ISSN: 1692-7257 - Volumen 1 – Número 35 - 2020

Revista Colombiana de Tecnologías de Avanzada

https://doi.org/10.24054/16927257.v35.n35.2020.3926

Recibido: 18 de octubre de 2019
Aceptado: 15 de diciembre de 2019

ANALYSIS OF ALGORITHMS COMPLEXITY: APPLICATION CASES

ANÁLISIS DE COMPLEJIDAD EN ALGORITMOS: CASOS DE APLICACIÓN

Msc. Efrén Romero-Riaño[*], Msc. Gabriel Mauricio Martínez-Toro[**]
Msc. Dewar Rico-Bautista[***]

[*] **Universidad Industrial de Santander**
Grupo de Investigación INNOTEC.
Bucaramanga, Santander, Colombia.
Celular: +57 3016427051. E-mail: efren.romero@saber.uis.edu.co

[**]**Universidad Autónoma de Bucaramanga**
Facultad de Ciencias Económicas, Administrativas y Contables, Programa De Administración de Empresas Formación Dual Universitarias, Grupo GENIO.
Bucaramanga, Santander, Colombia.
Celular: 3005126223 E-mail: gmartinez714@unab.edu.co

[***]**Universidad Francisco de Paula Santander Ocaña**
Departamento Sistemas e Informática. Grupo de ingeniería en innovación, tecnología y emprendimiento (GRIITEM).Ocaña, Norte de Santander, Colombia.
Celular: 3123973390. E-mail: dwricob@ufpso.edu.co

Abstract: Computational complexity related to time and space is a research topic that many researchers have been working on. Develop mechanisms that are able to deal with problems in a reasonable time and algorithms that minimize the use of computational memory it's the milestone to researchers in the computational field. The objective of the article is to present the analysis of cases of application of complex algorithms. Selected cases, especially connectivity problems, in which some problems are classified according to their complexity in time and space, as well as the description of the algorithms used to solve these problems.

Keywords: Algorithms; Computational complexity, NP-HARD.

Resumen: La complejidad computacional desde el enfoque de tiempo y espacio ha sido un tema de investigación que ha sido abordado por diversos investigadores. La creación de mecanismos capaces de resolver problemas en tiempo razonable y de algoritmos que minimizan el uso de memoria computacional, tema de discusión relevante para los investigadores en el ámbito de la computación. El objetivo del artículo es dar a conocer el análisis de casos de aplicación de complejidad de algoritmos. Casos seleccionados, especialmente problemas de conectividad, en los cuales se clasifican algunos problemas según su complejidad en tiempo y espacio, así como la descripción de los algoritmos usados para solucionar estos problemas.

Palabras clave: Algoritmos; Complejidad computacional; NP-HARD.

1. INTRODUCION

Mientras se libró la segunda guerra mundial, entre los años 1.937 y 1.940 el matemático inglés Alan Turing confrontaba un problema computacional que definiría el futuro del mundo (Mathematical Society of Japan, 1987; Lavalle, 2012; Revista digital para profesionales de la enseñanza, 2010).En el marco de los intentos por desencriptar los mensajes nazis de la máquina Enigma, en tiempos de guerra, Alan Turing creó una máquina de computación teórica, como modelo idealizado para cálculos matemáticos. La máquina se compuso por una línea de células, denominadas "cintas", con movimiento hacia adelante y hacia atrás; un elemento activo, denominado "cabeza", una propiedad de "estado" que cambia el color de una célula activa que se encuentra bajo la cabeza; y de unas instrucciones para modificar la célula activa y mover la cinta (Mathematical Society of Japan, 1987); (L Tangarife et al., 2017).

Cada paso de la máquina modifica el color de la célula activa y cambia el estado de la cabeza; después de lo cual se mueve la cinta una unidad a la izquierda o la derecha, en otras palabras, la máquina funcionaba como un lector ceros y unos en una larga banda de papel. La máquina tiene memoria y puede cambiar las direcciones en las que realiza la lectura (de izquierda a derecha y viceversa) basándose en las instrucciones que lee, los bits en el rollo de papel(Mathematical Society of Japan, 1987); (J Plaza, M Núñez, 2017).

Turing logró demostrar que una máquina es teóricamente capaz de ejecutar casi cualquier algoritmo, aunque para la época en que se creó esta máquina no se conocían formalmente estos conceptos (Lavalle, 2012). Probó que las tareas se pueden descomponer en pequeñas piezas, hasta llegar a una pregunta binaria (de sí y no), de tal forma una máquina podría interpretar lenguaje mediante cifrado binario. El problema computacional que enfrentóTuring se relacionócon la complejidad tanto de tiempo, como de espacio. Para una máquina de Turing la complejidad de tiempo se refiere al número de veces que la cinta se mueve cuando se inicializa la máquina para algunos símbolos de entrada y la complejidad del espacio es el número de células de la cinta escritas (Mathematical Society of Japan, 1987); (O Suarez at el, 2018)

Una máquina de Turing puede utilizarse como reconocedor de lenguaje, pues dada una cadena w la maquina decide si pertenece a un lenguaje o no, generando la salida adecuada; sin embargo, en ocasiones el número de combinaciones necesarias para desencriptar un mensaje hace los problemasirresolublesal consumir demasiado tiempo para que la información sea útil, para lo cual Turing notó que podía descartar parte de la complejidad en etapas (Universidad de Huelva, 2016; Mathematical Society of Japan, 1987).

Este artículo se encuentra dividido en tres apartados organizados de la siguiente manera, inicialmente se enuncia (i)Problemas según su complejidadde las clases P, NP, PSPACE y PSPACE-COMPLETE(ii), se dan a conocer los casos de aplicación relacionados con tópicos emergentes en el área y (iii), finalmente las conclusiones.

2. METODOLOGÍA DE LA REVISIÓN DE LITERATURA

El presente trabajo es una investigación teórico descriptiva de tipo documental, dado que el procedimiento implica la búsqueda, organización y análisis de un conjunto de documentos electrónicos, ver figura 1, sobre los temas de algoritmos y Complejidad computacional, en el período comprendido en los últimos años.

La revisión documental se realizó en revistas de alto impacto publicados en las bases de datos *AMJ, SCOPUS, SCIENCE DIRECT, SciELO, Directoryof Open Access Journals (DOAJ), TheNationalAcademiesPress, REDALYC, y LATINDEX*.

Como criterios de búsqueda, se incluyeron los siguientes descriptores: "Algoritmos"; "Complejidad computacional", "análisis de complejidad", "casos de uso". Estos descriptores fueron combinados de diversas formas al momento de la exploración con el objetivo de ampliar los criterios de búsqueda.

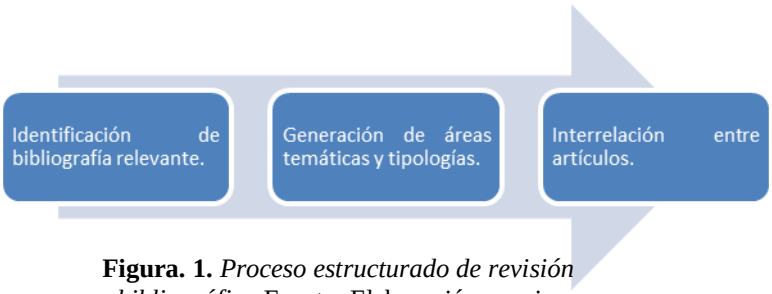

Figura. 1. *Proceso estructurado de revisión bibliográfica.*Fuente: Elaboración propia.

El objetivo de la primera fase es realizar la búsqueda de los documentos, en cada una de las bases de datos, se preseleccionaron artículos, al final en total cincuenta referencias, de acuerdo con los criterios de inclusión y exclusión. No se tomaron en consideración para el análisis aquellos artículos que no hacían alusión a los

núcleos temáticos y/o aquellos que no se encontraban en revistas indexadas. (Sánchez , 2011)

El objetivo de la segunda fase es la organización de los documentos, se creó un archivo de Word, con los siguientes campos: título del artículo, autor, año, revista, información de la revista, problema de investigación, objetivos, tipo de investigación, método, descripción, instrumentos utilizados, resultados y núcleo temático. Una vez organizada la información, se agruparon los documentos en tres núcleos temáticos, a saber: Algoritmos; Complejidad computacional, y análisis de complejidad.

El objetivo de la tercera fase es identificar interrelaciones a partir de un análisis de las similitudes, diferencias y contraposiciones de los conceptos planteados entre los artículos revisados y concluir sobre las perspectivas respecto al tema.

Posteriormente, se realizó el análisis de cada uno de los núcleos temáticos, identificando los problemas abordados, metodologías, instrumentos, población y resultados, definiendo lo más relevante y describiendo los aspectos comunes y divergentes entre los documentos seleccionados, mediante un ejercicio de comparación constante. Finalmente, se realizó un análisis global mediante el cual se identificaron las convergencias y divergencias del análisis de cada uno de los núcleos temáticos, se formularon ciertas hipótesis y conclusiones.(Sánchez , 2011)

3. PROBLEMAS SEGÚN SU COMPLEJIDAD

La complejidad computacional se divide en complejidad temporal y de espacio, los problemas clasificados como P y NP hacen parte de la complejidad temporal, este tipo de complejidad indica que a medida que las entradas son más grandes, es difícil de definir el tiempo de ejecución en una MT, los problemas clase P se resuelven en tiempo determinístico polinomial y los clase NP se resuelven en tiempo no determinístico polinomial(Rosenfeld & Irazábal, 2013)(Riaño, Rico-Bautista, & Martínez, 2018).

En cuanto a la complejidad en espacio hay que entender que el concepto de espacio limitado en la computación, los requerimientos de memoria que una tarea computacional requiere para ser ejecutada (Arora & Boaz, 2007), y el espacio total tomado por el algoritmo con respecto al tamaño de las entradas, a este tipo de complejidad pertenecen los problemas PSPACE y PSPACECOMPLETE. En la Tabla 1 se enuncian unos problemas de la clase P, NP, PSPACE y PSPACE-COMPLETE.

Tabla 1. Problemas según su complejidad.

CLASE	***PROBLEMA***	***CUÁL ES EL PROBLEMA***	***COMPLEJIDAD***
P	*Camino mínimo de un grafo* (Rosenfeld & Irazábal, 2013)	Determinar si en un grafo existe un camino de al menos longitud K entre dos vértices V1 y V2.	*O(n3) pasos por medio de un MT que reconoce el lenguaje del Shortestpath*
NP	*Connected set cover*(Zhang, Wu, Lee, & Du, 2012)	*Dado un sistema (V,S) donde V es un set de elementos y S es una familia de subsets de V, un set cover C de V es una subfamilia de S dado que cada elemento en V está en al menos uno de los subsets de C*	*O(k^2n^2) usando un algoritmo que busca el mínimo de connected set cover si cada Spanningtree de un grafo G tiene como máximo k hojas.*
PSPACE	*Back tracking*(Bock & Borges Hernandez, 2005)	*Algoritmo enumerativo, el cual se usa cuando no queda otra alternativa que enumerar los casos posibles a la hora de resolver un problema*	*Este algoritmo es generalmente de orden exponencial en tiempo, pero no en espacio, ya que la profundidad del árbol se acota polinomialmente. En casos particulares es exponencial o expo-* *potencial lo que hace que se encuentre en esta categoría.*
PSPACE-COMPLETE	*QBF-QSAT*(Bock & Borges Hernandez, 2005)	*Formulas Booleanas Cuantificadas. Árbol Binario completo donde cada variable booleana es un nivel del árbol*	*Cada hoja hace una evaluación de la fórmula booleana en la talla de la fórmula, siendo n la profundidad del árbol, por lo cual se necesita espacio polinomial*

Fuente: Elaborado por el autor.

4.CASOS DE APLICACIÓN

4.1 Complejidad de los problemas de conectividad

Dentro de los problemas NP se encuentran los problemas que se enfrentan a la conectividad, en los que se mueven objetos a través de una red de nodos. Problemas de ruteo en los cuales se plantea un Grafo G con dos terminales diferentes y dos números enteros p y k, la pregunta se centra en conectar las terminales en al menos p rutas que pasen por la mayoría de los k nodos.

El problema del Mínimo de Bordes Compartidos (*MinimumSharedEdges*) MSE es determinado como un problema NP-hard.(Omran, Sack, & Zarrabi-Zadeh, 2013)hacen una explicación detallada acerca de la complejidad computacional del problema al usar una versión del MSE y probar a través de la prueba de reducción del "*Set CoverProblem*" (SCP), su complejidad. A continuación, se hace una traducción libre del artículo llamado "*findingpathswithminimumsharededges*" de 2012 para mostrar la prueba de NP.

El SCP está definido por: Conociendo (X,C, l), donde X es un set finito de elementos, C es una colección de subgrupos de X y l es un número entero, hay un subgrupo C' $\subseteq$ C dado que lC'l $\leq$ l tal que los miembros de C' contengan a X. Se demuestra que la versión del MSE usado en el artículo es NP-completo (como lo es el *Set CoverProblem*) teniendo los siguientes parámetros del problema: G es un grafo con 2 nodos s y t, y k, h son dos números. ¿Hay un set P de k caminos de s hacia t tal que el número de bordes compartidos en P es como máximo h?.

Se reduce el SCP a MSE Transformando cada parte de (X,C, l) del SCP al (G,k,h) del MSE de la siguiente manera:

Se adiciona a G el grupo de nodos $V=V_xUV_cU\{t\}$, donde $V_x=(v_x \mid x \in X)$ y $V_c=\{v_{ci} \mid Ci \in C\}$. Se conectan todos los nodos de $v_x \in V_x$ a un nodo $v_{ci} \in V_c$ directamente si $x \in Ci$, además se conecta cada nodo $v_{ci} \in V_c$ con *t*, se adiciona a *G* el nodo *s* y se conecta con todos los nodos de $v \in V_xUV_c$ de tamaño l +1, (ver figura 1), cada uno de estos caminos se denomina cadena, y se completa la transformación definiendo $k = |X| + |C|$ y $h = l$.

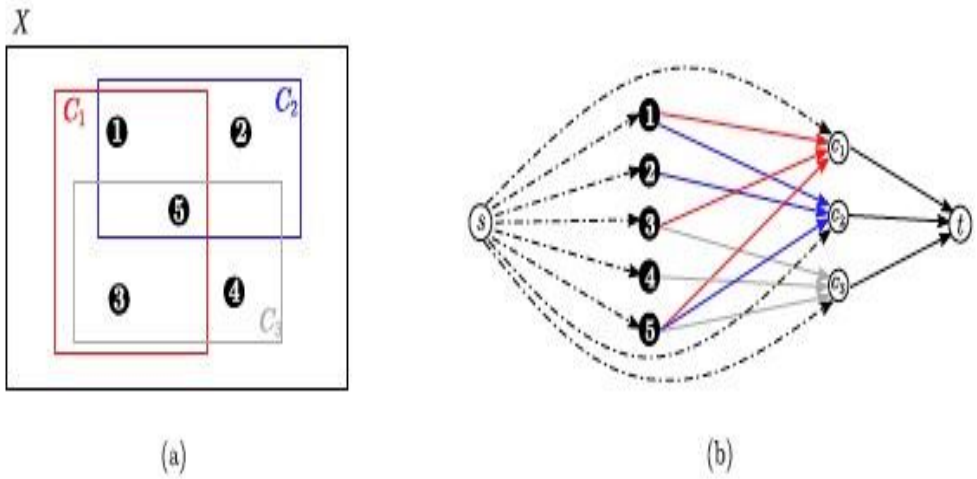

Figura. 1. Representación del SCP (a) y la reducción al MSE (b). Fuente (Omran, Sack, & Zarrabi-Zadeh, 2013)

Supóngase que hay un grupo *P* de *k* (*s*,*t*)-caminos en *G* con un máximo de h bordes compartidos. Como se, mostro previamente C' ⊆ C con |C'| ≤*l* que cubre a X. Se puede observar que cada cadena aparece máximo en un (*s*,*t*)-camino, ya que de lo contrario más de *h*(=*l*) bordes serían compartidos. Si el grado de salida de *s* es igual al número de caminos *k* indica que cada cadena es usada exactamente una vez, entonces cada vértice de $v_x \in V_x$aparece en un (*s*,*t*)-camino, de la misma manera solo una línea de salida de cada $v_x \in V_x$es usada en *P* se infiere que los bordes compartidos solo se presentan en los caminos que llegan *t*. Siguiente, se considera V'={v $\in V_c$| (*v*,*t*) es un borde compartido} .

Considerando a (*s*,*t*)-camino que va a través de un nodo v_x∈y un nodo v $\in V_c$. Se define v ∈ V', de lo contrario el nodo v participaría en 2 caminos, uno viniendo de v_xy el otro a través de la cadena, causando que el borde (*v*,*t*) sea usado en al menos 2 caminos, lo que es una contradicción. De tal forma se incluye un subgrafo G(P), cada nodo en $v_x \in V_x$es conectado a un nodo v ∈ V'. El grupo C' ={C_i |v_{ci}∈ V'} está cubriendo a X con |C'|= *l* . Haciendo C' ⊆ C cubra a X con |C'|≤*l* .Se muestra que en el grafo G hay un grupo P de *k* caminos con máximo *h* bordes compartidos. Haciendo V'={$v_{ci} \in V_c$|C_i∈ C'} . Para x ∈X, se define un (s,t) camino $P_x$de la siguiente manera: Se empieza por *s* y se sigue la cadena hacia v_x. Ya que x está cubierto por la colección de $C_i \in$ C' hay un borde (v_x, v_{ci}) tal que v_{ci}∈ V'. Se usa el borde (v_x, v_{ci}) para llegar de v_xav_{ci}y luego seguir hacia *t*. El grupo P_x={P_x|x∈ X} consiste en |X| (*s*,*t*) caminos.

Ahora se define P_c de |C|(s,t) caminos por concatenación, para cada $C_i \in$ C, la cadena de s a v_{ci}y los bordes (v_{ci}, t). Se define a P = P_xU P_c .Se observa que solo los bordes entre V_c y t pueden ser usados en más de un camino de P. Finalmente al ver que los nodos en Vc \ V' no son tocados por los caminos en P_x, cada borde (v,t) para v ∈Vc \ V'es usado exactamente una vez en P, determinando que el número de bordes compartidos en P es como máximo |V'| = *h*.

- **Algoritmos y descripción de su complejidad (tiempo o espacio)**

La calidad de los algoritmos se mide en función a su eficiencia, eficiencia en el tiempo y eficiencia en el costo de escribirlo, si un algoritmo se tiene que ejecutar pocas veces, es posible que su eficiencia no sea tan valorada como su costo de escribirlo, en el caso contrario la eficiencia resulta el factor más relevante. La eficiencia en un algoritmo en términos computacionales se refiere a la cantidad de recursos utilizados por el algoritmo cuando es ejecutado, en tal virtud, la eficiencia se puede medir de acuerdo con el tiempo usado o a la memoria necesaria para ejecutarlo(Bisbal Riera, 2009). En la Tabla 2 se resumen las funciones que frecuentemente surgen en el análisis de algoritmos:

Tabla 2. Funciones para el análisis de algoritmos

VELOCIDAD DE CRECIMIENTO	NOM.
K	Constante
LOG(N)	Logarítmica
N	Lineal
N LOG(N)	Cuadrática
n^2	Polinómica
2^n	Exponencial

Fuente: (Bisbal Riera, 2009)

A continuación, se hará la descripción de la complejidad del algoritmo usado para resolver el Problema del camino mínimo en un grafo.

- **Clase P: Problema de camino Mínimo en un Grafo**

Este problema consiste en determinar si en un grafo existe un camino de al menos longitud K entre dos vértices V1 y V2. Proponiendo un algoritmo que indica: Si A^h(i, j) es la longitud del camino mínimo entre los vértices i, j, y no pasa por ningún vértice de una longitud mayor que h, se cumple lo siguiente:

$$A^{h+1}(i,j) = \min(A^h(i, h+1) + A^h(h+1, j), A^h(i, j))$$

Usando una MT que trabaja en tiempo polinomial que represente el problema en términos del problema de la ruta más corta definida en los parámetros SP = {(G, v1, v2, K), donde G es un grafo con vértices V1 y V2, que tienen un camino entre ellos de máximo K. Teniendo la entrada w = (G, v1, v2, K), M obtiene A^m(v1, v2) como el camino mínimo en el grafo G entre sus vértices, el cual acepta si y solo si A^m(v1, v2)≤K. Se tiene

matrices de A^i de m x m que almacenan los valores que se van calculando.

1. Si w no es una entrada válida, rechaza.
2. Para todo i, j ≤ m, hace:
   a. Si G incluye un arco (i, j), entonces $A^1[i, j] := 1$, si no $A^1[i, j] := m$.
3. Para todo h = 2, 3, …, m, hace:
   a. Para todo i, j ≤ m, hace:
4. $A^h[i, j] := \min(A^{h-1}[i, h] + A^{h-1}[h, j], A^{h-1}[i, j])$.
5. Si $A^m[v1, v2] \leq K$, entonces acepta, si no rechaza.

El paso 1 verifica que G es válido (sus vértices van de 1 hasta m, sus arcos (E) no se repiten y todos sus vértices están en V), lo cual toma un tiempo O(|V|) + O($|E|^2$) + O(|V||E|) , adicionalmente que V1 y V2 son válidos y distintos y K es un número natural menor que m, lo cual toma un tiempo lineal. El tiempo que se consume el paso 1 es $O(n^2)$. En el paso 2 el valor que toma $A^1[i, j] :=$m es grande, lo cual determina que el valor de $A^1[i, j]$ es grande también. Los pasos 2-4consumen O(m^3) = $O(n^3)$ tiempo, por lo cual el costo en tiempo de la Maquina de Turing Determinista M es de $O(n^2) + O(n^3) = O(n^3)$ pasos (Rosenfeld& Irazábal, 2013).

4.2 Clase P: Ordenamiento de secuencias

Dada una entrada $(a1, a2, \dots, an)$, de tal forma que $a'1 \leq a'2 \leq \dots \leq a'n$. Por ejemplo, la entrada (8, 2, 4, 9, 3, 6), genera la salida (2, 3, 4, 6, 8, 9). Para solucionar este problema, se puede utilizar el algoritmo Insertion Sort (Cormen, Leiserson, Rivest, & Stein, 2001) que puede resumirse en el siguiente código:

Algoritmo Insertion sort. Fuente: (Cormen, Leiserson, Rivest, & Stein, 2001)
Insertion-Sort(A, n)

1. para $j \leftarrow 2$ to n
2. hacer: $key \leftarrow A[j]$
   2.1. $i \leftarrow j - 1$
   2.2. Mientras $i > 0$ y $A[i] > key$
3. Hacer: $A[i + 1] \leftarrow A[i]$
   3.1. $i \leftarrow i - 1$
   3.2. $A[i + 1] = key$

El tiempo de ejecución depende de la entrada, siendo una secuencia ya ordenada más sencilla. Para analizar este algoritmo es bueno empezar por parametrizar el tiempo de ejecución por el tamaño de la entrada, pues las secuencias cortas son más fáciles de ordenar que las largas (Eugenio & Rhadamés, 2001). Por lo general se buscan límites superiores en el tiempo de ejecución, pues establecen una garantía, los tiempos promedio o mínimos son relativos. El análisis se puede dividir en esos casos: el peor de los casos que es el más usado se refiere al tiempo o espacio para una entrada; el caso promedio es poco usual y se refiere al tiempo esperado o memoria consumida relacionado con todas las entradas de tamaño n; y el mejor de los casos evalúa los trucos con los que un algoritmo regular funciona bien con algunas entradas.

Cuando acontece el peor de los casos para este algoritmo se puede asumir que la velocidad de la máquina que está ejecutando, sin embargo, con el ánimo de ignorar las condiciones de la máquina se acude a una idea general. Se evalúa el algoritmo cuando el tamaño de la entrada tiende a infinito, $n \to \infty$; una idea que se denomina Big O notation. La idea matemática de esta notación, establece que existen constantes positivas $c1, c2,$ y $n0$ tal que $0 \leq c1g(n) \leq f(n) \leq c2g(n)$ para todo $n \geq n0$, y se representa como $O(g(n))$. Por ejemplo, considerando la función $3n^3 + 90n^2 + 5n + 6046 = O(n^3)$. De ese modo, el peor de los casos para el algoritmo Insertion sort, que se da cuando la entrada está ordenada al revés, puede describirse con la función

$$\sum_{j=2}^{n} O(j) = O(n^2)$$

Para el caso promedio, de la misma manera,

$$\sum_{j=2}^{n} O(j/2) = O(n^2).$$

Ahora, en cuanto a la complejidad espacial, es fácil notar que este algoritmo solo requiere una cantidad de memoria del orden de $O(n)$, adicionalmente, este es un algoritmo que transforma la entrada sin usar alguna estructura auxiliar, luego no requerirá más de una cantidad de menoría proporcional a $O(1)$, dada por la variable auxiliar que almacena a la variable key.

Por supuesto, se puede concluir que el algoritmo soluciona el problema el problema de ordenamiento en tiempo y espacio polinomial.

4.3 Árboles de expansión mínima – enfoque tradicional

Este apartado corresponde a la presentación de los hallazgos asociados a la solución de problemas mediante algoritmos clásicos. A continuación, se aborda, la descripción del problema del árbol de expansión mínimo, MST, desde la óptica de tres algoritmos de complejidad de tiempo Polinomial. Los nombres de estos algoritmos clásicos son: *Kruskal*, *Prims y Boruvka*. A continuación, se describen el problema y los algoritmos.

La entrada del problema de MST, se describe como un grafo G = (V, E) con bordes ponderados. La solución se plantea en términos de encontrar el subconjunto de peso mínimo de los bordes E´ ⊂ E que forman un árbol en V. El árbol de expansión mínimo (MST) de un grafo, define el subconjunto más barato de aristas que mantiene el grafo conectado en un componente.

MST presenta el enfoque seminal de todos los problemas de diseño de red(Skiena, 1982). Las principales características del MST son: i) se pueden calcular rápida y fácilmente, ii) proporcionan una manera de identificar *clusters* en conjuntos de puntos, iii) pueden ser utilizados para dar soluciones aproximadas a problemas duros como el árbol Steiner y el vendedor ambulante, iv) proporcionan evidencia gráfica de que los algoritmos avariciosos, pueden proporcionar soluciones óptimas.

El primero de los algoritmos clásicos de MST planteados es Kruskal. Este plantea que cada vértice comienza como un árbol separado y estos árboles se fusionan agregando repetidamente el borde de menor costo que abarca dos subárboles distintos (es decir, no crea un ciclo). A continuación, se presenta el algoritmo, donde n se refiere al número de vértices en un grafo, mientras que m es el número de aristas.

Kruskal(G) Sort the edges in order of increasing weight
count = 0
while (count < n − 1) do
get next edge (v,w)

if (component (v) ≠component(w))
add to T
component (v) = component(w)

El segundo de los algoritmos clásicos analizados es Prim. Este comienza con un vértice v arbitrario y "crece" un árbol a partir de él, encontrando repetidamente el borde de menor costo que enlaza algún nuevo vértice en este árbol. Durante la ejecución, etiquetamos cada vértice como en el árbol, en la franja (es decir, existe un borde desde un vértice del árbol), o invisible (es decir, el vértice se encuentra a más de un borde del árbol). A continuación, se presenta el algoritmo.

Prim (G)
Selectanarbitraryvertextostart
While (there are fringe vertices)
select minimum-weight edge between tree and fringe
add the selected edge and vertex to the tree
update the cost to all affected fringe vertices.

El tercero de los algoritmos clásicos de MST, es Boruvka. Este se basa en la observación de que el borde de menor peso incidente en cada vértice debe estar en el árbol de expansión mínimo. La unión de estos bordes resultará en un bosque que abarca un máximo de n / 2 árboles. Ahora, para cada uno de estos árboles T, seleccione el borde (x, y) de menor peso tal que x ∈ T y y∈ T. Cada uno de estos bordes debe estar de nuevo en un MST, y la unión de nuevo resulta en un bosque de extensión con máximo la mitad de los árboles que antes. A continuación, se presenta el algoritmo.

Boruvka(G)
Initialize spanning forest F to n single-vertex trees While (F has more than one tree)
for each T in F, find the smallest edge from T to G − T
add all selected edges to F, thus merging pairs of tres.

La mayoría de los problemas, se pueden plantear en términos de algoritmos basados en grafos (p.e. minimización del ancho de banda o el mejoramiento de autómatas de estado finito). El problema y los tres algoritmos presentados se incluyen en razón a que existen algoritmos eficientes para su solución, su tiempo de ejecución crece de forma polinomial con el tamaño del grafo.

Las descripciones de los tiempos de los algoritmos son en su orden: kruskal, obtiene un algoritmo de tipo O (m lg m); el algoritmo de Prim, con base a datos sencillos obtiene una implementación en

tiempo (O n2); y el algoritmo de Boruvka, después de como máximo, log n iteraciones, cada una de las cuales toma tiempo lineal, obtiene un tiempo O (m log n).

4.4 Enrutamiento en redes de datos

Una de las funciones de los dispositivos de capa tres (capa de red), es que pueda tener conocimiento de las diferentes rutas hacia él y cómo hacerlo, y así poder determinar la ruta hacia un destino. El *router* como principal representante coloca dichas rutas en su propia tabla de enrutamiento, la cual es actualizada después de aprenderse las fuentes, previo contacto con sus *routers* vecinos.(Laporte & Osman, 1995)

La toma de decisiones basadas en la tabla de enrutamiento determina los puertos de salida que debe utilizar un paquete hasta su destino, si la red de destino está conectada directamente, el dispositivo sabrá de antemano el puerto a reenviar el paquete, de lo contrario debe aprender y calcular la ruta más óptima a usar para reenviar los paquetes.(CCNA, 2014)

Al ser dispositivos multiprotocolos, es decir, pueden utilizar diferentes protocolos al mismo tiempo, se debe otorgar la prioridad de uno sobre el otro, a esto se le llama distancia administrativa. La tabla de enrutamiento se construye mediante uno de estos dos métodos:(Laporte & Osman, 1995)

- **Rutas estáticas.**

El administrador establece las rutas manualmente, quién debe actualizar cuando tenga lugar un cambio en la topología. Es decir, se establece un control preciso del enrutamiento según parámetros. (Feamster, Balakrishnan, Rexford, Shaikh, & van der Merwe, 2004)

- **Rutas dinámicas.**

Rutas aprendidas de forma automática a través de la información enviada por sus routers vecinos, después de ser configurado un protocolo de enrutamiento para el aprendizaje de dichas rutas.(CCNA, 2014)(Angel, Benjamini, Ofek, & Wieder, 2008)

- **Principio de optimización.**

Llegar a destino, en tiempo y forma, puede requerir que el algoritmo de enrutamiento, que es el encargado de escoger las rutas y las estructuras de datos, cumpla con ciertas propiedades que aseguren la eficiencia de su trabajo.

Estas propiedades son: corrección, estabilidad, robustez, equitatividad, sencillez y optimalidad. La corrección y la sencillez casi no requieren comentarios; no así la necesidad de robustez, la cual se refiere a que el algoritmo debe ser diseñado para que funcione dentro de la red por años, sin fallas generales. El algoritmo deberá estar preparado para manejar cambios de topología y tráfico sin requerir el aborto de las actividades o el reinicio de la red.

La equitatividad y la optimalidad resultan con frecuencia contradictorias, ya que muchas veces se requiere una concesión entre la eficacia global (optimización) y la equitatividad; es decir, antes de intentar encontrar un justo medio entre estas dos, se debe decidir qué es lo que se busca optimizar. Minimizar el retardo de los paquetes (disminuyendo escalas y ancho de banda) y maximizar el rendimiento total de la red sería la combinación más apropiada para un algoritmo de ruteo. (Ariganello, 2014)

Este postulado de optimización establece que, si el enrutador J está en la trayectoria óptima del enrutador I al enrutador K, entonces la trayectoria óptima de J a K también está en la misma ruta. Haciendo referencia a la figura 2, llamemos r1 a la parte de la ruta de I a J, y r2 al resto de la ruta. Si existiera una ruta mejor que r2 entre J y K, podría concatenarse con r1 para mejorar la ruta entre I y K, contradiciendo nuestra aseveración de que r 1 y r2 es óptima.

Como consecuencia directa del principio de optimalidad, podemos ver que el grupo de trayectorias óptimas de todas las de orígenes a un destino dado forma un árbol con raíz en el destino. Ese árbol que se forma, se llama árbol de descenso, donde la métrica de distancia es el número de escalas. El árbol de descenso puede no ser único, pueden existir otros árboles con las mismas longitudes de trayectoria.

Dado que un árbol de descenso ciertamente es un árbol, no contiene ciclos, por lo que cada paquete será entregado con un número de escalas infinito y limitado. En la práctica, no siempre sucede esto,

los enlaces y los enrutadores pueden caerse y reactivarse durante la operación, por lo que diferentes enrutadores pueden tener ideas distintas sobre la topología actual de la subred.

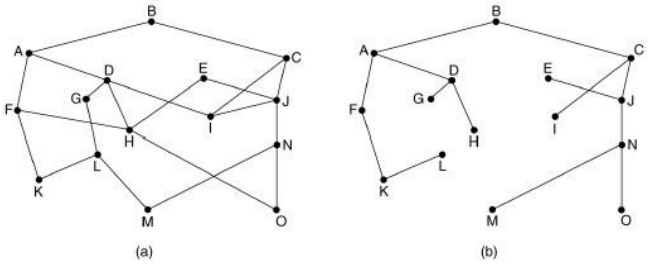

Figura. 2. (a) Subred. (b) Árbol de Descenso para el enrutador B. Fuente:(Laporte & Osman, 1995)

- **Algoritmo Dijkstra base del Protocolo OSPF**

Este protocolo utiliza el algoritmo de la mejor ruta es la de menor costo. Se basa en el algoritmo que fue desarrollado por Edsger Dijkstra, un especialista holandés en informática quien lo describió por primera vez en 1959.(Bollobás& Riordan, 1993)

El algoritmo considera la red como un conjunto de nodos conectados con enlaces punto a punto. Cada enlace tiene un costo. Cada nodo tiene un nombre. Cada nodo cuenta con una base de datos completa de todos los enlaces y por lo tanto se conoce la información sobre la topología física en su totalidad. Todas las bases de datos del estado de enlace, dentro de un área determinada, son idénticas.(Méndez, Rodriguez-Colina, & Medina, 2014)

El algoritmo de la ruta más corta calcula una topología sin bucles con el nodo como punto de partida y examinando a su vez la información que posee sobre nodos adyacentes. El algoritmo es una especialización de la búsqueda de costo uniforme, y como tal, no funciona en grafos con aristas de costo negativo (al elegir siempre el nodo con distancia menor, pueden quedar excluidos de la búsqueda nodos que en próximas iteraciones bajarían el costo general del camino al pasar por una arista con costo negativo). (Yin & Wang, 2010)

Teniendo un grafo dirigido ponderado de N nodos no aislados, sea x el nodo inicial, un vector D de tamaño N guardará al final del algoritmo las distancias desde x al resto de los nodos.(Bollobás& Riordan, 1993)(Yin & Wang, 2010)

1. Inicializar todas las distancias en D con un valor infinito relativo ya que son desconocidas al principio, exceptuando la de x que se debe colocar en 0 debido a que la distancia de x a x sería 0.
2. Sea a = x (tomamos a como nodo actual).
3. Recorremos todos los nodos adyacentes de a, excepto los nodos marcados, llamaremos a estos nodos no marcados v_i.
4. Si la distancia desde x hasta vi guardada en D es mayor que la distancia desde x hasta a, sumada a la distancia desde a hasta v_i; esta se sustituye con la segunda nombrada, esto es: si ($D_i > D_a + d(a, v_i)$) entonces $D_i = D_a + d(a, v_i)$
5. Marcamos como completo el nodo a.
6. Tomamos como próximo nodo actual el de menor valor en D (puede hacerse almacenando los valores en una cola de prioridad) y volvemos al paso 3 mientras existan nodos no marcados.

Una vez terminado al algoritmo, D estará completamente lleno.

- **Pseudocódigo**

Estructura de datos auxiliar: Q = Estructura de datos Cola de prioridad

```
DIJKSTRA (Grafo G, nodo_fuentes)
    para u ∈V[G] hacer
        distancia[u] = INFINITO
        padre[u] = NULL
    distancia[s] = 0
    adicionar (cola, (s,distancia[s]))
    mientras que cola no es vacía hacer
        u = extraer_minimo(cola)
        para todos v ∈adyacencia[u] hacer
            si distancia[v] > distancia[u] + peso (u,
            v) hacer
                distancia[v] = distancia[u] + peso (u,
                v)


                padre[v] = u
                adicionar(cola,(v,distancia[v]))
```

- **Complejidad**

Orden de complejidad del algoritmo: $O(|V|^2+|E|) = O(|V|^2)$ sin utilizar cola de prioridad, $O((|E|+|V|) \log |V|)$ utilizando cola de prioridad.(Barbehenn, 1998)

Se puede estimar la complejidad computacional del algoritmo de Dijkstra (en términos de sumas y comparaciones). El algoritmo realiza a lo más n-1 iteraciones, ya que en cada iteración se añade un vértice al conjunto distinguido. Para estimar el número total de operaciones basta estimar el número de operaciones que se llevan a cabo en cada iteración.

Se puede identificar el vértice con la menor etiqueta entre los que no están en S_k realizando n-1 comparaciones o menos. Después se hace una suma y una comparación para actualizar la etiqueta de cada uno de los vértices que no están en S_k. Por tanto, en cada iteración se realizan a lo sumo 2(n-1) operaciones, ya que no puede haber más de n-1 etiquetas por actualizar en cada iteración. Como no se realizan más de n-1 iteraciones, cada una de las cuales supone a lo más 2(n-1) operaciones, llegamos al siguiente teorema.

5. CONCLUSIONES

La complejidad computacional se puede dividir en complejidad de tiempo y de espacio, cada una de estas complejidades agrupan los problemas según sus características dentro de los grupos dentro de P, NP, PSPACE, NPSPACE, entre otros.

A través de la reducción del *Set CoverProblem* en una versión del MSE se logró comprobar la complejidad de este problema de conectividad.

La eficiencia de los algoritmos se determina a través de los recursos que son utilizados en el momento en que son ejecutados.

La complejidad computacional, trata de dar respuesta a la pregunta ¿Qué hace a algunos problemas computacionalmente difíciles y a otros fáciles?, y tiene como finalidad la creación de mecanismos y herramientas capaces de describir y analizar la complejidad de un algoritmo y la complejidad intrínseca de un problema, por lo que se constituye como uno de los parámetros de eficiencia que rige la computación, la eficiencia de un algoritmo se mide a través de la complejidad temporal y espacial (Hopcroft, Motwani, & Ullman, 2001; Sanchis, Ledezma, Iglesias, García, & Alonso, 2012).

REFERENCIAS

Angel, O., Benjamini, I., Ofek, E., &Wieder, U. (2008). Routingcomplexityoffaultynetworks. RandomStructures and Algorithms, 32(1), 71–87. https://doi.org/10.1002/rsa.20163

Ariganello, E. (2014). Redes Cisco. Guía de estudio para la certificación. CCNA Routing y switching (Primera ed.). México D.F., México: Alfaomega. Recuperado el 15 de Mayo de 2017

Arora, S., &Boaz, B. (Enero de 2007). ComputationalComplexity: A Modern Approach,Draftof a book. Princeton University.

Barbehenn, M. (1998). A note onthecomplexityofDijkstra'salgorithmforgraphswithweightedvertices. IEEE TransactionsonComputers, 47(2), 263. https://doi.org/10.1109/12.663776

Bisbal Riera, J. (2009). Manual de algorítmica: recursividad, complejidad y diseño de algoritmos. Barcelona: Editorial UOC.

Bock, M., & Borges Hernandez, C. E. (5 de Febrero de 2005). Un problema en el espacio PSPACE. Recuperado el 24 de 05 de 2017, de http://paginaspersonales.deusto.es/cruz.borges/Papers/05AlgComp.pdf

Bollobás, B., & Riordan, O. (1993). Dijkstra ' s Algorithm. Network, 69(1959), 036114. Retrievedfrom http://www.ncbi.nlm.nih.gov/pubmed/21282851

CCNA. (2014). Introducción al enrutamiento y reenvío de paquetes. Retrievedfrom https://sites.google.com/site/uvmredes2/1-introduccion-al-enrutamiento-y-reenvio-de-paquetes/1-3-construccion-de-la-tabla-de-enrutamiento

Cormen, T., Leiserson, C., Rivest, R., & Stein, C. (2001). Introductiontothealgorithns. Secondedition. London: The MIT Press. Recuperado el 23 de 05 de 2017, de http://is.ptithcm.edu.vn/~tdhuy/Programming/Introduction.to.Algorithms.pdf

Eugenio, S., &Rhadamés, C. (09 de 2001). Análisis de Algoritmos y Complejidad. Obtenido de http://ccg.ciens.ucv.ve/~esmitt/ayed/II-2011/ND200105.pdf

Feamster, N., Balakrishnan, H., Rexford, J., Shaikh, A., & van derMerwe, J. (2004). Thecase forseparatingroutingfromrouters. In Proceedingsofthe ACM SIGCOMM workshop on Future directions in networkarchitecture - FDNA '04 (p. 5). https://doi.org/10.1145/1016707.1016709

Hopcroft, Motwani, & Ullman. (2001). Introductiontoautomatatheory, languajes and computation. Segunda edición. Pearson Education.

JEG Plaza, MAR Nuñez, (2017) Formación en competencias específicas para la industria del software colombiano. Experiencias del uso del aprendizaje basado en proyectos. Revista Colombiana de Tecnologías de Avanzada, ISSN: 1692-7257

Laporte, G., & Osman, I. H. (1995). Routingproblems: A bibliography. AnnalsofOperationsResearch, 61(1), 227–262. https://doi.org/10.1007/BF02098290

Lavalle, J. (2012). El concepto de computabilidad de Alan Turing. Revista de la vicerrectoría de Investigación y estudios de posgrado , 1-16.

L Tangarife, M Sánchez, M Rojas (2017). Modelo de interventoría de tecnologías de información en el área de conocimiento de la gestión del alcance de PMBOK® y alineado con ISO 21500 y COBIT®. Revista Colombiana de Tecnologías de Avanzada, ISSN: 1692-7257

MathematicalSocietyofJapan. (1987). EncyclopedicDictionaryofMathematics. KiyosiIto.

Méndez, L., Rodriguez-Colina, E., & Medina, C. (2014). Toma de Decisiones Basadas en el Algoritmo de DIJKSTRA. Redes de Ingeniería, 4(2), 2013. Retrievedfrom http://revistas.udistrital.edu.co/ojs/index.php/REDES/article/view/6357/7872

O Suarez, C Vega, E Sánchez, A Pardo. (2018) Degradación anormal de p53 e inducción de apoptosis en la red P53-mdm2 usando la estrategia de control tipo pin. Revista Colombiana de Tecnologías de Avanzada, ISSN: 1692-7257

Omran, M. T., Sack, J. R., &Zarrabi-Zadeh, H. (2013). Findingpathswithminimumsharededges (Vol. 24). JournalofCombinatorialOptimization.

Revista digital para profesionales de la enseñanza. (10 de 09 de 2010). Enigma: las matemáticas ganaron la segunda guerra mundial. Obtenido de https://www.feandalucia.ccoo.es/docu/p5sd7422.pdf

Riaño, E., Rico-Bautista, D., & Martínez, M. (2018). ÁRBOL DE CAMINOS MÍNIMOS: ENRUTAMIENTO, ALGORITMOS APROXIMADOS Y COMPLEJIDAD. Revista Colombiana de Tecnologías de Avanzada, 1(31), 11–21. https://doi.org/https://doi.org/10.24054/16927257.v31.n31.2018.2780

Rosenfeld, D. R., & Irazábal, j. (2013). Computabilidad, complejidad computacional y verificación de programas. La Plata: Editorial de la Universidad Nacional de La Plata.

Sanchis, A., Ledezma, A., Iglesias, J., García, B., & Alonso, J. (22 de 05 de 2012). Universidad Carlos III de Madrid. Obtenido de http://ocw.uc3m.es/ingenieria-informatica/teoria-de-automatas-y-lenguajes-formales/material-de-clase-1/tema-8-complejidad-computacional

Scopus. (2017). Análisis de búsquedas con palabras clave . Scopus.

Skiena, S. (1982). Thealgorithmdesign manual. Chicago.

Takey, S. M., & Carvalho, M. M. (2016). Fuzzyfrontendofsystemicinnovations: A conceptual frameworkbasedon a systematicliteraturereview. TechnologicalForecasting and Social Change, 97-109.

Universidad de Huelva. (22 de 05 de 2016). Universidad de Huelva, Departamento de tecnologías de la información. Obtenido de http://www.uhu.es/francisco.moreno/gii_mac/docs/Tema_4.pdf

Yin, C., & Wang, H. (2010). Developed Dijkstra shortestpathsearchalgorithm and simulation. In 2010 International ConferenceonComputerDesign and Applications, ICCDA 2010 (Vol. 1). https://doi.org/10.1109/ICCDA.2010.5541129

Zhang, W., Wu, W., Lee, W., & Du, D.-Z. (2012). Complexity and approximationoftheconnected set-coverproblem. J GlobOptim, 53, 563-572.