



Ciencias de la
Computación
FACULTAD DE CIENCIAS
FÍSICAS Y MATEMÁTICAS
UNIVERSIDAD DE CHILE

Tarea 1 - Búsqueda en Texto

CC4102 - Diseño y Análisis de Algoritmos

Alumnos : Sebastián González

Patricio Isbej

Profesor : Gonzalo Navarro

Auxiliar : Jorge Bahamonde

Fecha : 12 de octubre de 2015

Resumen

En este trabajo se analizó 3 algoritmos distintos de búsqueda en texto, implementandolos, ejecutandolos para distintos casos y comparando estos resultados.

1. Introducción

Los algoritmos de búsqueda en texto son fundamentales debido a sus variadas aplicaciones como búsquedas dentro de documentos, búsqueda de cadenas de ADN, etc. Para este trabajo se compararon los siguientes algoritmos:

- Fuerza Bruta
- Knuth-Morris-Pratt
- Boyer-Moore-Horspool

Fuerza Bruta consiste en buscar de izquierda a derecha partiendo del primer caracter del texto y patrón, comparando caracter a caracter, si se encuentra una diferencia se vuelve a comparar comenzando con el caracter siguiente en el texto del que se comenzó, y si se recorrió todo el patrón entonces hay un calce.

Knuth-Morris-Pratt opera de manera similar a Fuerza Bruta, precalculando una tabla del tamaño del patrón a buscar donde cada elemento indica si en la posición donde se encuentra hay un sufijo que sea prefijo del patrón, para así al llegar a una diferencia se puede saber si hay tal sufijo y poder saltarse todos esos caracteres.

Boyer-Moore-Horspool precalcula una operación "siguiente" para cada caracter que indica la última ocurrencia en el patrón de ese caracter (sin considerar su último caracter). Una vez calculado comienza a comparar desde la derecha y si encuentra una diferencia, revisa que caracter se encontraba y avanza de acuerdo a la operación calculada (mientras más al principio se encuentre la última ocurrencia, más se va a avanzar en el texto).

Los algoritmos fueron ejecutados con los siguientes tipos de datos:

- Alfabeto Binario
- ADN real
- ADN sintético
- Lenguaje Natural, caso real
- Lenguaje Natural, caso sintético

2. Hipótesis

Asumiendo n como el tamaño de los datos, m como el tamaño del patrón y c como el tamaño del alfabeto se Formularon las siguientes hipótesis.

Para todos los algoritmos, el tener un c mayor significa aumentar la probabilidad de fallar, lo que hace que se tenga que avanzar en el texto con mayor frecuencia y así tener que hacer menos comparaciones y tomar menos tiempo.

2.1. Fuerza Bruta

Para Fuerza Bruta, en el peor caso la diferencia se va a encontrar siempre al final de la búsqueda, teniendo que hacer entonces nm comparaciones. Lo que significa que el tiempo y las comparaciones van a crecer proporcionales a m .

2.2. Knuth-Morris-Pratt

Este algoritmo depende de que se tengan muchas subcadenas que sean prefijo del patrón, lo que significa que va a ser mejor que el primer algoritmo, pero no por mucho.

2.3. Boyer-Moore-Horspool

Este algoritmo se va a comportar mejor que los 2 anteriores por la forma en que salta en el texto al encontrar diferencias (si el caracter no se encuentra o se encuentra muy al principio se salta m). Para un c pequeño, este algoritmo no va a ser tanto mejor que los anteriores, pero a medida que crezca, va a mejorar significativamente.

3. Diseño e implementación

Se crean con python los textos requeridos de tamaño 1MB.

Se implementan en C los 3 algoritmos, además de los métodos necesarios para ejecutarlos y guardar sus resultados.

Y finalmente se implementa en python métodos para analizar los resultados.

Más detalles se encuentran en el siguiente repositorio.

<https://github.com/Pato285/tareas-algoritmos/>

4. Experimentos y resultados

4.1. Algoritmos

Se ejecutaron los algoritmos 1000 veces para cada dato y largo de patrón, obteniéndose los siguientes resultados.

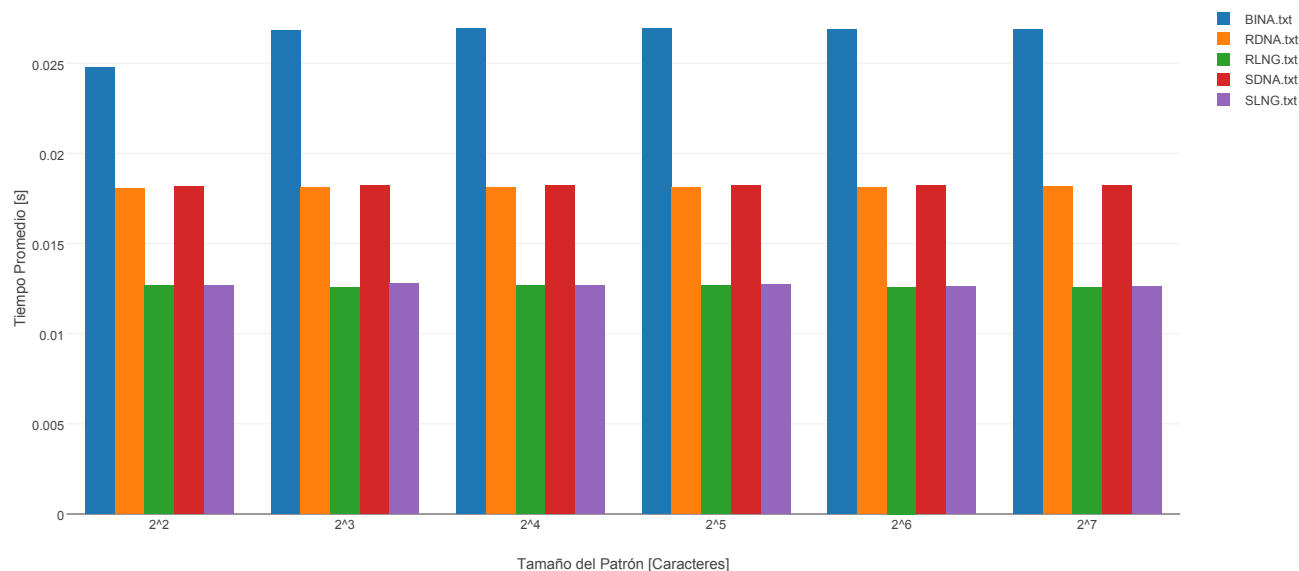


Figura 1: Tiempo de promedio de búsqueda para Fuerza Bruta

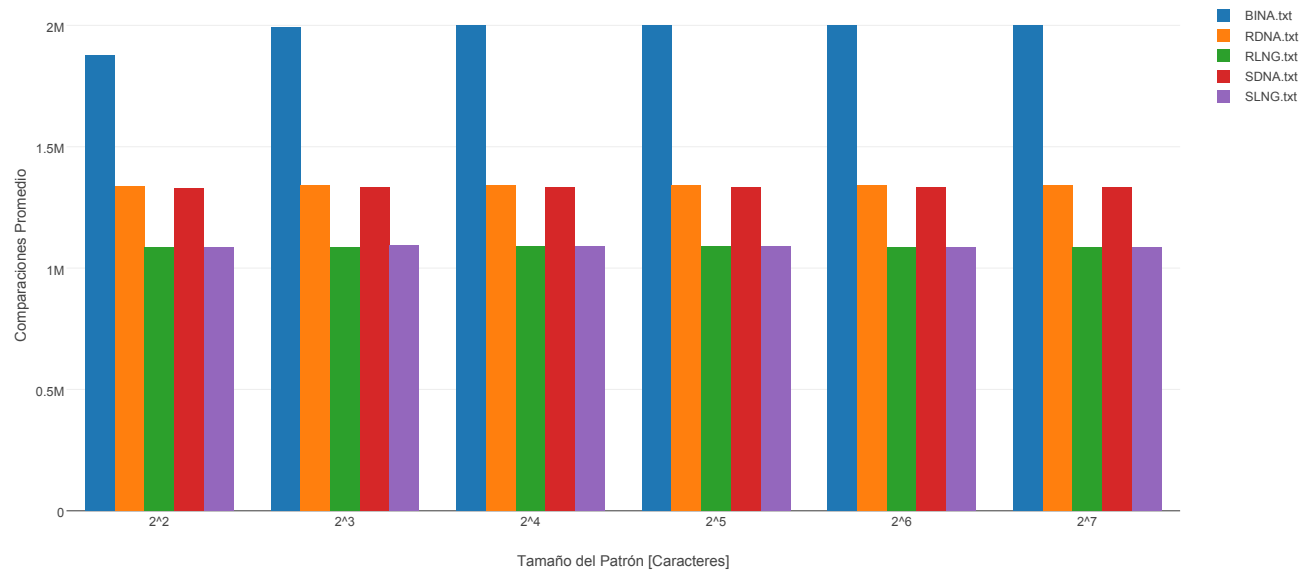


Figura 2: Comparaciones promedio por búsqueda para Fuerza Bruta

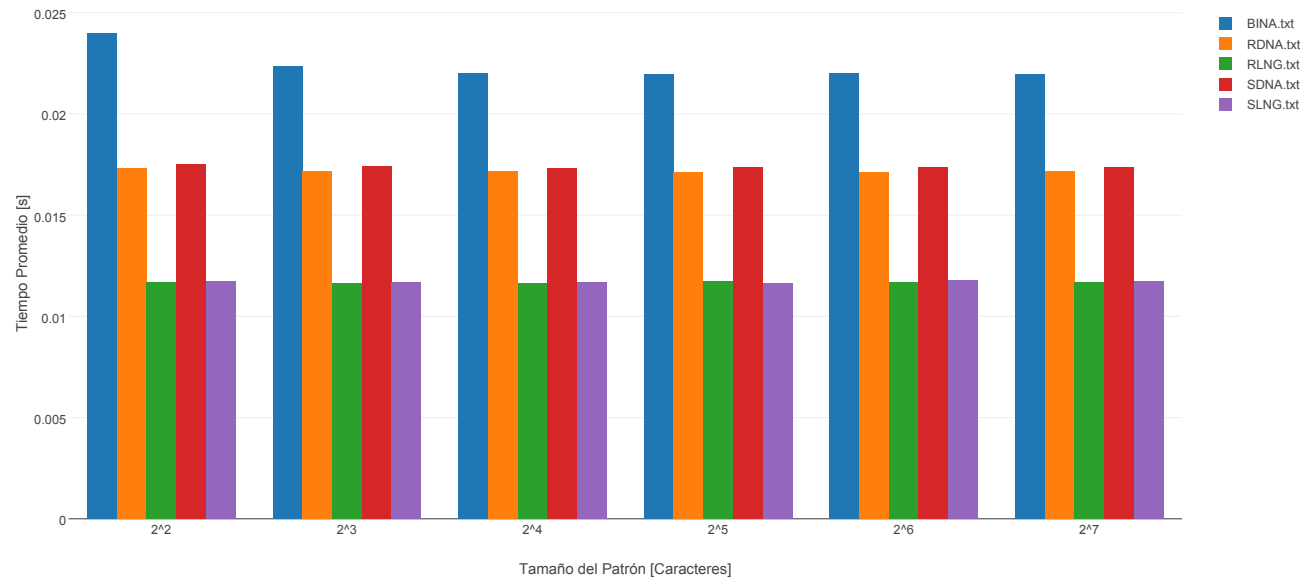


Figura 3: Tiempo de promedio de búsqueda para Knuth-Morris-Pratt

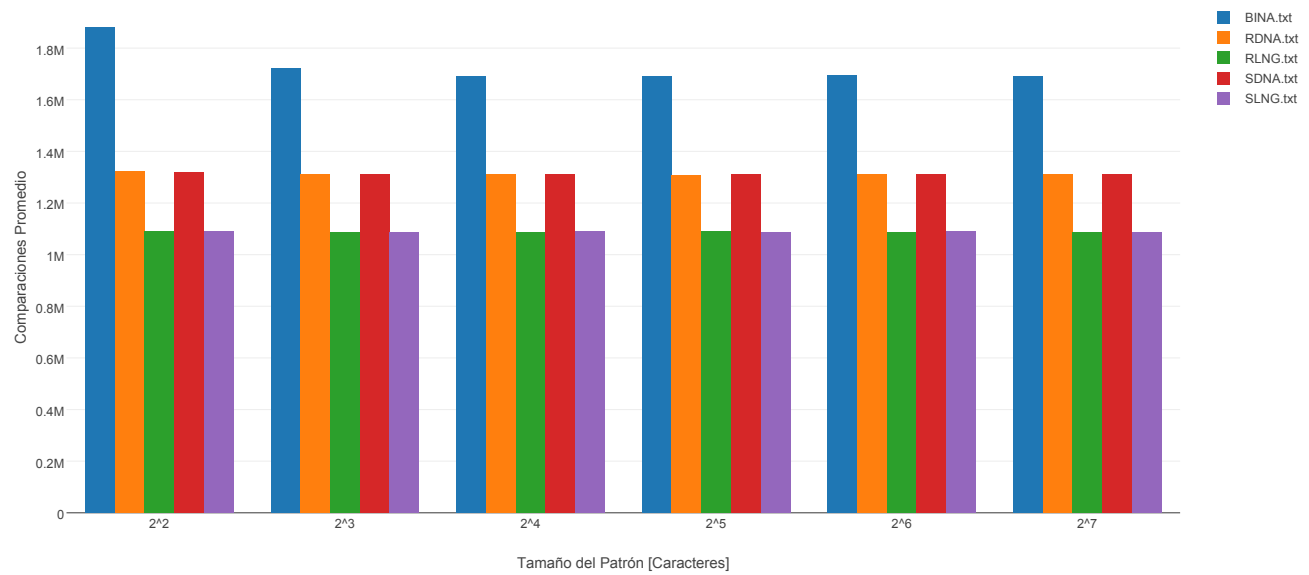


Figura 4: Comparaciones promedio por búsqueda para Knuth-Morris-Pratt

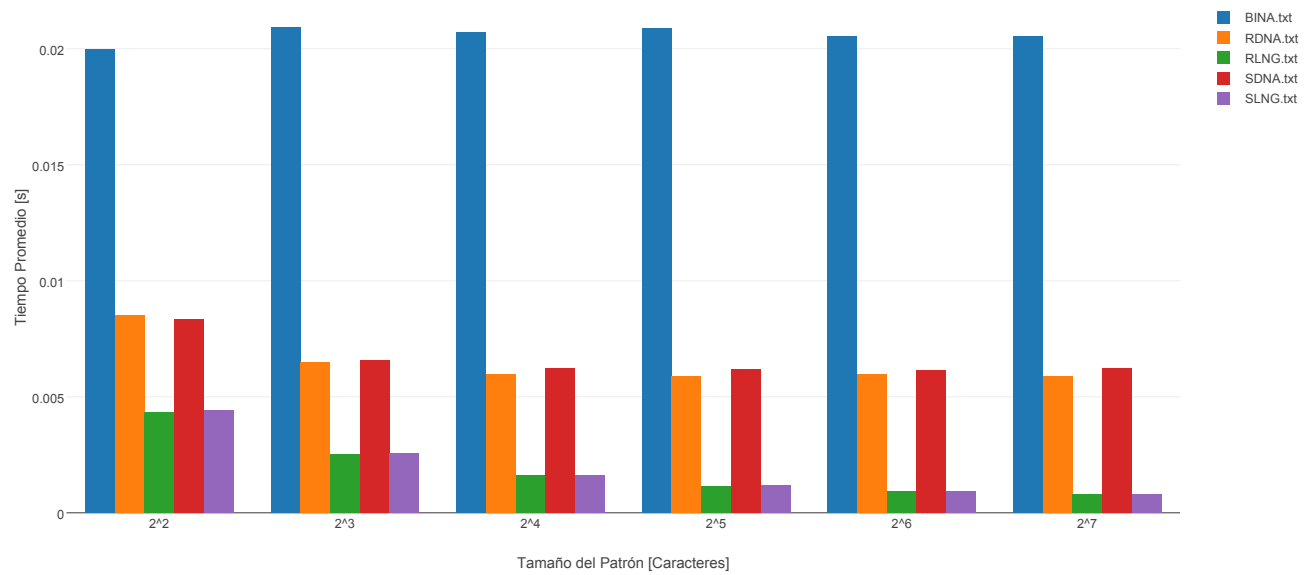


Figura 5: Tiempo de promedio de búsqueda para Boyer-Moore-Horspool

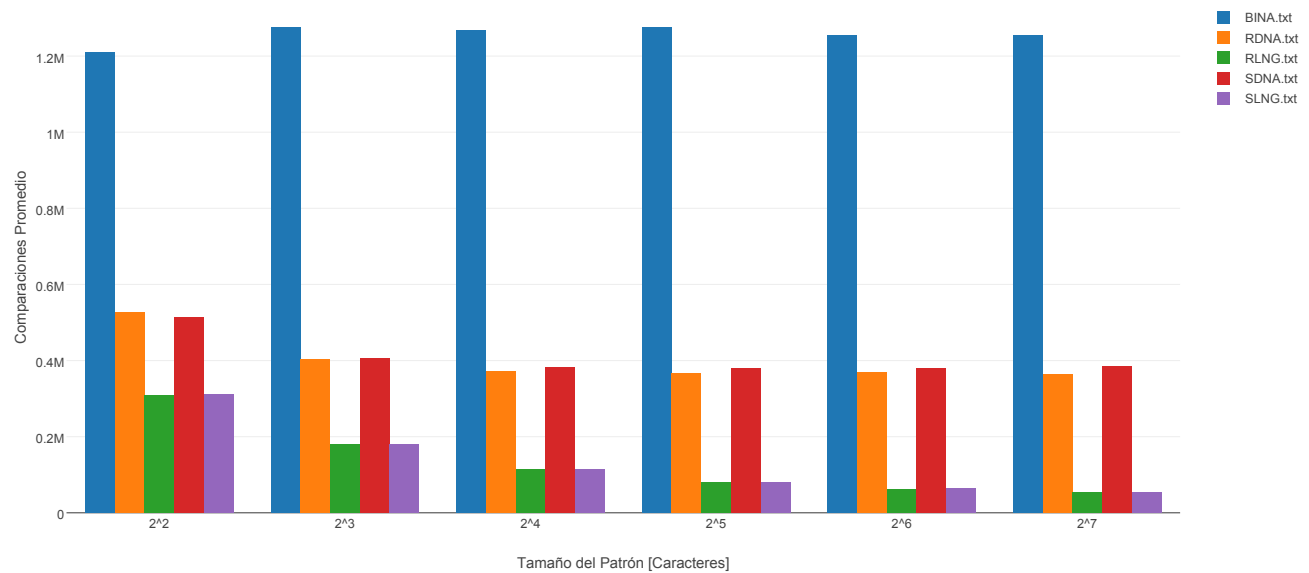


Figura 6: Comparaciones promedio por búsqueda para Boyer-Moore-Horspool

4.2. Datos

Se puede ver de las gráficas anteriores que los resultados sintéticos se asemejan muchos a los naturales. Este resultado es esperable dado que tienen un mismo c , y por tanto, para ser mas concisos al eliminar los casos sintéticos se obtiene:

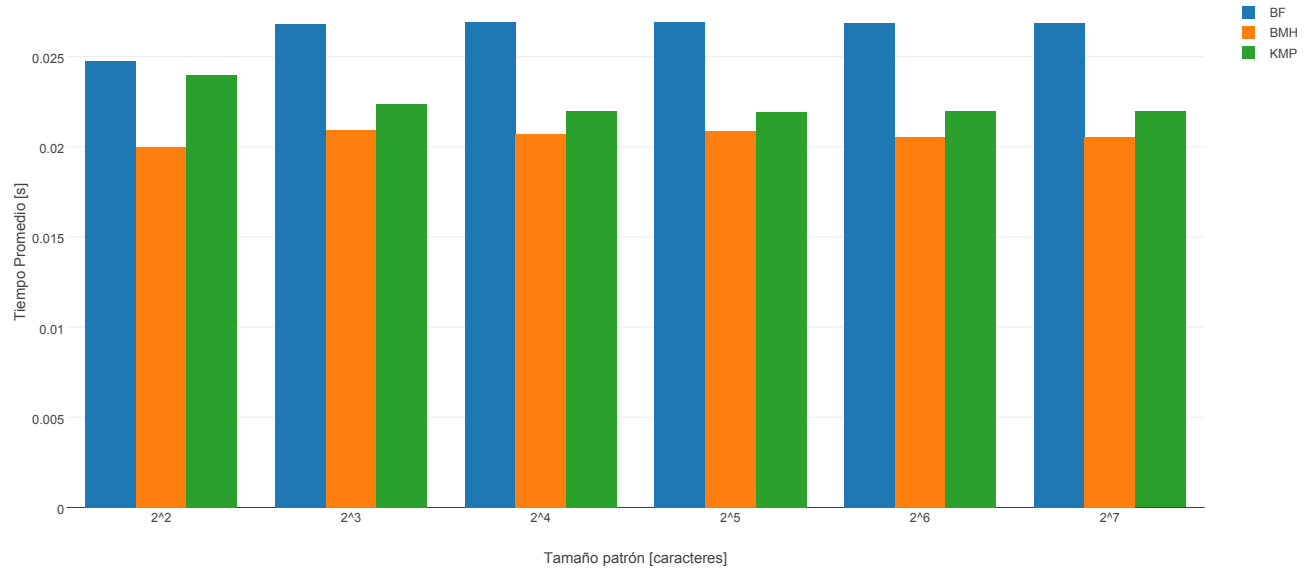


Figura 7: Tiempo promedio de búsqueda para texto binario

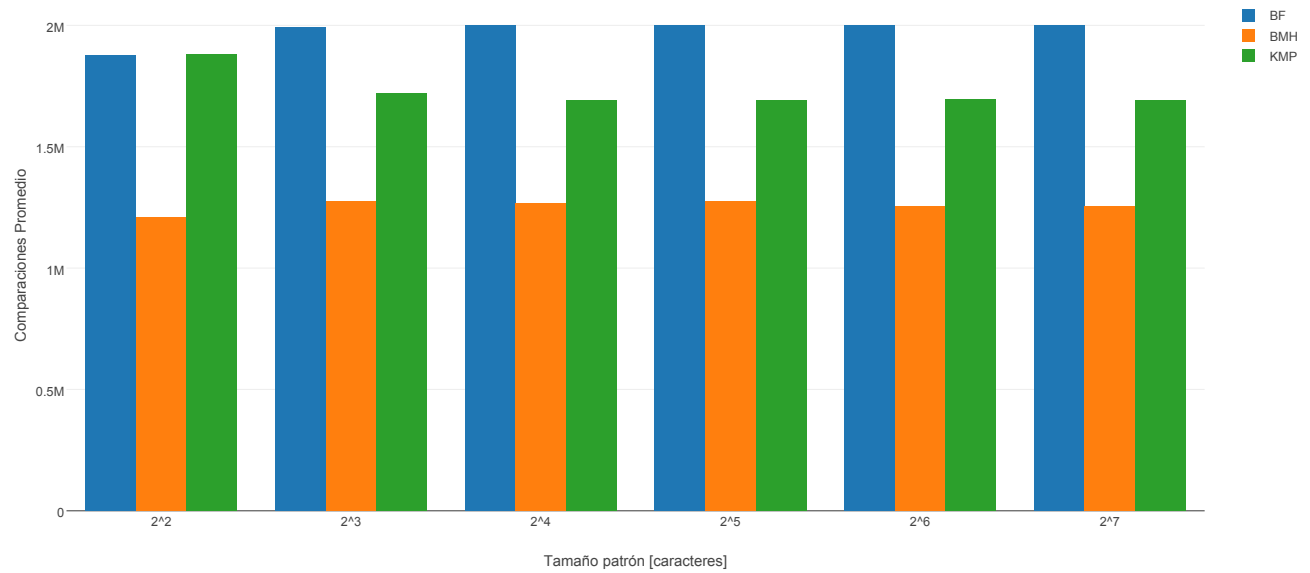


Figura 8: Comparaciones promedio por búsqueda para texto binario

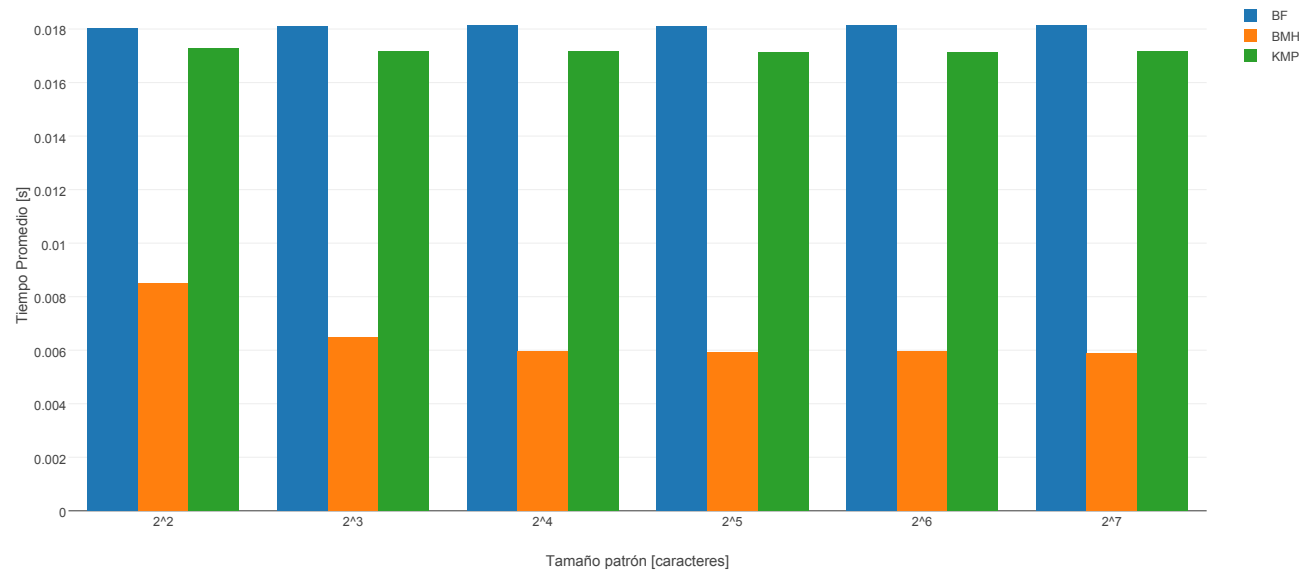


Figura 9: Tiempo promedio de búsqueda para cadenas de ADN

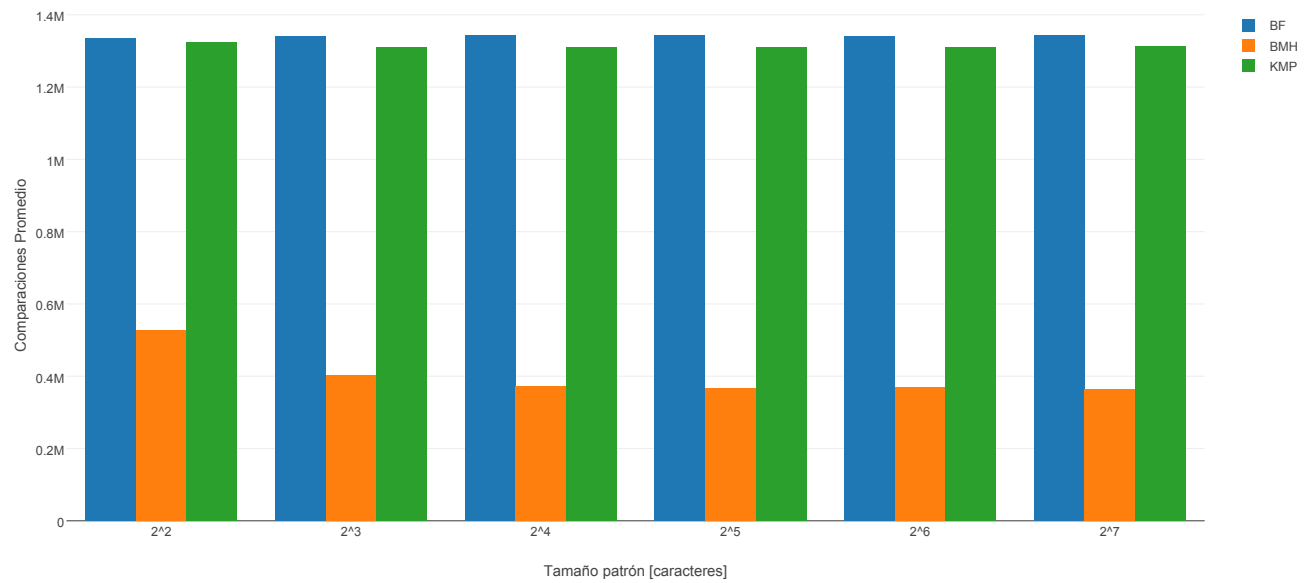


Figura 10: Comparaciones promedio por búsqueda para cadenas de ADN

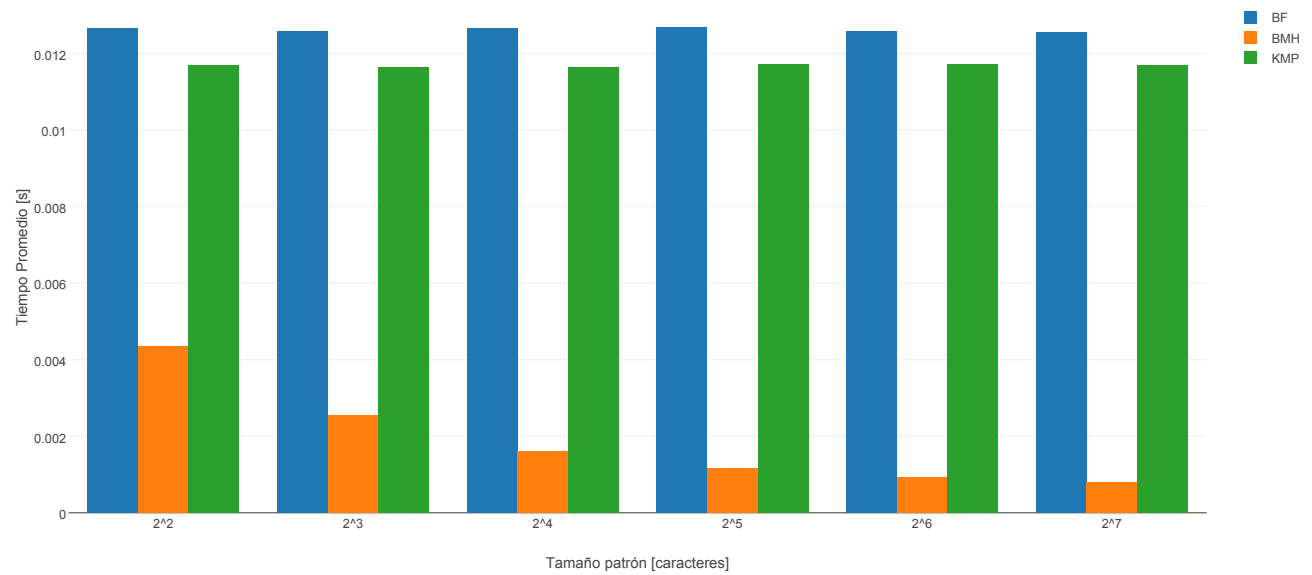


Figura 11: Tiempo promedio de búsqueda para lenguaje natural

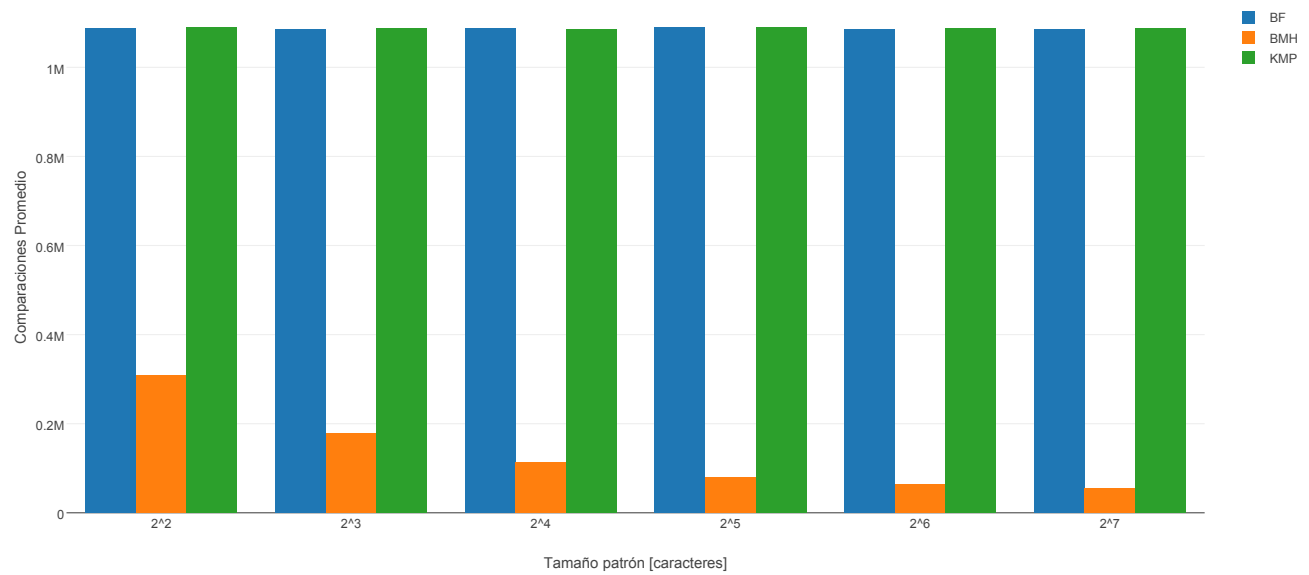


Figura 12: Comparaciones promedio por búsqueda para lenguaje natural

5. Análisis y Conclusiones

En los resultados de cada algoritmo se puede confirmar como aumentar c hace que mejore el rendimiento.

5.1. Fuerza Bruta

No ocurre lo esperado, ya que el rendimiento no empeora al aumentar m .

5.2. Knuth-Morris-Pratt

Este algoritmo cumple con lo esperado, al ser mejor que Fuerza Bruta, pero no por mucho.

Un detalle que se puede apreciar con este algoritmo es que en el caso binario este no empeora como Fuerza Bruta al agrandar m , lo cual es posible debido a que al tener tan sólo 2 caracteres, es mucho más probable tener subcadenas que sean prefijo del patrón.

5.3. Boyer-Moore-Horspool

Este algoritmo cumple con lo esperado, al no ser tan eficiente con c pequeño y mejorando significativamente al aumentarlo, y por ser mas eficiente al agrandar m .