Pontificia Universidad Javeriana
Departamento de Ingeniería de Sistemas
Estructuras de Datos
Taller 3: Árboles de Búsqueda, 2024-10

1 Objetivo

Evaluar la eficiencia de los árboles binarios ordenados en operaciones de búsqueda de elementos. En especial, se desea evaluar las habilidades del estudiante en el uso y análisis de las operaciones de inserción y eliminación de datos en árboles binarios ordenados, AVL y rojinegros (RN).

2 Recordatorio: compilación con g++

La compilación con g++ (compilador estándar que será usado en este curso para evaluar y calificar las entregas) se realiza con los siguientes pasos:

1. **Compilación**: de todo el código fuente compilable (**ÚNICAMENTE LOS ARCHIVOS CON EXTENSIONES** `*.c`, `*.cpp`, `*.cxx`)
   `g++ -std=c++11 -c *.c *.cxx *.cpp`
2. **Encadenamiento**: de todo el código de bajo nivel en el archivo ejecutable
   `g++ -std=c++11 -o nombre_de_mi_programa *.o`

   Nota: Estos dos pasos (compilación y encadenamiento) pueden abreviarse en un sólo comando:
   `g++ -std=c++11 -o nombre_de_mi_programa *.c *.cxx *.cpp`
3. **Ejecución**: del programa ejecutable anteriormente generado
   `./nombre_de_mi_programa`

ATENCIÓN: Los archivos de encabezados (*.h, *.hpp, *.hxx) **NO SE COMPILAN**, se incluyen en otros archivos (encabezados o código). Así mismo, los archivos de código fuente (*.c, *.cpp, *.cxx) **NO SE INCLUYEN**, se compilan. Si el programa entregado como respuesta a este Taller no atiende estas recomendaciones, automáticamente se calificará la entrega sobre un 25% menos de la calificación máxima.

2.1 Medición del tiempo de ejecución

Para la medición de tiempos de ejecución en C++, se provee la función `clock` (librería `ctime`), la cual retorna el tiempo de procesador utilizado por el programa. Para calcular el tiempo actual utilizado por un programa, el valor retornado por la función debe compararse con el valor retornado por un llamado previo a la misma función. A continuación se presenta un ejemplo de uso (adaptado de `http://www.cplusplus.com/reference/ctime/clock/`):

```
#include <ctime>
#include <math>
#include <iostream>

int main () {
  std::clock_t init_time = std::clock( );
  int freq = 99998;

  for ( int i=2; i<=n; ++i )
    for ( int j=sqrt(i); j>1; --j )
      if ( i%j==0 ) {
        --freq;
        break;
      }

  std::clock_t end_time = std::clock( );
```

```
  cout << ‘‘The number of primes lower than 100000 is: ’’ << freq << ‘‘\n’’;
  double calc_time = ( end_time - init_time ) / double( CLOCKS_PER_SEC );
  cout << ‘‘It took me ’’ << calc_time << ‘‘ seconds.\n’’;
  return 0;
}
```

3 Desarrollo del taller

Una empresa manufacturera almacena la información de sus productos utilizando como índice un código numérico (identificación) de 6 dígitos (0-9). La empresa busca una estrategia de almacenamiento de esta información para que la búsqueda de un producto particular o la extracción de grupos de productos de acuerdo a rangos de códigos sea fácil y eficiente.

El objetivo del taller consistirá en utilizar y comparar tres implementaciones diferentes de árboles para organizar los códigos de identificación de los productos: la implementación propia del árbol binario ordenado, la implementación propia del árbol AVL y la estructura `std::set<T>` de la STL (que implementa internamente un árbol rojinegro (RN) para el ordenamiento). Por simplicidad, se asumirá que sólo se dispone del código numérico de los productos como información asociada.

La información necesaria para poblar los árboles con los códigos de los productos se encuentra en un archivo de texto, donde línea por línea se indica la operación a realizar. Sólo dos operaciones son posibles: `A`: agregar el número de identificación del producto dado y `E`: eliminar el número de identificación de producto indicado. Por ejemplo, el siguiente archivo:

```
A 243535
A 546384
E 243535
```

Agregaría al árbol dos nuevos productos con códigos 243535 y 546384, para luego eliminar del árbol el producto identificado con código 243535. En el archivo, se garantiza que todos los códigos tienen longitud de 6 dígitos, por lo que pueden existir códigos con ceros a la izquierda.

El desarrollo del taller consistirá en diseñar (utilizando la plantilla de diseño de TADs vista en clase) e implementar un programa que permita almacenar ordenadamente los códigos de productos de un archivo de texto, utilizando un árbol binario ordenado, un árbol AVL y uno rojinegro (RN). En particular, el programa debe:

- Leer adecuadamente desde el archivo de texto la información de los códigos de productos y de las operaciones a aplicar (adición y eliminación).
- Utilizar la implementación propia del árbol binario ordenado, del árbol AVL y la estructura `std::set<T>` de la STL (árbol RN) para cargar la información de los códigos en los tres árboles diferentes. Además de que la información quede adecuadamente distribuída y organizada dentro de cada estructura, es de especial importancia realizar la medición del tiempo de ejecución del proceso de carga de la información en cada una.
- Implementar una función que permita extraer el recorrido en inorden de los árboles binario ordenado y AVL en una lista (`std::list<T>`). El resultado de esta función permitirá verificar si, luego del proceso de carga, el árbol binario ordenado, el árbol AVL y la estructura `std::set<T>` de la STL (árbol RN) almacenan exactamente la misma información.
- Utilizar las mediciones de tiempo y la comparación de los datos almacenados en los tres árboles, ejecutados sobre los diferentes archivos de ejemplo, para redactar un párrafo (dentro del documento de diseño) en el cual describa cuál estructura es la más adecuada para este problema.

Como entrada, el programa debe recibir por línea de comandos el nombre del archivo que contiene la información necesaria. Esto quiere decir que, una vez compilado el programa, debe ejecutarse por la línea de comandos de la siguiente forma:

```
./nombre_de_mi_programa archivo_entrada.txt
```

4 Evaluación

La entrega se hará a través de la correspondiente asignación de BrightSpace, antes de la medianoche del martes 9 de abril. Se debe entregar un único archivo comprimido (único formato aceptado: .zip) que contenga dentro de un mismo

directorio (sin estructura de carpetas interna) el documento de diseño (.pdf) y el código fuente (.h, .hxx, .cxx, .cpp). Si la entrega contiene archivos en cualquier otro formato, será descartada y no será evaluada, es decir, la nota definitiva de la entrega será de 0 (cero) sobre 5 (cinco).

La evaluación del taller tendrá la siguiente escala:

- **Excelente** (**5.0/5.0**): El estudiante diseñó correctamente (siguiendo la plantilla) e implementó una solución que utiliza correctamente el árbol binario ordenado, el árbol AVL y la estructura `std::set<T>` de la STL (árbol RN) para almacenar ordenadamente los datos, y además realiza la comparación entre las estructuras en base a datos y hechos recopilados.
- **Bueno** (**4.2/5.0**): El estudiante diseñó correctamente (siguiendo la plantilla) e implementó una solución que utiliza correctamente dos de las estructuras (árbol binario ordenado, árbol AVL o árbol RN) para almacenar ordenadamente los datos, y además realiza la comparación entre las estructuras en base a datos y hechos recopilados.
- **Aceptable** (**3.5/5.0**): El estudiante diseñó correctamente (siguiendo la plantilla) e implementó una solución que utiliza correctamente sólo una de las estructuras (árbol binario ordenado, árbol AVL o árbol RN) para almacenar ordenadamente los datos.
- **No fue un trabajo formal de ingeniería** (**3.0/5.0**): El estudiante implementó una solución completa o parcial, pero no la diseñó correcta o completamente.
- **Necesita mejoras sustanciales** (**2.0/5.0**): El estudiante diseñó y/o implementó una solución, pero no es completa o no soluciona lo pedido.
- **Malo** (**1.0/5.0**): El código entregado por el estudiante no compila en el compilador g++ (mínimo versión número 4.5).
- **No entregó** (**0.0/5.0**).