Estructuras de Datos y Algoritmos Avanzados

Proyecto 2: Conteo de Patrones

Alberto Esteban Ferrada Rivas
Francy Pilar Jelvez Jen
Diego Joaquín Andrés Venegas Anabalón

Noviembre 2023

1 Introducción

En este proyecto, vamos a analizar el comportamiento de tres algoritmos y estructuras de datos que tienen el objetivo de realizar búsquedas de patrones, estos serán Boyer-Moore Horspool, Suffix Array y FM-Index. Algunos objetivos de esto serán: Poner en práctica el uso de estas estructuras y algoritmos, no solo quedándonos en lo teórico, además de fortalecer nuestra comprensión de estos y mejorar nuestras habilidades de programación.

1.1 Boyer-Moore Horspool

Boyer-Moore Horspool consiste en la unión de los algoritmos de Boyer-Moore y Horspool, respectivamente. Su característica principal es que usa una sola tabla que guarda las distancias de los chars con respecto al final del patrón. La búsqueda consiste en comparar el patrón y el texto a partir del último char del patrón, y en caso de que no se encuentre, la posición del patrón analizado se mueve según la cantidad de espacios que diga la tabla de las distancias. La comparación entre el texto y el patrón es igual que la comparación a fuerza bruta, pero de atrás para adelante.

1.2 Suffix Array

Suffix Array como tal, consiste en un arreglo de sufijos ordenado lexicográficamente, por lo que también se usa en otras estructuras de datos, algoritmos de búsqueda y algoritmos de compresión. La búsqueda basada en Suffix Array consiste en una búsqueda binaria en el arreglo de sufijos para encontrar uno que coincida parcialmente con el patrón, si el sufijo es igual al patrón, se ha encontrado una ocurrencia del patrón y por ende se incrementa en uno el contador.

1.3 FM-Index

El FM-Index es una estructura de datos compacta que se construye a partir de un texto dado, y su principal característica es la compresión de la información. Primero, el texto se ordena lexicográficamente, creando una matriz de sufijos. Luego, se construye una estructura llamada Burrows-Wheeler Transform (BWT) que reorganiza los sufijos y agrupa caracteres similares. La BWT es la base del FM-Index y permite una búsqueda rápida y eficiente. Se crea una tabla de C (Count) que almacena el número de ocurrencias de cada carácter en el texto ordenado. Para buscar un patrón en el texto, el FM-Index aprovecha la estructura BWT y la tabla de C para buscar el patrón en orden inverso, reduciendo el número de comparaciones necesarias y acelerando la búsqueda. En resumen, el FM-Index permite buscar patrones en un texto grande de manera eficiente gracias a la compresión de datos y la estructura BWT, lo que lo hace especialmente útil en aplicaciones que involucran grandes conjuntos de datos de texto.

2 Implementaciones

2.1 Boyer-Moore Horspool

El funcionamiento de nuestro algoritmo de Boyer-Moore Horspool consiste en usar un array (Dentro de nuestra implementación, se llama PTBad (Pattern Bad) porque se ocupa cuando no coincide un patrón) que guarde las posiciones entre los chars del patrón y su cercanía con otro char del mismo tipo o, en caso contrario, el final de la palabra. La característica particular de este algoritmo es que el análisis del patrón es de derecha a izquierda, significando que se empieza a analizar desde el último char del patrón hacia atrás. Si se encuentra con un elemento que no coincide, se busca en nuestro array el valor del char del texto que no coincide con el patrón, y con ese valor, "corremos" el patrón en el texto, para volver a analizar. Esto se repite hasta que se termine el texto.

2.2 Suffix Array

El funcionamiento de nuestra implementación consiste en la creación de un vector que contiene los índices de todos los sufijos existentes en el texto a analizar, este vector será ordenado lexicográficamente mediante la función sort de la biblioteca algorithm, para esto ocupamos string views, con el objetivo de no ocupar memoria guardando cada sufijo.

Ya con el Suffix Array precomputado, podemos ocupar la función count, en este caso llamada contarPatron, la cual realiza una búsqueda binaria en el arreglo de sufijos hasta que coincida la primera letra del patrón buscado con la letra apuntada por med (valor central en la búsqueda binaria). A continuación, se buscan las ocurrencias en ambos lados de la actual iteración.

2.3 FM-Index

La implementación utilizada es aquella otorgada por Oliver Brito, está compuesto principalmente por un Suffix Array compacto *fm_index*, cuyo principal contenedor es un wavelet tree de huffmann, codificado con la transformada de Burrows Wheeler, implementado con bit_vector's. Además del csa, están las variables max_locations, pre_context y post_context que sirven únicamente para modificar el formato del output, no afectan el funcionamiento del FM-index. Una parte importante de los datos, es que la medida de datos de la construcción del FM-index dió 0 en tiempo, esto debido a que la implementación lo guardaba en binario y lo cargaba para no rearmarlo.

3 Análisis Teórico

A continuación, se presentarán los análisis de teóricos de las implementaciones anteriormente mencionadas:

3.1 Boyer-Moore Horspool

Para los siguientes análisis ocuparemos “n” para representar el tamaño del texto y ”m” para representar el tamaño del patrón.

Para el precomputo de Boyer-Moore Horspool, se crea un array de 256 elementos, uno para cada elemento tipo char. Este array se le da el valor del largo del patrón a todos los elementos, y luego cada elemento del patrón recibe un valor en la tabla. El precomputo siempre va a llenar una tabla de 256, y cuanto demore la asignación de valores, dependerá del largo del patrón, por lo que demorará O(m).

Para el computo, se analiza el patrón ”linealmente”; se mueve el patrón analizado según los espacios indicados por el array que se hizo en el precomputo. Por lo que, en el mejor caso (Que el texto consista únicamente de un patrón repetido, por ejemplo; ”Perro” y el texto es ”PerroPerroPerro”), este algoritmo se demorará O(n), pues avanza uniformemente (Realizará ’n/m’ saltos) y hará ’m’ análisis para comprobar si se cumple el patrón por cada salto, ya que es a fuerza bruta. Y por esto también podemos concluir que el peor caso será O(n*m), en donde forzosamente tenemos que correr por todo el texto (Demorará O(n)) y hacer ’m’ análisis para cada elemento.

3.2 Suffix Array

Para los siguientes análisis ocuparemos “n” para representar el tamaño del texto.

Para el precomputo de Suffix Array se crea un vector y se le asignan números del 1 a n a sus elementos, lo cual tiene una complejidad de O(n). Luego se usa sort, función que dependiendo de la cantidad de elementos utilizara diferentes métodos de ordenamiento, en este caso consideraremos el uso de merge sort, lo que tiene una complejidad de $O(nlogn)$. Con el pequeño análisis anterior, podemos concluir que la construcción del arreglo de sufijos toma tiempo O(n log n).

Para el algoritmo de conteo, tenemos que la búsqueda binaria como mucho puede tener complejidad O(log n), mientras que la búsqueda de ocurrencias, en el peor de los casos (Que se tenga que revisar todo el texto, o más bien que todos los sufijos comiencen con la misma letra) tendrá complejidad O(n). Con lo anterior podemos concluir que este conteo es O(log n+m) donde k es el número total de ocurrencias del patrón en el texto, puesto que el mejor de los casos para la búsqueda binaria es el peor caso para la búsqueda de ocurrencias, y viceversa.

Con lo anterior, podemos concluir que el algoritmo en total tomara complejidad O(log n) o O(n) dependiendo de la distribución y la repetición de los patrones en el texto.

Por otra parte, en este algoritmo el tamaño del patrón no debería afectar tanto, puesto que las comparaciones que se hacen son acordes a si coinciden en la primera letra y no al tamaño del texto, mientras que lo que más afecta es el tamaño total del texto, dado que mayor tendrá que ser el arreglo de sufijos y mayor será el precomputo.

Por otro lado si el texto tiene n char´s, el arreglo de sufijos guardara n unsigned int´s, lo que es equivalente 4*n bytes, por lo que se ocuparan O(n) bytes.

3.3 FM-Index

El FM-index está compuesto de dos estructuras principales, la BWT y el arreglo C (count) que guarda para todo elemento de Σ la cantidad de elementos menores a el, que hay en el mismo. La BWT se utiliza para comprimir y hacer las búsquedas. La complejidad asintótica de cada búsqueda es $O(m)$, donde m es el largo del patrón a ser buscado. Además la construcción del FM-index tiene una complejidad temporal de $O(n)$.

4 Hipótesis

4.1 Alberto Ferrada

En mi opinión, el menos eficiente de los tres, debiera ser el algoritmo de Boyer-Moore Horspool, pues lo que haremos más adelante será "cargar" el texto sobre el que se ejecutan las búsquedas y sobre eso hacer varias consultas, como sabemos, las estructuras de datos son más eficientes que los algoritmos de una pasada en ese sentido. Mientras que el FM-index debiera ser más rápido que el suffix array debido a los tamaños de textos que usamos, pues al ser este compacto y dado el tamaño de nuestro archivo, cabe en la memoria cache. La complejidad espacial del FM-index está acotada por $O(n)$.

4.2 Francy Jelvez

Mi hipótesis es que, de todos los algoritmos, los que se demorarán más serán Boyer-Moore Horspool y Suffix Array. Para Boyer-Moore Horspool, es porque todo análisis recae en la misma función de conteo. No subdivide el texto en fragmentos más pequeños, ni tampoco realiza algoritmos de búsqueda como Suffix Array y la búsqueda binaria, si no que depende de que tan cerca esté el patrón del punto analizado. Entre FM-Index y Suffix Array no habría mucha diferencia, pero yo creo que Suffix se demoraría más, ya que FM Index ocupa más estructuras compactas que Suffix en su búsqueda.

4.3 Diego Venegas

Según lo estudiado y visto en nuestras implementaciones, podría afirmar que Boyer-Moore Horspool será el algoritmo que mas tiempo tardara en el conteo de patrones, mientras que por el lado de Suffix Array y FM-Index no me atrevería a decir cual tardara menos, esto lo digo bajo los siguientes argumentos:

- Boyer-Moore Horspool tiene una muy pequeña precomputación, lo cual hace que, al momento de realizar el conteo, tenga como peor caso que el patrón sea tamaño cercano a 1 y por ende se recorra todo el string, de manera lineal.
- Por otro lado, Suffix Array y FM-Index sacrifican un poco más de memoria y tiempo realizando una precomputación, lo cual hace que count, requiera menos trabajo.

Además, en casos de búsqueda de patrones largos Boyer-Moore Horspool debería tener mejores tiempos que con patrones cortos.

5 Entorno de Hardware y Software

Usamos el lenguaje de programación C++ con el compilador de GCC, sin ninguna opción de optimización específica. Para realizar los experimentos, los ejecutamos con los servidores de chome.inf.udec.

6 Análisis Experimental

6.1 Boyer-Moore Horspool

En cuanto al Boyer-Moore Horspool, podemos decir que los resultados van de acuerdo con lo que se comentaba acerca del pequeño precomputo y el algoritmo de conteo que es O(n). Con respecto a los experimentos de tamaño de patrón, también podemos decir que son bastantes fieles a los que decíamos.

6.1.1 Variación en el tamaño del texto

Tamaño del Texto	Tiempo Precomputación (microsegundos)	Tiempo Contar Patrón (microsegundos)	Tiempo Total (microsegundos)	Varianza Precomputación (microsegundos)	Varianza Contar Patrón (microsegundos)	Varianza Total (microsegundos)
10000	1	237	238	4	15	14
20000	1	430	431	3	54	53
30000	1	549	550	5	122	138
40000	1	732	733	5	199	189
50000	1	899	901	3	307	299
60000	1	1084	1086	5	351	356
70000	1	1286	1288	4	414	429
80000	1	1480	1481	3	712	703
90000	1	1669	1671	5	736	716
100000	1	1868	1870	3	725	723
110000	1	2074	2075	3	720	693
120000	2	2296	2297	7	1110	1072
130000	1	2477	2479	5	1292	1249
140000	1	2652	2654	5	1033	1035
150000	2	2817	2818	3	1040	1018
160000	2	2993	2994	5	1355	1374
170000	1	3156	3158	4	1852	1843
180000	2	3322	3323	5	1581	1519
190000	2	3489	3490	6	1113	1130
200000	2	3670	3671	3	988	989

Figure 1: Tabla para diferentes tamaños de texto en BMH

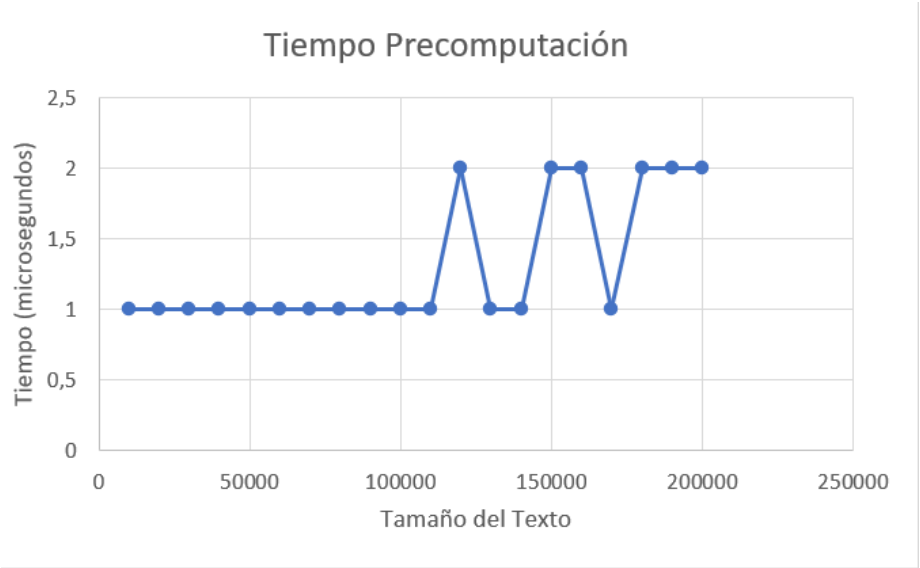

Figure 2: Gráfico para el Precomputo

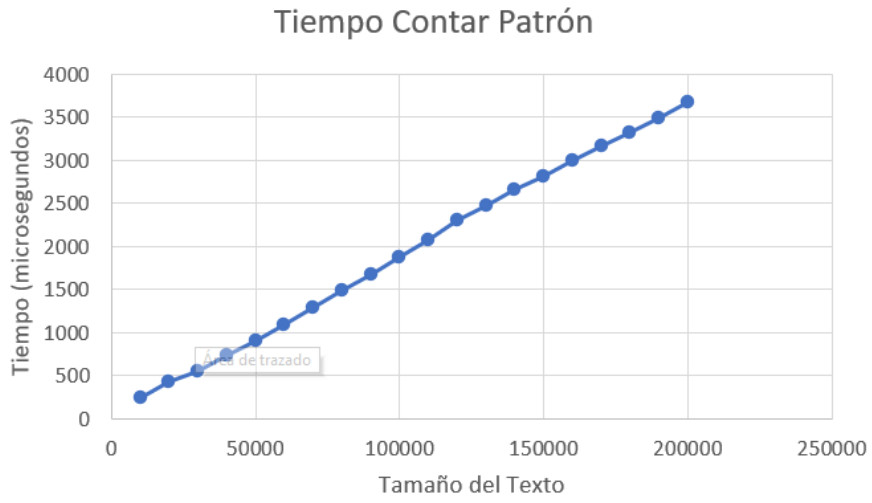

Figure 3: Gráfico para el Conteo

6.1.2 Variación en el tamaño del patrón

Largo del Patrón	Tiempo Precomputación (microsegundos)	Tiempo Contar Patrón (microsegundos)	Tiempo Total (microsegundos)	Varianza Precomputación (microsegundos)	Varianza Contar Patrón (microsegundos)	Varianza Total (microsegundos)
100	2	9529	9531	2	50927768	50929358
150	3	3885	3888	3	807	873
200	4	6030	6034	4	2147	2172
250	4	10364	10368	5	26469424	26467869
300	5	4606	4611	4	1654	1707
350	6	7204	7210	3	1512	1578
400	6	6595	6601	3	1383	1439
450	7	7533	7540	3	2409	2470
500	8	5362	5370	3	1074	1099
550	8	8255	8263	3	4292510	4290689
600	9	9809	9818	3	2471	2509
650	9	6892	6901	4	2033	2114
700	10	9517	9528	5	2439	2523
750	11	7897	7908	3	1637	1637
800	11	5358	5369	3	1334	1384
850	12	6261	6273	3	2432	2503
900	13	5933	5946	4	48959	49019
950	13	12472	12485	3	7998287	7999066
1000	14	7419	7433	3	3275	3372

Figure 4: Tabla para diferentes tamaños patrón en BMH

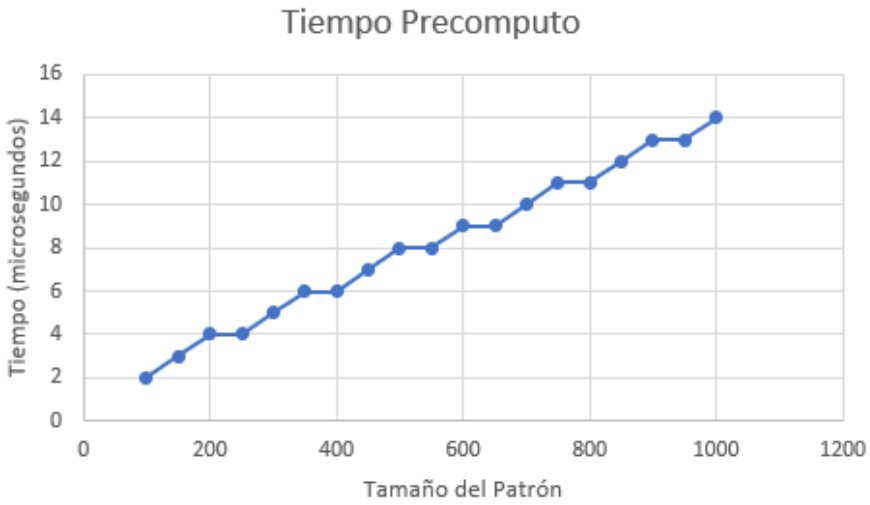

Figure 5: Gráfico para el Precomputo

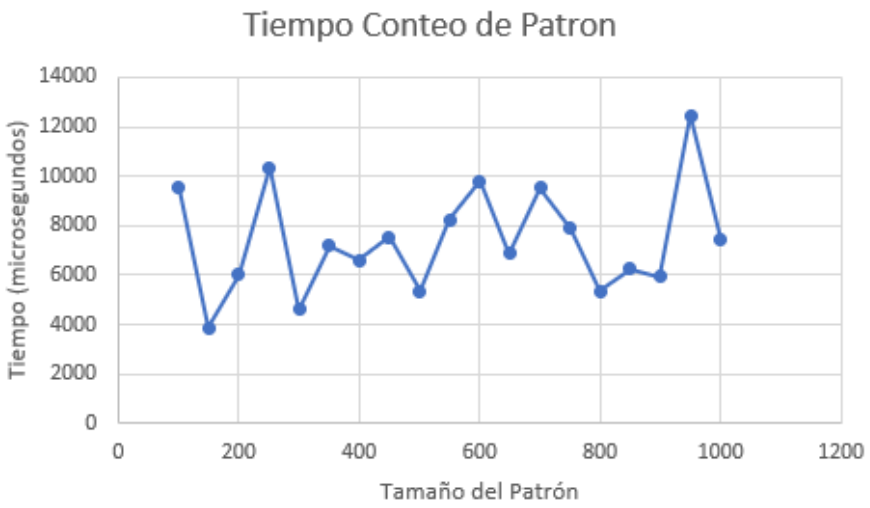

Figure 6: Gráfico para el Conteo

6.2 Suffix Array

Por parte del Suffix Array podemos decir que dio los resultados esperados, puesto que el precomputo del arreglo de sufijos es la operación que más tiempo lleva, mientras que el conteo es mucho menor, además por lo visto en los gráficos, se corresponden a algoritmos O(n log n) y O(log n) respectivamente. También podemos decir lo mismo en cuanto al análisis de diferentes tamaños de patrones, dado que no hay mayores cambios en los resultados.

6.2.1 Variación en el tamaño del texto

Tamaño del texto	Tiempo Precomputación (microsegundos)	Tiempo Contar Patrón (microsegundos)	Tiempo Total (microsegundos)	Varianza Precomputación (microsegundos)	Varianza Contar Patrón (microsegundos)	Varianza Total (microsegundos)
10000	21227	1	21228	51146210	3	51146887
20000	40374	2	40376	14633	4	14810
30000	61102	2	61104	50859	3	51267
40000	93802	2	93804	455156538	3	455156043
50000	117503	3	117505	281555064	4	281569860
60000	142292	3	142294	493241908	3	493241154
70000	169639	4	169642	915039487	3	915027147
80000	199134	5	199139	992925448	3	992930162
90000	219664	5	219669	361496024	4	361501101
100000	252401	5	252405	815203116	4	815200255
110000	278724	6	278731	3011472810	5	3011471230
120000	303688	6	303695	2361307029	5	2361310286
130000	340070	6	340076	2637697028	6	2637681880
140000	362907	7	362914	2661415787	4	2661501431
150000	398095	8	398103	4562006848	9	4561990990
160000	418781	8	418789	4093121466	3	4093143675
170000	474475	9	474484	5446891399	3	5446858342
180000	474830	8	474838	2845725020	6	2845780934
190000	520453	8	520461	2988166844	6	2988197071
200000	580457	8	580465	9520593813	4	9520663211

Figure 7: Tabla para diferentes tamaños texto en Suffix Array

Figure 8: Gráfico para el Precomputo

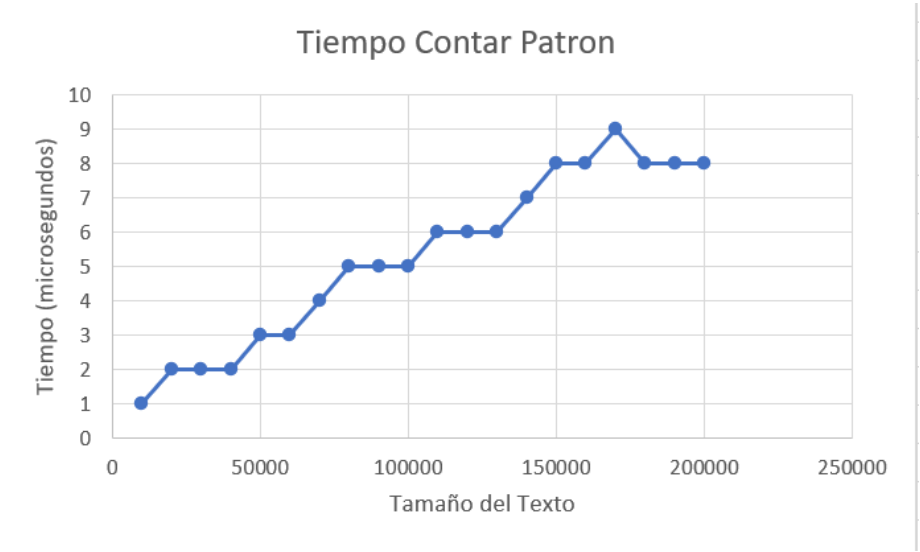

Figure 9: Gráfico para el Conteo

6.2.2 Variación en el tamaño del patrón

Tamaño del Patrón	Tiempo Precomputación (microsegundos)	Tiempo Contar Patrón (microsegundos)	Tiempo Total (microsegundos)	Varianza Precomputación (microsegundos)	Varianza Contar Patrón (microsegundos)	Varianza Total (microsegundos)
100	527430	7	527437	5006442996	6	5006456716
150	534658	9	534667	2455116934	11	2455270402
200	521997	9	522006	1522302879	19	1522434002
250	514056	6	514062	2006826295	4	2006828243
300	542544	9	542552	2255591770	12	2255685758
350	538359	7	538366	4098395740	23	4098681839
400	531706	7	531713	5058202754	8	5058275804
450	528876	7	528884	3254155259	13	3254286121
500	550580	7	550588	5906288475	12	5906472066
550	564391	8	564398	6601541051	10	6601615517
600	533772	7	533779	5129654881	11	5129720296
650	530752	9	530761	9401021635	8	9401102717
700	548107	8	548115	5474431229	13	5474537605
750	565125	8	565134	10183846317	9	10184170134
800	495228	10	495238	5020529995	14	5020710828
850	529631	8	529639	2841147014	12	2841144408
900	528924	7	528931	3030303001	12	3030437649
950	523456	9	523464	3873018469	10	3873090034
1000	524123	7	524130	1776979590	11	1777023744

Figure 10: Tabla para diferentes tamaños de patrón en Suffix Array

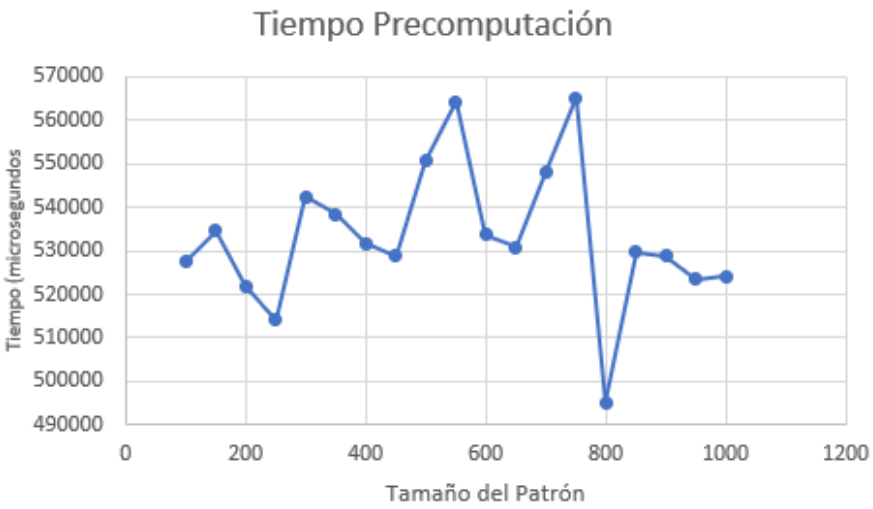

Figure 11: Gráfico para el Precomputo

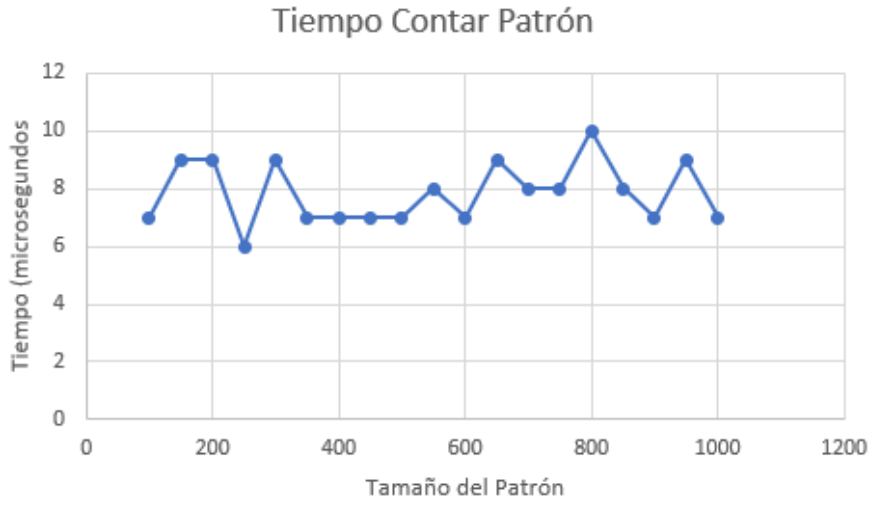

Figure 12: Gráfico para el Conteo

6.3 FM-Index

Acorde a los resultados, podemos decir que estábamos en los correcto al pensar que FM-Index seria el algoritmo que mejores resultados arrojaría en conteo, por otro lado, también hay que mencionar que la implementación es la ofrecida por Oliver Brito, y en esta, el precomputo se realiza una sola vez y es guardada en un archivo, de esta forma, para las siguientes búsquedas que se realizarán, no será necesario hacer un precomputo.

6.3.1 Variación en el tamaño del texto

Tamaño del Texto	Tiempo Precomputación (microsegundos)	Tiempo Contar Patrón (microsegundos)	Tiempo Total (microsegundos)	Varianza Precomputación (microsegundos)	Varianza Contar Patrón (microsegundos)	Varianza Total (microsegundos)
1000	0	71	0	0	127	200
2000	0	69	0	0	128	200
3000	0	70	0	0	111	240
4000	0	70	0	0	140	230
5000	0	71	0	0	126	202
6000	0	72	0	0	127	201
7000	0	78	0	0	128	221
8000	0	71	0	0	120	199
9000	0	70	0	0	127	204
10000	0	75	0	0	125	198

Figure 13: Tabla para diferentes tamaños de texto en FM-Index

Figure 14: Gráfico para el Conteo

6.3.2 Variación en el tamaño del patrón

Tamaño del Patrón	Promedio del Conteo (microsegundos)	Desviación estándar (microsegundos)
8	0.38178	0.0936486
32	0.3857	0.0975004
128	0.4164	0.152917
512	0.40208	0.147496
2048	0.38634	0.115276
8192	0.39396	0.130943
32768	0.47048	0.181362
65536	0.48366	0.2135

Figure 15: Tabla para diferentes tamaños de patrón en FM-Index

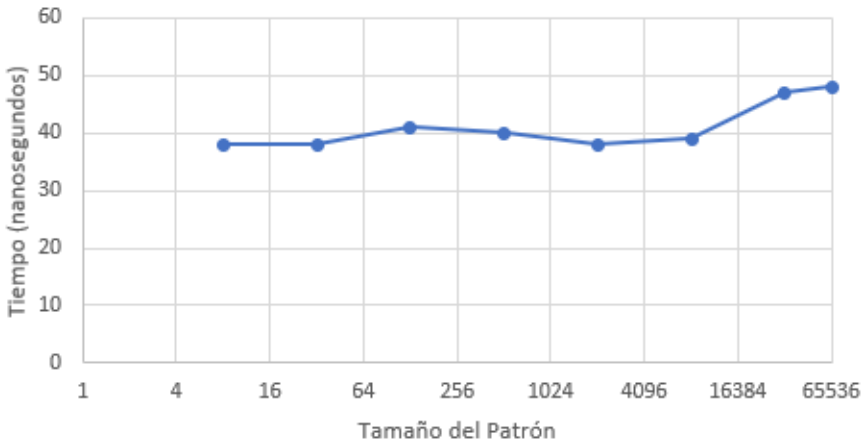

Figure 16: Gráfico para el Conteo

7 Conclusiones

Este proyecto ha sido una buena instancia para poner en práctica nuestros conocimientos, además de adquirir otros en la práctica, y con los anteriores análisis podemos concluir varias cosas.

Claramente, como en muchos casos, en el tema de algoritmos y estructuras de datos no existe un "mejor en todo" si no que cada uno de estos tiene sus fortalezas y es conveniente de ocupar en base al entorno que tenemos, teniendo en cuenta esto podemos dar las siguientes conjeturas:

- Boyer-Moore Horspool es un algoritmo que, si bien requiere un precomputo, este es muy pequeño, y no requiere ocupar grandes espacios en memoria, a consecuencia de esto, si requerimos hacer varias consultas esta no será la mejor opción.

- Suffix Array entre los dos otros algoritmos es el que mas espacio ocupa, debido a su precomputo, pero una de las ventajas que tiene esto, es que, si guardamos este arreglo, podremos hacer múltiples consultas que no llevaran un tiempo excesivo.

- FM-Index por tener la característica de ser una estructura compacta, prioriza el ocupar poco espacio en memoria, por lo que se encuentra entre el Boyer-Moore Horspool y el Suffix Array en cuanto a uso de espacio. Al igual que el Suffix Array, tiene la ventaja de que, si guardamos la estructura, podremos realizar múltiples consultas en poco tiempo.

- FM-Index tiene la ventaja de que no requiere el uso del texto para realizar las consultas, mientras que el conteo en el Suffix Array requiere el arreglo y además el texto original y Boyer-Moore Horspool requiere usar comparaciones con el texto.

8 Referencias

- Wikipedia: Boyer-Moore-Horspool Algorithm
- Mike Slade, en Youtube: Boyer-Moore-Horspool Algorithm Explanation
- Archivo usado para el análisis experimental proviene de Pizza&Chili