

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 pueden ser utilizados a lo largo del desarrollo de 3 problemas propuestos. Luego de esto, se desarrollaron soluciones a 3 problemas propuestos 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.

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 al paso 4. 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 $O(mn, m)$

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

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 .

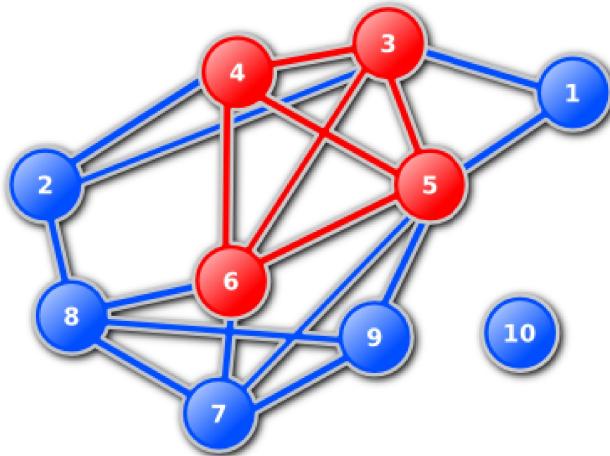


Figura 1: dado un grafo no dirigido G , y un número natural k , determinar si G posee un clique de tamaño k [8].

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:

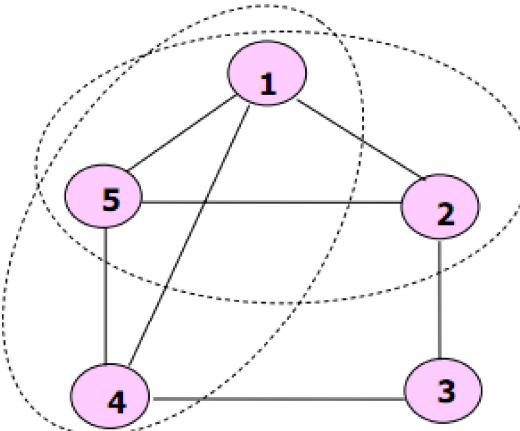


Figura 2: Cliques en grafos [8].

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^n 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 $O((3^{(n/3)}) -)$, 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

Objetivo 1

Para alcanzar el objetivo, es preciso seguir los siguientes pasos:

- Hallar los caminos menos costosos entre todos los nodos utilizando el algoritmo de Prim.
- Calcular el promedio de las aristas pertenecientes al nuevo árbol de expansión mínima.
- Recorrer las aristas pertenecientes al nuevo árbol
 - Si la distancia que representa la arista es mayor al promedio evaluado anteriormente
 - * Seleccionar la demanda mayor entre los nodos vecinos para construir la tienda
- Para ubicar las k tiendas en los nodos que presentan mayor demanda, se comienza con el nodo de mayor demanda.

Estos pasos se repiten hasta que todas las aristas del nuevo árbol haya sido recorridas.

Objetivo 2

Así que, para este se usa exactamente los mismos pasos del anterior algoritmo con la única condición de poner la cantidad n de tiendas. Esa cantidad se halla al comparar la demanda de cada nodo con la varianza, hallada de todos los costos del grafo.

Se elige la varianza porque su valor se aproxima más a los valores de mayor sesgo que el resultado de los promedios [1].

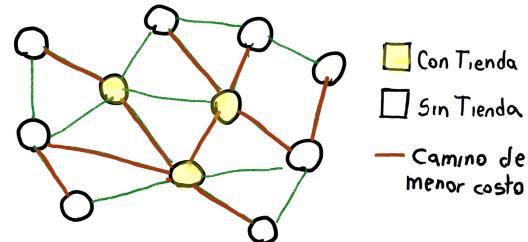


Figura 3: Aplicación algoritmo cadena de tiendas

1.3 Coloreamiento de aristas

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:

- $n = \text{numero de vertices}$
- $k = \text{numero de aristas azules}$
- $n-k-1 = \text{numero de aristas rojas}$
- $k \leq n-1$

El objetivo es desarrollar un algoritmo que encuentre un árbol de expansión sobre G con exactamente k aristas azules, y exactamente $n-k-1$ aristas rojas. Además de esto, buscamos determinar el tiempo de ejecución del algoritmo y finalmente, mostrar que es correcto[4].

Para solucionar el algoritmo se establece la siguiente secuencia de pasos:

- Elegir un nodo de inicio (cualquiera).

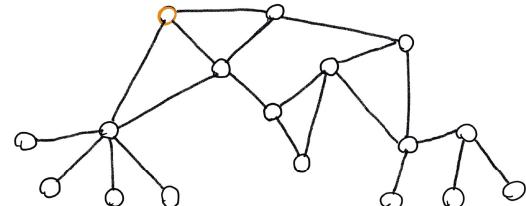


Figura 4: Selección del nodo de inicio

- Aplicar el algoritmo de Prim en el grafo, teniendo en cuenta el nodo de inicio seleccionado previamente. De esta manera se obtiene el árbol de expansión.

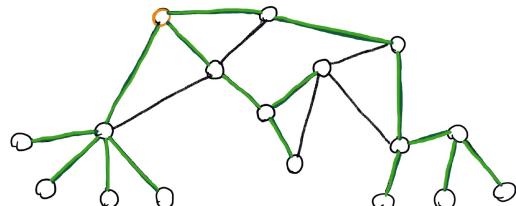


Figura 5: Aplicación del algoritmo de Prim

- Partir del mismo nodo de inicio y con base al árbol previamente obtenido, recorrer el número de aristas equivalente a k (aristas azules) mediante el algoritmo de expansión.

- Partiendo de los ultimos nodos alcanzados por las aristas k coloreadas, colorear como aristas rojas las restantes aplicando la formula $n-k-1$ [5].

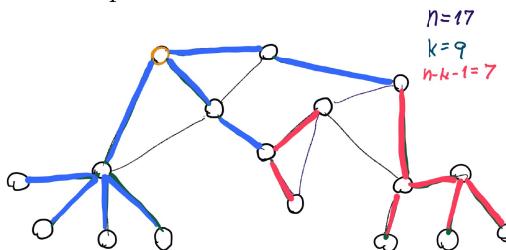


Figura 6: Algoritmo comprobado

- 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

En este algoritmo se debe encontrar un algoritmo "eficiente" para calcular el clique máximo en el grafo de Hamming (calcular el clique máximo es un problema NP-difícil)[5]. Se

le dará solución a partir de la teoría recopilada en la sección 1.1.4 Grafo de Hamming.

2 CONCLUSIONES

- Los algoritmos de árboles generadores mínimos son los que posibilitan la interpretación adecuada de grafos para su aplicación a problemas de mayor complejidad.
- Hay muchas maneras de recorrer un grafo o un árbol. Unas son más eficientes que otras pero cuál 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] nlisis 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] rez 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 Inca de Colombia 2004.
- [7] spinal, A. C., Flrez, J. C., Lpez, J. S. (2011). Aplicación de la teoría de grafos en la solución de problemas con impacto ambiental. Producción Ms Limpia, 6(1), 9-20.
- [8] <http://esteban-gzz.blogspot.com.co/2011/07/problema-de-clique.html>