Tarea 5:

Hash v/s Búsqueda binaria

Comparaciones en eficiencia

Profesor: Benjamín Bustos.

Profesor Auxiliar: Boris Romero

Sebastián Quezada

Alumno: Jorge Gutiérrez

Fecha: 24 de Noviembre de 2015

Introducción

Uno de los mayores avances de la humanidad ha sido la creación de maquinas capaces de hacer tareas productivas de forma rápida y eficiente. Así la humanidad ha sido capaz de aumentar sus esperanzas de vida y la calidad de esta, utilizando estas maquinas, tanto para la fabricación de alimentos, ropa , autos, documentos, maquinas, la automatización de la producción de bienes.,etc. Por otro lado estas maquinas además de estar presente en la producción de bienes están participando activamente en el proceso de organización de la vida en la sociedad. Es fácil notar que la gente usa aparatos electrónicos comúnmente y casi como si fuera algo normal .

Dada esta capacidad , la cual esta en forma de capacidad de memoria de almacenamiento y de procesamiento, somos capaces de almacenar mucha información, el problema es como acceder a ella , es decir, simplemente buscarla para tener acceso.

Así es como el humano para poder ser capaz de resolver este problema mediante computadoras a creado distintas formas de abordarlo, llamadas algoritmos.

En este informe se probara una forma de (ordenar y) acceder a datos en memoria mediante el uso de dos técnicas y/o estructuras, la primera mediante una lista de datos y accediendo a ella mediante búsqueda binaria y la ultima mediante una tabla Hash.

Fe de Erratas: Para hacer el siguiente informe mas comprensible repetiremos las secciones de Diseño de la solución y la de Implementación para cada uno de los métodos nombrados anteriormente. Ademas usaremos la función COMPARA para la parte de la experimentación ya que esta compara los dos métodos anteriores

Diseño de la Solución para Arreglo

Implementación de Lista y Búsqueda Binaria para función SCORE_ARREGLO:

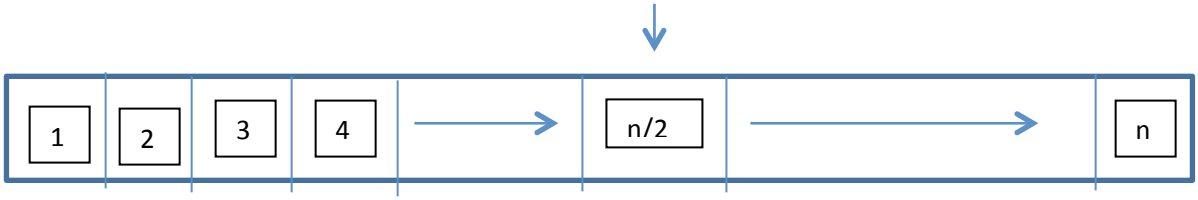
La función SCORE_ARREGLO que se pedia implementar tenia como objetivo entrgar la cantiad de palabras que existía de un set de palbras en otro. Para esto se simplifico esto a tomar una palabra del set de Strings y buscarla dentro del otro set de String y si era encontrada, entonces sumar uno a un contador.

Para entonces encontrar un String dentro de un set de Strings se procedio convirtiendo el set de palabras sobre el cual buscar en un arreglo de String nativo de Java (`String[]`))

Sobre esta estructura se aplico la búsqueda binaria. Este algoritmo se basa en buscar sobre un arreglo ordenado, para esto se ordeno la entrada usando un algoritmo parecido al “insertion sort” (Vease Anexo 1).

Este algoritmo (Busqueda binaria se puede ver en la siguiente sucesión de pasos:

1) Teniendo el arreglo ordenado desde palabras menores hasta las palabras mayores, comparamos el elemento del medio y lo comparamos con el valor buscado



Selección y comparación de elemento inicial en búsqueda binaria

2) Si este valor es mayor, entonces utilizamos este mismo método para buscar en la parte derecha del arreglo, la cual tiene a todos los elementos mayores a este, si por el contrario es menor, entonces, lo aplicamos al lado izquierdo. Por ultimo si este elemento es igual al elemento que estamos buscando entonces lo hemos encontrado.

Ejemplo de segundo paso (y paso iterativo) de búsqueda Binaria en caso de buscar en la parte derecha del arreglo

3) Así Aplicando esta búsqueda llegaremos a un ultimo elemento el cual debe ser igual a este si es que lo contiene y se aumenta el contador de elementos en uno.

Resultado final, asumiendo que la lista contenía el elemento

4) Por ultimo esto se repite este proceso para el siguiente String y se repite tantas veces como String haya que buscar .

Implementación para Arreglo

Ya conociendo el diseño de la solución planteada en el apartado anterior, entonces la implementación el lenguaje Java puede ser descrita de la siguiente manera:

```
public static int SCORE_ARREGLO(String D , String T){
          int contador =0; //Nuestro contador
             String[] list = sort(D.split("\\s+")); //Ordenando la
                                                              lista
           for (String s: T.split("\\s+")) { //Repetimos para cada
                                                          String S
                 if(indexOf(list,s)!=-1){ //Buscamos el elemento s
                                          (string en la lista ordenada)
                       contador+=1;       //Si es que es encontrada
                                            sumamos uno al contador
                 }
          }
          return contador;
     }
```

los detalles de la búsqueda binaria, representada por la función *sort* pueden ser vistos en el anexo

Diseño de la Solución para Hash

Implementación de Función Hash y Búsqueda para función SCORE_HASH:

La función SCORE_HASH, al igual que SCORE_ARREGLO ,pedia implementar tenia como objetivo entrgar la cantiad de palabras que existía de un set de palbras en otro. Para esto se simplifico esto a tomar una palabra del set de Strings y buscarla dentro del otro set de String y si era encontrada, entonces sumar uno a un contador.

Para entonces encontrar un String dentro de un set de Strings se procedio convirtiendo el set de palabras sobre el cual buscar en una tabla Hash, la cual para construirse se prosigue de la siguente forma:

1) Se crea una función Hash que nos convierta un String en un numero dentro de un rango el cual lo definimos de [0,n] donde n es la cantidad de palabras, esta función es iterativa y por cada letra, de cada palabra ,obtiene su numero ASCII y la suma a un total, luego a este total se aplica modulo de n.

2) Se inserta cada palabra en la posición que le indica la función hash, si ese lugar esta ocupado, entonces sigue avanzando hasta que encuentra un lugar desocupado (si avanza demasiado y llega a la ultima posición entonces sigue en el índice cero del arreglo).

3) Ahora para buscar una palabra desde el set de búsqueda solo necesitamos obtener el código Hash de la palabra y buscarla en el arreglo, siguiendo la mismas reglas de inserción, es decir si no es la palabra de la posición que nos entrega la función de hash entonces avanzamos.

Implementación para Hash

Ya conociendo el diseño de la solución planteada en el apartado anterior, entonces la implementación el lenguaje Java puede ser descrita de la siguiente manera:

```
public static int SCORE_HASH(String D, String T){
          int contador =0; //Contador
          String[] temp= D.split("\\s+"); //Generamos arreglo  con
                                                    las palabras
              String[] list = new String[temp.length]; //Generamos
                              nuevo arreglo para guardarlas según Hash
          for (String s:temp) list=insert(list,s); //Insertamos
cada palabra en su posición según su hash
          for (String s: T.split("\\s+")) {
                if(indexOf(list,s)!=-1) contador++; //Buscamos una
            palabra según su hash, si esta sumamos una al contador
          }
          return contador;
     }
```

Experimentación

La experimentación se hizo usando la función COMPARA que tiene como fin comparar el tiempo en que se demora cada una de estos dos métodos en calcular SCORE_ARREGLO y SCORE_HASH.
Así para la experimentación se utilizo un ejemplo disponible en el apartado “Material Alumnos” el cual contiene un diccionario de prueba y se probo utilizando un set de 10 palabras para buscar sobre este.

La implementación de Compara se puede ver a continuación

```
public static void COMPARA(String D,String T){
          long init=System.currentTimeMillis(); //Tiempo inicial
          long finish=0;
          long tiempo1 = 0;
          long tiempo2=0;
          String[] temp=D.split("\\s+"); //Preparacion del input
          String compare="";
                   for (String s:temp){ //iteración para crecer el
                        input,subdiccionarios del diccionario total
                compare=compare+s+" ";
                           Parte1.SCORE_ARREGLO(D, T); //Aplicamos
                                                    SCORE_ARREGLO
                 finish=System.currentTimeMillis(); // Tiempo final
                tiempo1=finish-init; //Calculamos intervalo
                   init=System.currentTimeMillis();//Tiempo inicial
                    Parte2.SCORE_HASH(D, T); //Aplicamos SCORE_HASH
                finish=System.currentTimeMillis();//Tiempo final
                tiempo2=finish-init; //Calculamos intervalo
     System.out.println(""+compare.split("\\s+").length+"\t"+tiemp
o1+"\t"+tiempo2);
          }
     }
```

Cabe decir que esta iteración solo se hizo hasta obtener 108 datos

Resultados y Conclusiones

Los resultados de la experimentación se pueden observar en el grafico del Anexo 5

A modo de conclusión y observando las ecuaciones que modelan los datos podemos ver que el tiempo de búsqueda dentro de una tabla hash es constante para una cantidad de datos sin importar su tamaño, es decir es O(1) casi siempre y la búsqueda para el uso de Array tiende a crecer mucho mas (casi 6 veces mas) que la del Hash, lo cual es de esperarse por que cada búsqueda tiene índices de comienzo distintos que en el caso de la búsqueda binaria es por azar y en la tabla Hash tiene una cercanía mucho mayor al elemento que se busca.

Así comprobamos la eficiencia de la tabla Hash para la búsqueda de elementos con una eficiencia Θ(1)

Anexos

1) Método Sort arreglo de String

```
public static String[] sort(String[] Array){
        String tempArray[]=Array;
       String  temp;
       for(int j=0; j<tempArray.length-1;j++){
           for (int i=j+1 ; i<tempArray.length; i++){
               if(tempArray[i].compareTo(tempArray[j])<0){
                   temp= tempArray[j];
                   tempArray[j]= tempArray[i];
                   tempArray[i]=temp;
              }
           }
        }
```

2) Funcion Hash

```
    private static int doHash(String str, int len) {
       int hash = 0;
       for (int i = 0; i < str.length(); i++) hash +=
str.charAt(i);
       return hash%len;
    }
```

3) Metodo de cracion de Tabla Hash

```
public static String[] insert (String[] list, String value){


          int key =doHash(value,list.length);
          for(int i=key;i<list.length;i++){
               if(list[i]==null || list[i].equals(value)) {
```

```
                    list[i]=value;
                    return list;
               }
          }
          for(int i=0;i<key;i++){
               if(list[i]==null || list[i].equals(value)) {
                    list[i]=value;
                    return list;
               }
          }
          return list;
     }
```

4) Metodo de Busqueda sobre tabla Hash

```
public static int indexOf(String[] a, String value) {
          int key=doHash(value,a.length);
          for(int i=key;i<a.length;i++){
               if(a[i].equals(value)) return i;
          }
          for(int i=0;i<key;i++){
               if(a[i].equals(value)) return i;
          }
        return -1;
         }
```

5)Grafico Resultado