UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

Estructura comprimida simplificada para indexar texto basada en gramáticas

PROPUESTA DE TEMA DE MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL EN COMPUTACIÓN

Cristóbal Fuentes Alvarado

PROFESOR GUÍA:
Gonzalo Navarro

SANTIAGO DE CHILE
2023

1. Introducción

El campo de las estructuras de datos compactos es de suma importancia hoy en día, considerando que el orden en que la acumulación de información crece cada día es varios órdenes de magnitudes mayores a la capacidad de procesamiento de los computadores modernos. Para facilitar el análisis de contenido sobre estos textos existen estructuras y algoritmos que de forma rápida y con poco espacio extra logran reportar las ocurrencias de patrones de texto en los datos de entrada. La elección de cuál algoritmo y/o estructura utilizar depende del desempeño de estos según las los parámetros de los patrones y textos de búsqueda. En muchos casos, distintas estructuras presenta desempeños similares en el análisis teórico, sin embargo, implementaciones muestran empíricamente que algunas se comportan mejor en función de ciertas características los datos. Por esto, es necesario aseverar según las características de los datos que se desea procesar qué estructuras son mejores para cada una de las operaciones que se requieran, y para eso es esencial desarrollar implementaciones para las estructuras hasta ahora solo teorizadas.

La estructura comprimida simplificada para indexar texto basada en gramáticas presenta una solución al problema de reportar las ocurrencias de un patrón de texto cualquiera en un texto dado. No es la única estructura de su tipo que resuelve el problema, y presenta sus propias ventajas y desventajas según los parámetros de los textos de búsqueda y patrón. Sin embargo, la simplicidad de esta estructura en términos de los componentes de esta (Secuencias compresibles, secuencias con permutaciones[1, Capítulo 6.1] y grillas representadas con *Wavelet Trees*[1, Capítulo 10.1]) motiva a un potencial buen desempeño. El valor teórico del desempeño en términos de tiempo y espacio de esta no es suficiente para determinar la factibilidad de la estructura; se requiere una implementación que permita determinar de forma empírica y, en términos comparativos, cuantitativa si es mejor usar esta estructura que otra de orden similar.

El problema abordado en este trabajo es lograr una buena, esto es, optimizada y congruente al espacio y tiempo teórico de la estructura, implementación de lo descrito en el libro *Compact Data Structures* (*Indexed Searching in Grammar-Compressed Text*)[1, Capítulo 10.5.6] y hacer un análisis empírico en función de los parámetros de entrada, obteniéndose conclusiones sobre el desempeño de la estructura en comparación a otras implementaciones de otros algoritmos sobre un mismo problema.

2. Situación Actual

La búsqueda de ocurrencias de un patrón de caracteres sobre un texto tiene muchas soluciones cada una con distintas ventajas y desventajas. Existen varios algoritmos y estructuras que resuelven el problema con distintos valores de órdenes de tiempo de operación y uso de espacio de la estructura (compresión).

De todas las soluciones existentes al problema de búsqueda de patrones de texto, una parte significativa de ellas tiene implementaciones en lenguajes de programación, por lo que los valores teóricos de los órdenes que describen el desempeño de estas estructuras tienen correspondientes valores empíricos, y por consiguiente, las evaluaciones comparativas de estos algoritmos tiene carácter explícitamente cuantitativo (siempre y cuando el lenguaje de las

implementaciones sea el mismo o de equivalente optimización), es decir, dos estructuras de orden teórico similar (y hasta equivalentes) pueden compararse prácticamente y así se puede determinar cuál usar dado ciertos parámetros del texto sobre el cual se quiere trabajar.

Nace entonces la necesidad de obtener el mismo carácter empírico para otras estructuras que no tienen implementaciones actualmente, entre ellas, la estructura que es objeto de análisis para este trabajo.

El fin de implementar la estructura comprimida simplificada para indexar texto basada en gramáticas es poder hacer análisis de esta y obtener un valor empírico a los órdenes de tiempo en función de los parámetros relevantes de un texto, además de hacer análisis comparativo con estructuras de carácter similar, y hacer conclusiones sobre que estructura se comporta mejor según los parámetros de texto de entrada.

En el caso del trabajo actual, reportar las ocurrencias de un *string* sobre un texto es un problema que concierne a, por ejemplo, motores de búsqueda, *indexers*, etc. Por esto es necesario cuantificar estas comparaciones entre estructuras, que solo se puede lograr implementando y analizando el comportamiento de estas en la práctica.

3. Objetivos

Objetivo General

El objetivo principal de este trabajo es evaluar de forma empírica la estructura comprimida simplificada para indexar texto basada en gramáticas simple descrito en el libro *Compact Data Structures* (*Indexed Searching in Grammar-Compressed Text*)[1, Capítulo 10.5.6] y comparar su desempeño con los algoritmos y estructuras actuales (y sus implementaciones) para la búsqueda de patrones en texto.

Objetivos Específicos

1. Implementar la estructura de forma correcta. Para evaluar esto se deben implementar *tests* que midan la robustez de la implementación con el fin de evitar *bugs*.
2. Obtener un *dataset* para evaluar el desempeño de la implementación. Los textos que componen este *set* deben ser analizados con el fin de obtener valores relevantes al análisis del desempeño del programa. Estos incluyen no solo el tamaño del texto sino también la entropía del texto (de distintos órdenes).
3. Implementar pruebas para evaluar el desempeño de la implementación según los distintos parámetros del set de prueba.
4. Obtener implementaciones de estructuras con fines equivalentes a la implementada con el fin de hacer análisis comparativos.
5. Analizar los resultados de las pruebas para obtener conclusiones respecto al desempeño empírico de la estructura, en contraste con otras estructuras existentes.

Evaluación

Considerando que el trabajo a realizar consiste en implementación de una estructura y luego el análisis de esta con el fin de obtener conclusiones respecto a su desempeño temporal y espacial, hay dos evaluaciones a hacer: evaluar la implementación en términos de código, y evaluar el análisis.

Para evaluar el funcionamiento correcto del código de la implementación de la estructura se utilizarán *unit tests*. Se deben evaluar los resultados obtenidos por el código en función de distintas entradas de prueba, incluyendo casos límites (como entradas sin datos o con entropía de orden cero nula). Además, se debe evaluar que la solución sea congruente con el análisis teórico del libro, esto es, que el tiempo de ejecución sea del orden teórico predicho. Una vez todos estos *tests* pasen con éxito se considerará exitosa la implementación.

El análisis de los resultados de los *tests* de desempeño temporal y espacial de la implementación será correcto siempre que las conclusiones obtenidas dependan totalmente de los resultados empíricos obtenidos en contraste a las predicciones teóricas del libro y los resultados, independiente si estos muestran un mejor o peor desempeño en comparación al estado del arte.

4. Solución Propuesta

Para implementar la estructura, se utilizará C++ como lenguaje, ya que presenta buena compensación entre robustez y eficiencia, además de ser el mismo lenguaje en el que muchas otras estructuras están implementadas, lo cual hará más justa la comparación de resultados.

En C++ se programarán los *unit test* para el código, utilizándose librerías como *Google Test* y *Microsoft Unit Testing* (decidir cuál librería usar dependiendo de las ventajas que cada una ofrezca será parte del trabajo).

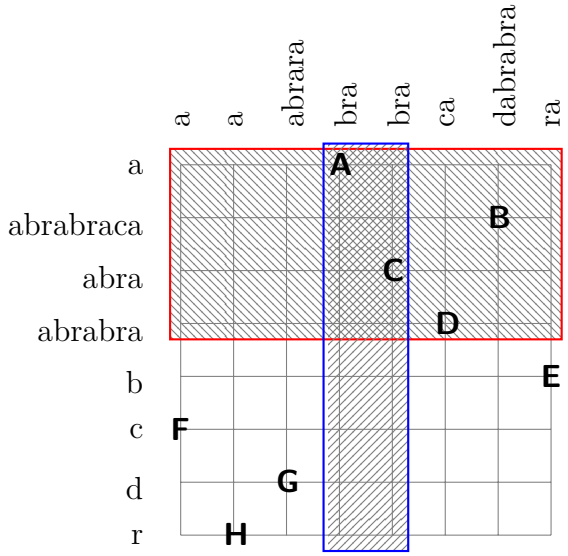
El algoritmo a implementar consiste en una representación como una grilla del texto comprimido a una serie de reglas de gramática tipo A $\rightarrow$ BC, donde A es un símbolo no-terminal y B y C son terminales o no-terminales. El texto es comprimido utilizando un algoritmo *RePair* recursivamente hasta obtener un solo símbolo no-terminal del cual se puede expandir el texto completo. El algoritmo *RePair* consiste en recorrer el texto reemplazando los dos caracteres más comunes por un símbolo no terminal, generando una regla de gramática, y repetir este proceso, hasta obtener un texto comprimido y una serie de reglas. Las reglas son entonces guardadas como una secuencia de pares de símbolos (un par para cada regla) y esta secuencia es a su vez representada usando permutaciones[1, Capítulo 6.1], que soporta las operaciones *Access*(i) (obtener el i-ésimo elemento de la secuencia) y $Select_c$(j) (Obtener el índice del j-ésimo c de la secuencia).

El conjunto de reglas será representado en una grilla de r $\times$ r donde cada regla $A_i \rightarrow B_iC_i$ es guardada en la posición (C_i,B_i). La grilla es representada utilizando *Wavelet Trees*[1, Capítulo 10.1]. Las filas son ordenadas de forma lexicográfica según los *strings* revertidos a los cuales expande B_i, mientras que las columnas se ordenan de forma lexicográfica según los *strings* a los que expande C_i.

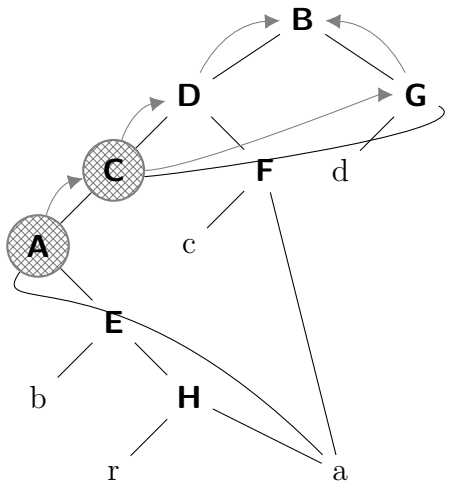
Finalmente, se guarda una secuencia L con los largos de los *strings* a los que expanden cada uno de los no-terminales de R (el conjunto de reglas).

Como ejemplo, para el texto T = abrabracadabrabra se obtienen las siguientes reglas de gramática:
A → a**E** (a|bra)
B → **DG** (abrabraca|dabrabra)
C → **AE** (abra|bra)
D → **CF** (abrabra|ca)
E → b**H** (b|ra)
F → ca (c|a)
G → d**C** (d|abrabra)
H → ra (r|a)

Las cuales son representadas en la siguiente grilla:



La grámatica puede ser visualizada por un Grafo acíclico dirigido (DAG):



La idea de la estructura es que para cualquier patrón P[1,m] que se desee encontrar, las ocurrencias de este corresponderán a una de dos: ocurrencias primarias, donde el patrón aparece dividido en B_i y C_i, y ocurrencias secundarias, correspondientes a ancestros (en el *DAG*) de nodos donde ocurrencias primarias o secundarias de P aparecen.

La búsqueda de ocurrencias primarias cosiste en encontrar, para cada posible par de prefijo y sufijo que concatenados conforman P, el rango de columnas que parten con el sufijo, y el rango de filas que terminan con el prefijo (aquí la razón de porque las filas están ordenadas por el *string* invertido) con lo cual, cada punto de este rango de filas y columnas corresponde a no-terminales A_i que al expandirse a *strings* contienen P. El lugar donde P aparece en cada A_i lo llamamos *offset* o_i.

Para cada par A_i y o_i encontrados en el paso anterior, las ocurrencias secundarias requieren encontrar todas las no-terminales que expanden a A_i directa o transitivamente hasta S, donde el offset será la posición de P en T. Para esto hay que hacer k = $Select_{Ai}$(j) sobre R por cada posible j, y por cada uno de estos k, D = $\lceil$ k / 2 $\rceil$ expande a XA_i (para algún X) si k es par, o a A_iX si k es impar. En el primer caso, el offset se mantiene, mientras que en el segundo se debe añadir el largo del *string* al que expande X (lo cual tenemos guardado en la secuencia L). El algoritmo se repite con D y así sucesivamente hasta llegar a S, donde el offset final es la posición de P.

El costo de reportar las *occ* ocurrencias de P[1,m] en T es $\mathcal{O}((m + \log n)m \log r \log\log r + occ \log n \log\log r)$. La implementación de la estructura debe comportarse en función a este costo y, por lo tanto, se debe testear que el tiempo de ejecución tenga este orden.

Para hacer análisis de los resultados se deben obtener archivos de texto de prueba. Existen muchos *datasets* con textos de distinto tipo (ya sean textos reales como libros, publicaciones, etc.) y texto basura (datos sinsentido) con distintos parámetros. Estos textos deben ser analizados para obtener valores primarios como el largo del texto, el alfabeto del texto, y valores secundarios, como la entropía de distintos órdenes del texto[1, Capítulo 2].

Se deben finalmente obtener las implementaciones de otros algoritmos de búsqueda de textos para poder hacer un análisis comparativo con la estructura a implementar. La idea es que muchos algoritmos tienen un costo equivalente al de la estructura actual, sin embargo, en la práctica puede que sean más o menos rápidos. Todo esto es parte del análisis.

5. Plan de Trabajo (Preliminar)

1. Analizar el algoritmo con el fin de detectar las partes independientes de la implementación.
2. Decidir el diseño de software de la implementación (Patrones de diseño) según el análisis anterior.
3. Organizar el flujo de trabajo con el fin de implementar las partes descritas en el paso anterior. Para esto se utilizarán métodos de trabajo ágil.
4. Implementar *tests* de prueba para el código.
5. Implementar cada una de las partes del programa según lo planificado en los pasos

anteriores, corrigiendo según los resultados de los *tests*.

6. Obtener archivos de prueba para evaluar el tiempo de ejecución de la implementarión.
7. Analizar archivos de prueba y obtener valores relevantes al análisis del desempeño de la implementación
8. Implementar *tests* de tiempo de ejecución (*Runtime tests*) utilizando los archivos de pruebas obtenidos en el paso anterior.
9. Evaluar programa usando los *tests* anteriores, obteniendo tiempos de ejecución en función de los valores y parámetros relevantes de los archivos de prueba.
10. Obtener Implementaciones de algoritmos para comparar.
11. Analizar Resultados de los *tests*, comparándolos con el estado del arte.
12. Escribir Memoria basándonos en el análisis efectuado.

Referencias

[1] Navarro, Gonzalo: *Compact Data Structures - A Practical Approach*. Cambridge University Press, 2016, ISBN 978-1-10-715238-0.