Informe del proyecto.

Algoritmos y Programación II

Prof.: José Padrón

SISTEMA DE INSCRIPCIONES EFICIENTE (SIE).

1) **Especificaciones funcionales:**

**Introducción:**

- El informe consta de descripción de las funciones usadas, parámetros asumidos, funciones adicionales, descripción de la estructura del proyecto, conclusiones.

El objetivo principal del proyecto fue que dos listas determinadas, en específico el Record Académico y las Materias funcionaran de manera adecuada y que cada acción sobre los mismos se generara correctamente, de manera que se pueda usar el proyecto para actualizar datos. Estas dos estructuras serian de tipo Árbol AVL, por ende en ambos registros se creó una variable FE(frecuencia de equilibrio) de tipo entero para que funcionara como contador de balance para los nodos de los árboles, como es la definición si su balance es -2 o 2 tiene que existir una rotación para balancear ambos árboles.

**Formulación del Proyecto:**

1.2) El programa consta con la capacidad de insertar y mostrar cada uno de los registros así como lo exige los requerimientos propuestos.

1.2.1) La función añadir de cada árbol consiste en la creación de una nueva estructura de tipo árbol (nodo), inserción de elementos para ese nodo, y finalmente se enlaza el nodo con los otros nodos de ese mismo tipo de estructuras.

1.2.2) La impresión en pantalla se basa en encontrar aquellos registros a imprimir y luego se utilizan funciones como el “printf” para mostrar en pantalla. Por ejemplo, para la consulta #1 se utilizó el procedimiento “Inorden” para mostrar el record académico de un alumno específico en orden cronológico en vista de que el árbol de record se ordena mediante el periodo de la materia cursada.

1.2.3) La estructura del proyecto consiste en crear 4 árboles: 2 AVL y 2 normales enlazados con semestre y alumno (tipo listas), donde los 2 normales se añaden el record y las materias de cada semestre donde se aplican las consultas y los 2 AVL se usaron para añadir también el record y las materias pero no están enlazados como requería el proyecto.

**Consideraciones Especiales:**

1.3) Para la elaboración del proyecto se asume que el usuario ingresara en aquellos campos la información correspondiente, es decir la introducción de valores como números en los campos que lo requieran. También se asumió que en la creación de registros el usuario no duplicara alguno de los mismos.

Existen procesos adicionales a la entrega de este proyecto que ya estaban realizados en el anterior proyecto.

**Conclusiones y Recomendaciones:**

1.4) El pro del diseño del proyecto, básicamente es que nos permite la creación e impresión de registros en pocas líneas de código; aparte de permitirnos hacer las consultas de manera más eficiente.

El contra del diseño es que los arboles AVL no se utilizan para las consultas, sino los normales.

**2)Especificaciones de programación:**

2.1) Diseño conceptual de las estructuras de datos

En el siguiente dibujo se expresa como se usaron los nodos, como fueron enlazados formando listas de cada tipo.

ESTRUCTURA

ARBOLES NORMALES ENLAZADOS CON LISTAS REQUERIDAS

ALUMNOS

SEMESTRE

MATERIA

RECORD

RECORD

MATERIA

MATERIA

RECORD

MATERIA

MATERIA

MATERIA

MATERIA

RECORD

MATERIA

ARBOLES AVL

RECORD

RECORD

MATERIA

MATERIA

RECORD

RECORD

RECORD

MATERIA

MATERIA

MATERIA

RECORD

RECORD

Como se puede contemplar en la imagen superior, se estructuró el proyecto mediante 4 árboles, en la primera imagen se realizó los arboles enlazados, el de record con los registros de alumnos tipo lista, los arboles de materia con los registros de semestres tipo lista, para las consultas e inserciones requeridas, y se hizo 2 árboles AVL no enlazados con ninguna estructura que almacenaban los records de los estudiantes y las materias de cada semestre, para demostrar las rotaciones, grados y niveles de ambos AVLs.

2.2) **Main:**

* Objetivo: Mostrar el menú principal del programa y convocación de los procesos requeridos por el usuario mediante la selección de opciones generadas por el programa.
* Descripción de los parámetros de entrada: se inicializa la variable op de tipo integer en -1(op hace referencia a la palabra opción), para que entre en el ciclo “while(op!=0)” este ciclo es el inicial y posee el menú del programa.
* Descripción de los parámetros de salida: mediante inserción del usuario la variable op será utilizada para elegir entre las distintas opciones, en pocas palabras op cambiara y si el usuario ingresa “0” saldrá del programa ya que interrumpe el ciclo inicial.
* Algoritmo:

Int opc=-1,while(op!=0), printf() de todas las opciones y decoraciones(Sistema de Inscripción de la UCAB por ejemplo), scanf() de la variable op ,hasta que op=0.

Agregar Nodo Principal de Arbol:

* Objetivo: Añadir uno de los nodos raíces al árbol para poder enlazar,
* Descripción de los parámetros de entrada: por referencia serian (semestre \*\*p) y (alumnos \*\*q) respectivamente en sus multilistas, estos punteros están inicializados en el Main, aparte se crea una variable auxiliar de tipo lista(referente al nodo) para los 2 procesos.
* Descripción de las salidas: nodo llenado con la información para cada estructura, enlazado a la lista ya existente.
* Algoritmo: lista \*aux=new lista, printf() de los datos a llenar, seguidos por scanf( , &aux->info) almacenando la información en el nodo, aux->sig=\*p/\*q (enlaza por cabeza el nodo a la estructura)y luego aux->aba=NULL (inicializando la lista que le procede), por ultimo (\*p/\*q)=aux.

Agregar Nodos Secundarios:

* **Objetivo**1: Añadir nodos (agregar record) en el primer árbol se le pide al alumno la C.I para verificar si ese alumno existe en el sistema. Después se le pedirá al alumno que añade el periodo académico para añadirse al árbol.
* **Objetivo2:** Añadir nodos (agregar materia) en el segundo árbol se le pide al alumno que ingrese su semestre para verificar si existe y pedirle que añade el código de la materia que desea registrar en ese semestre.
* **Objetivo3:** En los arboles AVL están inicializados en el main y en los llamados se pasan por parámetros (Insertar(&ArbolInt, n);) en el caso para los dos árboles avl, en el avl de materia se añade el código de la materia más su información de agregar semestre y número de créditos de la materia. Después se busca la altura en base de cómo se vaya armando el árbol y después se cuenta con el número de nodos y también se le pide al usuario que ingrese el código de la materia para saber la altura de ese nodo y su grado. Después se va a verificar si esta balanceado en el procedimiento de insertar para verificar que lado debe ubicarse el nodo si es a la derecha (mayores) o izquierda (menores). Pero a la vez de buscar que lado debe añadirse se verifica si el árbol padre existe para poder ubicar los nodos, y al final se equilibra con los procedimientos (1. RDD(rotación doble derecha) 2.RDI(rotación doble izquierda) 3.RSD(rotación simple derecha) 4.RSI(rotación simple izquierda)) donde esos procedimientos verifican la FE del grado de un nodo para saber su balanceo.
* **Objetivo4:** En los arboles AVL están inicializados en el main y en los llamados se pasan por parámetros (Insertar(&ArbolInt, n);) en el caso para los dos árboles avl, en el avl de records se añade el periodo académico más su información de agregar la cedula del alumno y el código de la materia y la nota obtenida de esa materia y la verificación de si APROBO o REPROBO . Después se busca la altura en base de cómo se vaya armando el árbol y después se cuenta con el número de nodos y también se le pide al usuario que ingrese el código de la materia para saber la altura de ese nodo y su grado. Después se va a verificar si esta balanceado en el procedimiento de insertar para verificar que lado debe ubicarse el nodo si es a la derecha (mayores) o izquierda (menores). Pero a la vez de buscar que lado debe añadirse se verifica si el árbol padre existe para poder ubicar los nodos, y al final se equilibra con los procedimientos (1. RDD(rotación doble derecha) 2.RDI(rotación doble izquierda) 3.RSD(rotación simple derecha) 4.RSI(rotación simple izquierda)) donde esos procedimientos verifican la FE del grado de un nodo para saber su balanceo.
* Descripción de los parámetros de entrada: por referencia serían los mismos de los Nodos Principales, se crea al encontrar el árbol deseada mediante el nodo padre y el numero que se añade.
* Descripción de las salidas: Cada nodo llenado con la información para cada estructura enlazado con el primer proyecto (enlazado para las consultas) siempre se verifica enlazado con el semestre y alumnos (estructuras) para acceder al árbol (normal) para agregar datos a esa estructura que se crea cada vez que el usuario agrega una a opción al árbol record o materia.
* Algoritmo: 1) AVL: En el insertar damos a verificar si el pNodo actual = \*a; existe ya que ese nodo será el dato que introduce el usuario, buscamos con un ciclo while(!Vacio(actual) && dat != actual->dato) si la condición del while se cumple; actual va a hacer el dato que se le asigna al padre padre = actual; y se compara si es menor o mayor que el dato que introduce el usuario; después verificamos si el actual existe if(!Vacio(actual)) Si existe retornamos (return); después verificamos si el padre existe if(Vacio(padre)) si existe creamos un nodo para de árbol de tipoNodo (\*a = (Arbol)malloc(sizeof(tipoNodo)) entonces la cajita contendrá el dato que introduce el usuario se lo asignamos y inicializamos izquierda y derecha (\*a)->izquierdo = (\*a)->derecho = NULL; inicializamos el padre también para que se pueda agregar otro dato actual->padre = padre; y la FE del nodo la inicializamos también pero equilibrar su grado, al hacer el proceso hacemos comparaciones de mayor o menor y lo equilibramos **EJEMPLO:**

MENOR:

else if(dat < padre->dato) {

actual = (Arbol)malloc(sizeof(tipoNodo));

padre->izquierdo = actual;

actual->dato = dat;

actual->izquierdo = actual->derecho = NULL;

actual->padre = padre;

actual->FE = 0;

Equilibrar(a, padre, IZQUIERDO, TRUE);

}

MAYOR:

else if(dat > padre->dato) {

actual = (Arbol)malloc(sizeof(tipoNodo));

padre->izquierdo = actual;

actual->dato = dat;

actual->izquierdo = actual->derecho = NULL;

actual->padre = padre;

actual->FE = 0;

Equilibrar(a, padre, IZQUIERDO, TRUE);

}

con el procedimiento de Equilibrar, este procedimiento recibe como parámetros el árbol y la variable tipo árbol que se crea en el procedimiento insertar y el enum de derecha o izquierda y un boolean, este proceso nos crea un boolean int salir = FALSE; que nos permite verificar en un ciclo si el nodo existen y el salir sea distinto de NULL. while(nodo && !salir) si se cumple el while verificamos si nuevo existe if(nuevo) si se cumple entonces preguntamos si a rama es izquierda, si se cumple podemos restarle a la FE, de lo contrario se suma, pero si ninguna operación se cumple preguntamos si nodo->FE es igual a 0, si se cumple la variable boolean le asignamos TRUE. Para otras condiciones las rotaciones es donde mas preguntamos después de verificar lo anterior, al preguntar si el grado del nodo es -2 if(nodo->FE == -2) si se cumple verificamos si if(nodo->izquierdo->FE == 1) si se cumple llamamos a la función RDD(a, nodo); donde le pasamos como dato “a” y árbol “nodo”; en la función RDD (rotacion doble derecha) creamos 6 punteros de tipo pNodo pNodo Padre = nodo->padre;

pNodo P = nodo;

pNodo Q = P->izquierdo;

pNodo R = Q->derecho;

pNodo B = R->izquierdo;

pNodo C = R->derecho;

en estos 6 punteros balanceamos el árbol en donde se encuentre el desbalanceo, preguntamos si el padre existe if(Padre), si existe entonces if(Padre->derecho == nodo) si se cumple esta condición el padre sub derecho será R, Padre->derecho = R; sino el izquierdo tomara R else Padre->izquierdo = R; si ninguna de las anteriores se cumple entonces la raíz tomara R \*raiz = R; después agarramos los punteros Q,P,R donde Q->derecho le asignamos a B y P->izquierdo le asignamos C y R->izquierdo le asignamos Q Y R->derecho le asignamos P; después de este proceso R->padre le asignamos al padre del árbol y P->padre le asignamos Q->padre, y a Q->padre le asignamos R; si B existe le asignamos a B su padre B-padre = Q; lo mismo para C si existe le asignamos if(C) C->padre = P; después de estas operaciones hacemos un case donde verifica que dato nos da R->FE, si ese dato es un numero de case en las opciones dadas (-1,0,1) entonces para opción -1 será : Q->FE = 0; P->FE = 1; break; y para la opción 0 será: Q->FE = 0; P->FE = 0; break; y para la opción 1 será: Q->FE = -1; P->FE = 0; break; al salir de las opciones la raíz sub FE será 0, R->FE = 0; esto mismo ocurre para la rotación RDI (rotación doble izquierda).

Al terminar el procedimiento de RDD volvemos al void Equilibrar, en el anterior caso habíamos preguntado si if(nodo->izquierdo->FE == 1) si ese caso no se cumplía entonces nos íbamos al void de RSD(rotación simple derecha) en este proceso es simple pero igual le pasamos la raíz y el nodo RSD(a, nodo); en este caso creamos 4 punteros de tipo pNodo, el mismo caso para el puntero padre que sera pNodo Padre = nodo->padre; para las demás variables P,Q,B de tipo pNodo donde P será nodo y Q se le asignara P->izquierdo y B se le asignara Q->derecho; después de ese proceso verificamos si el padre existe y si se cumple preguntamos si if(Padre->derecho == P) de esta manera si se cumple esta condición a Padre->derecho = Q; de lo contrario Padre->izquierdo = Q; y a la raíz la inicializamos \*a = Q;, después le asignamos P->izquierdo = B y Q->derecho = P; después de esto le asignamos p->padre = Q; y preguntamos si B existe para asignarle B->padre = P; lo mismo para Q->padre = padre; al finalizar las condiciones para los punteros de ir manejando su grado de nodo de cada uno de los datos inicializamos la FE de P->FE = 0; y Q->FE = 0; esto mismo ocurre para RSI(rotacion simple izquierda). Al regresar al procedimiento Equilibrar le asignamos TRUE al boolean salir y preguntamos si (nodo->padre) si existe este nodo entonces preguntamos al árbol del nodo sub padre sub derecho si es igual al dato que introdujo el usuario if(nodo->padre->derecho == nodo) si esta operación se cumple la rama le asignaremos DERECHA en caso contrario le asignamos IZQUIERDA y el nodo lo inicializamos en su padre para que sea árbol avl nodo = nodo->padre.

Aquí es el fin del algoritmo del AVL.

*Realizado por:*

Iván Moreno

Manuel Higueras

Marco Lozano.