Instituto Politécnico Nacional
Escuela Superior de Cómputo

Análisis de Algoritmos

Práctica 2: Pruebas a Posteriori (Algoritmos de Búsqueda)

Equipo:
CompilandoConocimiento

Integrantes:
Morales López Laura Andrea
Ontiveros Salazar Alan Enrique
Rosas Hernández Óscar Andrés

Profesor:
Franco Martínez Edgardo Adrián

Índice

1. Introducción

Uno de los típicos problemas dentro de un curso de programación es la búsqueda. Estos algoritmos son la base de muchos otros, además de que tenemos con ellos unas ideas interesantes a usar en otro tipo de algoritmos, como búsqueda binarai y derivados, las estructuras de datos y los algoritmos concurrentes.

1.1. Métodos

1.2. Definición

1.3. Algoritmos de Búsqueda

2. Planteamiento del Problema

3. Diseño de la Solución

4. Implementación de la Solución

4.1. Búsqueda Lineal

```
/*=============================================================================
=====================             LINEAL SEARCH             ====================
==============================================================================*/
/**
 * Is just lineal search ...
 *
 * @param Data               A pointer to the array of int to sort
 * @param DataSize           The size of the Data array
 * @param NumberToSearch     The number to search
 * @return                   Nothing...I'm modifying the raw data
 */
int LinealSearch(int Data[], int DataSize, int NumberToSearch) {     //==== LINEAL SEARCH ======

    for (int i = 0; i < DataSize; ++i) {                             //For each item in Data
        if (Data[i] == NumberToSearch)                               //If find it
            return i;                                                //Go
    }

    return -1;                                                       //Not found

}
```

5. Actividades y Prueba

5.1. Comparativas Individuales

5.2. Comparativas Globales

5.3. Preguntas

- **¿Cuál de los 3 algoritmos es más fácil de implementar?**

  El más sencillo (o en algoritmia conocido coloquialmente como la bruta) es la búsqueda lineal. Es la solución mas sencilla con apenas 3 o 4 línea de código.

  Además de ser sencilla de implementar es la que es menos probable que cometas un error a la hora de definir los índices o variables de apoyo.

- **¿Cuál de los 3 algoritmos es el más difícil de implementar?**

  El más difícil de implementar en la búsqueda usando un Binary Search Tree, porque nos obliga o bien a diseñar algoritmos que son comunmente recursivos en algo iterativos.

- **¿Cuál de los 3 algoritmos es el más difícil de implementar en su variante con hilos?**

  Curiosamente la respuesta creo que tiene que ir para la versión con hilos de la búsqueda lineal.

  Por un lado el trabajar con hilos siempre complica las cosas, sobretodo porque involucra problemas de sincronicidad y el manejo de variables que pueden ser posiblemente accedidas por varios hilos. Aquí hay una consideración importante que hay que hacer: Solo estamos buscando una variable, por lo que no me tengo que preocupar con la memoria, el primer hilo que me mande una respuesta, esa la tomaré y todos los demas hilos al notar el cambio en la variable se matan.

- **¿Cuál de los 3 algoritmos en su variante con hilos resulto ser mas rápido?**

  La búsqueda binaria, es cierto, que fue más rapido la versión lineal, pero aún así la versión con hilos es endemoniadamente rápida.

- **¿Cuál algoritmo tiene menor complejidad temporal?**

  La búsqueda binaria. Es uno de los algoritmos más famosos por tener un $O(\ln_2(n))$ aunque en ciertos casos, CON UN ÁRBOL BALANCEADO la búsqueda con árbol puede tener esa complejidad.

- **¿Cuál algoritmo tiene mayor complejidad temporal?**

  Sin duda alguna ese premio tiene que ir para la búsqueda lineal, en el peor de los casos tenemos un búsqueda de $O(n)$.

- **¿El comportamiento experimental de los algoritmos era el esperado? ¿Porque?**

  Si, al menos en rangos generales de los algoritmos se esperaba y se ve la gran diferencia entre los algoritmos de búsqueda rápidos y la búsqueda lineal.

  Por otro lado a simple vista uno puede pensar que una variante con hilos siempre será más rapida que una versión que no es concurrente, pero esto no tiene porque ser así, por ejemplo en la búsqueda binaria es más que obvio que como solo estamos haciendo serie de comparaciones entonces el añadir hilos no mejora nada y solo sobrecarga al sistema operativo con su creación.

- **¿Sus resultados experimentales difieren mucho de los análisis teóricos que realizo? ¿A que se debe?**

  No, como continuación de la respuesta anterior, si se analizaban con algo de tranquilidad los algoritmos se podia llegar a unas muy buenas estimaciones.

- **¿Los resultados experimentales de las implementación con hilos de los algoritmos realmente tardaron $\frac{F(t)}{\#hilos}$ de su implementación sin hilos? ¿Cuál es el variante con hilos? ¿Es lo que esperabas? ¿Por qué?**

  Jajaja, si hablamos de la búsqueda lineal, claro, es decir, no fue exactamente esa cantidad sobretodo por los grandes margenes de error que tenemos a la hora de medir tiempos tan pequeños pero el punto es que si, casi casi tenemos una mejora linealmente correspondiente con la cantidad de hilos, despues de todo dividimos el trabajo entre la cantidad de hilos.

  Pero en los otros dos al ser escencialemente una gran cantidad de comparaciones en lineal entonces no podemos llegar a practicamente ninguna mejora, incluso un desempeño un poco peor.

  En general tenemos que para n hilos tenemos una nueva función complejidad:

  - Búsqueda Lineal: $f_h(t) = \frac{f(t)}{\text{Número de hilos}}$
  - Búsqueda Binaria: $f_h(t) = 1f(t)$
  - Búsqueda BTS: $f_h(t) = 1f(t)$

- **¿Existió un entorno controlado para realizar las pruebas experimentales? ¿Cuál fue?**

  Sí, usamos una PC con las siguientes características:

  - HP EliteDesk 700 G1 SFF
  - 8GB de memoria RAM, DDR3 SDRAM non-ECC, 1600MHz
  - Procesador Intel(R) Core(TM) i5-4590 (4° generación), CPU @ 3.30GHz (4 CPUs), ∼3.3GHz. Intel vPro Technology
  - 1TB de disco duro HDD, Serial ATA-600 6Gb/s, 7200 rpm

  Y el software fue el siguiente:

  - Sistema operativo Linux
  - Distribución Elementary OS 0.4
  - Compliador GCC con soporte para C11
  - Python 3.6 con las bibliotecas `matplotlib` y `numpy`
  - No había ninguna aplicación abierta al momento de ejecutar los algoritmos

- **¿Si solo se realizará el análisis teórico de un algoritmo antes de implementarlo, podrías asegurar cual es el mejor?**

  Con un anális básico, es decir con el uso de cotas rápidamente podríamos descartar a la búsqueda lineal SI ES QUE NUESTRO ARREGLO ESTUVIERA ORDENADO, y ya que el BTS requiere otro tiempo inicial para la creación del árbol sería fácil decantarse por la sencilla búsqueda binaria.

  Pero si tenemos un arreglo que no esta ordenado las cosas cambian, ahí depende mucho del tamaño del problema y de la cantidad de hilos que podemos correr eficientemente lo que decanta la balanza por BTS o un búsqueda lineal.

- **¿Qué tan difícil fue realizar el análisis teórico de cada algoritmo?**

  - Búsqueda Lineal: Sencillo, de esos que son ejemplos básico en clase
  - Búsqueda Binaria: Algo más compleja, sobretodo por el cálculo de los valores límites, pero al verdad es que no hay mucho más álla.
  - Búsqueda BTS: Igualmente, el peor caso en un árbol BALANCEADO es bastante sencillo de obtener

- **¿Qué recomendaciones darían a nuevos equipos para realizar esta práctica?**

  - Hagan sus scripts jóvenes, pues aunque parezca más tedioso y crean que es más rápido anotar toda la información solicitada (que es mucha) a mano, es más eficiente por si se requiere cambiar algún algoritmo o los valores de entrada. Al menos que el script guarde los tiempos en algún archivo de texto con un formato que quieran.
  - Prueben sus algoritmos con arreglos pequeños antes de ordenar los 10 millones para ver si realmente los programaron bien.
  - Hagan sus programas generales, es decir, que reciban cualquier tamaño de subarreglo y el algoritmo a usar.
  - Usen Linux, pues en Windows no están las bibliotecas para medir los tres tiempos.
  - Cuando corran sus algoritmos, procuren cerrar todas las tareas en segundo plano para que los tiempos obtenidos sean lo más apegados a la realidad posible.

6. Errores Detectados

7. Posibles Mejoras

8. Conclusiones