En este caso el algoritmo funciona de una forma completamente distinta: Se comienza seleccionando desde el nivel más bajo de derecha a izquierda, si ese nodo no es una hoja, se añade al conjunto de seleccionados, en caso contrario se descarta. Una vez que se selecciona un nodo, se poda toda la rama descendiente (él incluido), lo que nos permitirá aplicar el mismo proceso en los niveles superiores. Esto se repite hasta que el conjunto de candidatos está vacío (ya no hay más nodos).

El algoritmo en pseudocódigo quedaría de la siguiente forma:

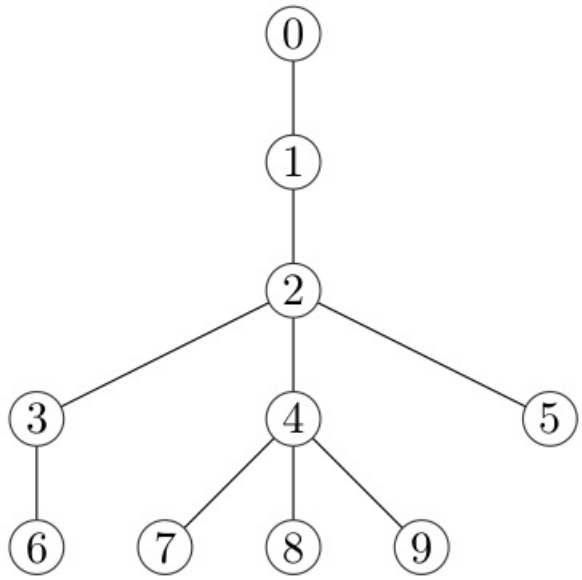
```
Recubrimiento minimal(C, MA)
    S = Ø

    mientras (C != Ø) hacer       //Función solución
        x = FS(C)                 //Seleccionar el más a la derecha
                                  //del nivel más profundo.
        C = C - {x}               //Eliminarlo del conjunto de candidatos.

        si FF(x, MA) entonces     //Si el vértice elegido es factible
                                  //(tiene al menos un hijo)
            S = S U {x}           //Añadir el vértice al conjunto de seleccionados.
            Podar(MA, x)          //Eliminar de la matriz de ayacencia
                                  //el subarbol decendiente de x.
        fin si
    fin mientras

    devolver S
fin
```

Además en este caso se ha optado por desarrollar el código que resuelva el problema en C++, el cual consta de una función principal y dos auxiliares para podar y comprobar si son nodos hoja:

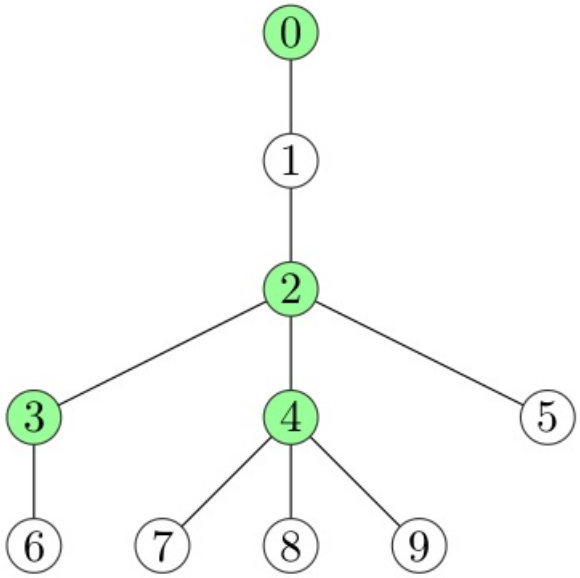
```
// Determina si un nodo es hoja o no, recibe la matriz de adaycencia y el tamaño.
bool isLeaf(int x, int** MA, int n){
  for(int i = 0; i < n; ++i)
    if(MA[x][i] == 1)
      return false;

    return true;
}
```

```
// Poda un subárbol. Aplica los cambios sobre la matriz de adyacencia.
void Podar(int** & MA, int x, int n){
  for(int i = 0; i < n; ++i)
    MA[x][i] = 0;

  int j = x-1;
  bool continua = true;
  while(j >= 0 && continua){
    if(MA[j][x] == 1){
      MA[j][x] = 0;
      continua = false;
    }
    j--;
  }


}

//Algoritmo voraz que busca el recubrimiento minimal de un arbol.
list<int> RecubrimientoMinimal(list<int> Candidatos, int** MA, int nodes){
  list<int> Solucion;
  int x;

  while(!Candidatos.empty()){
    x = Candidatos.front();
    Candidatos.pop_front();

    if(!isLeaf(x, MA, nodes)){
      Solucion.push_back(x);
      Podar(MA, x, nodes);
    }
  }

  return Solucion;
}
```

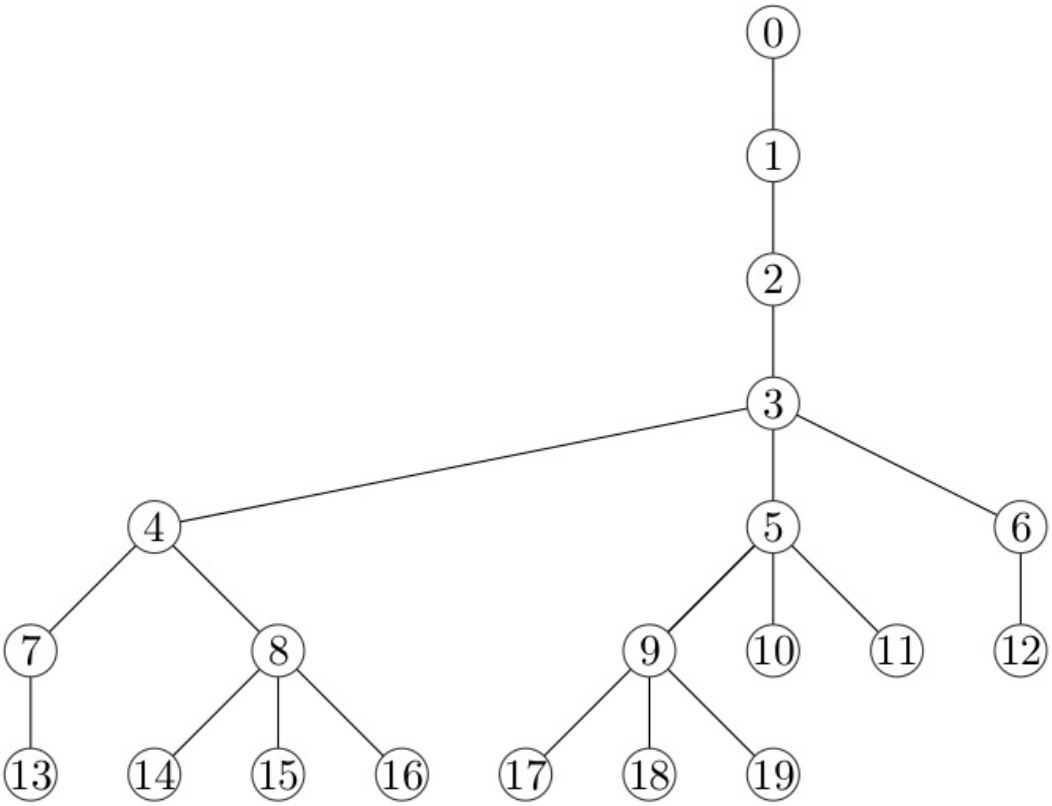
A continuación se muestran dos ejemplos con las soluciones que devolvería el algoritmo. Comenzaremos con el siguiente árbol de 10 elementos:



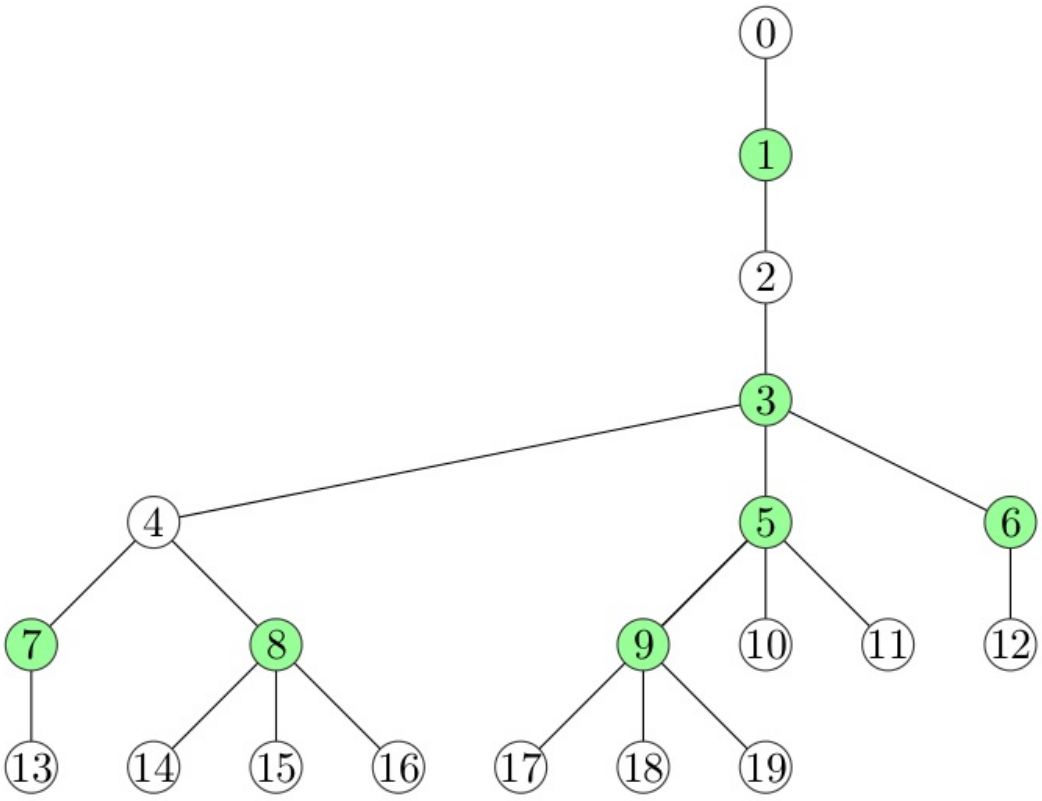
Cuando se resuelve con el algoritmo anterior, la solución obtenida es minimal:



Se procede a resolver con otro árbol, ahora de 20 elementos:



Se obtiene de nuevo la solución minimal al problema:



Según los resultados anteriores, siempre se obtiene la solución minimal. Esto se puede demostrar mediante reducción al absurdo:

> Sea A un árbol donde V es su conjunto de nodos, U su conjunto de aristas y S una solución cualquiera dada por el algoritmo implementado.
>
> - Supongamos que S no es una solución óptima, es decir, para el mismo árbol A existe una solución S' con al menos un nodo seleccionado menos:
>
> $|S| - |S'| > 0$
>
> - Si esto es así, podemos afirmar que al menos un nodo de S no aparece seleccionado en S':
>
> $\exists v \in V / v \in S \wedge v \notin S'$
>
> - Elegimos uno de esos nodos de S no seleccionados en S' y lo eliminamos de la solución. Además por propia implementación del algoritmo podemos asegurar que ese nodo no será una hoja, es decir siempre será un nodo interno.
>
> - Si v aparece seleccionado en S es porque tras podar el resto de los nodos seleccionados este tiene al menos un hijo, es decir existe al menos una hoja descendiente de v (llamémosla u). Como ya se ha mencionado anteriormente una hoja nunca es seleccionada por propia implementación del algoritmo, lo cual deriva en que la arista existente entre u y v no está cubierta. Si v ya no pertenece a la solución debemos seleccionar obligatoriamente a u. Dicho de otra forma, si v no pertenece a S', para que S' continúe siendo una solución, u debe pertenecer a S'. Esto también demuestra que S '$\not\subset$S , ya que v y u no pueden estar seleccionados simultáneamente en S:

$v \notin S' \Rightarrow u \in S'$ con $v, u \in V$ y $\{v, u\} \in U$

En el mejor de los casos v tiene un solo hijo, es decir la cardinalidades de S y S' se mantienen intactas, solo se permuta un nodo. Si esto se repitiese para todos los nodos de S no seleccionados en S' se tendría que las cardinalidades son forzosamente iguales:

$|S| - |S'| = 0$

En el peor de los casos, v tiene más de un hijo, es decir la cardinalidad de S' aumenta:

$|S| - |S'| < 0$

- En definitiva juntando ambos casos se tiene que:

$|S| - |S'| \leq 0$

lo cual supone una contradicción con lo postulado, puesto que o S y S' tienen la misma cardinalidad, o la cardinalidad de S es menor. En definitiva S es una solución óptima.