Universidad Santiago de Chile
Facultad de Ciencia
Licenciatura en Ciencia de la Computación

Complejidad de Algoritmos

"Algoritmos de búsqueda y ordenamiento"

2º Laboratorio

Datos Informe
Nombre alumnos: Sergio Salinas
Danilo Abellá
Nombre profesor: Dino Araya

Contents

1 Problema 1

1.1 Solución Encontrada

La solución que se encontró es que el número que más se repite es 5896 con 16 repitencias.

1.2 Estrategia

El algoritmo guarda todos los caracteres del archivo tex en un arreglo A y crea otro arreglo B de lleno de ceros que tiene 10000 espacios de memoria y se usa para almacenar las frecuencias que tiene cada número.

En un ciclo, se obtienen los primeros 4 dígitos de A y se usan como indice para acceder al espacio de memoria que corresponde a ese indice y aumenta la frecuencia de aparición de ese número en 1, luego se quita el primer elemento de A aumentando la posición del puntero en 1 y se vuelve al inicio del ciclo. Esto se repite hasta que se recorra A por completo.

Además hay otra variable que se encarga de guardar la mayor frecuencia durante el ciclo, para mostrar el número que más repite solo se busca en B el indice que tenga esa frecuencia.

1.3 Justificación

Se utiliza este método por su efectividad, en lo que es la memoria, el arreglo A es de largo n y el arreglo B siempre es constante ya que siempre tendrá 10000 espacios de memoria que son los necesarios para guardar la frecuencia de los números del 0 al 9999, por lo la memoria total utilizada es de n + 1000, por lo que es de orden O(n).

En lo que es la complejidad del algoritmo, es de orden O(n), esto es debido a que en el ciclo de aumentar la frecuencia de B dado un indice obtenido de A su operación básica se repite n veces, y en el ciclo de buscar el mayor número la operación comprar se repite 10000 veces, por lo que se tiene que la complejidad es n + 10000.

2 Problema 2

2.1 Solución Encontrada

Como solución se tienen las siguientes secuencias de 4 "dígitos" (hexadigitos) consecutivos que se repiten en el número hexadecimal dado:

n0006-n0068-n00BB-n00DF-n0115-n011A-n013C-n01AB-n01AD-n03A1-n0402-n04C0-n056A-n0584-n061B-n0734-n07EF-n0810-n085F-n08BA-n0A12-n0A47-n0BA6-n0BCB-n0DC7-n0DF0-n0F28-n1018-n105D-n10B3-n1133-n1156-n1159-n1183-n11A1-n1290-n12B4-n12DC-n133A-n13E0-n1421-n1462-n14BA-n14CD-n14D9-n14E5-n153C-n1548-n15D6-n1612-n1625-n1636-n16DF-n1811-n183B-n1856-n18B1-n1936-n1948-n19EE-n1BD3-n1C20-n1C36-n1C90-n1CE6-n1D00-n1D39-n1DC2-n1DDF-n1E0A-n1E63-n1E79-n1E8A-n1EAF-n1EBD-n1F2E-n1F6C-n1F9B-n202E-n20AD-n20C8-n2105-n21E7-n2299-n22B6-n2463-n2466-n2469-n2547-n25BD-n2623-n2690-n2796-n27A1-n2821-n2850-n2851-n2858-n2895-n28AB-n299B-n299F-n2AB3-n2ADA-n2B89-n2B8B-n2D38-n2D51-n2DFD-n2E13-n2E28-n2E9C-n2EB3-n2EF1-n2EF6-n2F32-n30DC-n314E-n317C-n31EA-n31F6-n320F-n33A3-n33E8-n3604-n361D-n369B-n3707-n3820-n3822-n38D8-n392E-n396A-n3A3A-n3AE1-n3B12-n3B3E-n3C6F-n3D5A-n3D7C-n3E81-n3E89-n3FD6-n4000-n4152-n4183-n42EF-n42F5-n42F6-n4324-n442E-n4574-n4595-n4614-n46DB-n46F6-n46FC-n479B-n4898-n48E4-n48FD-n49F1-n4A41-n4A99-n4B27-n4B47-n4BA3-n4BFB-n4CDD-n4E3D-n4E6C-n501A-n5056-n5094-n5133-n525F-n5282-n52DF-n52EC-n532E-n542F-n5470-n5546-n55AB-n55FD-n5605-n562E-n5664-n56E1-n5748-n5770-n586E-n5882-n593E-n59CB-n5AA5-n5BDD-n5C02-n5CB0-n5CFA-n5D88-n5EE3-n5F01-n5F79-n5F95-n5FEC-n602A-n6078-n6085-n612A-n614E-n618B-n6282-n6293-n62E9-n6369-n6382-n6629-n6636-n6645-n66A0-n67BC-n6842-n698D-n6A36-n6A37-n6A51-n6B2A-n6B6A-n6DFF-n6E2F-n6E37-n6EA6-n6F47-n6FB2-n6FF3-n7061-n7080-n7157-n7216-n724D-n74E6-n7509-n7560-n75B1-n7634-n7711-n771F-n7732-n7808-n78C1-n7960-n7A58-n7B8E-n7B9F-n7CCF-n7CD3-n7D9C-n7DA8-n7DD1-n7E6A-n800B-n80BB-n8128-n8129-n8162-n82EF-n8346-n83B5-n84A5-n8507-n8509-n8556-n858E-n85A3-n85F0-n86E3-n8740-n8839-n8986-n8A0B-n8B1D-n8B38-n8D01-n8D40-n8DB3-n8F48-n8FD2-n9045-n924A-n9317-n9361-n93D5-n93E8-n94B7-n94C2-n9586-n9662-n96A2-n9802-n991B-n993B-n9A45-n9A53-n9B04-n9B07-n9B6F-n9C2B-n9C60-n9D65-n9DCB-n9E15-n9E86-n9F25-n9F8D-n9FB4-nA0C1-nA0E2-nA124-nA14D-nA161-nA1D4-nA285-nA323-nA366-nA37F-nA3CB-nA476-nA58C-nA65C-nA7A9-nA835-nA995-nA9BE-nABEA-nABFC-nAC71-nACE1-nADA3-nAE4D-nAFD6-nB023-nB03A-nB155-nB1DC-nB21A-nB266-nB277-nB27B-nB285-nB2F3-nB331-nB38E-nB3B1-nB3EE-nB457-nB513-nB557-nB5FA-nB6A3-nB6C1-nB750-nB7D9-nB7E3-nB812-nB820-nB8B5-nB93D-nB9D3-nBA33-nBA99-nBA9B-nBB3A-nBB56-nBBCA-nBC2A-nBC9B-nBD31-nBE32-nBF00-nBF2C-nC115-nC25A-nC277-nC2DA-nC2DD-nC372-nC4BF-nC511-nC6E8-nC902-nC904-nC946-nC9E0-nCA92-nCB06-nCBEE-nCCC8-nCE0B-nCE77-nCF0B-nD04C-nD1B5-nD1CF-nD1D0-nD519-nD586-nD65F-nD6A1-nD6EB-nD83D-nD8FE-nD915-nD993-nD9B9-nDA2F-nDAE9-nDC09-nDC14-nDC25-nDD57-nDDF8-nDF01-nDFA1-nDFA6-nDFF8-nE03A-nE0E1-nE14E-nE153-nE169-nE1B0-nE1E7-nE1F9-nE305-nE39F-nE3D0-nE3D3-nE4C6-nE4D6-nE593-nE60B-nE647-nE69F-nE6AD-nE6BA-nE6C5-nE799-nE8A3-nE8AE-nE8EF-nE9DB-nEB26-nEB65-nEC6E-nECC8-nECF1-nED93-nEDFA-nEE30-nEE60-nEECC-nEF6A-nEF7D-nF011-nF017-nF01C-nF050-nF06A-nF092-nF0BD-nF0CA-nF0F7-nF19B-nF2B8-nF442-nF5EB-nF708-nF724-nF728-nF74B-nF74E-nF7DA-nF845-nF89E-nF8E7-nF8E8-nF953-nF95E-nFA3D-nFAD5-nFBC9-nFCED-nFD2C-nFE6E-nFF8E-nFFA3-nFFEA

2.2 Estrategia

Se utiliza un arreglo "a" para guardar todos los dígitos del archivo tex y "hexarray" (solo con ceros y 10000 en espacio de memoria) para guardar las secuencias de 4 dígitos hexadecimeales. Nota: Para trabajar con números hexadecimales se utilizó la función "strtol(char, NULL, 16)".

Luego se recorrió el arreglo "hexarray" ordenando todas las secuencias de 4 dígitos con el método de ordenamiento por inserción.

Una vez ordenado de menor a mayor, el arreglo es recorrido y se van mostrando las secuencias que se repiten al menos 1 vez con su respectiva frecuencia.

2.3 Justificación

Se decide crear una lista ordenada con todos los números de 4 dígitos del archivo tex debido a su facilidad de uso y porque el coste en memoria de este arreglo es de n-4. La complejidad de obtener todos los números de 4 dígitos del arreglo con los datos del archivo es de n-4.

Se decide usar el método de inserción dado a sus bajos requerimientos de memoria y estabilidad (nunca intercambia registros con claves iguales). Pese a su lento proceso y la numerosas comparaciones necesarias a realizar, muestra un comportamiento razonablemente bien en gran cantidad de situaciones, y esta no es la excepción, además que ser de fácil implementación.

El método de inserción tiene como complejidad en el peor de los casos O(n^2) y en el mejor de los casos O(n) (o sea, cuando el arreglo está ordenado), y el recorrido guardar las secuencias de 4 dígitos hexadecimeales una es de n-4.

Por lo que la complejidad del algoritmo sería:

En el mejor caso: n + (n-4)
En el peor caso: n^2 + (n-4)

Y su uso de memoria total es de 2n-4, debido a que n es lo que cuesta el arreglo con los dígitos de un archivo, n-4 el arreglo con todos los números de 4 dígitos y el insertion sort solo requiere O(1) de espacio adicional.

3 Problema 3

3.1 Solución Encontrada

Archivo$_3$*.tex* no arroja ningún resultado, por lo que no hay ninguna secuencia de 6 dígitos repetidas según el corte que se le dio.

3.2 Estrategia

Se uso un método similar al segundo ejercicio, se guarda la data del texto *Archivo*$_3$*.tex* en un arreglo A de largo n, este arreglo se recorre $\frac{n}{6}$ veces extrayendo en segmentos de 6 dígitos cada número del arreglo y guardandolos en un nuevo arreglo B en forma de INT.

Este arreglo B se ordena usando el método de ordenamiento quicksort, de esta forma el arreglo B tiene los elementos repetidos uno al lado del otro, por último se recorre el arreglo contando la frecuencia con la que aparece cada elemento repetido de manera seguida y los que tengan frecuencia mayor a uno se muestran.

3.3 Justificación

Se utilizo este método debido a que su memoria es lineal, solo ocupa dos arreglos de largo n y $\frac{n}{6}$, el quicksort ocupa $\log n$ de memoria, por lo que ocupa $\frac{7n}{6} + \log n$ de memoria.

Además de su rapidez, dividir el arreglo en secciones de 6 tiene complejidad de $\frac{n}{6}$ y ordenar el arreglo con el método quicksort tiene una complejidad de $n \log n$, por lo que su complejidad final es de $\frac{n}{6} + n \log n$, haciendo que el algoritmo sea de complejidad lineal logarítmica.

Se usa el algoritmo quicksort debido a su rapidez y que ser el largo del arreglo par es más seguro que el pivote elegido sea el correcto.