

# Métodos algorítmicos para problemas comunes

Cely Baez Marvin Daniel, Rojas Valbuena Laura Camila

**Abstract**—El presente artículo realizó una contextualización teórica respecto a las definiciones de los algoritmos que fueron utilizados a lo largo del desarrollo de los tres problemas propuestos. Luego de esto, se desarrollaron soluciones dichos problemas a partir de diferentes algoritmos existentes y propios.

**Index Terms**—Arbol de expansión, algoritmo de Prim, algoritmo de Kruskal, grafo de Hamming, cliques en grafos.

## 1 INTRODUCCIÓN

EN ESTE artículo se desarrollan soluciones a 3 problemas propuestos a partir de diferentes algoritmos existentes y propios. Previo a estas soluciones, se dan definiciones de los algoritmos que pueden ser utilizados a lo largo del desarrollo de éstas. Finalmente, para esta entrega definimos teóricamente el paso a paso

### 1.1 Descripción de los algoritmos a utilizar

Descripción de los algoritmos que se van a utilizar para resolver los diferentes problemas propuestos:

#### 1.1.1 Arbol de expansión

La búsqueda siempre avanza hacia un vértice no marcado, internándose profundamente en el grafo sin repetir ningún vértice. Cuando se alcanza un vértice cuyos vecinos han sido marcados, se retrocede al anterior vértice visitado y se avanza desde éste[7].

Sea  $G(V, E)$  un grafo conexo y  $v$  un vértice de  $V$ . El algoritmo de búsqueda en profundidad puede detallarse así:

1. Se comienza en un vértice  $v$  (vértice activo) y se toma como la raíz del árbol generador  $T$  que se construirá. Se marca el vértice  $v$ .

2. Se elige un vértice  $u$ , no marcado, entre los vecinos del vértice activo. Si no existe tal vértice, ir a 4. 3. Se añade la arista  $(v, u)$  al árbol  $T$ . Se marca el vértice  $u$  y se toma como activo. Ir al paso 2.4. Si se han alcanzado todos los vértices de  $G$  el algoritmo termina. En caso contrario, se toma el vértice padre del vértice activo como nuevo vértice activo y se vuelve al paso 2. La complejidad de este algoritmo es  $O(\max n, m)$

#### 1.1.2 Algoritmo de Prim

El algoritmo de Prim también conocido como el (Minimum Spanning Tree), ya que es uno de los métodos más sencillos de implementar. El algoritmo evita repetir cálculos de forma excesiva agregando una estructura de datos simple, ideal para grafos densos[6].

El algoritmo evita repetir cálculos de forma excesiva agregando una estructura de datos simple, ideal para grafos densos. De manera que  $V$  es el conjunto de nodos de un grafo pesado no dirigido. El algoritmo de Prim comienza cuando se asigna a un conjunto  $U$  de nodos un nodo inicial perteneciente a  $V$ , en el cual crece un árbol de expansión, arista por arista. En cada paso se localiza la arista más corta  $(u,v)$  que conecta a  $U$  con  $V-U$ , y después se agrega  $v$ , el vértice en  $V-U$ , a  $U$ . Este paso se repite hasta que  $V=U$ . El algoritmo de Prim es de  $O(N^2)$ , donde  $|V| = N$ .

Para empezar elegimos el nodo 0 y se apilan las aristas adyacentes a este nodo, en orden decreciente. La arista mínima es la que está en el tope de la pila, así que se desapila y se agrega al MST. Seguidamente se procede con el nodo 1 y se apilan sus aristas adyacentes, con la diferencia de que hay que verificar si dichas aristas representan un nuevo camino mínimo con respecto a las aristas que ya están introducidas en la pila. En este caso no se apila la arista 1-3 porque ya hay una arista que lleva a 3 con el mismo costo en la pila. Se toma 1-2 porque está en el tope y se procede con el nodo 2. Cuando se va a apilar la arista 2-3, se encuentra que ya hay un camino que lleve a 3 pero de mayor costo, así que se desapila 0-3 y luego se apila 2-3. Se agrega 2-3 al MST y termina el proceso, porque el conjunto de nodos en el MST es igual al conjunto de vértices del grafo original.

#### 1.1.3 Algoritmo de Kruskal

El algoritmo comienza a partir de un conjunto de árboles degenerados formados por un solo nodo, que son los nodos del grafo, y se comienzan a combinar los árboles de dos en dos usando la arista menos costosa posible, hasta que solo quede un solo árbol: El MST. Dada una lista de las aristas del

grafo, el primer paso del algoritmo de Kruskal es ordenarlas por peso (usando un quicksort por ejemplo). Luego se van procesando las aristas en el orden de su peso, agregando aristas que no produzcan ciclos en el MST. El algoritmo de Kruskal es de  $O(A \log A)$ , donde  $A$  es el número de aristas del grafo.

#### 1.1.4 Grafo de Hamming

Una red de computadores puede ser fácilmente representada por un gráfico, en el que cada máquina es un vértice

y los bordes son conexiones entre estas máquinas. En este gráfico, cada vértice tiene una etiqueta que sería la dirección de la máquina en la red. Sin embargo, estos marcadores pueden ser direcciones y reflejar directamente la distancia entre los vértices (máquinas). En una red cuya dirección sigue esta regla, el enrutamiento de paquetes se puede hacer mediante el análisis de forma óptima única información local. De un paquete siempre se conoce la dirección de destino. Cuando se llega a un vértice que compara esta dirección con el vértice vecino, va a lo que está más cerca de su meta[3]. Un gráfico que satisface esta propiedad es

n-cubo. Aquí, la distancia entre los vértices es exactamente la distancia de Hamming (o la distancia de edición) entre la secuencia de bits asignado a cada vértice. Para construir una red de esa manera se debe saber qué tipo de gráfico puede tener sus vértices dirigidos con las cadenas de bits de modo que la distancia de Hamming corresponde a la distancia entre los vértices en el grafo. Teorema (Djokovic -

1973):

Hay un esquema de direccionamiento utilizando cadenas binarias de los vértices de un grafo simple  $G$  tal que la distancia de Hamming de las cadenas coincide con la distancia de los vértices en el grafo si y sólo si:

- $G$  es bipartita.
- $G(b)$  es un conjunto cerrado para cada arista  $(a, b)$  a  $(g)$ .

Nota:  $G(b) = V \times (L) — d(x, a) = d(x, b) - 1$  r cerrado

para cualquier par  $a, b$  en  $T$ :  $x$  en  $V(G)$  y  $d(a, x) + d(b, x) = d(a, b)$  a continuación,  $x$  está en  $U$ .

#### 1.1.5 Cliques en grafos

El problema de clique es el siguiente: Dado un grafo no dirigido  $G$ , y un número natural  $k$ , determinar si  $G$  posee un clique de tamaño  $k$ .

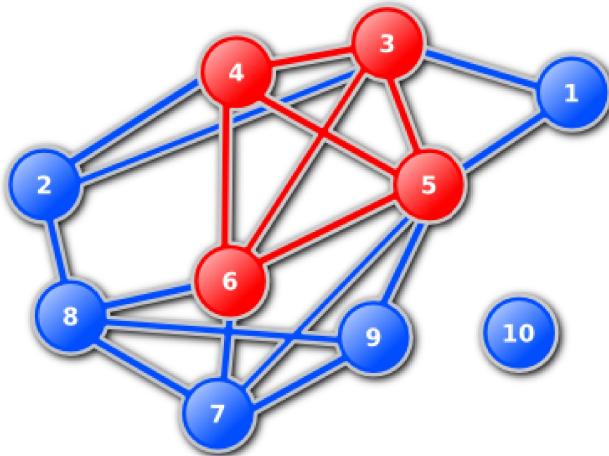


Figura 1: dado un grafo no dirigido  $G$ , y un número natural  $k$ , determinar si  $G$  posee un clique de tamaño  $k$  [8].

Dado un grafo no dirigido cualquiera  $G = (V, E)$ , en el cual  $V = 1, 2, \dots, n$  es el conjunto de los vértices del grafo y  $E$  es el conjunto de aristas. Un clique es un conjunto  $C$  de vértices donde todo par de vértices de  $C$  están conectados con una arista en  $G$ , es decir  $C$  es un subgrafo completo.

Este problema se puede enunciar como un problema de decisión si la pregunta que se hace es saber si existe una clique de tamaño  $k$  en el grafo. El correspondiente problema de optimización, consiste en encontrar una clique de tamaño máximo en un grafo.

Una vez que tenemos  $k$  o más vértices que forman una clique, es trivial verificar que lo son, por eso es un problema NP [8].

**Clique:** El término proviene de la palabra inglesa clique, que define a un grupo de personas que comparten intereses en común. En esta analogía, las personas serían los vértices y los intereses en común, las aristas. Cuando todas compartan un mismo interés, forman un grafo completo, es decir, forman un clique.

Un clique en un grafo no dirigido  $G$  es un conjunto de vértices  $V$ , tal que para todo par de vértices de  $V$ , existe una arista que las conecta, donde el tamaño de un clique es el número de vértices que contiene [8].

Ejemplo:

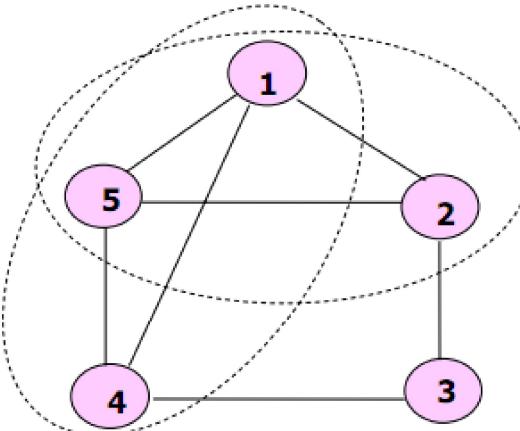


Figura 2: Cliques en grafos [8].

El grafo  $G$ :

- cliques 1,2,5 y 1,4,5 de tamaño 3.
- cliques 2,3 y 3,4 de tamaño 2.

Algoritmos para resolver este problema

- Algoritmo de fuerza bruta

Un ejemplo de algoritmo de fuerza bruta para encontrar una clique en un grafo consiste en listar todos los subconjuntos de vértices  $V$  y verificar para cada uno de ellos si forma una clique.

Ese algoritmo es polinómico si  $k$  es una constante, pero como no lo es para este caso, tenemos un exponencial de  $n$  a la  $k$ .

- Algoritmo Bron-Kerbosch

Este algoritmo consiste en arrancar con cliques de un solo elemento e intentar mezclar cliques para obtener otras más grandes, hasta que no queden más mezclas por intentarse. Dos cliques pueden ser mezcladas si cada nodo de la primera es adyacente a cada nodo de la segunda.

Es eficiente en el peor de los casos por un resultado de Moon and Moser, donde un grafo de  $n$  vértices tiene a lo sumo 3 a la  $(n/3)$  cliques máximos, y el tiempo de ejecución del peor caso del algoritmo Bron-Kerbosch, con una estrategia dinámica que reduce al mínimo el número de llamadas recursivas realizadas en cada paso, es  $O((3^{(n/3)})^3)$ , que coincidan con este límite.

Para estos dos algoritmos es fácil saber cuál es mejor que el otro. Tenemos que:

- El mejor es Bron-Kerbosch.
- El peor es Fuerza bruta.

## 1.2 Cadena de tiendas: distribución óptima

El objetivo es construir tiendas seleccionando barrios que cumplan unos criterios estratégicos.

Para representar este problema se optó por usar grafos, donde el vértice representa el barrio y el costo de construcción (ver Caso 2), y la arista representa la distancia entre barrios (ver figura 3):

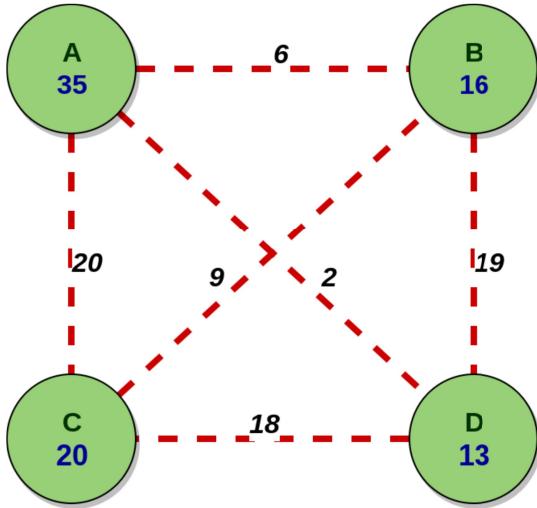


Figura 3: Representación grafo de barrios

Para encontrar los barrios donde se construirán las tiendas se tendrán en cuenta los siguientes casos:

- Caso 1: El criterio a seguir es construir un cierto número de tiendas donde minimice la distancia promedio de los clientes
- Caso 2: Se usa el mismo criterio del caso 1 y adicionalmente se selecciona la tienda minimizando el costo de construcción según el barrio donde esté ubicado

### 1.2.1 Algoritmo caso 1:

1. Lectura de datos del grafo (Ver figura X1), crear matriz de adyacencia y guardar cantidad de tiendas a construir.

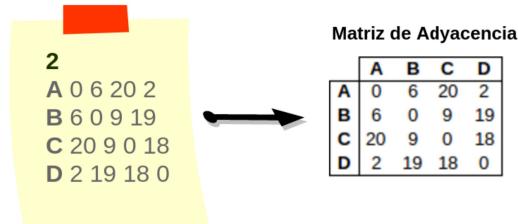


Figura 4: Lectura de archivos

2. Seleccionar tiendas 2.1. Calcular promedio entre la sumatoria de las distancias de un vértice dado contra todos los vértices restantes (Ver Imagen x2)

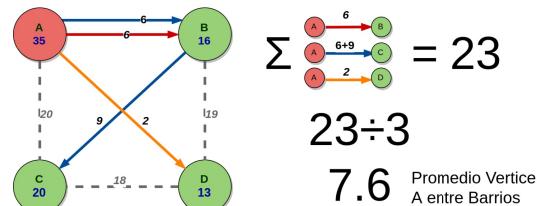


Figura 5: Promedio vértice A entre barrios Nota: Para seleccionar el camino de un vértice a otro vértice, se hace uso del algoritmo Dijkstra.

- 2.2. Repetir proceso para cada vértice
- 2.3. Del numero de tiendas solicitadas a construir, de la lista de promedios obtenidas seleccionar los promedios mas bajos.
- 3 Calcular distancias
- 3.1. Seleccionar vértices donde están las tiendas ubicadas y sumar sus promedios obtenidos del inciso 2.1.

### 1.2.2 Algoritmo caso 2:

1. El algoritmo es igual al caso 1, con la diferencia que en el promedio de cada vértice se saca un factor racional con los costos de construcción, es decir que se divide cada promedio obtenido por los costos de construcción de cada barrio.

Análisis: Como se utiliza recorridos de matrices con complejidad de  $O(n^2)$ , además del método Dijkstra que también tiene una complejidad de  $O(n^2)$ . Se concluye que todo el algoritmo tiene complejidad  $O(n^2)$

## 1.3 Coloreamiento de aristas

### 1.3.1 Abrir el archivo

Teniendo como base un grafo  $G=(V,E)$  con  $m$  vértices y  $n$  aristas, identificamos las siguientes variables que intervienen en el desarrollo del problema:

- Creamos un grafo vacío que será llenado posteriormente
- Por la forma de lectura que implica el formato de entrada, se debe tener un contador auxiliar para filas y otro para columnas.
- Si se puede abrir el archivo, se guarda la información por líneas, es decir, separando cada trozo del archivo.
- Reemplazamos inf por el número 0 para facilitar las validaciones posteriores.
- Separamos cada valor a partir del carácter "/" y guardamos el color de la arista (B/R) y su peso.
- En dos contadores, guardamos la cantidad de aristas rojas y azules que hay en total para hacer los cálculos posteriores.
- Se actualiza el grafo
- Se cierra el archivo
- Se retorna el grafo, el número de nodos y el k o aristas azules.

### 1.3.2 Definición de la clase constructorAr

Se inicializan los siguientes parámetros con el fin de poder utilizarlos más adelante en los demás métodos:

- Vértice inicio Determina donde se inicia el recorrido.
- Vértice fin Determina donde finaliza el recorrido.
- Color (azul o rojo) Determina el color de la arista.
- Peso de la arista Es el valor asociado a cada una de las aristas o como su nombre lo indica, el peso de las mismas.

### 1.3.3 Definición del método tieneSolucion(G,n,k)

Se verifican las siguientes condiciones:

- No debe haber más aristas azules que rojas o viceversa, depende del K Por lo menos tiene que tener K aristas azules ya que si tiene menos, el problema no tiene solución.
- Entra un K que define la cantidad de aristas azules que tiene el grafo inicial.
- Ejemplo: K=2 azules N=8 vértices N-K-1=8-2-1=5 El grafo inicial debe tener al menos 5 aristas rojas para que el problema tenga solución.

Luego de verificar las condiciones mencionadas anteriormente, se desarrolla el algoritmo a partir de los siguientes pasos:

- Recorrer las aristas con un ciclo for.
- Recorrer los vértices con un ciclo for.
- Contar la cantidad de aristas azules del grafo inicial.
- Contar la cantidad de aristas rojas del grafo inicial.
- Se debe verificar que se cumplan las siguientes condiciones para que el problema tenga solución:
- k debe ser menor o igual que la cantidad de aristas del grafo inicial. Si esto se cumple, se procede a calcular el nuevo número de aristas rojas que tendrá el grafo.
- CalculoRojo = número de nodos - número de aristas azules - 1 CalculoRojo = n - k - 1

- Si la variable CalculoRojo es menor o igual a la cantidad de aristas que tiene el grafo inicial, el problema tiene solución y el algoritmo retorna true. Si CalculoRojo fuera mayor, no se podría resolver pues el grafo no tendrá aristas rojas suficientes para colorear. En este caso, el algoritmo retorna false.

### 1.3.4 Definición del método burbuja(informacion)

Se organizan las aristas a partir de sus pesos utilizando el algoritmo de ordenamiento burbuja.

### 1.3.5 Definición del método organizarAr(G)

Para organizar las aristas, se deben llevar a cabo los siguientes pasos:

- Recorrer las aristas
- Recorrer los vértices
- Asignar el color R (rojo) ó B (azul)
- Agregar la información
- Organizar con el método burbuja
- Retornar la información

### 1.3.6 Definición del método minExp(G,n,k)

Si la verificación de que el problema tiene solución se cumple, se procede a realizar los siguientes pasos:

- Se organizan las aristas por peso
- Se construye el árbol final
- Se retorna el árbol final

### 1.3.7 Definición del método resultadoFinalArbol(informacion,k,n)

En este método se compila la información que conforma la solución final del problema. Para esto es necesario definir las siguientes variables:

- Nuevo valor azules
- Nuevo valor rojas
- Contador aristas azules
- Contador aristas rojas
- Definir un árbol

Finalmente, se lee el árbol y a continuación se obtiene el resultado (nuevo árbol y peso total) siempre y cuando el mismo cumpla con las restricciones iniciales.

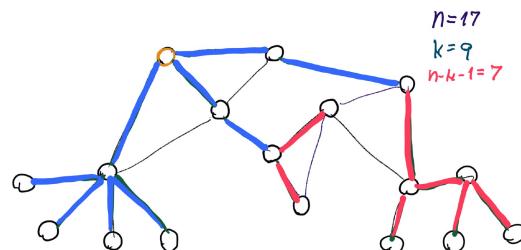


Figura 6: Algoritmo comprobado

Comprobar que al reemplazar los valores de  $n$  y  $k$  en la ecuación  $n-k-1$ , el resultado sea igual a  $n$ . Es decir,

$$k + (n-k-1) = n$$

De esta manera, está solucionado el algoritmo coloreamiento de aristas.

#### 1.4 Cliques en Grafos: distancia de Hamming

El objetivo es construir un algoritmo que calcule el clique máximo en el grafo de Hamming teniendo en cuenta que el clique máximo es un problema NP-difícil. Dicho esto los grafos de Hamming inicialmente serán grafos completos (Figura 1) dependiendo de la dimensión del grafo que se solicite  $n$  tendrá un numero de aristas que serán las que tengan una distancias de Hamming  $d$ .

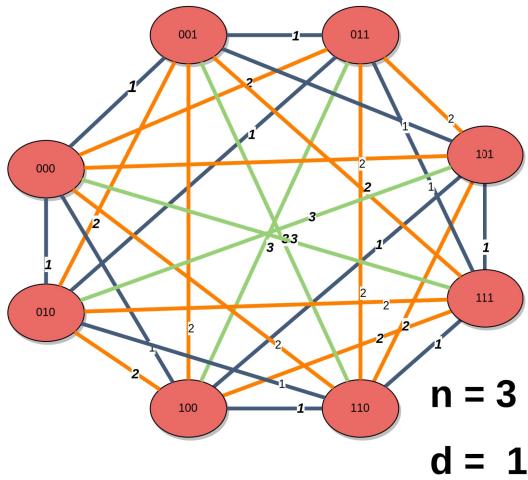


Figura 7: Grafo de Hamming de 3 dimensiones

De esta manera el nuevo grafo de dimensión  $n = 3$  con aristas mayores a  $d = 2$  será un grafo no completo (Figura 2), por eso mismo se buscará el tamaño del subgrafo más grande que sea completo.

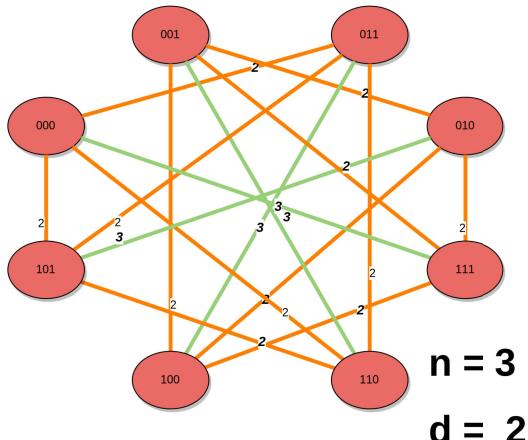


Figura 8: Grafo de Hamming A(3,2)

Antes de comenzar a construir la solución primero se utilizará el algoritmo de Bron-Kerbosch que consiste en enumerar todos los subconjuntos de vértices que cumplen la propiedad de colectividad de todos contra todos. [1] Como los algoritmos de Bron-Kerbosch tiene varias ver-

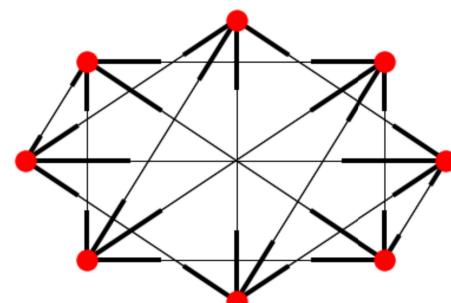
siones, se eligió la versión Sin Pivote, este usa backtracking. Entonces es algoritmo que busca los cliques máximos en un gráfico dado  $G$ . Generalmente, dado tres conjuntos  $R$ ,  $P$  y  $X$ , encuentra el máximo clique que incluyen todos los vértices en  $R$ , algunos de los vértices en  $P$ , y ninguno de los vértices en  $X$ . En cada llamada a la algoritmo,  $P$  y  $X$  siendo

conjuntos disjuntos cuya unión consiste en aquellos vértices que forman cliques al añadir a  $R$ . En otras palabras,  $P \cup X$  es el conjunto de vértices que están unidas a cada elemento de  $R$ . Cuando  $P$  y  $X$  son ambos vacío no hay elementos adicionales que se pueden agregar a  $R$ , por lo que  $R$  es un clique máximo y el algoritmo emite  $R$ .[2] La recursión inicia

mediante el establecimiento de  $R$  y  $X$  a ser un conjunto vacío y  $P$  a ser el conjunto de vértices de la gráfica. Dentro de cada llamada recursiva, el algoritmo considera los vértices  $P$  a su vez; Si no hay tales vértices, o bien se informa a  $R$  como un clique máximo (si  $X$  está vacío), o retrocede. Para cada

vértice  $v$  seleccionado entre  $P$ , se hace una llamada recursiva en la que  $v$  se añade a  $R$  y en la que  $P$  y  $X$  están restringidas al conjunto vecino  $N(v)$  de  $v$ , que encuentra  $v$  y los informes de todas las extensiones clique de  $R$  que contienen  $v$ . Entonces, se mueve  $v$  de  $P$  a  $X$  para excluirlo de consideración de futuros cliques y continúa con el siguiente vértice en  $P$ .[2]

- 1 Crear lista de vértices binarios
- 2 Solicitar  $n$  y  $d$  del nuevo gráfico
- 3 Crear Grafo
- 3.1 Agregar Vértices
- 3.2 Agregar Aristas según la distancia de hamming mínima
- 3.3 Dibujar Grafo
- 4 Generar lista de cliques con el algoritmo Bron-Kerbosch [3]
- 5 Buscar Tamaño del clique máximo El resultado será el grafo  $A(3,2) = 4$  como se muestra en la Figura 8.



$H(3,2)$   
El clique Máximo es 4

Figura 9: Ejemplo Solución Cliqué Máximo

## 2 CONCLUSIONES

- Los algoritmos de arboles generadores minimos son los que posibilitan la interpretacion adecuada de grafos para su aplicacion a problemas de mayor complejidad.
- Hay muchas maneras de recorrer un grafo o un árbol. Unas son mas eficientes que otras pero cual usar depende de la necesidad que se tenga[2].

## REFERENCES

- [1] .Coto, Algoritmos Básicos de Grafos, Caracas: Universidad Central de Venezuela, pp. 1820, 2003
- [2] nálisis comparativo de la ejecución del algoritmo voraz de PRIM en modo lineal y paralelo (LAM-MPI). (2003). Ingeniería y Desarrollo, (14), 6.
- [3] aragoza Salazar, J. E., Trueba Espinosa, A. (2015). Algoritmo computacional de mezcla dinámica de cliques para medir la resonancia de individuos en redes sociales. Acta Universitaria, 25(2), 28-39. doi:10.15174/au.2014.733
- [4] orca, G. T., Ruiz, J. A. (2014). UN ALGORITMO DEL MÉTODO DE INTEGRACIÓN DE VARIABLES PARA LA SOLUCIÓN DEL PROBLEMA MáXIMO CLIQUE PONDERADO. Investigación Operacional, 35(1), 27-35.
- [5] érez Aguilera, R. (2013). Una introducción a las matemáticas discretas y teoría de grafos. [N.p.]: El Cid Editor.
- [6] oreño Valencia, M. (2004). Teoría de grafos. Bogotá Universidad Incca de Colombia 2004.
- [7] spinal, A. C., Flórez, J. C., López, J. S. (2011). Aplicación de la teoría de grafos en la solución de problemas con impacto ambiental. Producción Más Limpia, 6(1), 9-20.
- [8] <http://esteban-gzz.blogspot.com.co/2011/07/problema-de-clique.html>
- [9] Finding All Cliques of an Undirected Graph”, dfki, 2017. [Online]. Available: <http://www.dfgi.de/~neumann/ie-seminar/presentations/findingcliques.pdf>. [Accessed : 27 – May – 2017].
- [10] BronKerbosch algorithm”, En.wikipedia.org, 2017. [Online]. Available: <https://en.wikipedia.org/wiki/Bron>
- [11] Implementing BronKerbosch algorithm in python”, it1me, 2017. [Online]. Available: <http://www.it1me.com/it-answers?id=13904636&tl=Implementing+Bron>