**Universidad de Buenos Aires**

**Facultad de Ingeniería**

**Departamento de Informática**

***Organización de Datos (75.06)***

**Voto Electrónico**

Cuatrimestre y año: 2do Cuatrimestre 2011

Docente a cargo del TP:Nicolás Pablo Fernández Theillet

Grupo: Lamas

Fecha de Entrega:

Integrantes: 2011-10-22

|  |  |  |
| --- | --- | --- |
| ***Padrón*** | ***Nombre*** | ***Email*** |
| 91187 | Gonzalez Durand, Juan Manuel | jmanuel.gonzalez.durand@gmail.com |
| 90762 | Ostrowsky, Gabriel | gaby.osto@gmail.com |
| 90728 | Schenkelman, Damián | damian.schenkelman@gmail.com |
| 91045 | Torrado, Alejandro | aletorrado@gmail.com |
| 90884 | Zamudio, Gonzalo | ahogadosderazon@gmail.com |

Diseño

A continuación se detallan las decisiones de diseño tomadas para la organización de los archivos de las diferentes entidades del problema. Para cada una de ellas se detalla:

* La organización que se utilizara para guardar los datos de relacionados con la misma.
* Las razones por la que se decidió dicha organización.
* Características particulares dentro de la organización elegida (ej: función de dispersión para el Hash).
* Estructura de los registros utilizados (tanto en índices como para guardar datos).
* Estructura de los bloques utilizados.
* Una imagen o gráfico que sirve para ilustrar el diseño elegido.
  1. Distrito
     1. Organización

Archivo de bloque con Registros de datos de longitud variable (RLV) guardados en un Hash de dispersión fija con zona de desborde. La zona de desborde será un archivo secuencial de RLV organizado por bloques para poder direccionar a otra cubeta.

* + 1. Razones
* Ninguna funcionalidad de la aplicación requiere recorrer distritos secuencialmente una vez que encontramos a uno.
* La cantidad de distritos (al menos en un escenario real) es fija, no aumenta. Solo se harán operaciones de consulta una vez que la carga inicial fue realizada.
  + De esta forma es posible diseñar una función de dispersión así como el tamaño de los buckets para poder recuperar un distrito en la primera lectura a disco (2da en caso de que la función de hash no sea lo suficientemente efectiva y se produzca un desborde).
    1. Características Particulares
* Función de Hash: A determinar.
* Tamaño Bucket/Bloque: A determinar.
* Factor empaquetamiento (registros ocupados / espacio disponible): A determinar.
* El bloque se considera desbordado cuando se algún distrito no pueda insertarse por tener tamaño mayor al espacio libre disponible. Sin embargo, si otro distrito es insertado y puede ser acomodado en el archivo de datos, el mismo puede ser dado de alta allí.
  + 1. Registros

Se utilizara solo un registro de datos, de la forma:

* **R**(longitudRegistro, longitudDistrito, (distrito)i)
  + 1. Bloques

Los bloques del archivo de datos y de desborde serán de la forma:

* **B**(bloqueDesborde, espacioLibre, (Distrito)\*)
  + 1. Ilustración

A continuación se muestra el posible contenido de un bloque:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Bloque Desborde | Espacio Libre | | Distrito 1 | | Distrito 2 | | Distrito 3 | | Distrito 4 | | | | |
| **4** | | 2 | | 6 | | Chaco | | 10 | Catamarca | 8 | Córdoba | 7 | Chubut |

Espacio ocupado: 4 (bloque desborde) + 4 (espacio libre) + 8 (longitud registro) + (6 + 10 + 8 + 7) = 46

* 1. Votante
     1. Organización

Archivo de bloque con Registros de datos de longitud variable (RLV) guardados en un Hash de dispersión fija con zona de desborde. La zona de desborde será un archivo secuencial de RLV organizado por bloques para poder direccionar a otra cubeta.

* + 1. Razones
* Ninguna funcionalidad de la aplicación requiere recorrer distritos secuencialmente una vez que encontramos a uno.
* Es necesario, para poder proveer una buena experiencia de usuario al electorado, poder acceder a la información de un votante en la menor cantidad de accesos a disco posible y de forma aleatoria.
* Una función de hashing, y tamaño de bloque (tanto de datos como de desborde) correctamente elegidos pueden hacer que en el momento más crítico (el día de las elecciones) la cantidad de lecturas necesarias para acceder a un votante sea baja. Veamos esto con un poco más de detalle:
  + Durante una elección, si un bloque se desborda debido al crecimiento de un registro, solamente se necesitaría de otro bloque para acceder a un votante. Es decir, que en la gran mayoría de los casos, como máximo se necesitarían 2 accesos a disco.
  + Una vez que las elecciones finalizan, se puede evaluar el estado del hash para determinar si una re-estructuración de mantenimiento es conveniente (en base a la cantidad de bloques con desborde).
    1. Características Particulares
* Función de Hash: Considerando que el DNI de un argentino tiene 8 dígitos, tomamos los últimos 6 para determinar el bloque a donde se debe guardar el votante.
* Tamaño Bucket/Bloque: 16 KB, seteado por configuración.
  + Teniendo en cuenta que el padrón electoral argentino en la última elección fue de 28000000 votantes usaremos bloques que puedan guardar aproximadamente 28 registros.
  + A partir del tamaño del [tamaño promedio del bloque](#votanteblockav) y que debemos usar el valor más cercano hacia arriba de 512 \* 2^n.
* Factor empaquetamiento (registros ocupados / espacio disponible): A determinar (debería ser cercano a 100%).
* El bloque se considera desbordado cuando se algún votante no pueda insertarse por tener tamaño mayor al espacio libre disponible o alguna actualización haga que el bloque se desborde.
  + 1. Registros

Se utilizara solo un registro de datos, de la forma:

* Votante (longitudRegistro, (DNI)i, longitudNyA, NombreyApellido, clave, domicilio(longitudCalle, calle, nro), longitudDistrito, (distrito)ie, ((eleccion(fecha(año, mes, dia), longitudCargo, cargo))ie)\*)

El campo de la longitud del registro es de 2 bytes. Los campos de control de cada campo son de 1 byte.

* + - 1. Tamaño promedio registro

Para calcular el tamaño promedio de un registro consideramos la siguiente longitud de sus campos.

* Control: 12 bytes.
* DNI: 4 bytes.
* Nombre y Apellido: En promedio consideramos 25 bytes.
* Clave: 4 bytes. La clave será 4 caracteres cualesquiera. Tomamos como ejemplo la clave bancaria.
* Calle: En promedio, 10 bytes.
* Numero: 2 bytes.
* Distrito: En promedio, 10 bytes.
* Fecha: 4 bytes.
* Cargo: En promedio, 10 bytes.
* Elecciones promedio: En el DNI hay 32 lugares para anotaciones. Consideramos un adicional del 25% ya que muchas veces se da la situación que estos lugares se llenan. Aproximadamente 20 elecciones como promedio, considerando que hay gente de todas las edades votando.

El tamaño promedio de un registro sería de 347 bytes.

* + 1. Bloques

Los bloques del archivo de datos y de desborde serán de la forma:

* **B**(bloqueDesborde, espacioLibre, (Votante)\*)

De esta forma el tamaño promedio de bloque será

28 votantes \* 347 bytes + 4 bytes + 2 bytes = 9722 bytes.

* 1. Eleccion
     1. Organización

Archivo de bloques secuencial indexado organizado con un árbol B+ y RLV. Adicionalmente, la mayor parte de la lista de distritos la guardaremos en un archivo aparte, organizado por bloques y con RLV.

* + 1. Razones
* En base a la funcionalidad del programa, tenemos la necesidad de realizar ABM y lecturas al azar.
* No encontramos una función de dispersión que pueda distribuir a los registros de una manera lo suficientemente homogénea como para que el un archivo de organización directa asegure una baja cantidad de desbordes, haciendo así que las operaciones puedan tener un costo alto. La característica balanceada del árbol B+, nos ayuda a mantener un orden razonable para las operaciones.
* Podríamos usar otro tipo de organización, por ejemplo un árbol B, ya que en realidad no necesitamos las características secuenciales del árbol B+. Sin embargo, por diferentes cuestiones (tiempo, robustez) preferimos usar un árbol B+, ya que sus capacidades de búsqueda secuencial si son requeridas para otras entidades.
  + 1. Características Particulares
* Tamaño nodo: 512 B.
* Política Split: Cuando se produce overflow, se calcula el tamaño de todos los registros (incluyendo el que causo el overflow). Se comienza a sumar los tamaños desde el primero hasta superar la mitad del tamaño total. Los registros que formaron parte de la suma se mantienen en el mismo nodo, los demás se guardan en un nodo nuevo.
* Política Merge: Cuando un nodo queda en underflow, se balancea con su hermano derecho sacando registros hasta que el hermano izquierdo supera en tamaño al hermano derecho, siempre y cuando haya espacio disponible en el hermano izquierdo. Se mergean dos nodos cuando la carga del hermano izquierdo más la del hermano derecho es menor al tamaño del nodo.
* Búsqueda separador: Lineal.
  + 1. Registros

Usaremos tres registros pero diferentes, uno para ser usado en el sequence set, otro para el index set y otro para los registros de la lista de distritos a guardar en un archivo aparte.

* + - 1. Sequence Set

**SSR**(longitudRegistro, año, mes, dia, longitudCargo, cargo, longitudDistrito, primerDistrito, punteroListaDistritos)

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Long Registro | Año | Mes | Día | Long Cargo | Cargo | Long Distrito | Primer Distrito | Puntero Distritos |
| 29 | 1986 | 06 | 22 | 11 | Intendente | 8 | Cordoba | **4** |

En el sequence set guardaremos solamente el primer distrito y un puntero a la lista de distritos en caso de haberlos (-1 en caso contrario). Esto es porque asumimos que la mayoría de las elecciones tendrán un distrito y solo algunas más que esa cantidad. Adicionalmente, no queremos que haya mucha diferencia entre el tamaño de registros, lo que podría pasar entre una elección presidencial (todos los distritos) y una de intendente en una pequeña localidad.

* + - 1. Index Set

**ISR**(longitudRegistro, año, mes, día, longitudCargo, cargo)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Long Registro | Año | Mes | Día | Long Cargo | Cargo |
| 16 | 1986 | 06 | 22 | 11 | Intendente |

* + - 1. Lista Distritos

**LDR**(longitudRegistro, longitudDistrito, distrito)

* + 1. Bloques

Tendremos tres tipos de organizaciones en bloques. Unas para el sequence set, otra para el index set y otro para la lista de registros en un archivo separado.

* + - 1. Sequence Set

**SSB**(nivel, espacioLibre, siguienteNodo, SSR+)

* + - 1. Index Set

**ISB** (nivel, espacioLibre, ISR+, punteroNodoHijo+)

* + - 1. Lista Distritos

**LDB** (espacioLibre, LDR\*)

Lista

Organización

Archivo de bloques secuencial indexado organizado con un árbol B+ y RLV.

Razones

* En base a la funcionalidad del programa, tenemos la necesidad de realizar ABM y lecturas al azar.
* No encontramos una función de dispersión que pueda distribuir a los registros de una manera lo suficientemente homogénea como para que el un archivo de organización directa asegure una baja cantidad de desbordes, haciendo así que las operaciones puedan tener un costo alto. La característica balanceada del árbol B+, nos ayuda a mantener un orden razonable para las operaciones.
* Podríamos usar otro tipo de organización, por ejemplo un árbol B, ya que en realidad no necesitamos las características secuenciales del árbol B+. Sin embargo, por diferentes cuestiones (tiempo, robustez) preferimos usar un árbol B+, ya que sus capacidades de búsqueda secuencial si son requeridas para otras entidades.

Características Particulares

* Tamaño Bucket/Bloque: A determinar.
* Política Split: cuando hay overflow se calcula el tamaño de todos los registros (con el que provoca el overflow incluido) y se suman los tamaños desde el primero hasta superar la mitad del tamaño total. Los registros que formaron parte de esa suma permanecen en el mismo nodo, los restantes se guardan en un nodo nuevo.
* Política Merge: al quedar un nodo en underflow, se balancea con su hermano derecho sacando registros hasta que el nodo izquierdo supera en tamaño al nodo derecho, siempre que haya espacio disponible en el primero. Se mergean dos nodos cuando la suma de las cargas de los hermanos izquierdo y derecho es menor al tamaño del nodo.
* Busqueda separador: lineal.

Registros

Se utilizará un solo registro.

**R**(longitudRegistro, año, mes, día, longitudCargo, cargo, longitudNombre, nombre)

El campo de longitud de registro es de 2 bytes. Los campos de control son de 1 byte.

Tamaño del registro

Control: 2 bytes

Fecha: 4 bytes

Cargo: 10 bytes (promedio)

Nombre: 20 bytes (promedio)

Tamaño promedio del registro: 38 bytes

Ejemplo

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Longitud Registro | Año | Mes | Día | Longitud Cargo | Cargo | Longitud Nombre | Nombre |
| 24 | 2009 | 04 | 18 | 11 | Intendente | 7 | Lista0 |

Tamaño del registro ejemplo: 24 bytes

Bloques

* SSB(nivel, espacioLibre, punteroNodo, R+)
* ISB(nivel, espacioLibre, ISR+, punteroNodoHijo+)

Tamaño promedio del bloque: 1 + 2 + 4 + (38 \* 13) = 501 bytes