

Laboratorio 3: String matching

12 de Noviembre, 2025

Alumno Mirko Peñailillo Vasquez
Profesor José Fuentes Sepúlveda
Ayudante Luciano Argomedo
Nº Matricula 2021900322

1 Resumen

El objetivo de este laboratorio es implementar, comprender y analizar algoritmos de búsqueda de patrones en cadenas de texto (string matching), así como evaluar experimentalmente dichas implementaciones. Para el desarrollo de este boletín se estudió el FM-Index y el algoritmo Rabin-Karp, comparando su desempeño en términos de tiempo de construcción, tiempo de búsqueda y uso de memoria.

2 Análisis Teórico

2.1 FM-Index

El FM-Index (Full-text Minute-space Index) es una estructura de datos comprimida que permite realizar búsquedas de patrones eficientemente. Se basa en la Transformada de Burrows-Wheeler (BWT) y utiliza funciones de rango para navegar por el texto.

Características principales:

- **Construcción:** $O(n)$ en tiempo, donde n es el tamaño del texto.
- **Búsqueda:** $O(m + occ)$ en tiempo, donde m es el largo del patrón y occ es el número de ocurrencias.
- **Espacio:** Requiere espacio proporcional al texto comprimido, típicamente menor que el texto original.
- **Ventaja:** Excelente para búsquedas repetidas sobre texto estático, con bajo consumo de memoria.
- **Desventaja:** Tiempo de construcción considerable para textos grandes.

2.2 Rabin-Karp

El algoritmo de Rabin-Karp es un algoritmo de búsqueda de patrones basado en hashing que utiliza una función hash rolling para verificar coincidencias potenciales.

Características principales:

- **Construcción:** No requiere preprocessamiento del texto, $O(1)$.
- **Búsqueda:** $O(n + m)$ en tiempo promedio, $O(nm)$ en el peor caso.
- **Espacio:** $O(1)$ adicional al texto original.
- **Ventaja:** Simple de implementar, sin preprocessamiento.
- **Desventaja:** Tiempo de búsqueda lineal con el tamaño del texto.

3 Implementación

3.1 FM-Index

Para la implementación del FM-Index se utilizó la biblioteca SDSL-lite (Succinct Data Structure Library), específicamente la clase `csa_wt<>` que implementa un Compressed Suffix Array con Wavelet Tree. La función principal utilizada es:

- `construct_im()`: Construye el índice en memoria a partir del texto.
- `count()`: Retorna el número de ocurrencias del patrón en el texto indexado.

El código fue adaptado de ejemplos de la biblioteca SDSL (en particular `sds1-lite/examples/fm-index.cpp`) y la documentación disponible en [1].

3.2 Rabin-Karp

La implementación del algoritmo Rabin-Karp fue obtenida de el repositorio de github del curso [2]. Sus características relevantes son:

- **Base:** $d = 256$ (alfabeto extendido ASCII).
- **Módulo:** $q = 101$ (número primo para reducir colisiones).
- **Hash rolling:** Se calcula el hash de manera incremental para cada posición del texto.
- **Verificación:** Cuando los hashes coinciden, se verifica carácter por carácter para confirmar la coincidencia real.

3.3 Entorno de desarrollo

Todos los experimentos fueron realizados en una máquina virtual Linux (WSL - Windows Subsystem for Linux) para poder utilizar las bibliotecas necesarias, particularmente SDSL-lite que tiene mejor soporte en entornos Unix. La máquina virtual contaba con 7 GB de memoria RAM disponible.

4 Resultados Experimentales

4.1 Decisiones de inicialización

4.1.1 Texto utilizado

Se utilizaron los archivos `proteins` y `proteins.200MB` del repositorio Pizza&Chili [3]. Estos archivos contienen secuencias de proteínas y tienen tamaños 1GB y 200MB respectivamente (aproximado).

Nota importante: Si bien las instrucciones del laboratorio solicitan utilizar un texto de más de 1 GB, debido a las limitaciones de memoria de la máquina virtual (7 GB RAM disponible), se limitó el tamaño máximo del texto a 500 MB. Textos más grandes agotaban la memoria disponible durante la construcción del FM-Index, causando que el sistema operativo terminara el proceso.

4.1.2 Experimento 1: Efecto del tamaño del texto

- Se probaron tamaños de texto de 10, 20, 50, 100, 200 y 500 MB.
- Para cada tamaño, se cargó una porción del archivo `proteins.txt`.
- Se generó un patrón aleatorio de longitud 10 caracteres extraído del propio texto.
- Se realizaron 30 ejecuciones (runs) de búsqueda para cada algoritmo.
- Se midió el tiempo de construcción del FM-Index.
- Se midió el tiempo promedio de búsqueda para ambos algoritmos.
- Se registró el uso de memoria de cada estructura.

4.1.3 Experimento 2: Efecto del tamaño del patrón

- Se utilizaron dos configuraciones:
 - Texto de 200 MB (`proteins200MB.txt`)
 - Texto de 500 MB (porción de `proteins.txt`)
- Se probaron longitudes de patrón: 5, 10, 15, 20, 25, 30, 40, 50, 75 y 100 caracteres.
- El FM-Index se construyó una sola vez al inicio del experimento.
- Para cada longitud de patrón, se generó un patrón aleatorio extraído del texto.
- Se realizaron 30 ejecuciones de búsqueda para cada longitud de patrón y algoritmo.

4.2 Características de la máquina donde se realizaron los experimentos

- Sistema Operativo: WSL (Windows Subsystem for Linux)
- Procesador: AMD Ryzen 5 3600 (3.6 GHz)
- Memoria RAM física: 16 GB DDR4 (3600 MHz)
- Memoria RAM disponible para WSL: 7 GB

4.3 Descripción de los resultados mediante tablas y gráficos

4.3.1 Experimento 1: Efecto del tamaño del texto

Table 1: Resultados del experimento de tamaño del texto (patrón de longitud 10)

Tamaño (MB)	Algoritmo	Construcción (ms)	Búsqueda (μs)	Memoria (MB)
10	FM-Index	5426.6	76.68	4.20
10	Rabin-Karp	0	107,715	10
20	FM-Index	11,480.8	72.03	8.69
20	Rabin-Karp	0	213,226	20
50	FM-Index	32,331.4	89.65	24.85
50	Rabin-Karp	0	537,509	50
100	FM-Index	69,014.2	80.29	49.96
100	Rabin-Karp	0	1,068,920	100
200	FM-Index	142,389	78.44	97.02
200	Rabin-Karp	0	2,134,010	200
500	FM-Index	326,763	95.89	224.42
500	Rabin-Karp	0	5,400,370	500

Observaciones:

- El tiempo de construcción del FM-Index crece aproximadamente de forma lineal con el tamaño del texto.
- El tiempo de búsqueda del FM-Index crece bastante poco respecto del tamaño del texto.
- El tiempo de búsqueda de Rabin-Karp crece linealmente con el tamaño del texto, como se esperaba teóricamente.
- El FM-Index utiliza aproximadamente la mitad de la memoria comparado con el tamaño original del texto, confirmando su naturaleza comprimida.
- Para un texto de 500 MB, Rabin-Karp es aproximadamente 56,000 veces más lento que FM-Index en búsqueda.

4.3.2 Experimento 2: Efecto del tamaño del patrón

Table 2: Resultados para texto de 200 MB

Long. Patrón	Algoritmo	Búsqueda (μ s)	Ocurrencias
5	FM-Index	31.39	233
	Rabin-Karp	2,133,060	233
10	FM-Index	82.30	2
	Rabin-Karp	2,085,530	2
20	FM-Index	166.15	2
	Rabin-Karp	2,183,530	2
50	FM-Index	567.38	1
	Rabin-Karp	2,156,340	1
100	FM-Index	869.92	3
	Rabin-Karp	2,210,870	3

Table 3: Resultados para texto de 500 MB

Long. Patrón	Algoritmo	Búsqueda (μ s)	Ocurrencias
5	FM-Index	35.20	8032
	Rabin-Karp	5,530,190	8032
10	FM-Index	74.12	127
	Rabin-Karp	5,382,920	127
20	FM-Index	156.81	1
	Rabin-Karp	5,588,680	1
50	FM-Index	454.51	1
	Rabin-Karp	5,478,410	1
100	FM-Index	925.17	2
	Rabin-Karp	5,477,880	2

Observaciones:

- El tiempo de búsqueda del FM-Index crece aproximadamente de forma lineal con la longitud del patrón, como se espera teóricamente ($O(m)$).
- El tiempo de búsqueda de Rabin-Karp se mantiene relativamente constante respecto a la longitud del patrón, dominado por el tiempo de recorrer todo el texto.
- Para patrones cortos, FM-Index es significativamente más rápido.
- A medida que aumenta la longitud del patrón, la diferencia se reduce pero FM-Index sigue siendo más eficiente.
- El número de ocurrencias encontradas es idéntico para ambos algoritmos, validando la correctitud de las implementaciones.

5 Conclusiones

5.1 Validación de complejidades temporales teóricas

Los resultados experimentales confirman las complejidades teóricas esperadas:

5.1.1 FM-Index

- **Construcción:** Se observa un crecimiento aproximadamente lineal ($O(n)$) con el tamaño del texto. Para 500 MB se requieren aproximadamente 327 segundos.
- **Búsqueda:** El tiempo de búsqueda es independiente del tamaño del texto y crece linealmente con la longitud del patrón, confirmando la complejidad $O(m + occ)$ donde m es la longitud del patrón. Los tiempos se mantienen en el orden de microsegundos incluso para textos de 500 MB.
- **Memoria:** El uso de memoria es aproximadamente 45% del tamaño del texto original, demostrando la naturaleza comprimida de la estructura.

5.1.2 Rabin-Karp

- **Construcción:** No requiere preprocesamiento, confirmando su complejidad $O(1)$.
- **Búsqueda:** El tiempo de búsqueda crece linealmente con el tamaño del texto, confirmando la complejidad $O(n + m)$. Para un texto de 500 MB, el tiempo promedio es de aproximadamente 5.4 segundos por búsqueda.
- **Memoria:** Utiliza únicamente el espacio del texto original sin estructuras adicionales.

5.2 ¿Qué algoritmo usar y cuándo?

5.2.1 Usar FM-Index cuando:

- Se realizarán múltiples búsquedas sobre el mismo texto.
- El texto es estático y no requiere modificaciones frecuentes.
- Se necesita optimizar el tiempo de búsqueda, especialmente para textos grandes.
- Se dispone de tiempo para la construcción inicial de la estructura.
- La memoria es una preocupación pero se puede tolerar el overhead de compresión (aproximadamente 45-50% del texto original).
- **Ejemplo práctico:** Búsqueda en bases de datos genómicas, motores de búsqueda de documentos, análisis de logs históricos.

5.2.2 Usar Rabin-Karp cuando:

- Se necesita realizar una o pocas búsquedas sobre el texto.
- El texto cambia frecuentemente o es generado dinámicamente.
- No se dispone de tiempo o recursos para preprocesamiento.
- El tamaño del texto es relativamente pequeño (< 10 MB).
- Se requiere una implementación simple sin dependencias externas.
- **Ejemplo práctico:** Búsqueda en streaming de datos, validación en tiempo real, detección de plagios en documentos pequeños.

5.2.3 Análisis de punto de equilibrio

Basándose en los resultados experimentales, el FM-Index se vuelve más eficiente que Rabin-Karp cuando se realizan aproximadamente 3-4 búsquedas o más sobre el mismo texto, ya que el costo de construcción se amortiza rápidamente con el tiempo ahorrado en las búsquedas.

Para un texto de 200 MB:

- Construcción FM-Index: ~142 segundos
- Diferencia por búsqueda: ~2 segundos
- Punto de equilibrio: ~71 búsquedas

5.3 Limitaciones del estudio

Es importante mencionar que este estudio tuvo limitaciones debido a la memoria disponible en la máquina virtual (7 GB). Si bien las instrucciones del laboratorio solicitaban experimentar con textos de más de 1 GB, se optó por limitar el tamaño máximo a 500 MB para evitar:

- Agotamiento de memoria durante la construcción del FM-Index
- Terminación forzada del proceso por parte del sistema operativo
- Tiempos de experimentación imprácticamente largos

En un entorno con más memoria disponible, se esperaría que las tendencias observadas se mantengan para textos más grandes, aunque los tiempos absolutos de construcción del FM-Index aumentarían proporcionalmente.

6 Referencias

- SDSL-lite library: <https://github.com/simongog/sdsl-lite>
- Pizza&Chili Corpus: <http://pizzachili.dcc.uchile.cl/texts.html>
- Material del curso de Estructuras de Datos y Algoritmos Avanzados