“Documentación Entrega 2 del proyecto”

Juan A. Jaramillo Penagos, Daniela Camacho, Yann Chové

Este archivo documenta la solución propuesta para la entrega 2 del proyecto de la clase Diseño y Análisis de Algoritmos

Universidad de los Andes, Bogotá, Colombia

Fecha de presentación: Mayo 3 de 2023

Tabla de contenido

1 Identificación de los autores

Juan Andrés Jaramillo P. Código: 201821305

Daniela Camacho. Código: 202110974

Yann Chové. Código: 202316916

2 Algoritmo de solución

Empezamos por mencionar que hubo 2 alternativas de implantación diferentes. La primera, se apoyó esencialmente de recorridos BFS sobre un grafo representado como una lista de adyacencias. Los recorridos se deben realizar para todos los vértices, es decir, desde cada posible vértice origen a cada posible vértice destino; el algoritmo también se apoya de una matriz de tamaño $|V| * |V|$ donde V: $vertices$ para guardar el tipo de conexión que tiene cada pareja de vértices (si es que tiene). La segunda se apoyó del algoritmo *Union-Set* implementado para encontrar el *MST* (árbol de recubrimiento mínimo) con la metodología de *Kruskal* para determinar si existe redundancia en la red. El algoritmo se utilizó específicamente para agrupar las nuevas conexiones (entrantes) con la red ya existente (asumiendo que la red existente puede o no estar conexa, y por lo tanto el grafo resultante puede representarse como un bosque con 2 o más arboles). Con lo anterior en mente, se decidió mantener 2 grafos, uno para representar la red conectada por cable óptico y otro para representar la red conectada por cable coaxial. En este caso, cada vez que se agrega una conexión se llama al método `Union()` y acto seguido se revisa si en la red global(óptico y coaxial) existe redundancia.

Al final de ambas implementaciones se decidió proceder con la segunda debido principalmente a sus mejores tiempos de ejecución. Dicho esto, a continuación se da una explicación más a detalle del segundo algoritmo:

El algoritmo está compuesto en 2 métodos principales y una clase interna de apoyo. El primer método, `ampliarRed()` se encarga de recibir todas las conexiones y agregarlas al grafo correspondiente a través del método `Union()` de la clase interna `Particion`; luego de agregar la conexión al grafo correspondiente, procede a revisar si existe redundancia en la red global (conformada por ambos grafos), para esto, hace uso del segundo método `testRedundancy()` el

cual se encarga de verificar que ambos grafos tengan el mismo número de vértices y que además tengan el mismo número de componentes conectados, puesto que si lo anterior no se cumple, se descarta la redundancia. En caso de que se cumplan ambas condiciones, el método procede a validar si cada componente conectado de un grafo, conecta los mismos computadores que algún componente conectado en el otro grafo, si encuentra un vértice para el cual no se cumple, entonces retorna false, de lo contrario retorna true y la red es redundante. Para lograr lo anterior, este método se apoya de un método de la clase interna `Particion` llamado `Find()` que retorna el vértice que representa el árbol al que pertenece un vértice dado.

Supuestos adicionales:

- Un computador no puede tener una conexión consigo mismo.
- No se permiten conexiones duplicadas.
- Una vez se agrega una conexión, esta no se modifica.

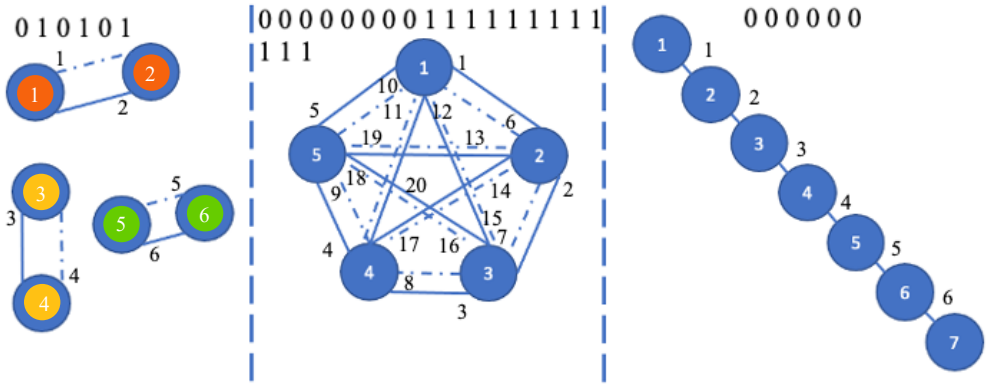

Figura 1. Posibles esquemas de redes probados con el algoritmo, el color de los vértices representa la partición a la que pertenecen.

3 Análisis de complejidad espacial y temporal

El enfoque del análisis de complejidad espacial y temporal será sobre los métodos principales del programa, es decir, los métodos `ampliarRed()` y `testRedundancy()`.

Como se ha mencionado anteriormente, el método principal `ampliarRed()` es el encargado de hacer el llamado a `testRedundancy()`, y por lo tanto su complejidad temporal está sujeta a la complejidad temporal de este último método.

Si se juntan todas las operaciones que son constantes bajo una sola constante $k = c_1 + c_2 + \cdots + c_n$ y se concentran los esfuerzos en determinar el mayor grado del polinomio que caracteriza la complejidad temporal del método principal `ampliarRed()`, se llega a una complejidad temporal del orden de $O(v)\ donde\ v\text{: }cantidad\ de\ vertices$, que se debe al llamado que hace este método al método de `testRedundancy()`. Por cuestiones de espacio, el detalle del análisis de complejidad temporal se dejó comentado en el código fuente entregado como anexo.

Vale la pena aclarar que dicha complejidad $O(v)$ es por cada vez que se inserta una conexión, y por lo tanto, si se quiere medir la complejidad temporal total observada en el programa al correr un archivo de prueba, esta seria del orden de $O(c * v)\ donde\ c\text{: }cantidad\ de\ conexiones$.

Por otro lado, con respecto al análisis de complejidad espacial se tiene lo siguiente: el programa utiliza 2 instancias de la clase `Partición`, cuya estructura principal son 2 arreglos de tamaño $v\text{: }numero\ de\ vértices$ a esto se le suma un arreglo adicional, también de tamaño $v\text{: }numero\ de\ vértices$ que contiene las ubicaciones de cada computador en la red global, y

finalmente se le deben sumar 2 tablas de hash también proporcionales a v utilizadas para optimizar la búsqueda de redundancias en la red. Dicho lo anterior, la complejidad espacial de este algoritmo es $E(v) = 4v + v + 2v = O(v)$.

4 Respuestas a los escenarios de comprensión de problemas algorítmicos

4.1 Escenario 1

El algoritmo diseñado mantiene un grado para cada tipo de conexión esto es representado por dos particiones en el código. Es por ello que en este escenario al añadir un nuevo tipo de conexión es necesario crear una nueva partición y por ende un nuevo grafo el cual va a representar la conexión inalámbrica.

Debido a este cambio, hay que generar más verificaciones al momento de responder si la red es redundante o no. Por ello, hay que modificar la función ampliar red para que esta tenga en cuenta la nueva partición generada, así mismo, al verificar la redundancia hay que comparar los 3 grafos y los componentes que se encuentran dentro cada uno de ellos. Esto generaría que la complejidad tanto temporal como en memoria aumentara debido a la nueva partición y lo que esta conlleva.

4.2 Escenario 2

En este escenario, se estaría eliminando la restricción de que todas las conexiones intermedias deben tener la misma tecnología, ya que solo necesitamos saber si las dos computadoras están conectadas, independientemente de la tecnología utilizada. Esto implica verificar la conectividad para cada par de computadoras en lugar de verificar la redundancia para cada posible par de computadoras y tecnología de conexión.

La implementación de un algoritmo que resuelva el escenario planteado es más simple que la del problema original y, teniendo en cuenta la solución implementada anteriormente, deberíamos verificar si todos los pares de nodos tienen el mismo representante, sin verificar el tipo de tecnología. El hecho de que los nodos tengan el mismo representante significa que se encuentran en el mismo componente conexo, por lo tanto, están conectados de forma directa o indirecta (a través de otros nodos). Esto se traduce al hecho de que se puede enviar un mensaje entre las computadoras representadas por estos nodos. Además, en lugar de utilizar dos grafos (uno para cada tipo conexión), se debería utilizar uno solo (para saber si hay o no una conexión). Por otro lado, la estructura de datos planteada no necesitaría ser modificada.