INGENIERIA INFORMATICA
Escuela Politécnica Superior
Universidad Autónoma De Madrid

Algoritmia

Práctica 3

David Teófilo Garitagoitia Romero
Daniel Cerrato Sánchez

Pareja 08 Grupo 1261
15/12/2022

Índice de Contenidos

Introducción

En esta práctica se asentarán y practicarán conceptos relacionados con la programación dinámica y los problemas de selección en listas de números desordenados.

El problema de selección

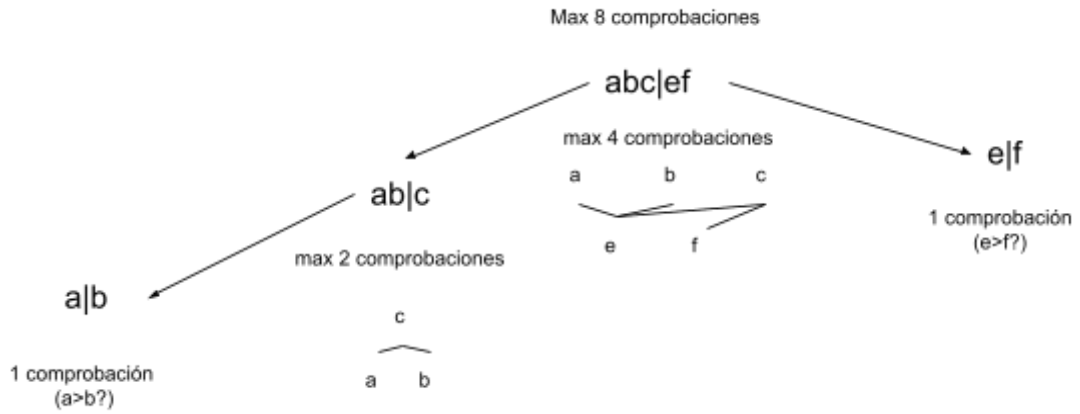
Cuestiones sobre QuickSelect y QuickSort

Cuestión 1

Argumentar que MergeSort ordena una tabla de 5 elementos en a lo sumo 8 comparaciones de clave

En una tabla de 5 elementos, MergeSort realiza la tarea "*partir*" que divide por la mitad en dos subtablas la tabla inicial hasta dividirla elemento a elemento. Después, va haciendo comparaciones entre subtablas para ordenarlas, generando nuevas tablas hasta que la tabla original queda ordenada.

La argumentación del máximo de comprobaciones dentro del algoritmo lo mostraremos a continuación en formato visual:



Cuestión 2

En realidad, en qsel_5 solo queremos encontrar la mediana de una tabla de 5 elementos, pero no ordenarla. ¿Podríamos reducir así el número de comparaciones de clave necesarias?

Se puede conseguir en únicamente 6 comprobaciones empleando el siguiente algoritmo:

- Ordenamos las primeras dos parejas, lo que supone una comparación por cada pareja.
  [a b c d e] -> *a <? b* - 1 comparación; *c <? d* - 1 comparación
- Obtenemos el menor de esas dos subtablas, lo que supone otra comprobación:
  [a < b c < d e] -> *a <? c* - 1 comparación
  El menor de esa comparación puede ser "eliminado", ya que sabemos que tiene 3 elementos mayores, por lo que no puede ser la mediana.
  a [b c < d e]
- Comparamos el último elemento con el elemento que se quedó sin pareja tras la "eliminación".
  a [b c < d e] -> *b <? e* - 1 comparación
- Volvemos a tener dos parejas de elementos, repetimos el paso 2: comprobamos el menor de esas parejas y lo eliminamos, ya que sabemos que es el menor de todos y, por tanto, también tiene 3 elementos por encima.
  a [b c < d e] -> *b <? e* - 1 comparación; *b <? c* - 1 comparación
  a b [c < d e]
- El último paso es comparar el elemento que se queda solo con el menor de la otra pareja. Este será la mediana, pues es el que tiene garantizado tener 2 elementos por encima y 2 por debajo (los eliminados).
  a b [c < d e] -> *c <? e* - 1 comprobación

En total se hacen 6 comprobaciones. Veamos un ejemplo con [7 3 4 2 1]:

7 <? 3 ; *4 <? 2* —> [3 < 7 2 < 4 1] - 2 comprobaciones
3 <? 2 —> 2 [3 < 7 4 1] - 3 comprobaciones
4 <? 1 —> 2 [3 < 7 1 < 4] - 4 comprobaciones
3 <? 1 —> 2 1 [3 < 7 4] - 5 comprobaciones
4 <? 3 —> 2 1 (3) 4 7 - 6 comprobaciones

Cuestión 3

¿Qué tipo de crecimiento cabría esperar en el caso peor para los tiempos de ejecución de nuestra función qsort_5? Intenta justificar tu respuesta primero experimental y luego analíticamente.

El mayor problema del QuickSort radica en la elección de pivote, que puede desembocar en un peor caso cuando el pivote es siempre el primer o último elemento de la tabla ordenada. En este caso, cada iteración deberá realizar n-1 operaciones, lo que conlleva un coste de:

$$\sum_{i=0}^{n}(n-i) = n^2 - \sum_{i=0}^{n} i = n^2 - \frac{n*(n+1)}{2} = n^2 - \frac{n^2}{2} - \frac{n}{2} = \frac{n^2-n}{2}$$

es decir, es del orden de O(n^2), muy por encima del caso medio de O(n*lg(n))

Caso mejor

{8, 6, 7, 9, 10}

/ \

{6, 7} {9, 10}

/ \ / \

{} {7} {} {10}

Caso peor

{6, 5, 4, 3, 2}

/

{5, 4, 3, 2}

/

{4,3,2}

– {4,3} - {3}

Para mejorarlo se emplea “qsort_5”, que mejora la elección del pivote mediante las medianas de 5 elementos. Esto nos permite evitar el caso peor gracias a las medianas, que nos garantizan una partición de la tabla prácticamente a la mitad, teniendo un caso medio y peor de O(n*lg(n)).

Programación dinámica

Cuestiones sobre las funcione de programación dinámica

Cuestión 1

El problema de encontrar la máxima subsecuencia común (no consecutiva) se confunde a veces con el de encontrar la máxima subcadena común consecutiva. Véase, por ejemplo, la entrada *Longest common substring problem* en Wikipedia. Describir un algoritmo de programación dinámica para encontrar esta subcadena común máxima consecutiva entre dos cadenas S y T, y aplicarlo "a mano" para encontrar la subcadena consecutiva común más larga entre las cadenas *bahamas* y *bananas* .

```
max_substring = ""
for i in range(len(S)):
   k = i
   actual = ""
   for j in range(len(T)):
      if k >= len(S):
         break
      if S[k] == T[j]:
         actual += S[k]
         l += 1
      else:
         if len(actual) > len(max_substring):
            max_substring = actual
         actual = ""
   return max_substring
```

Cuestión 2

Sabemos que encontrar el mínimo número de multiplicaciones numéricas necesarias para multiplicar una lista de *N* matrices tiene un coste *O(N^3)* pero ahora queremos estimar dicho número *v(N)* de manera más precisa. Estimar en detalle suficiente dicho número mediante una expresión *v(N) = f(N) + O(g(N))*, con *f* y *g* funciones tales que *|v(N) - f(N)| = O(g(N))*.

En nuestro código usamos tres bucles:

- El primero recorre las filas de la matriz, es decir, recorre todas las matrices.
- El segundo está pensado para poder moverse por las diagonales, es decir, recorre todas las matrices empezando por la que indique la fila.
- El último bucle se encuentra en una función auxiliar que calcula las multiplicaciones mínimas de la posición de la matriz en la que se encuentra el algoritmo. Este bucle simplemente recorre la tabla de dimensiones, desde la dimensión que indica las filas de la primera matriz hasta la que indica las columnas de la última matriz, sin contar estas.

El número de operaciones básicas que se realizan son 5: 2 multiplicaciones, 2 sumas y 1 comparación.

Esto, traducido a sumatorios para calcular el coste, queda de la siguiente manera:

$$\sum_{j=1}^{N}\sum_{i=j}^{N}\sum_{p=i-j+1}^{i+1} 5 = \sum_{j=1}^{N}\sum_{i=j}^{N} 5 * [i+1-(i-j+1)] = \sum_{j=1}^{N}\sum_{i=j}^{N} 5j = \sum_{j=1}^{N} 5j * (N-j) =$$

$$= \sum_{j=1}^{N} 5Nj - 5j^2 = 5N\sum_{j=1}^{N} j - 5\sum_{j=1}^{N} j^2 = 5N\frac{N(N+1)}{2} - 5\frac{N(N-1)(2N-1)}{6} = \frac{5}{6}N^3 - \frac{5}{6}N$$

Es decir, $v(N) = \frac{5}{6}N^3 + O(\frac{-5}{6}N)$, donde $f(N) = \frac{5}{6}N^3$ y $g(N) = \frac{-5}{6}N$

[FINAL DE DOCUMENTO]