ESTRUCTURAS Y ESTRATEGIAS DE PROGRAMACIÓN

PRÁCTICA CURSO 2017-2018

Alumno: Iván Adrio Muñiz

Correo: ivan.adrio@gmail.com

Teléfono: 660634998

Sumario

PREGUNTAS TEÓRICAS LISTAS

1.1. ¿Cómo depende el tamaño de almacenamiento del número de consultas diferentes en el registro de consultas? ¿Cómo depende del número de repeticiones?

Cada consulta diferente ocupara un nodo en nuestra lista. Por lo tanto, la dependencia es lineal.

Las repeticiones ocupan el mismo lugar en nuestra lista, por lo tanto, el tamaño de almacenamiento no depende del numero de repeticiones.

1.2. ¿Cómo depende el tiempo de localizar todas las posibles sugerencias de búsqueda del número de consultas diferentes? ¿Y del número de repeticiones? ¿Y de la longitud máxima de las consultas medida como el número de caracteres de su texto? ¿Y del tamaño del conjunto de caracteres permitido? Razona en términos del coste asintótico temporal en el caso peor que se podría conseguir para el método listOfQueries con este diseño. Consideremos una estructura de datos alternativa en la que disponemos de una lista de consultas similar a la considerada por cada carácter inicial (es decir, tendríamos una lista con todas las consultas que empiezan por el carácter “a”, otra con las que empiezan por “b”, etc.) ¿Cuánto se podría reducir el tiempo de búsqueda en caso peor? ¿Cambiaría el coste asintótico temporal?

El tiempo máximo en localizar todas las posibles sugerencias depende de forma lineal del número de consultas diferentes. Solamente debemos recorrer la lista 1 vez.

En cuanto al número de repeticiones, estas no influyen en el tiempo de ejecución, ya que las repeticiones se almacenan dentro del mismo objeto query.

La longitud máxima de las consultas y el tamaño del conjunto de caracteres permitido tampoco influye en el tiempo de computación. Cada cadena de texto se almacena como un String, y es este objeto el que se compara en las búsquedas. En ningún momento es necesario recorrer el String.

El coste asintótico temporal en el caso peor lo vamos a calcular como suma de dos componentes. Por un lado tendremos el coste asintótico temporal de filtrar la lista y por otro lado el coste de ordenarla.

El coste asintótico temporal de la fase de filtrado lo describimos anteriormente. Debemos recorrer la lista de inicio a fin, para ello se puede emplear el iterador de la lista, por lo que solo se accede a cada elemento una vez. Además, hay que tener en cuenta el coste de la comparación con el string de búsqueda y el coste de la inserción en la nueva lista. El coste de esta comparación se puede despreciar frente al coste del recorrido de la lista. El coste asintótico temporal en el caso peor de esta parte sería por lo tanto O(n).

En cuanto al coste de la ordenación, tenemos dos listas, la primera de ellas es la obtenida del paso anterior y la segunda es una lista auxiliar sobre la que iremos insertando los elementos ordenados.

El proceso de ordenación será el siguiente:

Cogemos un elemento i perteneciente a la lista1, la cual tiene tamaño “n”. Este elemento debemos compararlo con los elementos de la lista2, la cual tiene tamaño j<n. Acotando el problema superiormente, tendríamos que hacer n x n operaciones, por lo que el coste asintótico en el caso peor de esta parte es igual a: $O(n^2)$.

El coste asintótico temporal en el caso peor de nuestro problema viene marcado por el método de ordenación. Es por lo tanto: $O(n^2)$.

En cuanto a la estructura de datos alternativa, el coste asintótico temporal no variaría. El resultado de dividir igual a dividir la longitud de la lista entre un número constante (m → número máximo de caracteres). Sin embargo esto no modifica el coste asintótico temporal en el caso peor:

$$O(n^2/m) \rightarrow O(n^2)$$

1.3. ¿Se ajusta este diseño a los requisitos del problema? Explica por qué

Si bien el espacio de almacenamiento usado es correcto, no es así el tiempo de consulta de las sugerencias. El coste asintótico temporal en el caso peor del algoritmo es de orden cuadrático, algo que lo hace prácticamente inútil para este ejercicio.

PREGUNTAS TEÓRICAS ARBOLES

2.1. ¿Cómo depende el tamaño de almacenamiento del número de consultas diferentes en el registro de consultas? ¿Y del tamaño máximo de las consultas? ¿Y del tamaño del conjunto de caracteres permitido? Consideremos un conjunto de 50 caracteres y un tamaño máximo de consulta de 10 caracteres. Compara el tamaño máximo del árbol en la segunda aproximación con el tamaño máximo de la lista en la primera aproximación. ¿La diferencia se agrandará o se reducirá para conjuntos de caracteres mayores y consultas más largas?

No podemos determinar una proporción exacta entre el tamaño de almacenamiento y el número de consultas, pero si podemos acotarlo superiormente. Como máximo, por cada consulta diferente será necesario añadir “m” nodos + 1 (nodo de frecuencia) al árbol, siendo m el número máximo de caracteres admitido en una consulta. Así, siendo “n” el número de consultas, el tamaño de memoria sería proporcional a n x m. En cuanto al número de repeticiones no influye en el tamaño de almacenamiento.

El conjunto de caracteres permitido si afecta al tamaño del árbol. Siendo “c” el número máximo de caracteres, cada subárbol podrá tener como máximo un número “c+1” de hijos.

El tamaño máximo del árbol, contando los nodos de frecuencia, se puede calcular como:

$$a(n,c)= \sum_{i=0}^{n} 2\,xc^{i}=\frac{1-c^{n+1}}{1-c}\ x2$$

Para los datos del enunciado el resultado sería: 2 x 9.9649235e+16 nodos +

En el caso de las listas, el número máximo de consultas es igual al número de variaciones de c elementos tomados de n en n. Siendo n el tamaño máximo de la consulta. Con la particularidad de que las consultas de tamaño (n-1) caracteres ocuparan espacios distintos, mientras que en el árbol no lo harán.

La fórmula matemática sería es: $l(n,c)= \sum_{i=0}^{n} c^{i}$

Para los datos del enunciado el resultado es: 9.765625e+16 nodos

El numero máximo de nodos del árbol es el doble que el de la lista debido a la necesidad de añadir un nodo de frecuencia por cada consulta diferente. Esta proporción será constante e independiente del tamaño del problema.

2.2. ¿Cómo depende el tiempo de localizar una posible sugerencia de búsqueda del número de consultas diferentes? ¿Y del número de repeticiones? ¿Y de la longitud máxima de las consultas medida como el número de caracteres de su texto? ¿Y del tamaño del conjunto de caracteres permitido?

En el peor caso posible deberíamos recorrer todo el árbol para encontrar la sugerencia que queremos. Por lo tanto, debemos acceder 1 vez a cada nodo de nuestro árbol. En el peor caso posible, una consulta de longitud l añadirá l nodos nuevos al árbol, por lo que, si tenemos n consultas diferentes, como máximo tendremos n x l nodos en el árbol.

Con esto podemos acotar superiormente el coste temporal de nuestro problema como:

$$O(n)$$

El número de repeticiones no influye en el coste temporal.

Siendo c el tamaño del conjunto de caracteres permitido y l la longitud máxima de la consulta, el número de nodos que debemos recorrer en nuestro árbol será igual a:

$$\sum_{i=0}^{l} c^i = \frac{1-c^{l+1}}{1-c}$$

De esta fórmula deducimos lo siguiente:

El coste asintótico temporal en el caso peor en función de la longitud máxima de la consulta será:

$$O(k^n$$

Siendo k el tamaño del conjunto de caracteres permitido y n la longitud máxima de la consulta.

El coste asintótico temporal en el caso peor en función del tamaño del conjunto de caracteres permitido será:

$$O(n^k$$

Siendo n el tamaño del conjunto de caracteres permitido y k la longitud máxima de la consulta.

2.3. ¿Cómo depende el tiempo de localizar todas las posibles sugerencias de búsqueda del número de consultas diferentes? ¿Y del número de repeticiones? ¿Y de la longitud máxima de las consultas medida como el número de caracteres de su texto? ¿Y del tamaño del conjunto de caracteres permitido? Razona en términos del coste asintótico temporal en el caso peor que se podría conseguir para el método listOfQueries con este diseño.

Deberemos recorrer el subárbol correspondiente para buscar todas las posibles consultas. Esto sería en el peor caso que el prefijo de búsqueda fuese igual a la cadena vacía. Por lo tanto, el tamaño del problema es directamente proporcional al tamaño del árbol, ya que deberemos visitar cada nodo una vez.

En el peor de los casos, una consulta de longitud máxima “l”, añadirá “l” nodos nuevos a nuestro árbol. Por lo tanto, consideran “n” como el número de consultas diferentes, nuestro árbol tendría un tamaño máximo de n x l nodos.

El coste asintótico temporal en el caso peor para localizar todas las consultas de nuestro árbol, en función del número de consultas diferente será O(n).

El número de repeticiones no afectaría ya que se almacenarían en el nodo de frecuencia.

Siendo c el tamaño del conjunto de caracteres permitido y l la longitud máxima de la consulta, el número de nodos que debemos recorrer en nuestro árbol será igual a:

$$\sum_{i=0}^{l} c^i = \frac{1-c^{l+1}}{1-c}$$

De esta fórmula deducimos lo siguiente:

El coste asintótico temporal en el caso peor en función de la longitud máxima de la consulta será:

$$O(k^n$$

Siendo k el tamaño del conjunto de caracteres permitido y n la longitud máxima de la consulta.

El coste asintótico temporal en el caso peor en función del tamaño del conjunto de caracteres permitido será:

$$O(n^k$$

Siendo n el tamaño del conjunto de caracteres permitido y k la longitud máxima de la consulta.

El coste asintótico temporal en el caso peor del método listOfQueries será el resultado de sumar los costes de la búsqueda de las consultas y de la ordenación, el cual en función del numero de consultas diferentes sería igual a:

$$O(n) + O(n^2) \rightarrow O(n^2)$$

2.4. A la vista de las respuestas a las preguntas anteriores y de los requisitos de nuestro problema, ¿consideras éste un diseño más adecuado para implementar QueryDepotIF?

El coste de almacenamiento del árbol siempre va superior al coste de la lista, por lo que en este aspecto la lista sería más aconsejable.

En cuanto al coste temporal para la búsqueda de sugerencias, en el caso peor ambos métodos están igualados, con la diferencia de que en el caso del árbol el tamaño de nuestro problema se reduce a medida que el prefijo de búsqueda se hace más largo. Además, para un conjunto de caracteres pequeño y un numero de consultas muy elevado, el coste asintótico en el caso peor calculado en el apartado anterior y el tamaño del árbol máximo calculado también anteriormente son cálculos demasiado conservadores y los costes medios estarían muy por debajo de esos valores.

Finalmente la elección de un método u otro depende en gran medida del peso que demos a las variables “coste de almacenamiento” y “coste temporal de las sugerencias”. Personalmente me decantaría por el segundo método con el fin de proporcionar una mejor experiencia de uso y agilizar las consultas.

2.5. Compara el coste de encontrar todas las sugerencias posibles con el coste de ordenarlas por frecuencia (consideremos que el coste de ordenación en el caso peor sea del orden O(n·log(n))). ¿Sería aconsejable comenzar a realizar sugerencias antes incluso de que el usuario comience a teclear, con este diseño? ¿Por qué?

El coste de ordenación O(n·log(n))) es de orden superior al coste de la búsqueda de las posibles sugerencias O(n). Por este motivo es una prioridad filtrar las posibles sugerencias antes de ordenarlas y

así reducir el tamaño del problema. Esto desaconseja totalmente realizar sugerencias antes de que el usuario teclee, ya que, si esperamos a que el usuario introduzca el primer carácter, en una gran parte de las ocasiones nos evitaremos tener que buscar en el árbol completo.

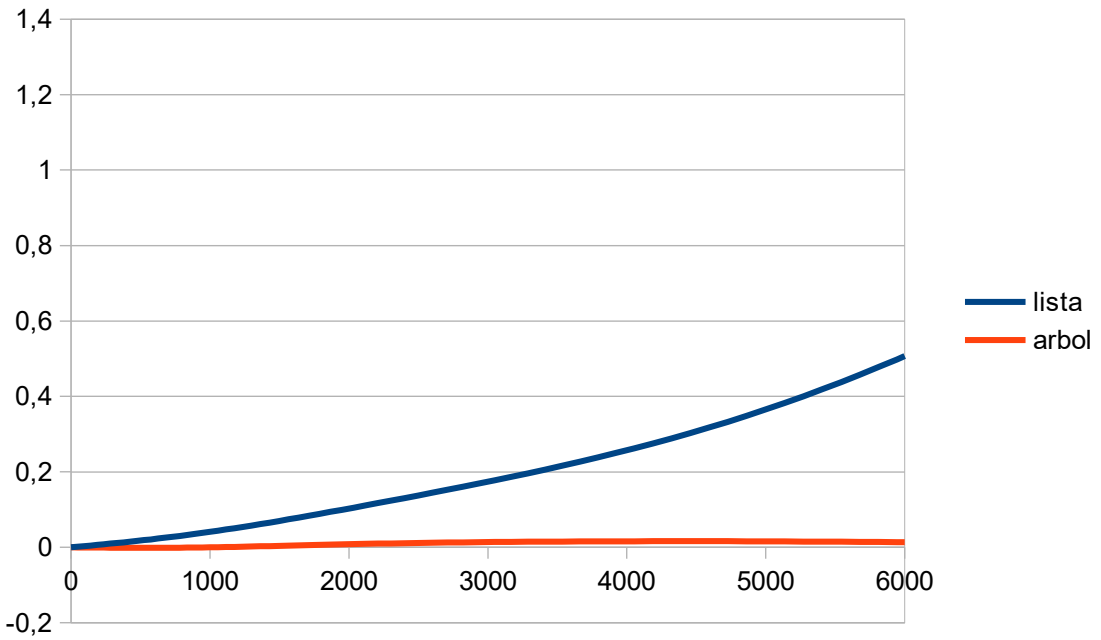
ANÁLISIS EMPÍRICO DE COSTES

Para realizar el estudio empirico de costes se han preparado una serie de juegos de consultas con un número diferente de consultas que varía desde las 1000 a las 50000. Se han realizado las mismas operaciones en cada uno de los juegos de consultas (10 operaciones de consulta de frecuencia y 5 de consulta de sugerencias) y se ha calculado el valor medio.

Una tercera parte del estudio analiza la evolución del coste del método listOfQueries en el caso peor. Para realizar este calculo se ha partido del supuesto de que el programa no ofrece sugerencias para la cadena vacía. Por lo tanto se ha realizado la consulta de sugerencias para cada una de las legras del abecedario y se ha calculado su media.

Las gráficas que vemos a continuación son el resultado del estudio:

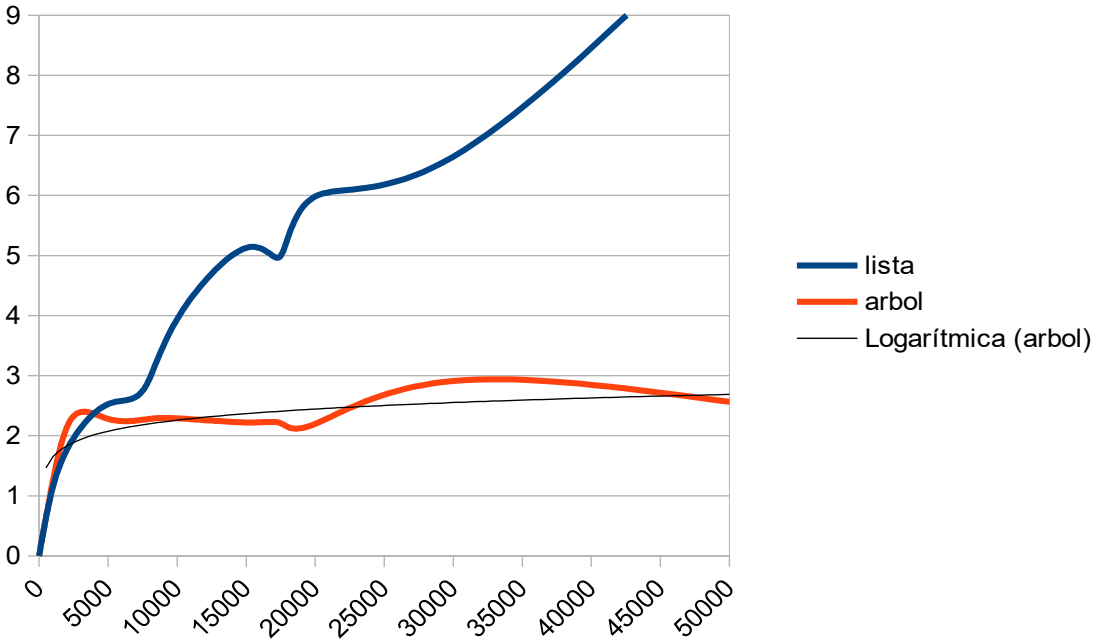
1. GetFreqQuerie



La grafica obtenida para la lista se corresponde con una ecuación lineal con un R^2=0,97, siendo la curva de ajuste que mejor se adapta.

En el caso del árbol podemos ver que el tiempo de las consultas apenas depende del número de consultas diferentes. Esto es así debido a que el factor crítico del que más depende este tiempo es la longitud máxima de las consultas.

2. listOfQueries



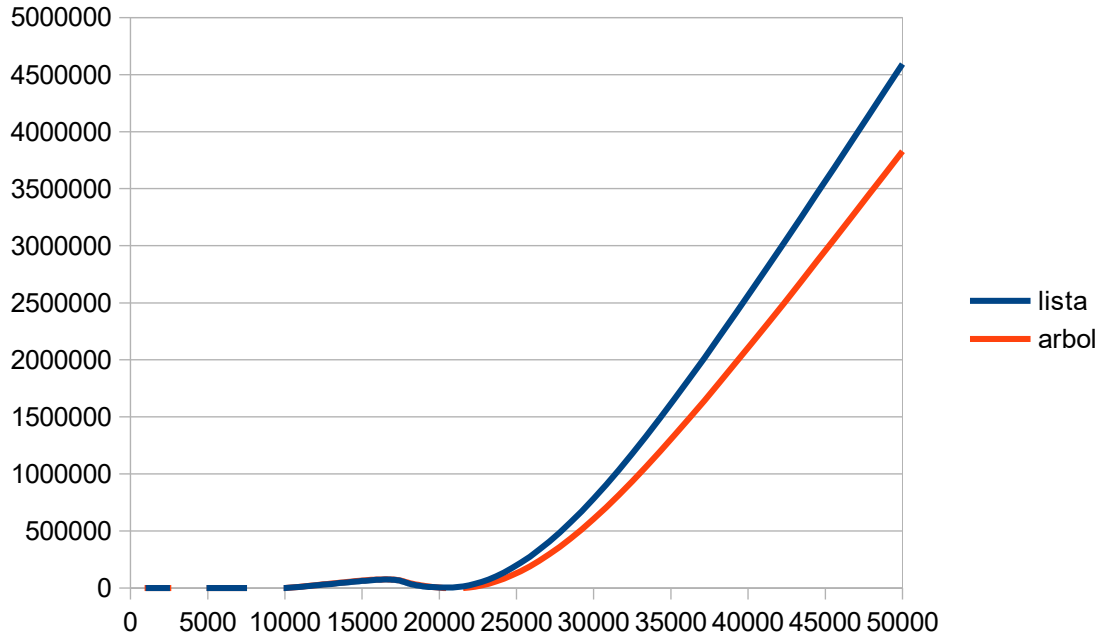
En este caso la gráfica obtenida en el caso de la lista se ajusta a una función cuadrática de grado 2 con un ajuste R^2=0,965

En el caso del árbol no encontramos una función que ajuste de manera adecuada. Si nos fijamos a medida que el número de consultas crece el tiempo apenas varía. Esto se podría explicar con el mismo razonamiento que seguimos en la gráfica anterior. Llegado un punto el tamaño del árbol apenas crece al añadir una nueva consulta. Solo debemos añadir aquellos caracteres situados al final de nuestra consulta que todavía no se encuentran en el árbol. Es decir, las cadenas mas largas que las ya existentes.

Las graficas anteriores no nos sirven para poder contrastar los análisis teóricos hechos en los primeros apartado, para ello debemos recurrir a los casos peores.

La siguiente gráfica muestra el análisis empírico del caso peor tal como se describe al principio de este apartado:

2. Caso peor listOfQueries



En este caso vemos que no hay una gran diferencia entre los dos métodos. Si bien el árbol es ligeramente más rápido. Esto esto así debido a que para "n" suficientemente alto, el algoritmo que cobra más peso es el de ordenación de las queries..

Si buscamos una curva que se ajuste a las graficas en ambos casos tendríamos un polinómio de grado 2, con un R^2 superior a 0,99 en los dos casos, por lo que podemos dar nuestro análisis teórico como valido.