[[1]](#footnote-1)

Búsqueda de Firma de Virus, Gusanos y Malware  
Aho-Corasick

Natalia Laserna, Xiomara Gaona, Dixon Medina.,   
Institución Universitaria Politécnico Grancolombiano, Ingeniería de Sistemas, Bogotá, Colombia  
  
Correos: [nalaserna@poligran.edu.co](mailto:nalaserna@poligran.edu.co), xigaonat@poligran.edu.co, damedina@poligran.edu.co

*Resumen*—En este articulo podemos encontrar la solución a búsquedas de múltiples cadenas en un texto, este caso se presenta en la búsqueda que hacen los antivirus en cada uno de los archivos que se encuentran en nuestras computadoras. El antivirus necesita recorrer el texto de cada archive y revisar si no tiene firmas insertadas de los virus o software malicioso.

*Palabras claves*—Antivirus; Virus; Aho-Corasick; Cadena; Tri.

# INTRODUCCIÓN

E

n este artículo se presentara el problema de buscar múltiples cadenas en un texto, de tal manera que demos solución al problema que presentan los antivirus al tener que realizar la búsqueda de firmas de virus en sistemas de computación .

## Justificación

Mantener los sistemas de computación seguros es una tarea diaria, la cual llevan los antivirus, y revisar cada uno de los archivos de un sistema es una tarea bastante extensa, por eso cada proceso que ejecute un antivirus debe ser eficiente y eficaz, tanto en tiempos y en uso de recursos. Por eso es necesario tener algoritmos eficientes que ejecuten estas tareas de una manera óptima, en este caso validar si un archivo tiene una firma de un virus.

## Objetivo General

Encontrar una solución eficiente a la búsqueda de múltiples cadenas en un texto o diccionario de palabras, y con esto solucionar el problema que tienen los antivirus de validar las firmas de todos los virus en su base de datos en un archivo de un sistema.

# MARCO TEORICO

Para poder realizar la búsqueda de múltiples firmas de virus en un archivo debemos afrontar el problema de buscar o coincidir cadenas en un texto, es un problema muy común en ciencias de la computación [1,2,3]. Varios algoritmos podrían dar solución a nuestro problema, realizando procesos de fuerza bruta, con los cuales darían solución a nuestro problema, pero no necesariamente de una manera óptima, teniendo en cuenta que, para nuestro problema de las firmas de los antivirus, debemos realizar el proceso de la manera más rápida y eficiente,

Tomando cada una de las firmas en cada uno de los archivos que contenga la máquina, sabemos que las firmas son miles al igual que los archivos de la máquina.

## Descripción del problema

El problema se basa en buscar las firmas que deja cada uno de los virus o Software malicioso en nuestros archivos, con lo anterior podemos llegar al problema de buscar patrones de cadenas en un texto. Este es un proceso puede llevar mucho tiempo y recursos de nuestra maquina si no se realiza de una manera optima o con un algoritmo capaz de realizar el proceso simultaneo de cada una de las cadenas, en este caso las firmas de los virus.

## Solución propuesta

La definición de este problema se declara de la siguiente forma; un conjunto de cadenas C: {c1, c2, c3, c4…cn} y buscar estas cadenas en un texto que llamaremos T [1], actualmente la solución a este problema se soluciona con el algoritmo Aho-Corasick el cual localiza todas las ocurrencias en un texto que contiene finitas palabras [5]. Este algoritmo esta basado en un autómata finito, el algoritmo esta definido en dos partes, la primera parte es la construcción de un árbol de prefijos de cada una de las cadenas en el conjunto C, en el paso dos vamos a procesar dentro de este árbol cada uno de las cadenas en el texto T y validar si se encuentra cada una de las palabras que se encuentran en T [5].

Para el primer paso debemos realizar la construcción de una estructura conocida como Trie en la cual vamos almacenar cada uno de los prefijos de las cadenas que vamos a buscar por ejemplo tenemos la siguiente cadenas C: {a, ab, bc, aab, aac, bd } la estructura del trie se creara un diccionario de cada uno de los prefijos de las cadenas en forma de árbol partiendo de un nodo inicial en el cual podemos definir como un carácter el cual no ocurrirá en las cadenas, por ejemplo el #, en la figura 1 podemos ver como se visualiza el árbol de prefijos del conjunto de cadenas C [8].

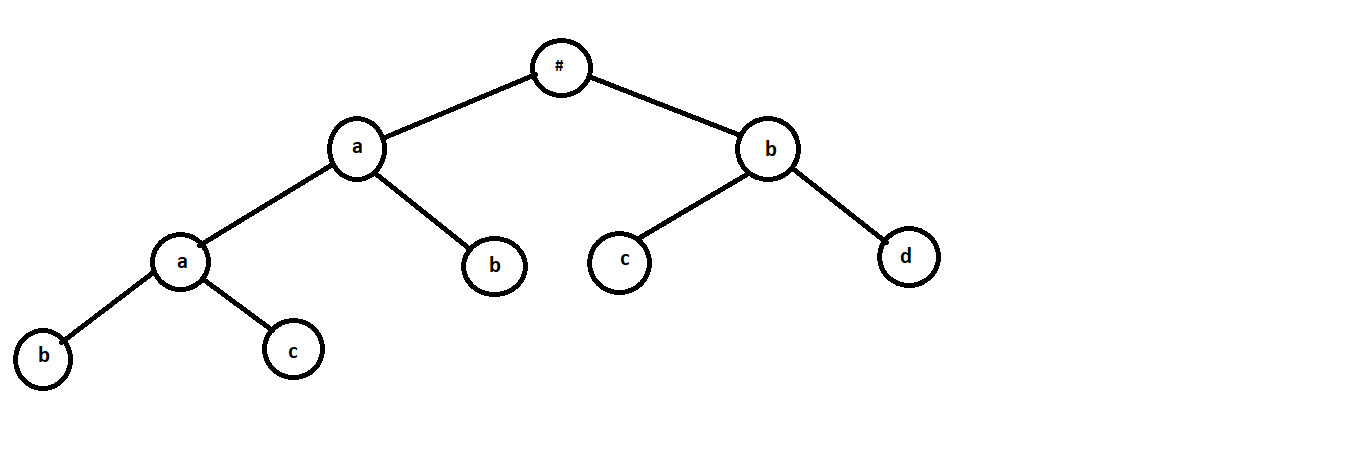


Fig. 1. Árbol de prefijos Trie

Al terminar la construcción del árbol de prefijos encontraremos que al llegar a cada una de las hojas finales, tendremos cada una de las cadenas que tenemos en C [5].

Para nuestro problema el conjunto de cadenas C, estará confirmada por fada uno da las firmas en la base de datos del antivirus, para obtener las firmas de virus, podemos basarnos en la prueba EICAR (Standar Anti-Virus Test) esta es una prueba desarrollada por European Institute for Computer Antivirus Reseach, en la cual se basa en copiar una cadena de texto específico en un archivo con extensión .com [6]. Con esta prueba podemos realizar la validación de nuestro algoritmo. Con la referencia [7] podemos encontrar la forma de validación y la descarga de los archivos .com en los que encontraremos la firma de prueba para los antivirus y la cual usaremos para realizar la prueba del Aho-Corasic.

Después de ingresar cada una de los caracteres en el Trie, se debe realizar la preparación depara eso debemos construir un árbol de fails, en el que cada uno de los nodos tendrá su primer antecesor con el mismo carácter del nodo que revisamos, para nuestro ejemplo en la figura 1, podemos ver que los dos primeros nodos del árbol a y b no tienen antecesor y por eso el fail de estos es el nodo principal el cual contiene #, en la figura 2 podemos ver cómo queda nuestro árbol de fails.

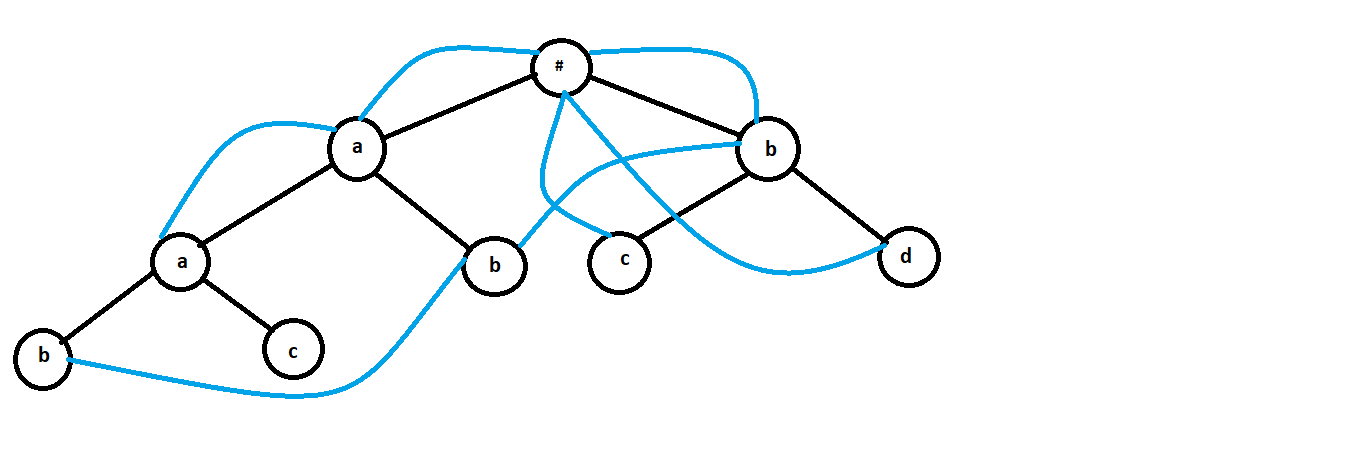


Fig. 2. Arbol de Fails

Al realizar el proceso de creación del árbol de fails genera una salida de cada una de las palabras se vamos de desde un nodo hasta su ancestro con el mismo carácter, al terminar este proceso que hace parte de la construcción del Trie, debemos pasar al segundo paso que es realizar la comparación de nuestra cadena M en el trie, a continuación, se describen las salidas que genera calcular el Trie de fails [9];

{a, b, {aa}, {ab}, {bc}, {bd}, {aab}, {aac}}

Debemos recorrer cada uno de los caracteres de nuestra cadena M cada carácter de M será validado en nuestro Trie, buscando el fail de cada carácter y revisando si este tiene algún salida generada por el fail, en cada iteración guardaremos el fail del carácter anterior de M y empezaremos el mismo proceso desde este nodo del Trie, para validar si hace parte del carácter anterior, si no se debe empezar desde el nodo inicial que es el #, de esta forma cuando lleguemos a un fail que tenga una salida imprimiremos cual fue la posición de la cadena M donde se encontró el carácter y cuál es el índice de la cadena del conjunto C [10].

## Metodología

Para la solución del problema se debe llevar a cabo la implementación del algoritmo Aho-Corasick , el cual tendrá una estructura Trie, en la que se adicionara cada uno de las cadenas del conjunto de firma el cual podemos armar a partir de la documentación de EICAR [7] con estas cadenas conformaremos nuestro conjunto de cadenas C. Igualmente de EICAR [7] descargaremos los archivos .com con los cuales conformaremos nuestro Archivo el cual debemos validar cada una de las cadenas de C, el cual llamaremos M. al implementar el algoritmo en Java o Python debe dar como resultado una lista con la siguiente información; índice donde se encontró la firma del virus en M y el índice de cual fue la cadena de C que encontró

Referencias

1. [1]Sun Liangxu, Li Linlin, Improve Aho-Corasick Algorithm for Multiple Patterns Matching Memory Efficiency Optimization, <https://www.researchgate.net/publication/269780289_Improve_Aho-Corasick_Algorithm_for_Multiple_Patterns_Matching_Memory_Efficiency_Optimization>
2. Elvan Owen, Aho – Corasick Algorithm in Pattern Matching,

<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2014-2015/Makalah2015/Makalah_IF221_Strategi_Algoritma_2015_032.pdf>

1. Akhtar Rasool1 , Gulfishan Firdose Ahmed2 , Raju Barskar3 , and Nilay Khare1, Efficient Multiple Pattern Matching Algorithm Based on BMH: MP-BMH, <https://iajit.org/PDF/November%202019,%20No.%206/15585.pdf>
2. Eric C. Rouchka, Pattern Matching Techniques and Their Applications to Computational Molecular Biology -- A Review, <http://bioinformatics.louisville.edu/localresources/papers/WUCS-99-09.pdf>
3. Alfred V, Aho and Margaret J.Corasick, Efficient String Matching: An Aid to Bibliographic Search, <https://cr.yp.to/bib/1975/aho.pdf>
4. Wikipedia, EICAR, <https://es.wikipedia.org/wiki/EICAR>
5. EICAR, Download Anti Malware Test File, <https://www.eicar.org/?page_id=3950>
6. David Esposito, AhoCorasickBuildTree, <https://www.youtube.com/watch?v=ePafMI_rSJg>
7. David Esposito, AhoCorasickBuildFail, <https://www.youtube.com/watch?v=qPyhPXPl3T4>
8. David Esposito, AhoCorasickMatching, <https://www.youtube.com/watch?v=IcXimoT_YXA>

1. . [↑](#footnote-ref-1)