Voto Electrónico

Etapa 1

Organización de Datos – Cátedra Servetto 2do cuatrimestre 2011

21/10/2011

Alfredo Scoppa	89149
Cecilia khalil	87996
Fernando Romera Ferrio	88406
Juan Manuel Romera Ferrio	88405

Tabla de contenidos

Organización de registros

Los registros tienen métodos para leerse y escribirse en un bloque en memoria, no utilizan un delimitador o indicador de longitud sino indicadores de longitud de los campos variables los cuales se almacenan en un byte antes del campo variable. Para leerse y escribirse reciben un puntero a la posición de memoria donde comienza y luego de escribirse o leerse deja el puntero en el byte siguiente al fin del registro. Todos los registros tienen una clave que se usa para comparar con otros registros, las claves son clases anidadas a los registros. Todos los registros implementan métodos de comparación al igual que las claves. Las claves tienen un comportamiento similar a los registros ya que tienen métodos para leerse y grabarse en un bloque de memoria e implementan al igual que los registros un método size() que devuelve el tamaño que ocupa al escribirse.

Las organizaciones de archivo implementadas tienen un “template” con el tipo de registro que maneja, por lo que todos los registros deben respetar ciertas firmas de métodos, entre ellas read(),write(),size(),getKey(),etc.

Organización Balanceada: Árbol B+

Hojas (Sequence Set)

Las hojas almacenan los registros, estos pueden ser de tamaño variable o tamaño fijo. Además de los registros almacenan tres campos de control: el espacio libre, el nivel y el nodo siguiente. El espacio libre se almacena como un entero sin signo de 2 bytes, permitiendo un tamaño máximo de hoja de 64KiB, y representa la cantidad de bytes libres en la hoja, los cuales estarán compactados al final de la hoja. El nivel guarda el número cero en 1 byte, sirve para diferenciar los nodos internos de las hojas. El nodo siguiente es un campo de 4 bytes que almacena el número de bloque de la hoja siguiente. Las hojas almacenan una lista en memoria con los registros que contiene y cada registro “sabe” como escribirse y leerse en el bloque de la hoja, por lo que no se necesitan campos que delimitan la longitud del registro. Los registros de tamaño fijo son tratados de igual manera que los registros variables.

Cuando se inserta un nuevo registro se busca secuencialmente en la lista de registros en memoria la posición donde se insertará el registro, en una inserción la hoja no se escribe en el archivo, sino que se deja esta responsabilidad al ámbito superior (nodo padre) para ahorrar escrituras. La inserción puede fallar por varios motivos. Si se intenta insertar un registro cuyo tamaño es superior al tamaño de bloque se lanza un error, si el registro ya se encuentra en la hoja se lanza otro error y finalmente si el espacio libre es menor que el tamaño del registro se lanza “overflow” , el cual será manejado desde el ámbito superior.

El overflow de la hoja es manejado por el padre de la siguiente manera. Se crea una nueva hoja en el archivo y se distribuyen los registros, incluyendo al que se quiere insertar, dejando a la hoja no nueva con la cantidad de registros necesaria para superar la mitad de su capacidad y el resto de los registros se ubican en la hoja nueva. Los registros de mayor valor se colocarán en la hoja nueva. Luego se corrigen los valores de nodo siguiente y finalmente se inserta en el nodo padre la clave del primer registro de la hoja nueva.

En el borrado de un registro primero se busca el registro en la lista al igual que en la inserción y si no se encuentra se lanza un error. Una vez encontrado se analiza el espacio libre, si este antes del borrado es mayor que la mitad de la capacidad de la hoja en bytes, se borra el registro aumentando el espacio libre y se lanza “underflow”. En caso contrario se borra el registro aumentando el espacio libre pero al igual que en la inserción la hoja no se escribe en disco.

El “underflow” es manejado por el padre del siguiente modo. Si la hoja tiene hermano derecho se lo lee, si no se lee el izquierdo. Se fija si el hermano tiene capacidad mínima, es decir, si su espacio libre es mayor que la mitad de la capacidad de la hoja. En caso positivo se fusionan las hojas

liberando la hoja de la derecha luego de actualizar el siguiente nodo de la hermana y guardando todos los registros en la hoja de la izquierda. Como se liberó la hoja de la derecha se borra la clave que la indexa en el nodo padre. Si el hermano no tiene capacidad mínima se balancean los registros de manera que la diferencia de espacio libre entre las hojas sea mínima, conservando el ordenamiento de los registros. Por último se actualiza la clave que indexa a la hoja derecha.

Nodos Internos (Index Set)

Los nodos internos almacenan las claves o identificadores de los registros y el número de bloque de los nodos hijos. Cada clave tiene un hijo derecho y uno izquierdo, de manera que el hijo derecho de una es el izquierdo de la siguiente. Para almacenar los números de bloque de los hijos se usaron 4 bytes y se almacenan en el archivo intercalado con las claves. Las claves pueden ser de longitud fija o variable y se delega la escritura y lectura en el bloque al comportamiento de la clase que las representa, por lo tanto no hizo falta agregar un delimitador en el archivo. Además de claves e hijos los nodos internos guardan campos de control, los cuales son el espacio libre y el nivel. El espacio libre al igual que en las hojas representa la cantidad de bytes libres del nodo y se almacena en 2 bytes, el nivel representa la altura del nodo, es decir, cuantos nodos tengo que leer para llegar a una hoja, se almacena en 1 byte permitiendo hasta 255 niveles.

La inserción y borrado de claves e hijos se da cuando se produce un overflow o underflow en algún nodo hijo. Como se mencionó anteriormente cuando una hoja produce overflow se debe insertar una nueva clave junto con su hijo derecho. Si no hay suficiente espacio libre el nodo interno lanza “overflow” y le pasa al padre la clave e hijo que no se pudieron insertar, es responsabilidad del padre manejar el overflow.

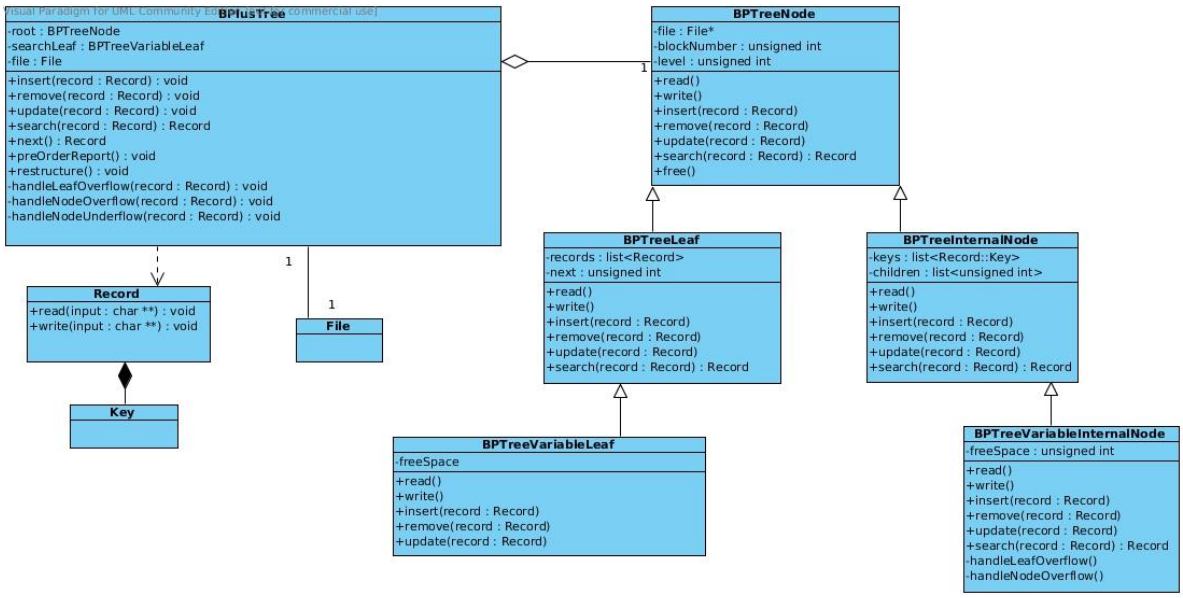
Un nodo interno soluciona el overflow de su hijo nodo interno de esta manera. Obtiene la lista de hijos y de claves del hijo, agrega a las listas la clave e hijo que se quiso insertar. Luego deja al nodo hijo con las claves e hijos necesaria para superar la mitad de la capacidad, la siguiente clave la reserva y el resto de las claves e hijos los inserta en un nodo interno nuevo que será hermano derecho del primero. Finalmente inserta la clave reservada con el nodo nuevo como hijo derecho.

Cuando un nodo interno intenta borrar una clave al igual que las hojas se fija si el espacio libre es mayor a la mitad de la capacidad, en caso positivo lanza un underflow luego de borrar la clave y su hijo derecho. En otro caso simplemente borra la clave y su hijo derecho aumentando el espacio libre.

Un nodo interno maneja el underflow de su hijo nodo interno de la siguiente manera. Leo el hermano derecho o si no tiene al izquierdo y le pregunta si tiene capacidad mínima. En caso

afirmativo toma las claves e hijos de los dos hijos le agrega la clave padre en el medio y fusiona todas estas claves e hijos en el nodo que produjo underflow, se libera el hermano y se remueve su referencia en el nodo padre. Si el hermano no tenía capacidad mínima se juntan todas las claves e hijos de los hijos junto con la clave padre de ambos y se distribuyen las claves e hijos de forma ordenada entre los dos nodos hijos haciendo que la diferencia de espacio libre entre ambos sea mínima, por último la clave que quedo en el medio se coloca como la nueva clave padre.

Diagrama de clases



Organización Directa

Se implementó dispersión estática. Los desbordes se resuelven usando direccionamiento abierto y doble dispersión. Se usa una función para obtener la dirección base y otra para dar los saltos en caso de desborde.

Funciones

Función principal: Resto de División o Módulo. La clave se divide entre el número de direcciones. Para asegurarnos una dispersión eficiente nos aseguramos que el tamaño de la tabla sea un número primo.

Función secundaria: También es Resto de División o Módulo solo que esta función es alimentada con el resultado de una función de hashing previa.

Se pensó el hash para trabajar con registros de tamaño variable que se almacenaran en cubetas con capacidad para varios registros.

Cubetas

Estructura de una cubeta:

Overflow (1 byte)	FreeSpace (2 bytes)	Count (2 bytes)	bytes

El tamaño de las cubetas es parametrizable como también el tipo de dato que se almacena en la tabla. De acuerdo al tipo de registro a insertar se tomaron distintos múltiplos de 512 para determinar el tamaño de la cubeta.

Para determinar la cantidad óptima de cubetas en diferentes escenarios se consideraron diferentes parámetros:

C: número de registros por cubeta

R: número total de registros almacenados

DE: Densidad de empaquetamiento la cual se fijó en 0.7

C y R son parámetros estimativos que deben determinarse conociendo los registros a insertar.

Así se despeja el numero óptimo de cubetas D, como D = R/(C*DE).

Finalmente se verifica que D sea primo y en caso contrario se toma el primo más cercano.

Primitivas implementadas

Se implementaron las siguientes primitivas:

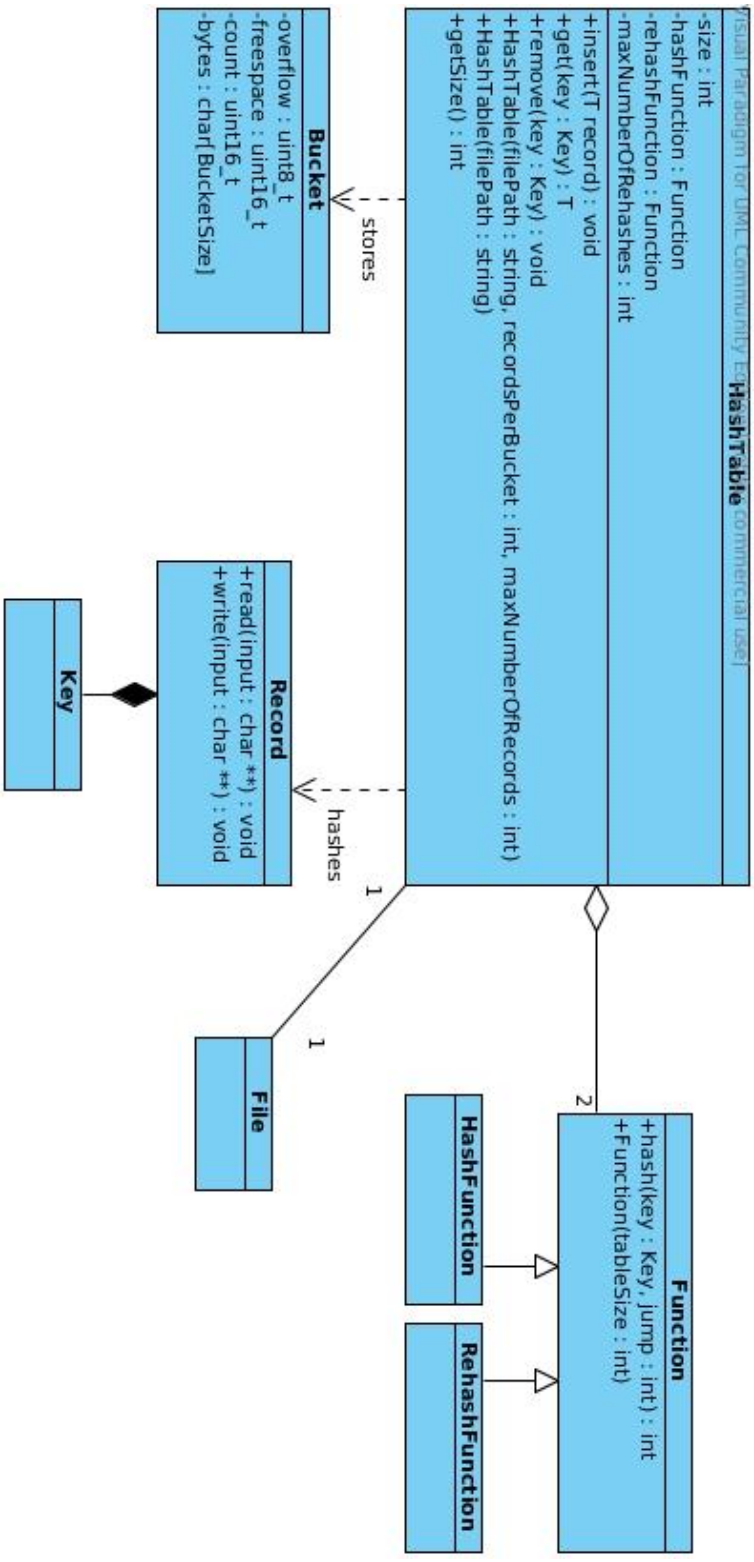
- Creación: Se crea un nuevo archivo y se calcula la cantidad de cubetas necesarias.
- Carga: Se abre un archivo y se determina la cantidad de cubetas existentes.
- Inserción: Se intenta insertar un registro y lanza una excepción en caso de que ya exista.
- Recuperación: Se intenta recuperar un registro y lanza una excepción en caso de que no exista.
- Actualización: Se intenta actualizar un registro y lanza una excepción en caso de que no exista.
- Borrado: Se intenta borrar un registro y lanza una excepción en caso de que no exista.

Archivos

Se utiliza un solo archivo que es el que contiene la totalidad de las cubetas.

Diagrama de clases

A continuación se presenta un diagrama de clases resumido de las clases involucradas en esta organización.


Visual Paradigm for UML Community Edition [commercial use]
HashTable
-size : int
-hashFunction : Function
-rehashFunction : Function
-maxNumberOfRehashes : int
+insert(T record) : void
+get(key : Key) : T
+remove(key : Key) : void
+HashTable(filePath : string, recordsPerBucket : int, maxNumberOfRecords : int)
+HashTable(filePath : string)
+getSize() : int
stores
Bucket
-overflow : uint8_t
-freespace : uint16_t
-count : uint16_t
-bytes : char[BucketSize]
hashes
Record
+read(input : char **) : void
+write(input : char **) : void
Key
File
1
1
2
Function
+hash(key : Key, jump : int) : int
+Function(tableSize : int)
HashFunction
RehashFunction

Manual de Usuario

Ver documento adjunto: ManualDeUsuario.pdf