Luis Fernando Muñoz A00046396

Sebastian Arango A00130532

Santiago del Campo A00137608

**Fase 1: Identificación del problema**

**Síntomas y necesidades**

**-**Almacenar grupos de datos de un mismo tipo.

-Es mejor tener los datos ordenados.

-La búsqueda de datos se facilita al tenerlos almacenados en un lugar específico.

-Una compañía necesita un software para analizar datos.

-La compañía maneja grupos de datos muy grandes que deben ser ordenados para su entendimiento.

**Definición del problema**

Una compañía encargada del análisis de datos, ha solicitado el desarrollo de una base de datos para manejar grandes cantidades de información, la búsqueda en esta base de datos tiene que ser muy eficiente pues se manejan millones de datos.

**Listado de requerimientos**

**Requerimientos funcionales**

|  |  |
| --- | --- |
| Nombre | R1 – Realizar consulta de registros sobre la información. |
| Resumen | Realiza la búsqueda de información específica de manera eficiente. |
| Entradas | |
| Dato que se quiere encontrar en la tabla. | |
| Resultados | |
| Toda la información asociada al dato que se busco. | |

|  |  |
| --- | --- |
| Nombre | R2 –Cargar información para generar la base de datos. |
| Resumen | Genera la tabla que contiene todos los datos. |
| Entradas | |
| Documento que contiene un grupo de información. | |
| Resultados | |
| Base de datos en forma de tabla que contiene toda la información ingresada. | |

**Requerimientos no funcionales**

-Para tres columnas de la tabla la búsqueda debe ser muy eficiente.

**Fase 2: Recopilación de información**

**Definiciones**

**Base de datos:** Un conjunto de datos pertenecientes a un mismo contexto y almacenados sistemáticamente para su posterior uso.

**Arboles AVL:** Es un árbol binario de búsqueda que intenta mantener su altura, o el número de niveles de nodos bajo la raíz, tan pequeños como sea posible en todo momento, automáticamente.

Siempre equilibrados de tal modo que para todos los nodos, la altura de la rama izquierda no difiere en más de una unidad de la altura de la rama derecha o viceversa. Gracias a esta forma de equilibrio (o balanceo), la complejidad de una búsqueda en uno de estos árboles se mantiene siempre en orden de complejidad O(log n).

**Árboles Rojinegros:**  Son un tipo de árbol balanceado que a cada nodo le agregaron el atributo de un color y siguen las siguientes características.

1)Todo nodo es o bien rojo o bien negro.

2)La raíz es negra.

3)Todas las hojas (NULL) son negras.

4)Todo nodo rojo debe tener dos nodos hijos negros.

5)Cada camino desde un nodo dado a sus hojas descendientes contiene el mismo número de nodos negros.

**Fase 3: Búsqueda de soluciones creativas**

(1)

En la lluvia de ideas se propone guardar toda la información en una matriz de tamaño MxN, con el fin de tener un orden en los datos.

(2)

El grupo cree que una buena manera para manejar esta información es creando una lista enlazada que contenga N objetos con M atributos, por ejemplo:

|  |
| --- |
| Estudiante |
| -Codigo:String  -Nombre:String  -Saldo:Double |
| ... |

(3)

En esta solución se crea la tabla en la cual van a haber tres Campos representados por árboles de búsqueda ,donde la key va a ser El valor del campo y el value el índice del registro que le corresponde dicho valor

Como se puede observar, si existen varios registros con el mismo campo, el nodo apuntará a una lista enlazada de nodos con valor (campo, registro)

**Fase 4: Transición de la formulación de ideas a los diseños preliminares**

(1)

-Permite un orden muy evidente e intuitivo en los datos proporcionados.

-Cada “Objeto” ocupa una fila determinada.

(2)

-Guarda los datos de manera muy ordenada.

-Al encontrar un objeto se puede acceder en tiempo constante al resto de su información.

(3)

-La búsqueda es rápida

-la información está ordenada

-El acceso a cada posición es fácil.

**Fase 5: Evaluación y selección de la mejor solución**

**Criterios de evaluación**

­ **Criterio A.** La solución permite acceder a los datos en un tiempo:

[2] <O(n)

­ [1] >O(n)

­**Criterio B.** El modelo de la base de datos en la solución es:

­ [2] Intuitivo y de fácil entendimiento para los programadores

­ [1] Rebuscado y tedioso de codificar para los programadores

**Criterio C.** El nivel de aprendizaje por implementar la solución es:

[3]Alto

[2]Intermedio

[1]Bajo

**Criterio D.** La solución permite agregar datos nuevos dinámicamente:

[2] si

[1]no

**Criterio E.** es una estructura en la cual naturalmente se aplique la recursión

[2]si

[1]no

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Soluciones | Criterio A | Criterio B | Criterio C | Criterio D | Criterio E | Sumatoria |
| (1) | 1 | 2 | 2 | 1 | 1 | 7 |
| (2) | 1 | 2 | 2 | 2 | 1 | 8 |
| (3) | 2 | 2 | 3 | 2 | 2 | 11 |

**Fase 6: Preparación de informes y especificaciones**

**Definición de los TAD**

|  |
| --- |
| **TAD AVL tree** |
| **Representación: X,L,r donde L y r son sub-arboles de x, donde la altura de r menos la altura de L no difiere en más de 1.** |
| **Invariantes:**  **|Factor de balanceo|<=1** |
| **Operaciones:**  **CrearArbolAVL ----> AVL tree**  **Eliminar Key ----> booleano**  **Insertar AVLNode ----> void** |

|  |
| --- |
| **CrearAVLTree()** |
| **Pre: true** |
| **Post: Se crea el arbol.** |

|  |
| --- |
| **Eliminar(K key)** |
| **Pre: true** |
| **Post: si se encuentra key, se eliminan los nodos con todos los valores asociados a ella, luego, se balancea completamente. Retorna true si fue eliminada** |

|  |
| --- |
| **Insertar(AVLNode x)** |
| **Pre: x diferente de null** |
| **Post: existe el nodo en el árbol y se balancea completamente** |

|  |
| --- |
| **TAD RedBlackTree** |
| **Representación: X,L,r donde L y r son sub-arboles de x.** |
| **Invariantes:**  1)Todo nodo es o bien rojo o bien negro.  2)La raíz es negra.  3)Todas las hojas (NIL) son negras.  4)Todo nodo rojo debe tener dos nodos hijos negros.  5)Cada camino desde un nodo dado a sus hojas descendientes contiene el mismo número de nodos negros. |
| **Operaciones:**  **CrearRedBlackTree ----> RedBlackTree**  **Eliminar Key ----> booleano**  **Insertar RedblackNode ----> Void** |

|  |
| --- |
| **CrearRedBlackTree()** |
| **Pre: true** |
| **Post: se crea un arbol** |

|  |
| --- |
| **Eliminar(K key)** |
| **Pre: true** |
| **Post: si se encuentra key, se eliminan los nodos con todos los valores asociados a ella, luego, se balancea suficientemente, retorna true si fue eliminada** |

|  |
| --- |
| **Insertar(RBTNode x)** |
| **Pre: x diferente de null** |
| **Post: el nodo existe en el árbol y queda suficientemente balanceado** |

|  |
| --- |
| **TAD ABBTree** |
| **Representación:**  **X,L,r donde L y r son sub-arboles de x.** |
| **Invariantes:**  **Subarbol izquierdo de x contiene valores menores que x y el subarbol derecho contiene valores mayores x.** |
| **Operaciones:**  **CrearABBTree ----> ABBTree**  **Consultar Key ---->ABBNode**  **RotarIzquierda ABBNode ----> void**  **RotarDerecha ABBNode ----> void**  **Insertar Key,Value ------>ABBTree**  **eliminar Key ------>boolean** |

|  |
| --- |
| **CrearABBTree()** |
| **Pre: true** |
| **Post: el arbol ha sido creado** |

|  |
| --- |
| **Insertar(Key k,Value v)** |
| **Pre: true** |
| **Post: existe en el árbol nodo con llave k y valor v** |

|  |
| --- |
| **Consultar(K key)** |
| **Pre: true** |
| **Post: retorna null si no encuentra el nodo, y retorna el nodo si coincide con key dada (pueden haber varios clones con diferentes values)** |

|  |
| --- |
| **RotarIzquierda(ABBNode x)** |
| **Pre: x diferente de null y el hijo derecho de x diferente de null** |
| **Post: x rotado a la izquierda con su hijo de la derecha como padre** |

|  |
| --- |
| **RotarDerecha(ABBNode x)** |
| **Pre: x y su hijo izquierdo diferente de null** |
| **Post: x rotado a la izquierda con su hijo de la derecha como padre** |

|  |
| --- |
| **eliminar(K key)** |
| **Pre: true** |
| **Post: si se encuentra, elimina los nodos de key con todos sus valores asociados a ella, retorna true si fue eliminada, false en caso contrario** |

**Diseño de pruebas**

**AVL TREE**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Clase | Método | Escenario | Entrada | Resultados |
| TestAVLTree | insertar() | escenario1() | ("a",3)  (“HOLA”,2) | Agregado, se mantienen las propiedades. |
| TestAVLTree | insertar() | escenario1() | (null,null) | No se añade nada |
| TestAVLTree | insertar() | escenario1() | (“hhhhhhhhhhhhhhhhhhhhhhffdssassdfgjmnbvgfr”,240245904 ) | Agregado, se mantienen las propiedades de balanceo. |
| TestAVLTree | eliminar() | escenario2()  Árbol con 15 elementos | “h” | Se elimina y se mantienen las propiedades de balanceo. |
| TestAVLTree | eliminar() | escenario2()  Árbol con 15 elementos | "b" | Se elimina y se mantienen las propiedades de balanceo. |
| TestAVLTree | eliminar() | escenario2()  Árbol con 15 elementos | “a” | Se elimina y se mantienen las propiedades de balanceo. |
| TestAVLTree | eliminar() | escenario2()  Árbol con 15 elementos | “m” | Se elimina y se mantienen las propiedades de balanceo. |
| TestAVLTree | eliminar() | escenario2()  Árbol con 15 elementos | null | No se modifica nada en el árbol, se retorna nulo. |
| TestAVLTree | consultar() | escenario2()  Árbol con 15 elementos | “e” | Retorna el nodo con valor 75 |
| TestAVLTree | consultar() | escenario2()  Árbol con 15 elementos | “l” | Retorna el nodo con valor 489 |
| TestAVLTree | consultar() | escenario2()  Árbol con 15 elementos | null | Retorna null |
| TestAVLTree | consultar() | escenario2()  Árbol con 15 elementos | “hlbb” | Retorna null ya que no se encuentra en el arbol |

**REDBLACK TREE**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Clase | Método | Escenario | Entrada | Resultados |
| TestRedBlackTree | insertar() | escenario1() | (5,3) | Agregado, se mantienen las propiedades de balanceo. |
| TestRedBlackTree | insertar() | escenario1() | (6,2) | Agregado, se mantienen las propiedades de balanceo. |
| TestRedBlackTree | insertar() | escenario1() | (125698745,254698752) | Agregado, se mantienen las propiedades de balanceo. |
| TestRedBlackTree | insertar() | escenario1() | (null,null) | No se modifica nada del arbol |
| TestRedBlackTree | eliminar() | escenario2()  Arbol de 8 elementos | 2 | Se elimina y se mantienen las propiedades |
| TestRedBlackTree | eliminar() | escenario2()  Arbol de 8 elementos | 456 | El elemento no se encuentra en el árbol, retorna nulo |
| TestRedBlackTree | eliminar() | escenario2()  Arbol de 8 elementos | 1 | Se elimina y se mantienen las propiedades |
| TestRedBlackTree | eliminar() | escenario2()  Arbol de 8 elementos | null | Retorna nulo, no se modifica el arbol |
| TestRedBlackTree | consultar() | escenario2()  Arbol de 8 elementos | 546 | El elemento no se encuentra en el árbol, retorna nulo |
| TestRedBlackTree | consultar() | escenario2()  Arbol de 8 elementos | 3 | Elemento encontrado con valor de 3 |
| TestRedBlackTree | consultar() | escenario2()  Arbol de 8 elementos | 2 | Elemento encontrado con valor de 2 |
| TestRedBlackTree | consultar() | escenario2()  Arbol de 8 elementos | null | Retorna nulo |

**DBMS**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Clase | Método | Escenario | Entrada | Resultados |
| TestDBMS | cargarDatos() | escenario1() | ./DatosEnCSV/EjecucionPresupuestoImpuestosAlcaldiaVistaHermosa.csv | Datos cargados registros creados |
| TestDBMS | consultarDatos() | escenario1() con registros de ./DatosEnCSV/EjecucionPresupuestoImpuestosAlcaldiaVistaHermosa.csv | (1,"$31650.000000000000000000000000000000000")  (1,"INGRESOS ADMINISTRACION CENTRAL")  (3,"$2825005520.00") | Se han encontrado todos los registros sin problema |

Bibliografia

<http://ingecomp.mforos.com/674005/3075255-arboles-avl-2-3-y-splay/>

<https://web.archive.org/web/20050801080205/http://webpages.ull.es/users/jriera/Docencia/AVL/AVL%20tree%20applet.htm>

<https://web.archive.org/web/20070926220746/http://www.eecs.usma.edu/webs/people/okasaki/jfp99.ps>