Nombre: Alejandro Mena
Materia: Análisis de algoritmos (AA17II)

Análisis de implementaciones de la estructura Symbol Table.

Descripción del Proyecto

El objetivo de este proyecto es comparar la eficiencia de varias implementaciones de una Tabla de Símbolos.

Para probar la eficiencia usaremos la implementación de un cliente llamado "Frequency Counter". Este cliente construye una Tabla de Símbolos donde las llaves son las palabras de una longitud mayor a "L" y el valor es la frecuencia con que aparece esa variable en un texto "T".

Para la implementación de la Tabla de Símbolos utilizaremos 3 implementaciones:

- Búsqueda Binaria
- Árboles Rojo-Negro
- Tabla Hash con Consulta Lineal

Para cada uno de estos algoritmos probar con palabras más largas de (valor de L):

- 3 caracteres
- 6 caracteres
- 9 caracteres
- 12 caracteres

Utilizando los siguientes textos (T):

TinyTale.txt (decenas de palabras)

Tale.txt (miles de palabras)

Leipzig1M.txt (millones de palabras)

Desarrollo

Para este proyecto las implementaciones de Binary Search Tree, Black Red Tree y Hash Table Linear probing fueron obtenidas del sitio web:

https://algs4.cs.princeton.edu/30searching/

Estas implementaciones están desarrolladas en el lenguaje JAVA.

Calculo del tiempo

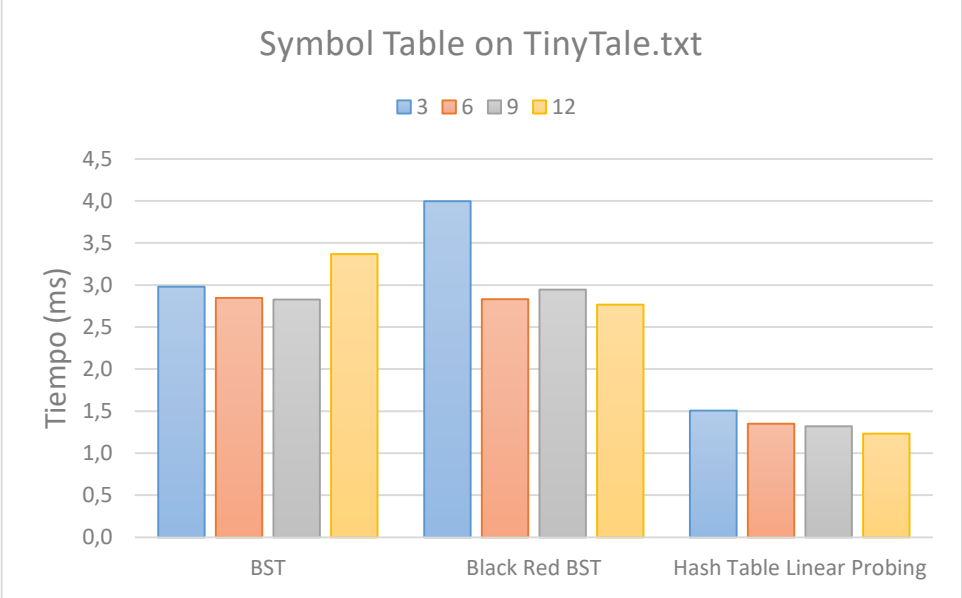
Ejemplo de código del cálculo de tiempo:

```
int minlen =3 ; // key-length cutoff
....
long time_1, time_2,difference=0;
long sum=0; //contains the total time of creation  Symbol Table

    time_1 = System.nanoTime();
    BST<String, Integer> st = new BST<>(); //create the reference
    time_2= System.nanoTime();
    difference=time_2 - time_1 ;
    sum+=difference; //add time

    while (input.hasNext())
    { // Build symbol table and count frequencies.
       String word = input.next();

       if (word.length() < minlen) continue; // Ignore short keys.

       time_1 = System.nanoTime();
       //Do search and insert operations
       if (!st.contains(word)) st.put(word, 1);
       else st.put(word, st.get(word) + 1);
       time_2= System.nanoTime();

       difference=time_2 - time_1 ;
       sum+=difference; //add time
    }
    System.out.println("The difference is: " +sum); //print total time
```

Análisis

A continuación, se analiza los tiempos de ejecución del cliente Frequency counter con distintos valores de longitud “L” y tres implementaciones de Symbol Table:

Archivo TinyTale.txt



En el archivoTinyTale.txt los tiempos de ejecución de BST y Black Red BST son similares y no llegan a un segundo. Debería suceder que Black Red BST se ejecute más rápido, mas no tan rápido ya que según los tiempos de ejecución de estas implementaciones, expuestos por el Prof. Sedgewick[1] en uno de videos[2]:

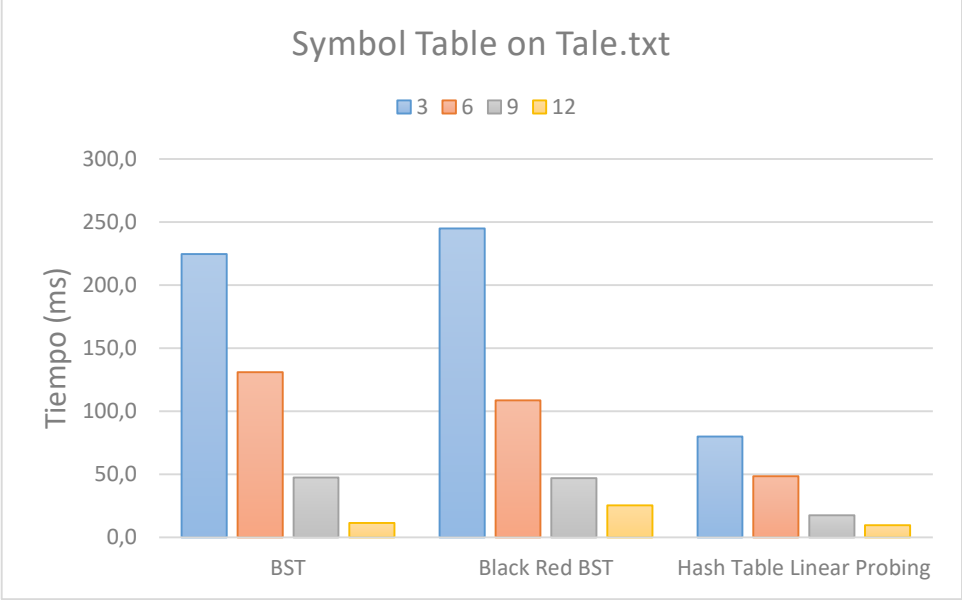
implementation	worst-case cost (after N inserts)			average case (after N random inserts)			ordered iteration?	key interface
	search	insert	delete	search hit	insert	delete		
BST	N	N	N	1.38 lg N	1.38 lg N	?	yes	compareTo()
red-black tree	2 lg N	2 lg N	2 lg N	1.00 lg N	1.00 lg N	1.00 lg N	yes	compareTo()

El factor que multiplica a log (N) es muy cercano en ambas implementaciones. Es por esto que los tiempos de ejecución son similares.

La diferencia que se observa en los tiempos de ejecución también es producto de la concurrencia.

Además, *la implementación Hash Table ofrece un mejor tiempo de ejecución para los distintos valores de longitud*, en comparación con las otras dos implementaciones.

Archivo Tale.txt

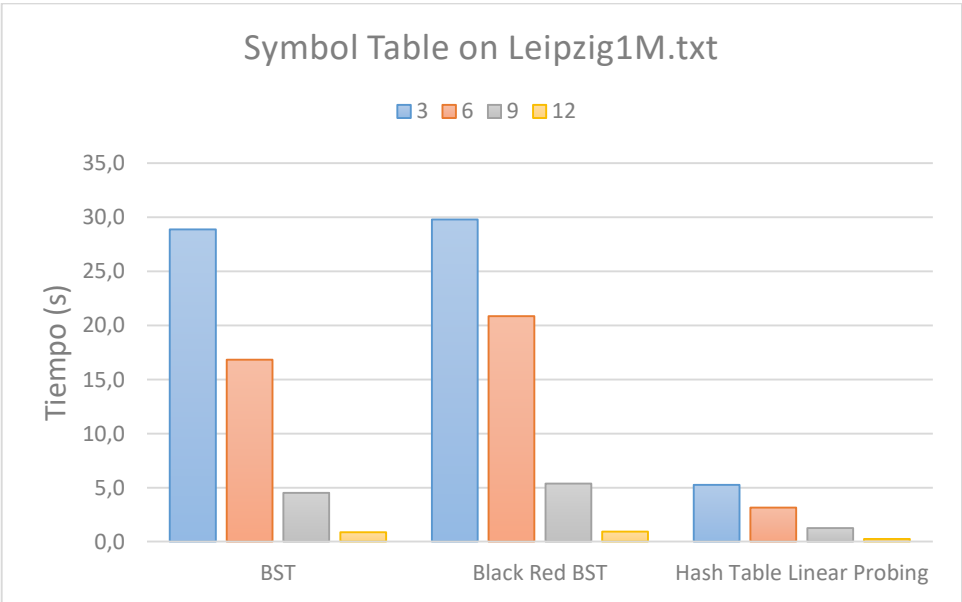


Aquí, Hash Table sigue siendo mejor que las otras implementaciones en el tiempo de ejecución y los tiempos de ejecución siguen sin superar un segundo.

Al igual que con el archivo anterior, BST y Black Red BST son similares en el tiempo de ejecución por lo explicado en el análisis del archivo anterior.

Cabe destacar que la tendencia también cambia con respecto a los valores de longitud de las palabras, esto es debido a la cantidad de palabras que contiene el archivo.

Archivo Leipzig1M.txt



Aquí los tiempos de ejecución entran al orden de los segundos. Las implementaciones de BST y Black Red BST siguen siendo similares entre si y la tendencia se mantiene

afectada por la longitud de los valores. Pero es importante observar que BST y Black Red BST pueden tardar hasta 30 segundos en la ejecución.

La implementación de Hash Table destaca aquí por su rapidez en la ejecución.

Observaciones

En caso de existir más de una palabra con un mismo número de frecuencias, el cliente Frequency Counter muestra el primero que se encontró.

Conclusiones

- Al usar el cliente Frequency Counter, con los archivos de texto de diferente cantidad de palabras (decenas, miles y millones) la implementación Hash Table con prueba lineal, siempre fue mejor en el tiempo de ejecución.
- Aunque Hash Table es mejor en el tiempo de ejecución, en comparación con BST y Black Red BST al usar el cliente Frequency Counter. La diferencia se aprecia cuando el número de palabras está en el orden de los millones. Esto quiere decir que, para decenas y miles de palabras, estaría bien usar cualquiera de las tres implementaciones.
- Los tiempos de ejecución de BST y Black Red BST , aunque ambos corren en tiempo log(N) , el factor en ambos es muy cercano. Y bajo ciertas circunstancias como el lenguaje de programación y la concurrencia, los tiempos pueden llegar a ser muy similares.

Referencias y Bibliografía

[1] https://en.wikipedia.org/wiki/Robert_Sedgewick_(computer_scientist)

[2] https://www.youtube.com/watch?time_continue=855&v=m4j7clgt5HQ

[3] https://en.wikipedia.org/wiki/Binary_search_tree

[4] https://en.wikipedia.org/wiki/Red%E2%80%93black_tree

[5] https://en.wikipedia.org/wiki/Hash_table