Simulador para el análisis de la complejidad computacional de Códigos Libres de Prefijo Optimos

por *David Muñoz*

Una propuesta de trabajo de título
presentada a la Universidad de *Chile*
para el cumplimiento
y obtención del grado académico
Ingeniería Civil
en
Computación

Santiago de Chile, 2015

Índice general

Capítulo 1

Motivación

Un problema bastante conocido en el área de la computación, es el del cálculo de los Códigos Libres de Prefijo, mejor conocidos en ingles como 'Optimal Prefix Free Codes' (en adelante OPFC). Este problema basicamente consiste en encontrar para cada elemento perteneciente a un conjunto $[1...n]$ de símbolos, una codificación de tal forma que ningún código sea prefijo de otro.

Esta codificación además tiene que ser óptima. Esto en el sentido que se busca optimizar el espacio usado por cada símbolo según algún parámetro conocido. En particular se suele considerar la frecuencia de aparición de cada símbolo dentro de un texto, pero a modo general se asume que se tiene como parámetro una lista con los pesos de cada elemento.

Este problema ha sido ampliamente estudiado a lo largo de la historia y existen varios resultados interesantes. El resultado más conocido (y enseñado) es el algoritmo de Huffman [2]. Dicho algoritmo es tan conocido que los OPFC son ampliamente conocidos como 'Códigos de Huffman'.

Usando el algoritmo de Huffman para determinar los OPFC de un set de n símbolos dada su lista de frecuencias y usando una cola de prioridad, se consigue un tiempo de $\theta(n \log n)$ en el modelo de comparaciones. Es decir, en el peor caso eso es lo que va a tomar calcular un OPFC con una entrada de tamaño n.

Sin embargo, Van Leeuwen [3] definió un algoritmo en el que el cálculo de la codificación de Huffman podía llevarse a cabo en tiempo $O(n)$ si la lista de frecuencias era entregada inicialmente de forma ordenada.

Existen varios resultados (que veremos más adelanto). Pero solo con esta información se puede plantear la siguiente interrogante, que es la que se pretende contestar durante la realización del presente trabajo de titulación:

¿Es posible computar los OPFC en tiempo mejor que $\theta(n \log n)$ al sacar ventaja de ciertos 'casos fáciles' específicos?

Capítulo 2

Alternativas analizadas y opción de solución elegida

Siendo el problema a resolver algo bastante teorico, se hace necesario tratar de acotar un poco el problema para poder abordarlo de forma más concreta.

Primero, es necesario una definición un poco más formal del problema de los OPFC. Este puede ser enunciado de la siguiente manera (siendo aún bastante informal):

"Sea T un texto conformado por símbolos pertenecientes a un lenguaje L donde $|L| = n$. Se sabe que cada símbolo p perteneciente a L ocupa una cantidad $k = log(n)$ bits. Esto implica que el texto tiene tamaño $|T| = k \sum f(p)$ bits. Donde $f(n)$ representa la cantidad de apariciones del símbolo p en el texto T

Dada la lista de frecuencias de aparición de cada símbolo L en T, se pide encontrar una nueva codificación para cada palabra de este lenguaje de forma que se genere un nuevo texto T' a partir de T tal que $|T'| \leq k \sum f(p)$ bits. Se requiere además, que esta codificación sea óptima."

Como ya se enuncio, el algoritmo definido por Huffman ya resuelve este problema de forma exacta y en tiempo $\theta(n \log n)$ para el peor caso con entradas de tamaño n. Pero la pregunta propuesta, y el trabajo en general, apuntan a que se puede lograr algo mejor si consideramos que existen varios casos que son fáciles de detectar y resolver.

Algunos ejemplos de que casos que se considera fáciles de resolver son:

1. Si la lista de frecuencias viene ordenada
2. Si la frecuencia de todos los símbolos es la misma
3. Si todos los elementos de la lista estan en el rango $[x..,2x-1]$

Tambien se podría sacar ventajas de otras soluciones adaptivas que ya existen y que podrían tener una influencia sobre la resolución de este problema. Por ejemplo, existen algo-

ritmos de ordenamiento adaptivos, que pueden ordenar en tiempo menor a la cota $\theta(n \log n)$.

Finalmente se tienen intuiciones que se pueden desarrollar para ayudarnos a lograr el objetivo planteado. En particular, y la que se pretende usar fuertemente en el desarrollo de este proyecto es que no es necesario ordenar el arreglo para realizar el calculo de los OPFC. Un buen ejemplo de esto es el caso ya nombrado: si todos los elementos de la lista de frecuencias estan en el rango $[x..,2x-1]$.

A todo esto hay que agregar que ya existen varios resultados adaptivos. La mayoría de estos resultados asume que la lista de frecuencias viene ordenada. Sin embargo proveen resultados (y algoritmos) bastante interesantes que podrían dar una pista de lo que se quiere lograr. En particular hay 3 resultados que resultan de especial interés:

1. Moffat y Turpin [6] definieron un algoritmo que toma tiempo $O(r(1 + log(N/r)))$, con r el largo del código más grande genenerado.

2. Belal y Elmasry [1] definieron un algoritmo que calculo los OPFC en tiempo $O(16^k n)$.

3. Pessoa et al. [4] definio un algoritmo que toma tiempo $O(n)$, pero que optimiza el uso de espacio.

En el sección de 'Trabajo avanzado' se entrara en más detalle sobre los resultados de Pessoa et al.y Moffat-Turpin.

Una vez identificado todo esto solo queda decidir como y con que enfoque se va abordar el problema. Para tener un buen punto de partida, se decidió abordar por la intuición de que no es neceario ordenar (completamente) el arreglo de frecuencias para realizar el cálculo de los OPFC. La idea general es que aunque necesito acceder a partes del arreglo, no se necesita acceder a todas las posiciones.

Para poder acceder a posiciones 'ordenadas' de una lista desordenada, es necesario crear una estructura de datos que me permita hacer las consultas pertinentes. Para esto se decidió hacer uso de lo que se conoce como 'Deferred Data Structures' (en adelante DDS). La idea principal de las DDS es que al hacer una consulta por algúna posición de la lista, puedo calcular el elemento en dicha posición, marcar la posición como ya calculada (para futuras consultas a la misma posición) y dejar el resto de la lista desordenada. De esta forma cada vez que haga una consulta por una posición nueva voy a ir 'ordenando' la lista de a poco.

Esto nos da una pregunta a contestar un poco más específica y pertinente al enfoque que se le dará al problema: ¿Existe algún algoritmo para calcular los OPFC usando DDS de forma que al finalizar de hacer el cálculo, no todas las posiciones de la DDS hallan sido calculadas?

Capítulo 3

Objetivos

Generales:

1. Crear una plataforma que permita éjecutar un análisis y testeo rápido (y empírico) sobre algoritmos de OPFC.
2. Identificar una medida de dificultad y definir un algoritmo adaptivo para el cálculo de los Optimal Prefix Free Codes.

Específicos:

1. Hacer un estudio teórico más profundo acerca de los OPFC
   a) Estudiar algoritmo de Moffat y Turpin
   b) Estudiar algoritmo de Pessoa et al.
2. Definir una estructura de datos 'auxiliar' (DDS) para el manejo de accesos a una lista desordenada
3. Definir un algoritmo adaptivo para el cálculo de OPFC: En particular definir variaciones de los algoritmos estudiados y usar DDS para evitar ordenar las listas de frecuencias.
4. Desarrollar un plataforma que permita comparar la eficiencia de algoritmos utilizado para las calcular OPFC: A grandes rasgos:
   a) Implementar los algoritmos de cáculo de OPFC más conocidos.
   b) Establecer formas de comparar algoritmos de OPFC.
5. Implementar el algoritmo (o algoritmos) definidos.
6. Testear los algoritmos en la plataforma desarrollada (usando los métodos de comparación definidos) de forma de obtener datos empíricos y hacer que el experimento sea repetible.

7. Hacer un análisis teórico sobre los algoritmos definidos (siempre y cuando los resultados de la comparación sean significativos).

Capítulo 4

Metodología

La metodología de trabajo a seguir esta profundamente ligada con los objetivos propuestos. En particular la forma en la que fueron expuestos los objetivos, es la misma secuencia de trabajo que se va a realizar/se esta realizando (sin considerar partes del trabajo que se hacen en forma paralela). Por lo que en particular cabe destacar algunos detalles con respecto a esto.

Primero, notar que se va a empezar con un enfoque muy practico. Primero se definirá el algoritmo, esto al mismo tiempo que se implementa una plataforma de comparación de algoritmos (es plataforma será exclusiva para comparar algoritmos de OPFC). Luego se implementará el algoritmo y será puesto a prueba en la plataforma creada. Finalmente, y si es que la plataforma lanza resultados interesantes, se llevará a cabo un análisis teórico.

La idea es comprobar empíricamente la utilidad de un algoritmo antes de hacer el análisis teórico. Esto puede llevar a que sucedan varias iteraciones en la 'definición' del algoritmo, puesto que si los resultados empíricos de nuestro algoritmo no son mejores que los normales (digase Huffman o Van Leeuwen), no valdría la pena hacer un análisis teórico.

Tambien es interesante notar que la creación de una DDS es tambien parte de los objetivos. A pesar que el estudio no se centra en DDS, una parte importante es poder implementar una DDS de la forma más eficiente posible, y por lo tanto esto también puede llevar a varias iteraciones.

Es importante decir que los algoritmos a definir serán variaciones de algoritmos adaptivos que ya existen. La idea entonces no es empezar de cero, si no que refinar el trabajo que ya existe y en particular abordar un aspecto del problema que ha sido poco estudiado: considerar el caso de recibir una lista desordenada.

Finalmente notar que aunque una gran parte del trabajo es teórico, la plataforma definida no es exclusiva de las definiciones de algoritmos que se estan proponiendo y podrá ser usada como una herramienta para cualquier persona que quiera hacer estudio sobre OPFC. Es decir esta será una herramienta de apoyo a la investigación.

Capítulo 5

Trabajo adelantado

El trabajo avanzado durante el semestre se dividió en 2 partes que se desarrollaron de forma paralela. La primera parte fue el estudio de los resultados ya existentes, en particular las trabajos de Moffat-Turpin, y Pessoa.

La segunda parte fue la implementación de test los cuales serán aplicados al algoritmo que se pretende definir. Esto nos permitira comprobar la eficiencia del algoritmo con respecto a los algoritmos más tradicionales ya existentes (en particular el algoritmo de Van Leeuwen). Aparte de los test que se aplicaran sobre el algoritmo, tambien se definieron test a ser aplicados sobre la DDS, que es la estructura de datos que será usada por el algoritmo a definir.

El estudio teórico consistio en, ademas de leer los papers pertinentes, en escribir (a modo de ejercicio) resumenes para lograr una mejor comprensión de los temas a abarcar. Dichos resumenes se incluyen en la sección de apendices del presente informe.

Ademas los códigos de los test creados estan disponible en github en el siguiente repositorio: [TODO insertar url repositorio github]

Apéndice A

Resumen sobre el trabajo de Pessoa

Pessoa describe en su artículo 'Three Space-Economical Algorithms for Calculationg Minimum-Redundancy Prefix Free codes' tres algoritmos para el cálculo de los OPFC. Estos algoritmos no fueron definidos para optimizar el tiempo que toma calcular los OPFC, si no que la idea es optimizar el uso de espacio.

En particular, los algoritmos que definen, requieren durante su cálculo espacio incluso menor que $O(n)$, siendo n el tamaño de la entrada. Para lograr esto se hace uso de un algoritmo que los autores denominaron 'Lazy Traversal Algorithm'.

Dicho algoritmo define una forma de recorrer el árbol de Huffman (el obtenido al aplicar el algoritmo de Huffman) sin tener que almacenar la estructura del árbol por completo.

Entonces básicamente los algoritmos definidos en el árticulo definen variaciones del 'Lazy Traversal Algorithm' que permiten calcular los OPFC optimizando el uso de espacio.

Los alogoritmos y resultados definidos en el artículo son:

1. F-LazyHuff Algorithm: tiempo $O(n)$ y espacio $O(min\{H^2, n\})$
2. E-LazyHuff Algorithm: tiempo $O(n + nlog(n/H))$ y espacio $O(H)$
3. B-LazyHuff Algorithm: tiempo $O(n)$ y espacio $O(H)$

En estos algoritmos, n representa el tamaño de la entrada (la lista de frecuencias de cada símbolo) y H representa el largo del código más largo generado (o equivalentemente, la altura del árbol de Huffman).

Estos resultados son bastante interesantes (independiente de que el artículo es difícil de entender), pero no apuntan a lo que se quiere lograr en el presente proyecto. Sin embargo la definición del algoritmo base ('Lazy Traversal Algorithm') presenta una idea interesante que puede servir bastante bien para los objetivos que se quieren lograr.

En particular, el 'Lazy Traversal Algorithm' hace predicciones de que elementos de la lista

de frecuencias van a ser hermanos dentro del árbol de huffman. Esto lo hace sin la necesidad de acceder al elemento de por sí. De hecho, el algoritmo no accede a ningúna posición del arreglo a menos que sea necesario (por lo mismo se define como un algoritmo Lazy).

Con esto podemos pensar que usando una DDS y una variación de este algoritmo, se pueden calcular los OPFC sin acceder a todas las posiciones de la lista.

Apéndice B

Resumen sobre el trabajo de Moffat y Turpin

Bibliografía

[1] A. A. Belal and A. Elmasry. Distribution-sensitive construction of minimum-redundancy prefix codes. *CoRR*, abs/cs/0509015, 2005. Version of Tue, 21 Dec 2010 14:22:41 GMT, with downgraded results from the ones in the conference version.

[2] D. A. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the Institute of Radio Engineers*, 40(9):1098–1101, September 1952.

[3] J. V. Leeuwen. On the construction of Huffman trees. In *Proceedings of the International Conference on Automata, Languages, and Programming (ICALP)*, pages 382–410, Edinburgh University, 1976.

[4] R. L. Milidiú, A. A. Pessoa, and E. S. Laber. Three space-economical algorithms for calculating minimum-redundancy prefix codes. *IEEE Trans. Inf. Theor.*, 47(6):2185–2198, Sept. 2001.

[5] A. Moffat and J. Katajainen. In-place calculation of minimum-redundancy codes. In *WADS*, volume 955 of *Lecture Notes in Computer Science*, pages 393–402. Springer, 1995.

[6] A. Moffat and A. Turpin. Efficient construction of minimum-redundancy codes for large alphabets. *IEEE Transactions on Information Theory*, pages 1650–1657, 1998.