1. **IDENTIFICACIÓN:**

Camilo Anzola (201529838)

Santiago Rodríguez González ()

1. **ALGORITMO DE SOLUCIÓN:**

El programa que resuelve este problema lo que hace es, luego de recibir los datos de entrada, pone los nodos en dos conjuntos (de acuerdo a las conexiones dadas por los datos de entrada) tales que no haya conexiones entre dos nodos de un mismo conjunto y al final calcula la diferencia entre el tamaño de los dos conjuntos. Esto se logra de la siguiente manera:

Con los datos de entrada se llena un arrayList con todas las conexiones que forman el grafo G que es una BC-suma. Luego se procede a llenar dos arrayList que corresponden a los dos conjuntos (Conjunto X y conjunto Y) de acuerdo a la regla que dice que no puede haber una conexión entre dos nodos de un mismo conjunto. Para esto se consulta el arrayList de conexiones y se verifica que no hay conexiones entre el nodo que se quiere agregar a un conjunto y algún nodo que ya hace parte del conjunto. Al finalizar este procedimiento, todos los nodos se encuentran en alguno de los dos conjuntos y se retorna entonces la diferencia entre el tamaño de los dos conjuntos.

Este programa no hace uso de clases adicionales, es decir, no se modelaron las cosas con grafos ni nodos como tal, sino que se trabajó sobre el contenido de lo que serían los nodos (número) y se utilizó un arrayList de String que modela el conjunto de arcos. Esta decisión se tomó por simplicidad, ya que no se vio la necesidad de modelar utilizando la clase Grafo, o la clase Nodo o la clase Arco.

Cabe resaltar que había otra alternativa de solución que consistía en utilizar el algoritmo clásico DFS para verificar si el grafo es bipartito y entonces ir contando los elementos de cada conjunto para al final restar la cantidad de elementos de cada conjunto. En este caso sí hubiera sido necesario utilizar la clase Grafo, la clase Nodo y la clase Arco para efectuar DFS correctamente. Sin embargo, no se llevó a cabo de esta manera porque consideramos que implicaba una complejidad extra que en realidad sobra, ya que parte del problema es conectar los grafos de manera que quede al final un solo grafo que sea bipartito, por lo tanto, tener un algoritmo que verifique que el grafo sea bipartito es innecesario.

Otra opción que también tuvo consideración fue la de incluir la técnica de programación dinámica a la hora de calcular la diferencia entre los dos conjuntos de grafos. En este caso, los dos conjuntos se irían armando a medida que se construye la solución y al final se obtendría la solución final. Sin embargo, no se optó por esta vía porque no logramos definir completamente la función objetivo y por consiguiente no funcionaba el algoritmo en todos los casos, de manera que no era un problema fácil de modelar con programación dinámica así que optamos por una solución sencilla que sí funciona.

1. **ANÁLISIS DE COMPLEJIDADES:**

COMPLEJIDAD ESPACIAL:

Este programa hace uso de varios arrayList. Dos de ellos hacen referencia a los conjuntos de nodos, de los cuales el más grande podría ser a lo sumo de tamaño la cantidad de nodos de todos los grafos menos uno. Otro de los arreglos hace referencia a las conexiones, el cual puede ser a lo sumo de tamaño la suma de los arcos de todos los grafos multiplicado por dos. En cualquiera de los casos el espacio que ocupa en memoria depende de n (la cantidad de grafos).

COMPLEJIDAD TEMPORAL:

Restar el tamaño de los dos conjuntos no tiene mucha complejidad, sin embargo, agregar cada elemento a uno o a otro conjunto requiere recorrer el arrayList de conexiones y el arrayList del conjunto para verificar que no exista ningún elemento en el conjunto con una conexión al elemento que se quiere añadir. Este procedimiento se realizaría por cada nodo de cada grafo por lo tanto tenemos que:

Donde

x = cantidad total de nodos de todos los grafos

y = cantidad total de arcos de todos los grafos \* 2

Se multiplica por dos porque un arco representa dos conexiones (en ambos sentidos)

1. **COMENTARIOS FINALES:**

Si bien este programa no tiene una estrategia o una táctica que asegure eficiencia de respuesta, se puede observar un buen comportamiento a la hora de digitar los datos de entrada, incluso al probar con una gran cantidad de grafos (se probó con 70).