Trabajo Práctico - Etapa 1
Voto Electrónico

Martín Hernán Gómez, *Padrón Nro. 85.780*
martinhgomez@yahoo.com.ar

Ignacio Marambio Catán, *Padrón Nro. 82.694*
ignacio.marambio@gmail.com

Martín Eduardo Quiroz, *Padrón Nro. 86.012*
martinedq@yahoo.com.ar

Daniel Shlufman, *Padrón Nro. 88.040*
incorporado@gmail.com

Lucas Damian Tarcetti, *Padrón Nro. 87.165*
lucas.tarcetti@yahoo.com.ar

2do. Cuatrimestre de 2011
75.06 Organización de Datos – Cátedra Lic. Arturo Servetto
Facultad de Ingeniería, Universidad de Buenos Aires

26/10/11

Índice

1. Introducción

El trabajo consiste crear una aplicación capaz de mantener un sistema de voto electrónico.

1.1. Objetivos

El objetivo de este trabajo práctico es que la aplicación sea capaz de mantener información sobre las entidades que componen el sistema de votaciones (votantes, elecciones, candidatos, etc.), y de proveer la posibilidad a un votante de emitir su voto para la correspondiente elección.

2. Documentación técnica

2.1. Elección del lenguaje de programación

La resolución del trabajo práctico debe será realizada en plataforma Linux y en lenguaje C++, aprovechando el uso de la programación orientada a objetos.

2.2. Entidades

- Distrito ((distrito)i)
- Votante ((DNI)i, NombreyApellido, clave, domicilio, (distrito)ie, ((eleccion)ie)*)
- Eleccion ((fecha, (cargo)ie)i, ((distrito)ie)+)
- Lista (((eleccion)ie, nombre)i, cantidadVotos)
- Candidato (((lista)ie, (votante)ie, (cargo)ie)i)
- Cargo ((cargo)i)
- Administrador ((usuario)i, clave)

2.3. Estructuras de datos

Las estructuras principales elegidas para este trabajo han sido elegidas según la necesidad de cada entidad. Las mismas son:

- Árbol B+
- Dispersión Extensible
- Archivos de Bloques
- Registros Variables

2.4. Funcionalidades

En el ámbito del administrador se encuentran las siguientes funcionalidades:

- Mantener Distritos
- Mantener Votantes
- Mantener Elecciones
- Mantener Cargos
- Mantener Listas
- Mantener Candidatos
- Informar Resultados

2.5. Descripción de la arquitectura utilizada

Para una mejor observación de los diagramas de clases de la arquitectura utilizada en el programa, las mismas se encuentran documentadas en un archivo externo junto con su descripción, se puede acceder a el a través del archivo '00 Index.html', el mismo se encuentra en la carpeta 'documentación'. Desde allí se puede navegar a través de todas las clases existentes.

2.6. Descripción de cada clase utilizada

A continuación se enumeran las principales clases de la aplicación, no refiréndonos a sus nombres de archivo sino con su nombre conceptual.

- Clase ArbolBMas: Se utiliza para guardar las listas de votación y para la confección de informes.
- Clase Dispersión: Se utiliza para almacenar todas las demás entidades que no sean las listas de votación.
- Clase Archivo en Bloques: Se usa para darle un sustento en disco a las clases de Árbol B+ y de Dispesión. Ofrece una persistencia en disco en un archivo de bloques.
- Clases de Entidades: Las mismas se utilizan para instanciar cada entidad necesaria al momento de crear una votación. Las mismas incluyen: Distrito, Votante, Eleccion, Lista, Candidato, Cargo, Administrador.

2.7. Justificación de uso de cada clase utilizada

El uso del *Árbol B+* se eligió principalmente debido a la opción de poder hacer búsquedas parciales y además a la ventaja de poder acceder secuencialmente a los datos de forma ordenada. El uso del *archivo de dispersión* se eligió debido a que usa la menor cantidad de accesos posibles (solo un acceso) haciendo que se optimice el uso de estos archivos, al ser los mas accedidos en disco. El uso del *archivo de bloques* se eligió por ser el más compatible con respecto a la persistencia del Árbol B+ y del archivo de dispersión, haciendo mas efectivo el uso de los mismos.

2.8. Descripción de la organización de archivos auxiliares

Definiciones lógicas y físicas de todos los archivos que se utilicen (datos maestros, índices, trabajo, control, etc).Descripción de la organización de cada uno de los archivos. Para qué se utiliza. Por qué se utiliza (base teórica).

Dentro de los archivos utilizados para el funcionamiento del programa disponemos de las siguientes estructuras auxiliares:

- Archivo de configuración: El mismo se usa para obtener las rutas de los archivos de dispersión, archivo del árbol, tamaños de cubetas y de nodos.
- Índice: El índice se usa para la obtención de los informes de votaciones por distintos criterios de una misma base de datos.
- Archivo de control: Se utiliza para poder persistir datos referentes a las estructuras de control de las estructuras, como por ejemplo la tabla de dispersión dentro de la estructuras de dispersión.
- Archivo de password: Guarda el *user* y *pass* perteneciente al administrador de votos.

2.9. Planificación de tareas

Planificación (identificación de tareas, estimación de duración y asignación)

A grandes rasgos la planificación del proyecto se realizó dentro de las cinco semanas la siguiente manera:

- Semana 1: Planeamiento del problema, elección de las distintas estructuras para las entidades existentes, configuración del sistema operativo, herramientas de compilación, IDE's, etc. Para su correcto funcionamiento en el sistema Linux. Duración aproximada: 1 semana.
- Semana 2: Construcción de las entidades, creación de sus respectivas clases (1 semana). Creación de la clase encargada del manejo en disco, manejador de archivos (1 semana). Comienzo de creación de la clase de Árbol B+ (4 semanas).
- Semana 3: Comienzo de creación de la clase de Dispersión (3 semanas). Creación de la clase encargada del archivo de bloques (2 semanas). Creación de clase bucket y creación de clase nodo.
- Semana 4: Creación de pruebas individuales e integrales (1 semana). Serialización e hidratación de datos (1 semana). Creación del archivo de buckets (1 semana).
- Semana 5: Integración de los distintos módulos (1 semana). Creación de la lógica de votación (1 semana). Implementación de una interfaz para el usuario (1 semana).

2.10. Bugs conocidos

A continuación se enumeran ciertas situaciones donde el programa podría presentar dificultades en su proceso:

- La fechas deben tener tener el formato: aaaa/mm/dd para poder procesarse correctamente.
- Si el archivo de configuración no tiene el delimitador '//' dentro del archivo, el mismo no parsea ningún dato, ya que toma al archivo entero como comentario
- El archivo de bloques debido a su estructura interna debe tener un tamaño de bloque que sea múltiplo de 4 bytes, si no es de esta forma podría referenciar erroneamente a un bloque. Esto se solucionó validando el tamaño de bloque al iniciar un archivo.
- En una elección determianda se podría agregar cualquier distrito sin que se verifique si realmente existe.
- El borrado de algún archivo de control, configuración y/o datos en tiempo de ejecución llevará a una malfunción del programa.

2.11. Archivos de Control para las Entidades

2.11.1. Archivos de Control

Se encuentran todos dentro de un directorio, especificado a través de un archivo de configuración de la aplicación, y pueden tener jerarquía de subdirectorios interna. Es donde se guarda toda la información necesaria para poder funcionar.

2.11.2. Archivos con resultados

Como respuesta a toda interacción, el sistema generará archivos de registro de operaciones (LOGs) en el directorio donde se llame a la aplicación.

3. Parte teórica

En la siguiente sección se enumeran distintos conceptos teóricos acerca de las estructuras usadas en la resolución del TP.

3.1. Física - Organización

3.1.1. Organización de registros

- La longitud de un registro y/o de un campo variable se delimita con el uso de un indicador de longitud, el cual se antepone a los datos pertenecientes al registro o campo. De esta forma se puede conocer donde termina la sección de datos.
- En todos los casos se guarda como mínimo la información correspondiente a la longitud del segmento de datos pudiendo, en ciertos casos, guardarse mas información, según sea la estructura.

3.1.2. Consideraciones acerca del Archivo de Bloques

1. Los bloques tienen tamaño fijo, determinado por primera y única vez al crearse el archivo.

2. Existen 4 tipos de bloques: Data, Metadata, Removed y Head

3. Head: Existe un solo bloque de este tipo por archivo y siempre se ubica al principo del mismo (en la posición 0).

   El bloque Head tiene en su estructura:

   |currmetadata (int)|maxblocknum(int)|blocksize(int)| Espacio sin uso (int)|... ...|Espacio sin uso (int)|

4. Data: Este bloque se usa íntegramente para guardar datos del usuario

   El bloque Data tiene en su estructura:

   |datos(int)|...|datos(int)|...|datos(int)| (Todo el espacio reservado para datos, sin metadata).

5. Metadata: Bloque que se encarga del control de los bloques de datos borrados (Removed). Aquí se guardan las referencias a bloques removed.

   El bloque de Metadata tiene en su estructura:

   | Metadata Anterior(int)| currPos(int)| ID bloqLibre (int)| ...| ID bloqLibre (int)|

6. Removed: Son bloques de datos que han sido borrados por el usuario, no se utiliza ningún atributo para identificarlos, los mismos están referenciados en el bloque de metadata como 'libre'.

7. Internamente todos los atributos de metadata son manejados como int. En el caso donde desde afuera se piden los bloques de datos el bloque se obtiene como un char*.

8. El currpos empieza desde el primer byte del bloque, incluyendo los bytes cabezeras. O sea que el 1er dato de metadata está en el byte 8 (2*sizeof(int))

9. Cada vez que cambia el 'currmetadata' o 'maxblocknum' se escribe en disco con serializehead (que escribe 2 veces en disco por cada vez que se lo llama)

10. Cada vez que cambia el 'currpos' se escribe en disco (se escribe en el bloque de metadata). Esto sucede al pedir un bloque nuevo o al borrar un bloque de datos

11. Está contemplado el caso donde el metadata actual no tiene bloques libres, y entonces este mismo pasa a ser un bloque disponible (Caso límite).

12. Está contemplado el caso donde borro un data, pero el metadata actual está totalmente lleno, pasando el bloque 'D' a ser un bloque 'M' (Caso límite).

13. Si pido un metadata nuevo tengo que ver si es el 1ro, si esto es así, su valor ."anterior"(posición [1] dentro del bloque de int's) tiene que ser = Cero

14. No hace falta un getblock de 'R' porque esos bloques siempre van a ser accedidos a través de newblock (con parámetro 'D' o 'M')

15. No hace falta un newblock de 'R' porque esos bloques siempre van a ser creados a través de delblock

16. En una primera instancia, se propuso etiquetar cada bloque para su identificación. Pero luego, para simplificar el funcionamiento y para no invadir espacio en el bloque de Datos, se optó por no usar etiquetas en los bloques.

3.2. Índices - Búsqueda

3.2.1. Hashing

Función de hashing utilizada. Criterio de elección.

Para una rápida recuperación de la información se decide organizar dicho archivo con el método de dispersión extensible. Se utiliza un hashing extensible de valores sufijos. La función de hashing utilizada es f(x) = (x) mod (tamaño de tabla)

3.2.2. Árbol B+

Nuestro arbol B+ es relativamente genérico, a excepción de la clave de ordenamiento que es invariablemente un string, los datos son una entidad genérica y en general no me interesa que se guarda ahi.

Está implementado de tal manera que también esta separado casi completamente de la parte que lidia con el almacenamiento teniendo 2 precondiciones ineludibles. el lugar donde se encuentra alojada la raíz debe ser invariante y las estructuras que representan a los punteros a los bloques deben ser representadas como enteros.

La separación se logró mediante la clase ffile y tuvo tanto éxito que con mínimas modificaciones fue posible pasar de un árbol cuyo almacenamiento era la memoria a otro cuyo almacenamiento era un archivo de bloques.

El árbol esta implementado en 4 clases diferentes, la ya mencionada ffile, bplustree, inner _node y leaf_node estando estas ultimas intimamente relacionadas de manera tal que una esta declarada como friend de la otra. Esta decision se tomo para no exponer cosas criticas de las clases que necesariamente tenian que usarse entre inner_node y leaf_node.

En memoria, estas clases utilizan un vector de pares de enteros y cadenas de caracteres para los nodos internos (la clase inner_node) y pares de cadenas y vectores en las hojas (la clase leaf_node). Estas 2 clases, al serializarse se transforman en una cadena larga de caracteres (que se guarda en un vector) representada por pares longitud, datos. Ademas a esto se agrega un entero mas que representa la cantidad de pares, longitud, datos que existen en la cadena en adicion a un encabezado que es una I o una L segun sea el nodo un nodo interno o una hoja y un par de enteros mas que representan el nodo que contiene los datos cuya clave es mayor al nodo actual o el nodo que tiene datos menor al primer par del nodo.

En este momento, el arbol se comporta como un arbol B+ clásico para los agregados pero no hace rebalanceos en caso de los borrados.

Mas alla de los enteros que tienen tamaño fijo, el resto de las estructuras son dinamicas. El objetivo de esta implementacion fue utilizar la mayor cantidad de las funciones de la STL que fueran posible y, como desafortunadamente, la STL, si bien tiene una funcion de busqueda que funciona en contenedores como los vectores utilizados para almacenar los datos en la memoria de este árbol, ésta solo devuelve si un dato se encuentra o no en el contenedor pero no su posición o sus datos por lo que la búsqueda dentro de los nodos es lineal.

Esto también fue una pre condicion para poder implementar la función que luego de una búsqueda devuelva el dato inmediatamente mayor al ultimo devuelto.

Para ordenar los datos en memoria, sin embargo si se utiliza la función sort() del grupo de funciones incluidas en <algorithms>

4. Documentación de usuario

4.1. Instalación del sistema

Para la instalación del sistema se usará un archivo makefile, el cual realizará automaticamente toda la operación de compilación de todos los archivos de código fuente, recorriendo todas las subcarpetas necesarias para una operación exitosa.

Para la instalación del sistema debemos descomprimir el archivo conteniendo el código fuente en la carpeta donde se desea realizar la instalación. Una vez realizada la descompresión debemos ejecutar por consola, situados en la ruta elegida, el comando *make* el cual compilará todos los archivos fuente y como consecuencia obtendremos el archivo ejecutable listo para usar.

4.2. Ejecución del sistema

Una vez creado el archivo ejecutable debemos iniciar el programa[1] pasando por parámetro la ruta del archivo de configuración, esto es un requisito obligatorio para comenzar con el programa, mediante los argumentos -c <rutaArchivoConfiguración>

A su vez el archivo de configuración proveerá todos los requisitos necesarios para ubicar los demás archivos relativos al programa, ya sean archivos de datos, control, configuración o el archivo de password.

4.2.1. Interfaz de administrador

Al iniciar el programa el administrador de la votación deberá autenticarse para poder entrar al sistema, una vez ingresado tendrá un menú con opciones donde podrá:

- Mantener Distritos.
- Mantener Votantes.
- Mantener Elecciones.

[1] Para iniciar el programa, al administrador se debe autentificar. User: undomiel, Pass: aragorn

- Mantener Cargos.
- Mantener Listas.
- Mantener Candidatos.
- Informar Resultados.

4.2.2. Interfaz de usuario

Una vez iniciada una votación, se dispondrá de una interfaz para el usuario 'votante'. En esta instancia el votante podrá:

- Autenticarse
- Emitir voto
- Corregir voto
- Informar voto

5. Corridas de prueba

A continuación se detallan las pruebas realizadas sobre el funcionamiento del programa. Se emplearon las pruebas detalladas en el enunciado del informe, las cuales pasaron con éxito.

Corridas de prueba:

```
Ingrese su nombre de usuario: undomiel

Ahora ingrese su contraseña: aragorn

usuario: <undomiel>
password: <aragorn>
INGRESO APROBADO
Bienvenido al sistama de gestion de elecciones
¿ Desea eliminar la base de datos y comenzar de 0? S/N
N

Opciones:

1) Mantener Distritos
2) Mantener Votantes
3) Mantener Elecciones
4) Mantener Cargos
5) Mantener Listas
6) Mantener Candidatos
7) Informar Resultados
8) Habilitar Elecciones
9) Habilitar Votantes para elección
10) salir
```

```
Opcion:
.
.
.
```

6. Conclusiones

Conclusiones

7. Código en C++

A continuación se adjunta el código del programa en lenguaje C++

Referencias

[1] Documentación: Texmaker 3.0.2 (para la redacción de este informe)

`http://www.xm1math.net/texmaker/`

8. Apendice - Enunciado TP Etapa 1

A continuación se agrega el enunciado de la Etapa 1, correspondiente a este Trabajo Práctico.