Codificación Huffman Adaptativa

Jose Antonio Lorencio Abril

09/2021

Contents

1 Introducción

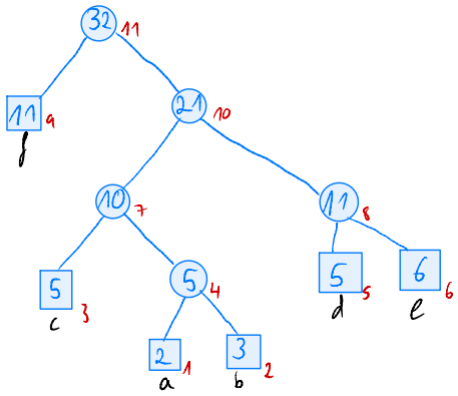
Me voy a basar en [VitterVitter1987] para la explicación. Fue en este artículo en el que J. S. Vitter describió el algoritmo que él planteaba para mejorar la implementación de la codificación Huffman. Vitter describe la forma de crear códigos Huffman estudiada en clase, a la que llama **método de dos fases de Huffman**, puesto que se necesita recorrer dos veces la cadena de texto para poder codificarla: el primer recorrido sirve para efectuar un conteo de las ocurrencias de cada caracter, y el segundo para codificar el mensaje una vez construido el árbol de Huffman. Esto es uno de los grandes inconvenientes de este algoritmo, además del overhead que requiere para transmitir el árbol.
Explica, además, el **algoritmo FGK**, pensado para sustituir al algoritmo de Huffman original e ideado para que solo necesite recorrer una vez la cadena de texto. La idea es que el árbol se puede ir construyendo sobre la marcha, haciendo que, para cada caracter procesado, el árbol originado desde el inicio hasta el momento actual, sea un árbol de Huffman para la subcadena recorrida, pero no necesariamente lo será para el resto de la cadena. Por tanto, el árbol deberá ser modificado en cada paso.
Para este propósito, se define la siguiente propiedad, que caracteriza los árboles Huffman:

Propiedad de los hermanos: un árbol binario con p hojas de peso no negativo es un árbol de Huffman si, y solo si,

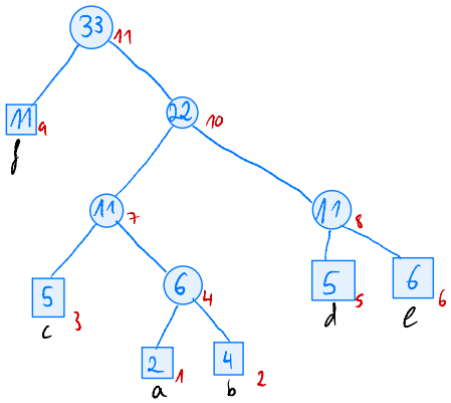
1. las p hojas tienen pesos no negativos $w_1, ..., w_p$ y el peso de cada nodo interno es la suma de los pesos de sus hijos; y
2. los nodos pueden ser numerados en orden no decreciente por peso, de forma que los nodos $2j-1$ y $2j$ son hermanos, $1 \leq j \leq p-1$, y su nodo padre común es superior en la enumeración

Dado que he elegido explicar el algoritmo de Vitter, respecto a FGK solo voy a mostrar un ejemplo, para que la idea quede clara.

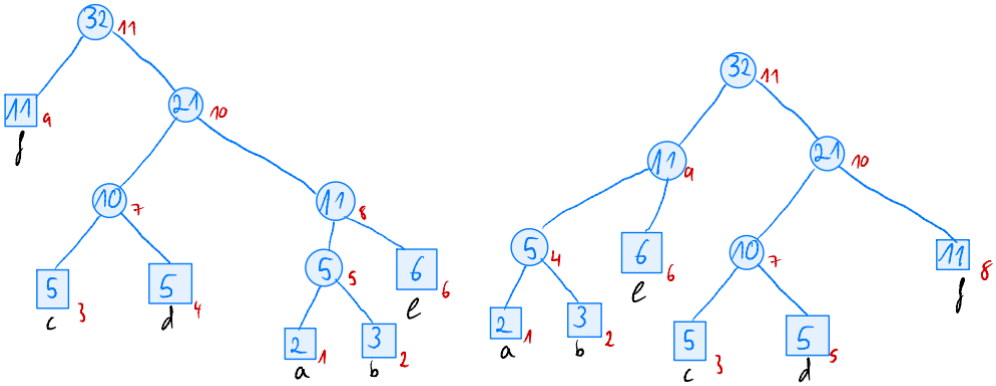
Supongamos que disponemos del siguiente árbol, que es de Huffman:



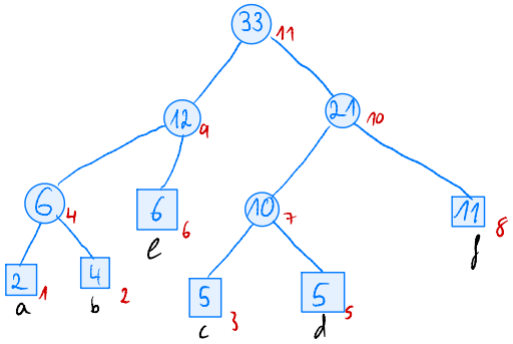
Este árbol representa que hemos recorrido una cadena de 32 caracteres, y ha habido 2 ocurrencias de a, 3 de b, 5 de c, 5 de d, 6 de e y 11 de f. Es fácil comprobar que se verifica la propiedad de los hermanos. Supongamos, ahora, que nos encontramos con una ocurrencia nueva de una b, entonces querremos actualizar los pesos de los nodos. Pero, si lo hacemos directamente, obtenemos el siguiente árbol:



Que no es de Huffman, pues no cumple la propiedad de los hermanos. La solución aportada en el algoritmo FGK es que, si queremos incrementar en 1 el peso de una hoja, debemos recorrer la rama que llega hasta dicha hoja, y, si encontramos algún nodo cuyo peso no sea único en el árbol, debemos sustituir ese nodo (con todo lo que de él cuelga) por el aquel nodo con el mismo peso y con el mayor valor en la enumeración de los nodos, hasta llegar a la raíz. En nuestro ejemplo, los pasos serían los siguientes:



Por último, aumentamos el peso de la hoja b y actualizamos los pesos necesarios:



Este último árbol verifica la propiedad de los gemelos y, por tanto, es de Huffman. Notemos que la codificación de b ha pasado de ser 1011 a 001.
Falta notar que se usa un nodo-0 para representar los caracteres que aún no han aparecido. Este nodo se bifurca en dos cuando un nuevo caracter se encuentra, y se procede como en el ejemplo.
Se denota $\uparrow$ cuando se intercambia un nodo por otro de un nivel superior, y $\rightarrow$ cuando se intercambia por un nodo del mismo nivel. Es importante notar que dos nodos del mismo peso no pueden distanciarse más de un nivel en el árbol, a no ser que uno de ellos sea el hermano del nodo-0. En tal caso, un intercambio que haga subir un nodo 2 niveles es posible, y se denota $\uparrow\uparrow$. Por último, los intercambios que mueven un nodo hacia abajo se denominan $\downarrow$.
Tras todo esto, Vitter propone un nuevo algoritmo, que el denomina Λ, pero que ha terminado en denominarse **algoritmo de Vitter**. Este algoritmo es óptimo dentro de los algoritmos de Huffman de una fase, y lo describo en detalle en la siguiente sección.

2 Algoritmo Vitter

Vamos ahora a describir el algoritmo que propone Vitter, que consta de una única pasada y demuestra que es óptimo dentro de los algoritmos de Huffman de una única fase.
Para ello, se define una nueva forma de numeración de los nodos, que ayuda a la ejecución del algoritmo:

> **Numeración implícita:** la numeración de un nodo se corresponde con la representación visual del árbol. Es decir, los nodos son numerados en orden creciente por nivel; los nodos de un nivel están numerados más bajos que los nodos en el siguiente nivel más alto. Los nodos en el mismo nivel se numeran en orden creciente de izquierda a derecha.

Vitter demuestra que usando esta numeración, se eliminan los intercambios de tipo $\downarrow$, y que si el nodo que se mueve hacia arriba en un $\uparrow$, entonces el nodo que se mueve hacia abajo debe ser una hoja.
El punto clave para minimizar $D_t - S_t$ (donde S_t es el coste de comunicación para una codificación Huffman estática del mensaje procesado hasta ahora, usando el árbol Huffman construido hasta ese momento, y D_t es el coste total de comunicación para las t letras) es hacer los cambios $\uparrow$ imposibles, excepto en la primera iteración el bucle while al actualizar.
Esto se puede conseguir manteniendo la siguiente propiedad invariante:

> $(*)$ Para cada peso w, todas las hojas de peso w preceden, en la numeración implícita, a todos los nodos internos con ese mismo peso w

Se puede demostrar que si un árbol Huffman satisface $(*)$, entonces también minimiza $\sum_j l_j$ y $\max_j \{l_j\}$ (l_j representa la distancia en el árbol desde la raíz hasta la hoja a_j).
También demuestra Vitter que si mantenemos $(*)$, entonces los intercambios $\uparrow\uparrow$ son imposibles, y que los $\uparrow$ solo pueden ocurrir al mover hacia arriba una hoja.
Llegamos ahora al teorema importante, el que demuestra la optimalidad del algoritmo de Vitter

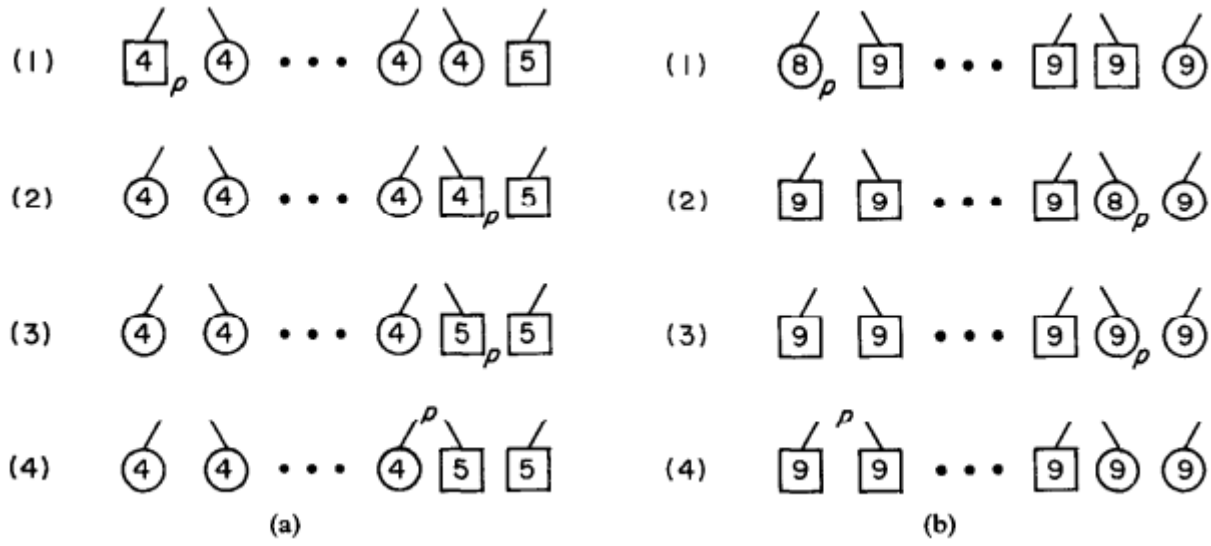
Theorem 1. *El algoritmo de Vitter minimiza la diferencia $D_t - S_t$ del peor caso, sobre todos los mensajes de longitud t, entre todos los algoritmos Huffman de una fase.*

2.1 El algoritmo

Se definen los **bloques** como clases de equivalencia de nodos definidos por $x \sim y$ sii los nodos x e y tienen el mismo peso y ambos son internos o ambos son hojas. El **líder de un bloque** es el nodo del bloque con numeración implícita mayor.
Para mantener $(*)$, se usa la operación $DeslizarEIncrementar$, que básicamente consiste en que:

- Si se va a incrementar un nodo hoja, antes se mueve a la derecha, hasta tener a la izquierda a todos los nodos internos que tienen su mismo peso.
- Si se va a incrementar un nodo interno, antes se mueve a la derecha, hasta tener a la derecha a todos los nodos hoja que tienen su mismo peso $+1$ (su peso actualizado).

La siguiente imagen es el ejemplo proporcionado en [VitterVitter1987] , que hace justo lo comentado ahora mismo:

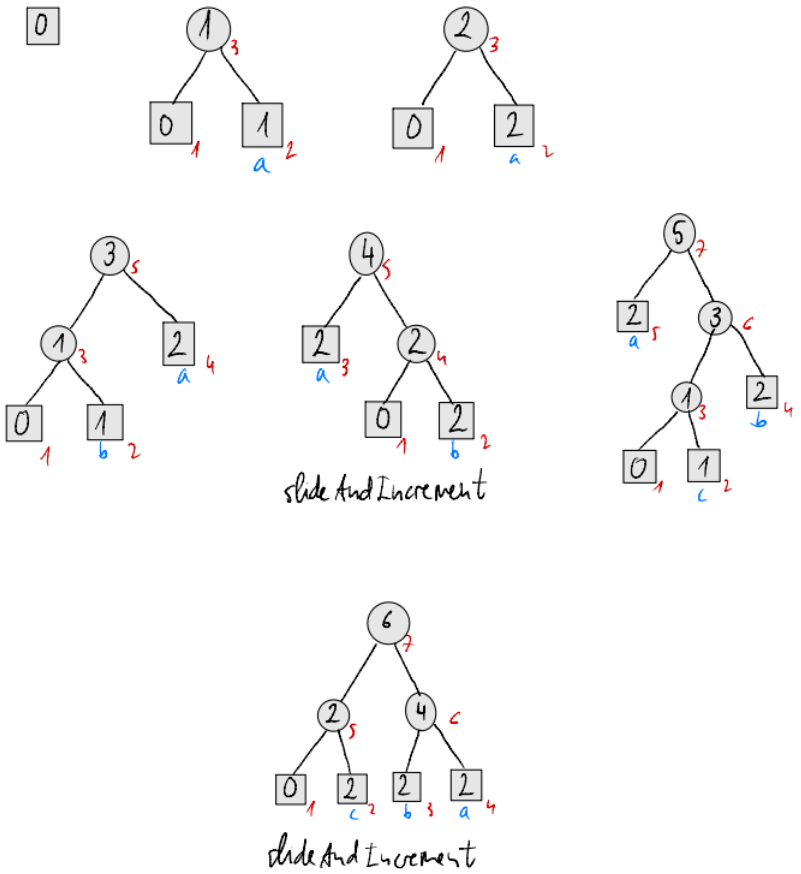


Además, se utiliza una estructura de datos que tiene las siguientes características:

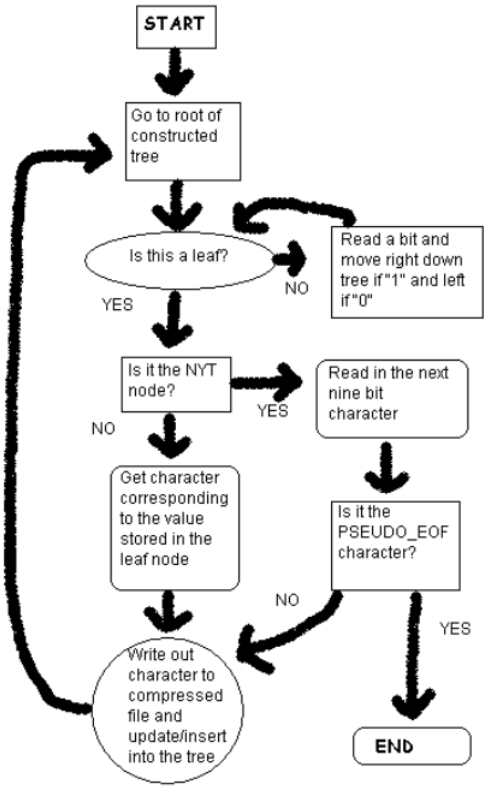
- Representa un árbol Huffman con pesos no negativos que mantiene $(*)$ invariante.
- Guarda una lista contigua de nodos internos en orden no decreciente por peso. Los nodos internos con mismo peso se ordenan con respecto a la numeración implícita. Otra lista similar se guarda para las hojas.
- Debe encontrar el líder de un bloque de nodos dado basándose en la numeración implícita.
- Debe intercambiar los contenidos de dos hojas del mismo peso.

- Debe incrementar el peso del líder de un bloque por 1, lo que pueda causar que la numeración implícita se "deslice" a lo largo de la numración de los nodos del siguiente bloque, haciendo que cada uno decrezca en una unidad.
- Debe representar la correspondencia entre las k letras del alfabeto que han aparecido en el mensaje y las hojas con peso positivo del árbol.
- Debe representar las $n - k$ letras que no han aparecido aún con un nodo-0.

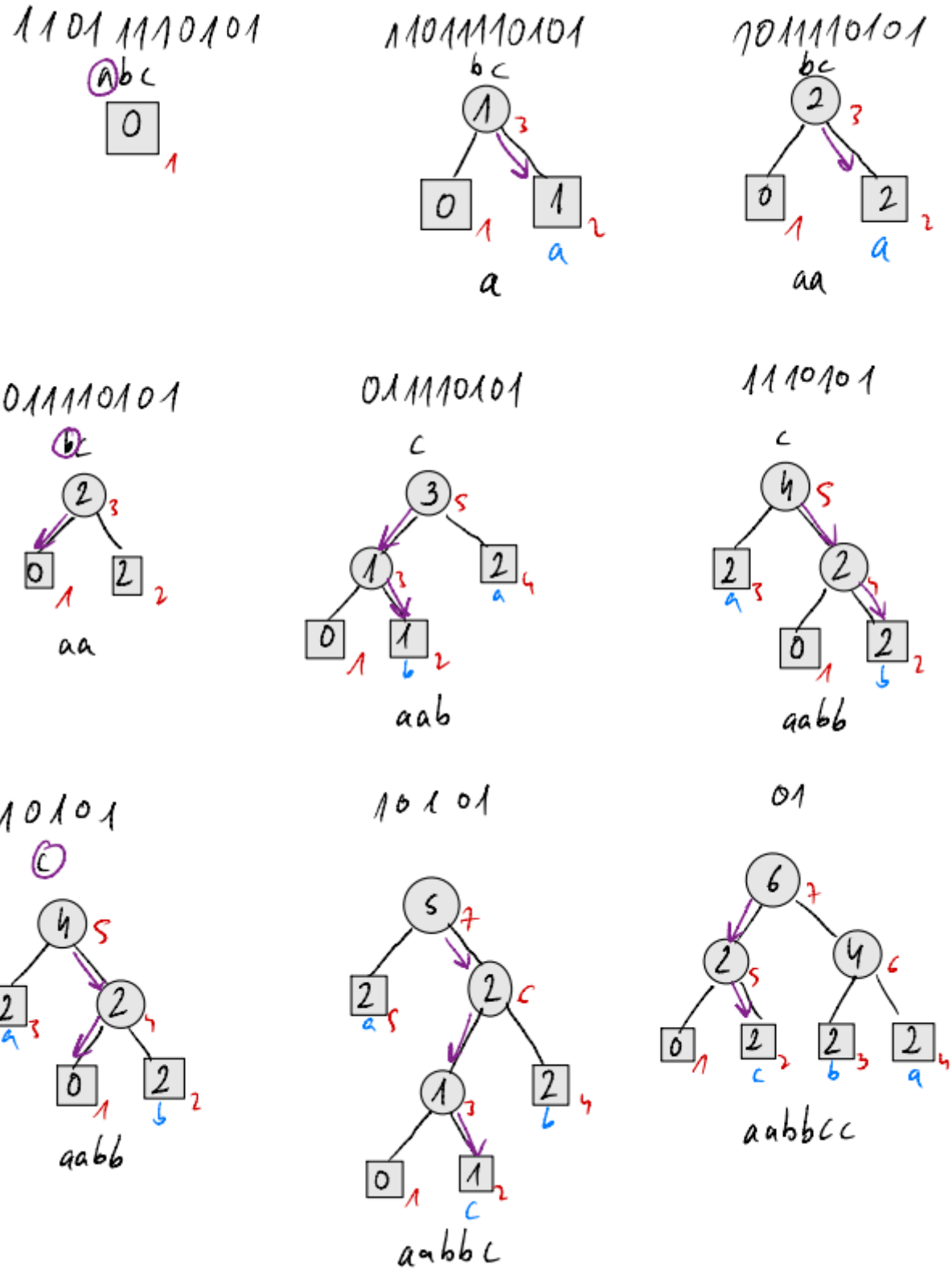
La estructura de datos usada se denomina **árbol flotante** porque los punteros al padre y al hijo de un nodo no se mantienen explícitamente. En su lugar, cada bloque tiene un puntero *parent* y un puntero *right_child* que apuntan al padre y al hijo derecho del líder del bloque. La localización de los padres e hijos de los otros nodos en el bloque se pueden calcular mediante un cálculo del offset desde el padre del bloque y el puntero *right_child*. Esto permite a un nodo deslizarse a través de un bloque entero sin tener que actualizar más que una cantidad constante de punteros.
Así, el algoritmo consiste en hacer como en FGK, pero con las consideraciones mencionadas. Veamos un ejemplo. Supongamos que queremos construir el árbol Huffman del mensaje aabbccdd. El proceso sería el siguiente:



Así, el mensaje aabbcc, se codificaría como 1 1 01 11 101 01 → 11011110101, pues cada caracter se codifica al vuelo, cuando le toca. Entonces, enviamos el mensaje codificado, y una lista con los caracteres ordenada por orden de aparición en el mensaje (sin repetir, o estaríamos mandando todo el mensaje).
Para decodificar, la mejor explicación la he encontrado en [SayoodSayood2000]:



Para nuestro ejemplo, sería el proceso siguiente:



Nótese cómo el árbol obtenido es el mismo.

3 Consideraciones prácticas

La mayor ventaja del algoritmo de Vitter frente al de Huffman y al FGK es que utiliza menos bits para mensajes cortos, donde las ligeras eficiencias de codificación son relativamente más significativas. Además, cuando la longitud de los mensajes crece, el coste de comunicación de los Algoritmos de una fase es asintóticamente igual al de dos fases. Esto lo justifica Vitter con diferentes tablas que, en general, muestran cómo funcionan los 3 algoritmos usados en diferentes fuentes de mensajes, y en todos los casos el algoritmo de Vitter actúa mejor, o como mínimo igual, que los otros dos.
Por ejemplo, en la siguiente tabla se muestran los resultados obtenidos al codificar un libro técnico escrito en TeX:

t	k	S_t	b/l	D_t^{Λ}	b/l	D_t^{FGK}	b/l
100	40	372	7.7	345	6.7	378	7.0
500	123	2566	7.5	2514	6.9	2625	7.1
1000	177	5904	7.6	5875	7.2	6029	7.3
10000	248	67505	7.0	67769	6.9	67997	7.0
100000	256	691897	6.9	692591	6.9	692858	6.9
588868	256	4170298	7.1	4171314	7.1	4171616	7.1

t es la longitud del mensaje, k el tamaño del alfabeto, S_t es el coste de comunicación con Huffman clásico, D_t^{Λ} es el coste de comunicación para el algoritmo de Vitter, y D_t^{FGK} para el FGK. b/l representa los bits por caracter en cada codificación.
Como hemos mencionado, vemos como el que proporciona mejores resultados es el algoritmo de Vitter, especialmente con los tamaños menores.

4 Conclusiones

Tras este análisis, podemos concluir que los algoritmos adaptativos son mucho más útiles para la transmisión de mensajes codificados para telecomunicaciones, en los que normalmente transmitiremos mensajes cortos, al dividirlos en paquetes. Al hacerlo en una única fase, puede hacerse con más velocidad y eficiencia, además de que los códigos generados ocupan menos espacio al necesitar una cantidad media de bits por caracter menor. Por si fuera poco, el algoritmo de dos fases necesita transmitir el árbol completo, mientras que los algoritmos de una única fase no, pues la decodificación consiste en la construcción del árbol. Esto reduce la información que se necesita añadir a un mensaje para que este sea decodificable.
Vitter menciona, además, que en muchas ocasiones puede resultar de utilidad realizar la codificación por bloques de caracteres, en lugar de hacer caracter a caracter. No obstante, se debe ser cauto al hacer esto y restringir la cantidad posible de palabras, pues si no el tamaño del árbol podría dispararse.

References

[SayoodSayood2000] Khalid Sayood. 2000. *Introduction to Data Compression.* Morgan Kaufmann Publishers.

[VitterVitter1987] Jeffrey Scott Vitter. 1987. Design and analysis of dynamic Huffman codes. *J. ACM* 34 (Oct. 1987), 825–845. https://doi.org/10.1145/31846.42227.