**ESTRUCTURAS DE DATOS**

Ing. Román Martínez M

Semestre Agosto-Diciembre 2013

**TAREA #7**

Nombre: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Matrícula: \_\_\_\_\_\_\_\_

*Esta tarea deberá resolverse en estas mismas hojas que serán entregadas como solución a la misma.*

**PROBLEMA #1** *(20 puntos)*

Responde los siguientes casos:

1. Una tabla de hashing de 100 posiciones está completamente llena y todas las llaves han llegado a la tabla por medio de una función de hashing perfecta. ¿Cuál es el promedio de comparaciones para determinar que una llave existe en la tabla? \_\_\_\_\_\_
2. Si en una tabla de 10 posiciones se han insertado 3 llaves en la primera, segunda y décima posición. ¿Cuál es el promedio de comparaciones para determinar que una llave NO existe en la tabla, sabiendo que se usó la PRUEBA LINEAL como estrategia para manejar las colisiones? \_\_\_\_\_\_
3. Para una tabla de 'n' posiciones, que contiene 'n-1' llaves, ¿cuál sería el número máximo de comparaciones para encontrar una llave? \_\_\_\_\_\_\_\_ ¿cuál el mínimo? \_\_\_\_\_\_\_\_
4. Se tiene una tabla de hashing con 28 posiciones y se usó la función H(llave)=H(d1d2d3)= d1+d2+d3. Si se usa la PRUEBA CUADRÁTICA para manejar las colisiones, ¿cuáles serían las siguientes 3 posiciones de la tabla que se verificarían si se inserta la llave 678 y la posición 21 estaba ocupada? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**PROBLEMA #2** *(20 puntos)*

Sobre una tabla de hashing se aplica la siguiente función de hashing: H(llave) = H(d1d2d3d4d5d6)= d4d5 % 10, y se usa como estrategia de manejo de colisiones la PRUEBA LINEAL. Después de hacer algunas inserciones la tabla contiene las siguientes llaves:

0 ------

1 143012

2 198911

3 153926

4 152621

5 520851

6 ------

7 191972

8 335872

9 151090

10 190890

11 150588

12 232371

En base a la información de esta tabla, responde a cada una de las siguientes preguntas:

(NOTA: Considera sólo las comparaciones contra llaves y contra vacío).

1. ¿Cuáles son las llaves que se insertaron sin problemas de colisiones? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
2. ¿En qué orden se insertaron las llaves para las cuales la función dio el valor de 7? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
3. ¿Cuántas comparaciones se requieren en promedio para encontrar una llave de la tabla? \_\_\_\_\_\_
4. ¿Cuántas comparaciones se requieren en promedio para determinar que una llave NO existe en la tabla? \_\_\_\_\_\_
5. ¿Cuántas colisiones ocurrirían al insertar la llave 154889? \_\_\_\_\_\_\_
6. ¿Qué tipo de agrupamiento provoca que al insertarse la llave 123456 ésta colisione? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
7. Si se diera de baja de la tabla la llave 150588, ¿cuántas comparaciones serían necesarias para determinar que esta llave ya no existe en la tabla? \_\_\_\_\_\_\_\_
8. Si se diera de baja de la tabla la llave 153926, ¿cuántas comparaciones serían necesarias para encontrar a la llave 152621? \_\_\_\_\_\_
9. Si se diera de baja de la tabla la llave 335872, ¿en qué posición se guardaría la llave 432178? \_\_\_\_\_\_
10. Reescribe la función de hashing, de tal forma que sea más eficiente.

**PROBLEMA #3** *(20 puntos)*

Sobre una tabla de hashing de 14 posiciones se utiliza la función H1(llave) = llave % 10 y se manejan las colisiones con la estrategia DOBLE HASHING con una función H2(llave) = (llave % 3) + 1.

1. Muestra a través de un esquema, cuál es el status de la tabla después de insertar las siguientes llaves en el orden mostrado: 27, 12, 15, 7, 35, 40, 75, 82, 99.
2. ¿Cuál es el factor de carga en la tabla? \_\_\_\_\_\_
3. ¿Cuántas comparaciones se requieren para encontrar la llave 75? \_\_\_\_\_\_\_\_
4. ¿Cuántas comparaciones se requieren en promedio para determinar que una llave existe en la tabla? \_\_\_\_\_\_\_\_
5. ¿Cuántas comparaciones se requieren en promedio para determinar que una llave NO existe en la tabla, si se sabe que H1(llave) = 5? \_\_\_\_\_\_\_\_\_\_

**PROBLEMA #4** *(40 puntos) – Tomado del tercer examen parcial del semestre AD12.*

Anexo encontrarás un archivo que contiene el código de un programa que tiene implementada una Tabla de Hashing en donde se ha utilizado la estrategia de encadenamiento externo utilizando árboles binarios de búsqueda para almacenar los datos colisionados. Este programa carga los datos de un archivo texto en la Tabla, y posteriormente, despliega la información indicando qué datos se colocaron en cada posición, cuántos datos colisionaron en la posición y qué altura tiene el árbol de la posición correspondiente. Revisa el código con el detalle que necesites y pruébalo con el archivo de datos llamado prueba.txt.

Tu tarea en este problema consiste en responder las preguntas que se muestran a continuación. Para responder a estas preguntas sustentando tus respuestas, deberás programar las instrucciones que consideres convenientes y eficientes en el mismo programa que se te facilitó. Como resultado copiarás el segmento de código que hayas programado para encontrar la respuesta cuando así sea el caso, añadiendo la explicación necesaria que justifique tu respuesta. Adicionalmente, como evidencia de tu trabajo, copiarás también el código completo del programa con las modificaciones que hayas realizado.

PREGUNTAS DEL CASO:

1. *(5 puntos)* ¿Cuál es el factor de carga de la Tabla? ¿Cuál sería el factor de carga de la tabla si se hubiera utilizado la prueba lineal para el manejo de colisiones? *(Esta pregunta no requiere evidencia de programación, sólo explicación breve de tu respuesta)*
2. *(5 puntos)* ¿Cuál es el método que se utilizó para diseñar la función de hashing que se utiliza en el programa? Descríbela en función de los dígitos. *(Esta pregunta no requiere evidencia de programación, sólo explicación breve de tu respuesta)*
3. *(5 puntos)* ¿Cuántas comparaciones de datos se requieren para encontrar el dato **785958** en la Tabla?
4. *(5 puntos)* ¿Cuántas comparaciones se requieren para saber que el dato **143012** no existe en la Tabla?
5. *(5 puntos)* ¿Cuál es el tamaño promedio de los buckets de la Tabla después de haber insertado los datos del archivo (# promedio de datos en cada posición)? Considerar sólo los buckets que tienen datos.
6. *(5 puntos)* ¿Cuál es la altura promedio de los árboles binarios de búsqueda que se forman en la tabla después de haber insertado los datos del archivo? Considerar sólo los árboles que tienen datos.
7. *(5 puntos)* El dato que obtuviste en el punto anterior puede ser utilizado como un indicador de la eficiencia de la búsqueda en la Tabla. Cambia la función de hashing del programa para que ahora sea **H(d1d2d3d4d5d6) = (d5d6)2 % Tamaño de la tabla** e indica si con esta función se tiene mayor eficiencia en la tabla con respecto a la función original.
8. *(5 puntos)* Cambia el tamaño de la Tabla para que ahora sea de 500 posiciones. Prueba la carga de los datos con la función de hashing original y con la que implementaste en el punto anterior, e indica con cuál hay mayor eficiencia según el indicador del promedio de altura de los árboles.

// Código para el tercer examen parcial de Estructuras de Datos, semestre AD12, elaborado por RMM

#include <stdlib.h>

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

#define TAM 100

//Implementación del nivel físico de un ABB

template <class T>

class NodoArbol

{ public:

T info;

NodoArbol<T> \*izq, \*der;

NodoArbol() { izq = der = NULL; }

NodoArbol(T dato) { info = dato; izq = der = NULL; }

};

template <class T>

class ABB

{

private:

NodoArbol<T> \*raiz;

public:

ABB() { raiz = NULL; }

void inserta (T dato);

void despliega() { inorden(raiz); }

int cuenta () { return contar(raiz); }

int altura () { return niveles(raiz); }

~ABB() { libera(raiz); }

};

template <class T>

void libera (NodoArbol<T>\* raiz)

{

if (raiz != NULL)

{

libera(raiz->izq);

libera(raiz->der);

delete(raiz);

}

}

template <class T>

void inorden (NodoArbol<T>\* raiz)

{

if (raiz != NULL)

{

inorden(raiz->izq);

cout << raiz->info <<" ";

inorden(raiz->der);

}

}

template <class T>

int contar (NodoArbol<T>\* raiz)

{

if (raiz == NULL)

return 0;

return(1+contar(raiz->izq)+contar(raiz->der));

}

template <class T>

int niveles (NodoArbol<T>\* raiz)

{

int altizq, altder;

if (raiz == NULL)

return 0;

altizq = niveles(raiz->izq);

altder = niveles(raiz->der);

return(1+ (altizq>altder? altizq : altder));

}

template <class T>

void ABB<T>::inserta (T valor)

{

NodoArbol<T> \*NuevoNodo = new NodoArbol<T>(valor);

NodoArbol<T> \*actual = raiz, \*anterior = NULL;

while (actual != NULL)

{

anterior=actual;

actual=(actual->info>valor? actual->izq: actual->der);

}

if (anterior==NULL)

raiz = NuevoNodo;

else

if (anterior->info > valor)

anterior->izq = NuevoNodo;

else

anterior->der = NuevoNodo;

}

// Implementación del nivel físico de una Tabla de Hash

template <class T>

class tablaHash

{

private:

ABB<T> tabla[TAM];

int cantidad;

public:

tablaHash() {cantidad = 0;}

void inserta (T dato);

void despliega ();

};

template <class T>

void tablaHash<T>::inserta (T dato)

{

int posicion = dato / 100 % TAM;

tabla[posicion].inserta(dato);

cantidad++;

}

template <class T>

void tablaHash<T>::despliega()

{

cout << "Cantidad de datos: "<<cantidad<<"\n";

for (int i = 0; i < TAM; i++)

{

cout << "Posicion "<<i<<": "<<tabla[i].cuenta()<<" datos en "<<tabla[i].altura()<<" niveles ";

tabla[i].despliega();

cout << "\n";

}

}

// Implementación del programa de aplicación

int main()

{

tablaHash<int> hashing;

ifstream arch;

string nomarch;

int dato;

cout << "Ingrese el nombre del archivo a cargar: ";

cin >> nomarch;

arch.open(nomarch);

while (!arch.eof())

{

arch >> dato;

hashing.inserta(dato);

}

arch.close();

hashing.despliega();

cin>>nomarch;

return 0;

}

**PROBLEMA ADICIONAL PARA PUNTOS EXTRAS**

Como lo comentamos, hace algunos años me tocó diseñar este problema para el concurso regional de programación de la ACM. Al ser un caso interesante de representación de un árbol binario, he querido compartirlo para que intenten resolverlo.

Quien lo desee, puede usar el juez en línea para que se lo revise a través de este link: <https://icpcarchive.ecs.baylor.edu/index.php?option=com_onlinejudge&Itemid=8&page=show_problem&problem=273>

No caigas en la tentación de resolver el problema con ayuda. La idea de este ejercicio es que sea un reto que resuelvas en forma personal.

2001 ACM Programming Contest

Mexico and Central America Region

BST in a matrix

Files: bst.in, bst.out, bst.c, bst.cpp, ..

**Background**

The structure (or shape) of a Binary Search Tree (BST) can be represented in memory as follows:

The keys of the BST are d1, d2, d3, … dn, where d1 < d2 < d3 < … dn .

The matrix M, named the “matrix of roots” for the BST contains the following values: for every i <=j, M[i,j] contains the index of the key that is the root in the BST that is formed with the keys form di to dj. The rest of the values in the matrix M are zero, because you can’t construct a BST with that conditions.

The matrix of roots has rows from *1* to *n+1* and columns from *0* to *n*, to facilitate the construction of the BST if you needed.

The next, is an example of a BST and its matrix of roots:





# Input

You will be given a number of cases. The input for each case consists of one line with the order of the matrix of roots that represents the BST, following with the values of the matrix, one line per row, and each row consist of the values of each column (separated by a tab). All values are non-negative and you can assume that a BST contains at most 100 elements. A line in which contains the value of -1 indicates the end of input.

# Output

For each test case, print the case number followed by messages indicating the height of the BST and the average of comparisons to search the keys of the BST, in the form:

Case X:

The height of the BST is: Y

The average of the search in the BST is: Z

where X,Y and Z are replaced by the appropriate values. Z could be displayed with 2 decimals.

# Sample Input

10

0 1 2 3 3 5 5 5 5 5

0 0 2 3 3 5 5 5 5 5

0 0 0 3 3 5 5 5 5 5

0 0 0 0 4 5 5 5 5 5

0 0 0 0 0 5 5 5 5 5

0 0 0 0 0 0 6 7 7 9

0 0 0 0 0 0 0 7 7 9

0 0 0 0 0 0 0 0 8 9

0 0 0 0 0 0 0 0 0 9

0 0 0 0 0 0 0 0 0 0

-1

# Sample Output

Case 1:

The height of the BST is: 4

The average of the search in the BST is: 2.89