

Análisis de problemas en grafos.

Rie Kaneko Bojacá

Ingeniería de Sistemas
Pontificia Universidad Javeriana
Bogotá, Colombia

Felipe Jiménez Borrero

Ingeniería de Sistemas
Pontificia Universidad Javeriana
Bogotá, Colombia

Oscar David Virguez Castro

Ingeniería de Sistemas
Pontificia Universidad Javeriana
Bogotá, Colombia

Resumen. En el siguiente documento se analizarán tres diferentes problemas encontrados al manejar grafos en programación, para los cuales se identificarán sus componentes y definiciones, y se expondrán algunas posibles soluciones. Se tratarán problemas sobre: distancia y caminos más cortos entre nodos de un grafo, cliques en un grafo de Hamming, y árboles de expansión mínima.

Palabras clave— grafo; clique; distancia de hamming; árbol de expansión mínima; algoritmo.

I. INTRODUCCIÓN

Al trabajar con múltiples datos al momento de programar, se debe entender que en bastantes casos dichos datos van a estar relacionados de cierta manera. Las relaciones entre datos pueden ser bastante sencillas, de modo que los datos en conjunto se comporten como una lista; o bastante complejas, de forma que las relaciones entre los datos se expresen teóricamente como un grafo.

Al trabajar con un grafo de cualquier tipo, surgen problemas cuyas soluciones muchas veces pueden llegar a ser heurísticas, en vez de algorítmicas, o soluciones bastante complejas de por sí. Para dichos casos se pueden llegar a aproximaciones por medio de algoritmos, sin embargo, suelen ser problemas NP-completo, lo cual implica una muy alta complejidad.

A continuación, se presentarán tres problemas distintos de manejo de grafos, los cuales se analizarán para llegar a propuestas de soluciones a los mismos.

II. CLIQUES EN GRAFOS DE HAMMING

La distancia de Hamming $dist(u,v)$ entre dos vectores binarios $v=(v_1, v_2, v_3, \dots, v_n)$ y $w=(w_1, w_2, w_3, \dots, w_n)$ es el número de índices k tal v_k sea diferente de w_k . Una pregunta fundamental en la teoría de la codificación es determinar el número:

$$A(n,d) = \max\{|S| \subset \{0,1\}^n \mid dist(u,v) \geq d \text{ for all distinct } u,v \in S\}$$

El máximo número de vectores binarios de longitud n que uno puede encontrar tal que dos vectores diferentes tienen una distancia de Hamming mayor o igual a d .

Encuentre un algoritmo "eficiente" para calcular el clique máximo en el grafo de Hamming (calcular el clique máximo es un problema NP-difícil).

En esta problemática se busca encontrar el clique más grande dentro del grafo, dicho subgrafo se construye dependiendo de la distancia de Hamming mínima que se desea tener entre los dos vértices. Para construir este subgrafo, los vértices son conectados cuando la distancia de Hamming es mayor o igual al parámetro ingresado (d).

Un grafo es un conjunto de vértices (nodos) unidos por aristas (arcos), que relacionan los nodos, donde permiten representar relaciones binarias entre los elementos de un conjunto. El grafo se puede describir de la siguiente forma:

$$G = (V, E)$$

Donde G es un grafo, V son los conjuntos de vértices y E es un conjunto de aristas.

Existen dos tipos de grafos: el grafo dirigido y el no dirigido. El grafo dirigido tiene un sentido definido en las aristas. Esto hace que importe la dirección entre un nodo y otro de manera que $\{a, b\} \neq \{b, a\}$. En el grafo no dirigido no importa las direcciones de las aristas entre los nodos de manera que $\{a, b\} = \{b, a\}$. Para esta problemática se usa el grafo no dirigido ya que no nos interesa la dirección de cada nodo sino la cantidad de nodos que están conectados.

La distancia de Hamming es la cantidad de pasos necesarios para encontrar la diferencia entre los dos datos. Por ejemplo, si se tiene una palabra "Casa" y otra palabra "Mapa" entonces la distancia de Hamming sería de 2 ya que su diferencia es de dos letras. En este problema se maneja la misma lógica, pero con números binarios, un ejemplo para este caso sería "100" y "000" en el cual su distancia de Hamming es de 1.

El código de Hamming es un código para la corrección de errores que se puede representar como grafos no dirigidos. En redes de comunicación se usa este código para verificar si los datos que se transmiten de un computador a otro llegan correctamente, y el caso de no lo se corrigen. Para ello se usa la siguiente fórmula:

$$2^n \geq n + m + 1$$

Donde n es el número de bits de Hamming y m es el número de bits del dato. Este es uno de los usos que se le da a la distancia de Hamming en la vida real.

La construcción del grafo de Hamming se hace de la siguiente manera:

1. Elegir la dimensión: Esto significa que se escoge el n donde n es la cantidad de dígitos de los números binarios.
2. Sacar todas las posibles combinaciones: Se debe sacar todas las posibles combinaciones de 1's y 0's dentro de las dimensiones elegidas, es decir el total de combinaciones es (2^n).
3. Elegir la distancia mínima de Hamming: Se decide que distancia de Hamming mínima para hacer las conexiones se desea tener.
4. Conectar los nodos: Conectar los números binarios que tengan dicha distancia de Hamming. En la Figura 1 se puede ver un ejemplo de 3 dimensiones con distancia de Hamming igual a 1.

3 Dimensiones

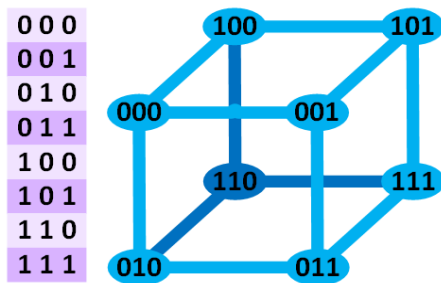


Figura 1, tomada de [4]

En la teoría de grafos, el clique es un subconjunto de vértices donde están conectados todos con todos creando un grafo completo.

Cuando se habla del clique máximo, hace referencia al clique que tiene mayor número de vértices, por lo tanto, en el problema que se trabaja, se busca encontrar el clique que tenga más nodos dependiendo de la distancia de Hamming ya que es de ella que se crean las aristas.

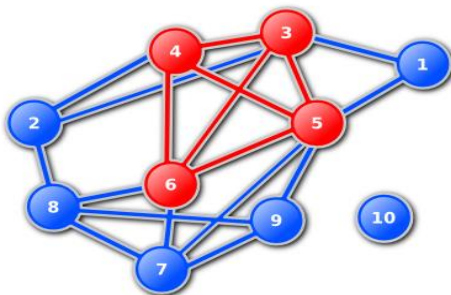


Figura 2, tomada de [5]

El cálculo del clique máximo es un problema NP-difícil. Para esto es posible construir un algoritmo que trabaje en tiempo polinómico para cualquier problema de NP, este es el objetivo del proyecto, encontrar un algoritmo que solucione el problema planteado.

A continuación, se plantearán algunos ejemplos y posibilidades de solución.

Ejemplo 1. Sea $A(n, d)$ donde n es la cantidad de dígitos de los números binarios y d la distancia de Hamming mínima que se desea, encontrar el número de clique máximo cuando $A(3, 2)$.

Diseño del grafo

- i. Encontrar las dimensiones

Como $n = 3$, entonces la es de 3 dimensiones.

- ii. Sacar todas las posibles combinaciones entre 0 y 1

Las combinaciones posibles son de $2^3 = 8$.

000	100
001	101
010	110
011	111

- iii. Encontrar la distancia de Hamming mínima

Como $d = 2$, entonces su distancia mínima es de 2.

- iv. Conectar nodos

Conectar los datos (punto ii), entre los que tengan distancia de haming mayor o igual a 2.

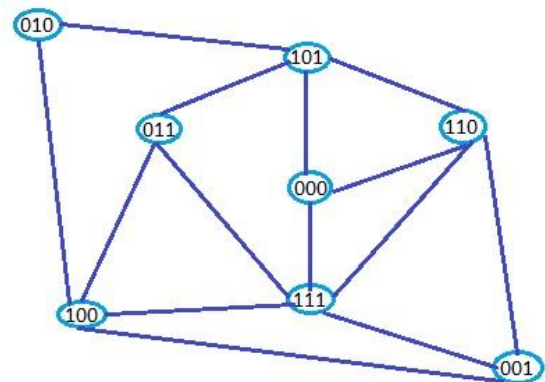


Figura 3

Ejemplo 2. Sea $A(n, d)$ donde n es la cantidad de dígitos de los números binarios y d la distancia de Hamming mínima que se desea, encontrar el número de clique máximo cuando $A(4, 4)$.

Diseño del grafo

i. Encontrar las dimensiones

Como $n = 4$, entonces la es de 4 dimensiones.

ii. Sacar todas las posibles combinaciones entre 0 y 1

Las combinaciones posibles son de $2^4 = 16$.

0000	1000
0001	1001
0010	1010
0011	1011
0100	1100
0101	1101
0110	1110
0111	1111

iii. Encontrar la distancia de Hamming mínima

Como $d = 4$, entonces su distancia mínima es de 4.

iv. Conectar nodos

Conectar los datos (punto ii), entre los que tengan distancia de haming mayor o igual a 4.

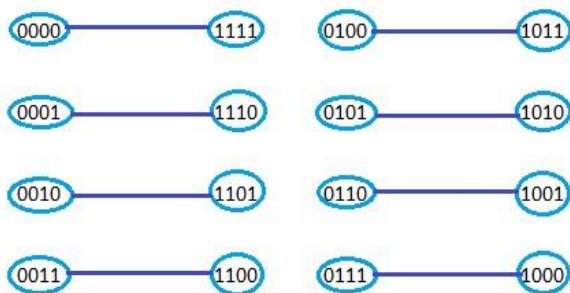


Figura 4

Solución. Encontrar el clique máximo

Para encontrar los cliques máximos se puede usar el Algoritmo de Bron-Kerbosch que consiste en pararse inicialmente en cualquier nodo y buscar un nodo adyacente, este proceso se realiza iterativamente siempre buscando nodos que tengan conexión con el clique formado he intentado mezclar cliques para encontrar otros más grandes.

III.

PROBLEMA DE DISTRIBUCIÓN DE TIENDAS

El problema consiste en que se planea construir en una ciudad una cadena de tiendas, a partir de unas ubicaciones estratégicas previamente identificadas en diferentes barrios. Se conoce también con las ubicaciones, la demanda de productos en cada uno de los barrios. Se quieren construir k tiendas, con lo cual se formulan las siguientes preguntas:

- ¿Dónde se deben localizar las tiendas para lograr minimizar la distancia promedio entre los clientes?
- Asumiendo que se conoce el costo de construcción de cada una de las tiendas, y se cuenta con la opción de variar el número de tiendas a construir, ¿En qué ubicaciones se deben construir las tiendas para lograr minimizar la distancia promedio entre los clientes y el costo total de producción?

Para dar solución a esta problemática desde una perspectiva algorítmica, debemos adaptarla a una estructura de datos. Para esto definiremos los conceptos necesarios.

Árbol de expansión: Es un árbol que recorre y conecta todos los nodos del grafo desde un nodo inicial teniendo en cuenta el costo o distancia de las aristas de tal manera que no haya ciclos dentro del mismo, este tipo de árboles generalmente son de tipo mínimos o máximos (suma de los costos o distancias de las aristas).

En este problema se hace indispensable el uso de grafos, para su representación más específicamente un grafo no dirigido, en donde los nodos o vértices son los puntos donde se encuentra la demanda y las aristas son los costos de desplazamiento. Es un problema muy relacionado con la vida cotidiana, un ejemplo de este es cuando una empresa quiere construir una o más plantas de producción en una determinada ubicación, puede ser ciudad o barrio, dicha localización no puede ser escogida de manera aleatoria, ya que dependiendo de las necesidades de la empresa, tiene que tener en cuenta factores como la cercanía de la o las plantas con los clientes o en dado caso, los costos de su construcción haciendo indispensable reducir gastos sin afectar la calidad de las plantas, es aquí donde surge la necesidad de encontrar los puntos más eficientes dentro del grafo.

A continuación, se plantearán algunas posibles soluciones a este problema.

a.

Árbol de expansión:

Una de las posibles soluciones para este problema es haciendo el uso de un árbol de expansión, en donde se analiza todo el grafo cada vez que se desee insertar una nueva planta, dicho árbol se crearía mediante el algoritmo de Prim, al cual deberá hacerse unas determinadas adaptaciones para el caso de que se desee construir una cantidad variable de plantas minimizando su

costo de construcción. Debido a que el algoritmo trabaja sobre las aristas, al haber costos de construcción debe también analizarse los valores de cada nodo. [1]

b. Algoritmo ADD

Es el algoritmo que más se ajusta a las necesidades del problema, ya que este si analiza el costo fijo (Costo de construcción), la demanda asociada a cada vértice y el costo variable (distancia entre vértices), dicho algoritmo fue propuesto por John Wiley en el año 1945.

En el caso de que se desea abrir una cantidad variable de tiendas su proceso es:

1. Se busca el vértice que minimiza los costos fijos y variables de todo el grafo (Árbol de expansión mínima) y se realiza la primera construcción en dicho punto.
2. Se realiza el proceso iterativo buscando más puntos en donde se pueda atender la demanda teniendo en cuenta los costos de las aristas.
3. Se continua el ciclo hasta que deje de existir un costo mínimo.

ADD trabaja creando una matriz de costos fijos y variables:

- Los costos fijos en este caso son dados con anterioridad para cada nodo
- Los costos variables están dados por la siguiente expresión:

$$\frac{\text{distancia entre nodos}}{\text{demanda}} = \text{Costo variable}$$

Una vez creada la matriz con los costos variables para cada par de nodos, se realiza la sumatoria de costes totales para cada nodo y se escoge el de menor valor, este sería el punto inicial de la construcción.

Para cada nueva construcción que se desea realizar es necesario recrear la matriz de costos debido a que, al existir plantas anteriores los costos variables van disminuyendo.

En el caso de que se desea abrir una cantidad K de tiendas su proceso es muy parecido al anterior, con la diferencia de que el proceso iterativo se realiza k veces. [1] [3]

Según este problema, se cuenta con un grafo, el cual podríamos representar teóricamente en una red telefónica, con m aristas y n vértices, en donde cada arista puede ser azul o roja. En los parámetros del problema hay también una variable numérica k , se debe encontrar a partir del grafo un árbol de expansión mínima que cuente con exactamente k aristas azules y $n-k-1$ aristas rojas. De el algoritmo que solucione este problema se debe evaluar su tiempo de ejecución y demostrar su correctitud.

Para entender bien los conceptos de este problema, se usarán las definiciones de grafo y de árbol de expansión mínima previamente definidas en este texto.

El algoritmo que dé solución a este problema debe contar con las siguientes entradas:

- Datos equivalentes a los vértices, cuyo contenido no es relevante dentro del problema.
- Matriz de relación, de tamaño $n \times n$ (considerando que el grafo tiene n vértices) en donde se indique qué vértices se relacionan entre si y de qué color es su relación (azul o roja).
- Variable k que indique al algoritmo, después del respectivo análisis, cuántas aristas azules y cuántas rojas tendrá que tener el árbol resultado.

El grafo de entrada, representado teóricamente tendrá que lucir como el siguiente ejemplo:

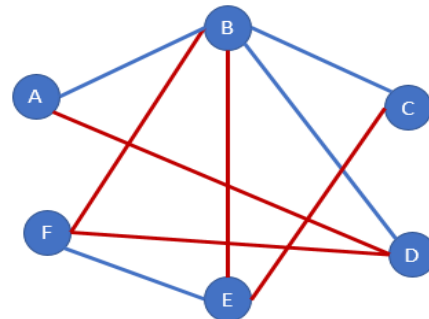


Figura 5

El algoritmo debe tener como única salida un conjunto de datos equivalentes a los datos de los vértices, junto con una matriz de relación en la cual se especifica qué vértices se relacionan entre sí y con qué color, de manera que formen un árbol de expansión mínima con las características requeridas por quién define la variable k .

Para el ejemplo de grafo de entrada mostrado anteriormente, una posible salida podría ser la siguiente:

IV. ÁRBOL DE EXPANSIÓN MÍNIMA

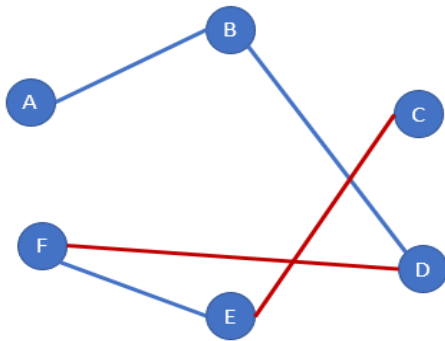


Figura 6

Para dar solución a la creación de árboles de expansión mínima a partir de un grafo existen diferentes algoritmos actualmente. Para este problema en particular, se debe tener en cuenta que las soluciones que se implementen, aunque puedan basarse en algún algoritmo ya existente, deben evaluar como condiciones adicionales para la construcción del árbol, los límites de aristas de cada tipo dependiendo del valor de la variable k .

Un algoritmo sobre el cual se puede establecer la base de la solución a este problema es el algoritmo de Kruskal, el cual es un algoritmo voraz. Este algoritmo se basa bajo el principio de agregar aristas al árbol siempre que ellas no formen un ciclo. Además, el algoritmo ordena las aristas de menor a mayor para trabajar también con los costos de las mismas. Un pseudocódigo del algoritmo sería el siguiente [9]:

```

1 método Kruskal(Grafo):
2     inicializamos MST como vacío
3     inicializamos estructura de unión
4     ordenamos las aristas del grafo
5     para cada arista e que une los vértices u y v
6         si u y v no están en la misma componente
7             agregamos la arista e al MST
8             hacemos la unión de las componentes de u y v

```

Para este caso en particular, en el que fuera del simple método del árbol de expansión, requerimos también verificar los colores de las aristas, se haría necesario modificar algunas partes de este proceso.

Primero, se tendrían que descartar los elementos del algoritmo que no son relevantes al problema en particular. Para este caso, el árbol no está trabajando con costos, así que el ordenamiento de las aristas del grafo se puede descartar sin afectar negativamente nuestro resultado.

Adicionalmente tenemos que añadir las condiciones que la inserción de la variable k implica. Debemos tener en cuenta el añadir como parámetro a la función la variable k , con la cual, dentro de la función, declaramos una variable $x = k$, la cual determinará el número de aristas azules; y una variable $y = k - n - 1$, la cual determinará el número de aristas rojas. Al momento de entrar a la condición interna, se deben añadir otras instrucciones, al agregar una arista al árbol, y antes de hacer la unión de los componentes de u , se debe restar en 1 el valor de x si la arista agregada es azul, o restar en 1 el valor de y si la arista agregada es roja. Por último, para que entre a la condición interna se debe evaluar también, que, si la arista a agregar es roja, el valor de y sea mayor a 0, y que, si la arista a agregar es azul, el valor de x sea mayor a 0.

De esta manera se podrá garantizar que el árbol de expansión mínima que se obtiene, cumpla con las características dadas por la variable k en los parámetros de la función.

V. REFERENCIAS

- [1] M. Soto, «YouTube,» 7 Noviembre 2014. [En línea]. Available: <https://www.youtube.com/watch?v=IB2rGgMWnM0>. [Último acceso: 15 Febrero 2017].
- [2] «YouTube,» 28 Diciembre 2008. [En línea]. Available: <https://www.youtube.com/watch?v=Kh60Juq-hwM>. [Último acceso: 2017 Febrero 15].
- [3] «SciELO,» Abril 2004. [En línea]. Available: http://www.scielo.br/scielo.php?script=sci_arttext&pid=S0101-74382004000100008. [Último acceso: 2017 Febrero 15].
- [4] R. Invarato, «Jarroba.com,» 9 Noviembre 2016. [En línea]. Available: <https://jarroba.com/hamming/>. [Último acceso: 20 Febrero 2017].
- [5] R. González, «Blogger,» 5 Julio 2011. [En línea]. Available: <http://esteban-gzz.blogspot.com.co/2011/07/problema-de-clique.html>. [Último acceso: 20 Febrero 2017].
- [6] L. Alcón, «Universidad Nacional de La Plata,» Marzo 20013. [En línea]. Available: http://www.mate.unlp.edu.ar/tesis/tesis_liliana.pdf. [Último acceso: 20 Febrero 2017].
- [7] Wikipedia, «Wikipedia,» 11 Febrero 2017. [En línea]. Available: https://en.wikipedia.org/wiki/Bron%E2%80%93Kerbosch_algorithm. [Último acceso: 21 Febrero 2017].

- [8] Wikipedia, «Wikipedia,» 17 Mayo 2016. [En línea]. Available:
https://es.wikipedia.org/wiki/Problema_de_la_clique.
[Último acceso: 21 Febrero 2017].

- [9] J. Arias «Algorithms and more» 19 Abril 2012. [En línea]. Available:
<https://jariasf.wordpress.com/2012/04/19/arbol-de-expansion-minima-algoritmo-de-kruskal/>
[Último acceso: 20 Febrero 2017].