Agradecimientos

Ariel Sobrado

A mis padres *María del Carmen y Ricardo*, por darme la vida y ayudarme desde un principio para poder llegar a esta instancia en mi vida.

A mi novia *Belén*, quién con su amor me brinda, día tras día, las fuerzas necesarias para afrontar los obstáculos de la vida.

A mi compañero de Tesina *Luciano*, quién le puso empeño y garra para poder lograr este trabajo.

Gracias.

Luciano Marrero

A mis padres *Nieves y Fernando*, por el incansable esfuerzo y dedicación para que pueda llegar a este momento en mi vida.

A mi hermano *Christian*, por el abrazo y apoyo incondicional en todas las decisiones camino a mis objetivos.

A mi compañero de Tesina, *Ariel*, quién confió en mí para emprender juntos este camino.

Gracias.

Ariel Sobrado y Luciano Marrero

A nuestros directores de Tesina Pablo y Pampa, por estar siempre presentes en los momentos difíciles y brindarnos todo el apoyo necesario para lograr este trabajo.

Gracias.

Índice

Capítulo 1

Introducción

En este capítulo se presenta la motivación de desarrollar una herramienta de software educativa, se describe la organización del trabajo, los objetivos y los resultados esperados.

1.1 Motivación

La motivación es aquello que lleva a una persona a realizar determinadas acciones e insistir en ellas hasta el cumplimiento de sus objetivos. El concepto también se encuentra vinculado a la voluntad y al interés. La motivación es la voluntad para hacer un esfuerzo y alcanzar ciertas metas.

Comunicar conocimientos es una experiencia reconfortante que requiere un alto compromiso de las partes intervinientes. Durante el proceso de enseñanza y aprendizaje se trata de ser lo más expresivo posible con el fin de que todo sea captado y entendido correctamente.

Más allá del esfuerzo de ambas partes, alumnos y docentes, por entender y comprender algún determinado tema, en la mayoría de los casos las dudas y preguntas de interés surgen en momentos que están fuera de los horarios de clase preestablecidos, por lo tanto, se debe esperar la próxima clase. Si bien actualmente existen diversos medios de comunicación y lugares donde es posible buscar información de un tema en cuestión, generalmente suele suceder que lo encontrado no encuadra en el marco conceptual adecuado, o no es fácil de interpretar, aplicar y adaptar a lo que realmente se debe resolver.

Una realidad actual es que en la mayoría de los hogares se dispone de algún medio informático, una computadora personal de escritorio, una notebook, una netbook, una tablet, entre otros. Aprovechar esta realidad de la vida cotidiana y utilizar un producto de software adecuado que actúe como asistente para evacuar una duda en cualquier momento, es un apoyo importante durante el proceso de enseñanza y aprendizaje de un tema específico.

En la experiencia como docentes de la asignatura *Introducción a las Bases de Datos de la Facultad de Informática de la UNLP*, los autores de esta tesina han notado un incremento en el interés en los alumnos y la rápida comprensión de un tema, cuando existe la posibilidad de interactuar con un medio informático.

En base a la experiencia de estar presentes en las aulas y compartir con los alumnos la utilización de herramientas educativas de software, como *CASER* (herramienta asistir el diseño de bases de datos) [11] [12], y *HEA*, (herramienta para simular la implementación de índices mediante la familia de árboles B) [10], es deseable disponer de un complemento adicional para asistir al alumno en el proceso de enseñanza y aprendizaje del tema "Dispersión de Archivos".

1.2 Objetivos

Se propone el desarrollo de un producto de software denominado *E-HASH* (Herramienta de Software para Dispersión de Archivos), el cual está diseñado para asistir al alumno durante el proceso de aprendizaje de dispersión de archivos. La herramienta se presenta como un complemento al proceso de enseñanza y aprendizaje, y no pretende reemplazar, la práctica tradicional de lápiz y papel.

Combinando la práctica tradicional con *E-HASH* el alumno podrá comprender y profundizar el tema en cuestión, verificando su proceder en la resolución de cualquier problemática, como también probar nuevos casos, producto de su curiosidad e imaginación.

1.3 Organización de la tesina

Este informe se encuentra dividido en diversos capítulos que abarcan temas relacionados entre sí.

Resumen del Capítulo 2

Cuando se trabaja con una base de datos, muy pocas veces se analiza cómo se almacena la información dentro de los archivos de datos y que permiten, posteriormente, recuperarla cuando sea necesario. Esto se debe a que los Sistemas de Gestión de Bases de Datos (SGBD) abstraen al usuario las cuestiones internas y brindan interfaces amigables para gestionar la información.

Una base de datos es un gran repositorio de información que aumenta de forma constante, y lógicamente radica en memoria secundaria. El acceso a la información en memoria secundaria es algo muy costoso en lo que respecta a tiempos computacionales, por lo tanto, se debe considerar la forma más eficiente de poder localizar un dato en memoria secundaria utilizando para esto el menor tiempo de respuesta posible. Es aquí donde la dispersión de archivos comienza a jugar un papel importante debido a que hace posible llevar a cabo, en la mayoría de los casos, el acceso directo a la información.

Resumen del Capítulo 3

En este capítulo se describe otra técnica muy importante para el acceso a la información sobre memoria secundaria, la indización. Es importante tener un punto de referencia para evaluar y comparar el rendimiento de la dispersión de archivos, dado que no es la única técnica con la que se puede acceder en forma eficiente a la información en memoria secundaria.

La indización como técnica para la búsqueda de información se basa en el concepto de utilizar estructuras auxiliares para acceder rápidamente a los datos deseados. Esta técnica agrega un nivel de direccionamiento.

Primero se debe armar el índice, estructura que contiene los valores claves para cada elemento, para posteriormente poder acceder y obtener la información de manera completa en memoria secundaria.

Una de las formas de implementar índices es a través de la familia de árboles B.

Está fuera del objetivo de este trabajo profundizar los conceptos de indización, sino que se describe como una técnica más de implementar para el acceso a la información en memoria secundaria.

Resumen del Capítulo 4

En este capítulo se explican detalladamente los conceptos involucrados en la dispersión de archivos, desde su definición hasta las estrategias de implementación.

Esta técnica de organización de archivos logra, en la mayoría de los casos, el acceso directo a la información. Previamente se deben analizar y estudiar varias cuestiones para poder llegar a implementar un ambiente de dispersión adecuado y eficiente.

Antes de dispersar un archivo, hay que seleccionar un conjunto de parámetros importantes, los cuales le darán la forma al ambiente de dispersión. Entre ellos se encuentran: el "*tipo de espacio de direccionamiento"*, la "*función de dispersión"*, la "*técnica de resolución de colisiones",* entre otros.

Se deben estudiar las distintas posibilidades de elección para cada parámetro con el fin de poder generar diversas combinaciones y comprender los distintos escenarios que se puedan presentar.

Resumen del Capítulo 5

En este capítulo se presenta *E-HASH,* una herramienta de software educativa que tiene como principal objetivo asistir al alumno en el proceso de enseñanza y aprendizaje de dispersión de archivos.

Cada concepto descripto para la herramienta de software fue adoptado siguiendo el marco conceptual provisto por la asignatura de *Introducción a las Bases de Datos* de la *Facultad de Informática* de la *UNLP*.

El alumno, mediante *E-HASH*, podrá probar sus propios casos de uso analizando los resultados obtenidos y comparándolos con el trabajo realizado en la guía práctica tradicional.

La herramienta cuenta con explicación de todos los conceptos necesarios para entender y comprender un ambiente de dispersión. Además, presenta una interface amigable e intuitiva que ayudará al alumno a la configuración de cada ambiente de dispersión.

Resumen del Capítulo 6

En este apartado se hace referencia y se describe la tecnología que se ha utilizado para desarrollar *E-HASH*. Se explican los aspectos más importantes de cada una de las tecnologías involucradas y que han servido de soporte para realizar las distintas acciones que componen la herramienta.

Se ha optado por utilizar tecnología de uso libre y multiplataforma; se pensó en una herramienta que el alumno pueda llevar y ejecutar en la computadora que desee, sin importar el navegador o sistema operativo que se utilice.

E-HASH se basa en dos tecnologías simples y conocidas, *HTML* y *JavaScript*. Ambas hacen que sea una herramienta flexible y muy estable. No se requiere de la instalación de ninguna librería adicional, facilitando así la instalación del producto por parte del alumno.

En lo que respecta a *JavaScript*, es una tecnología que ha evolucionado con el tiempo en cuanto a su manejo y utilización, brindando la posibilidad de desarrollar funcionalidad con muy poco tiempo y esfuerzo.

E-HASH combina ambas tecnologías para lograr una herramienta robusta y estable, y brindar así al alumno un soporte confiable y eficaz.

Resumen del Capítulo 7

Se presentan las conclusiones finales obtenidas y los trabajos e investigaciones futuras que se esperan realizar.

Capítulo 2

Conceptos generales de Bases de Datos

2.1 Introducción

Hace algunos pocos años, hablar de "*Bases de Datos (BBDD)*" se reducía sólo al ámbito informático. Actualmente, este concepto se ha popularizado, producto de la inserción informática en la vida cotidiana de las personas.

El avance de la tecnología informática, tanto en software como en hardware, ha logrado ubicar a los sistemas de BBDD como el anfitrión ideal de cualquier organización al momento de persistir información digitalizada. Las BBDD son componentes esenciales de los sistemas de información.

En el mercado informático existen, al alcance de cualquier potencial usuario, múltiples herramientas de software que posibilitan la persistencia de información a través de una BBDD. Esto último no implica que su utilización sea la adecuada, y en la mayoría de los casos, no se aprovecha el potencial que las BBDD poseen.

En la actualidad se puede tener un sistema de BBDD, configurado y disponible en una pequeña maquina de escritorio como así también en grandes y potentes servidores empresariales. Los sistemas de BBDD actuales brindan múltiples ventajas con respecto a otros repositorios de almacenamiento de información tradicionales; algunas de ellas son: compactación, velocidad de acceso, menor tiempo de trabajo, disponibilidad de la información y seguridad, entre otras.

Se define a una base de datos como una colección o conjunto de datos almacenados e interrelacionados con un propósito específico vinculado a la resolución de un problema del mundo real [2].

Actualmente, las BBDD son esenciales para la supervivencia de cualquier organización. Los datos estructurados constituyen un recurso esencial para todas las organizaciones, grandes o pequeñas, y usuarios individuales.

2.2 Sistema de Gestión de Bases de Datos (SGBD)

Los *Sistemas de Gestión de Bases de Datos (SGBD)* o *Database Management System (DBMS)* son aplicaciones que permiten a los usuarios definir, crear y mantener las BBDD que deseen. Un SGBD es la aplicación que interactúa con los usuarios de los programas de aplicación y la base de datos.

Actualmente, cualquier sistema de software necesita interactuar con información almacenada en una base de datos y para ello requiere del soporte de un SGBD. Por lo tanto, no es posible separar una base de datos de un SGBD.

Un SGBD posee diversos objetivos, entre ellos se pueden destacar dos muy importantes:

- Permite definir una base de datos y sus componentes mediante un *Lenguaje de Definición de Datos* (LDD) que permite especificar la estructura, tipos de datos y restricciones sobre los datos, almacenando toda esta información en la propia Base de Datos.

- Permite separar la descripción y manipulación (inserción, eliminación, actualización y consulta) de los datos, permitiendo flexibilidad de consulta y actualización, todo mediante el manejo de un *Lenguaje de Manipulación de los Datos* (LMD).

La definición del esquema de una base de datos implica el diseño de la estructura que tendrá efectivamente la misma; describir los datos, las relaciones entre ellos, la semántica asociada y las restricciones de consistencia. Para ello se utilizan los lenguajes mencionados. A través del LDD se obtiene un archivo llamado diccionario de datos que describe la estructura y componentes de la base de datos. Con LMD se puede recuperar información, agregar nueva información y modificar o borrar información existente en la base de datos [3].

Un SGBD cuenta con otros objetivos, algunos de ellos son: controlar la concurrencia, facilitar el acceso a los datos, proveer seguridad para imponer restricciones de acceso, mantener la integridad de los datos, entre otros.

Para conservar la consistencia, debe diseñarse un modelo de base de datos que almacene cada dato lógico en un solo lugar de la misma, evitando así la redundancia y

ahorrando espacio de almacenamiento. Es importante destacar, que en algunos casos puede convenir tener redundancia controlada con el fin de mejorar el rendimiento operacional de las consultas realizadas en la base de datos.

Los sistemas de BBDD proporcionan almacenamiento persistente a los objetos y estructuras de datos de los programas. Es habitual que los lenguajes de programación cuenten con estructuras de datos complejas. Los valores de variables de un programa se desechan una vez que éste termina, a menos que sean almacenados en archivos permanentes; para ello, suele requerirse la conversión de esas estructuras complejas a un formato adecuado para su almacenamiento en archivos.

Si bien los SGBD presentan muchos beneficios y ventajas, también poseen algunos inconvenientes que no deben dejarse de mencionar. Los SGBD son conjuntos de programas muy extensos y complejos con muchas funcionalidades. Es preciso comprender estas funcionalidades para sacar ventaja de ellos, como también es necesario gran cantidad de espacio en disco y en memoria principal para poder trabajar de forma eficiente [3].

Arquitectura de tres niveles de un SGBD

La arquitectura de tres niveles, presentada en la Figura 2.1, es un método de gran aceptación para explicar el funcionamiento de los sistemas de BBDD, fue formalizado en 1975 y mejorado en 1978 y es también conocido como arquitectura ANSI/SPARC, así llamada por la Standards “Planning and Requirements Committee of the American National Standards Institute”, en español el Comité de Estandarización de Requerimientos y Planificación del Instituto Nacional de Estandarización Americano [14] [15] [20].

El objetivo de la arquitectura de tres esquemas es separar las aplicaciones del usuario y la base de datos física. En esta arquitectura se definen esquemas en los siguientes tres niveles:

- El nivel interno tiene un esquema interno, que describe la estructura física de almacenamiento de la base de datos. El esquema interno emplea un modelo de datos físico y describe todos los detalles para su almacenamiento, así como los caminos de acceso para la base de datos.

- El nivel conceptual presenta un esquema conceptual, que describe la estructura de la base de datos completa para una comunidad de usuarios. El esquema conceptual oculta los detalles de las estructuras físicas de almacenamiento y se concentra en describir entidades, tipos de datos, vínculos, operaciones de los usuarios y restricciones.
- El nivel externo o de vistas incluye varios esquemas o vistas de usuario. Cada esquema externo describe la parte de la base de datos que interesa a un grupo de usuarios y oculta a ese grupo el resto de la base de datos.

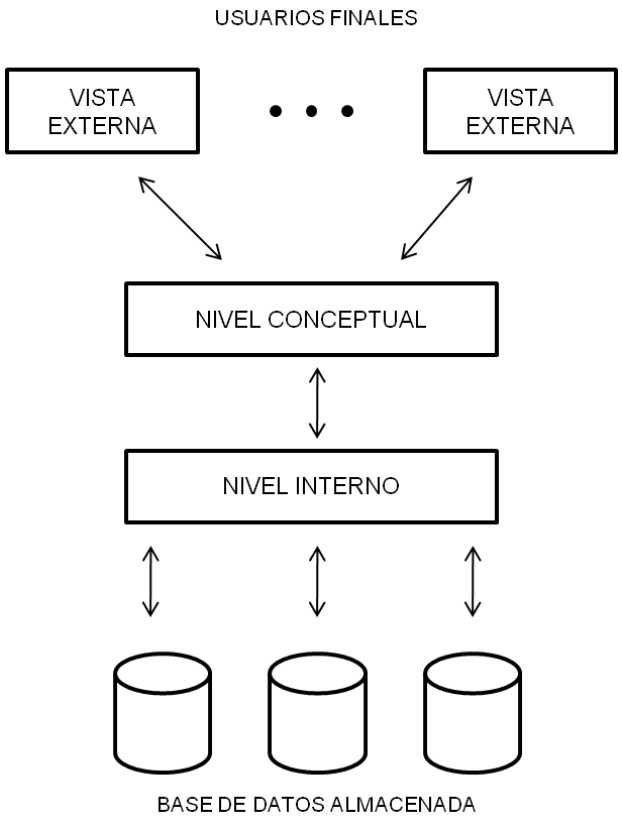


Figura 2.1. Arquitectura de tres niveles.

La arquitectura de tres niveles es una herramienta adecuada para que el usuario visualice los niveles del esquema de un sistema de bases de datos. La mayoría de los SGBD no separan los tres niveles completamente, pero algunos soportan, en cierta medida, la arquitectura de tres niveles. Cabe destacar, que los tres esquemas no son más que descripciones de los datos, los únicos datos que existen realmente están en el nivel físico.

En un SGBD basado en la arquitectura de tres niveles cada grupo de usuarios hace referencia exclusivamente a su propio esquema externo; por lo tanto, el SGBD debe transformar una solicitud expresada en términos de un esquema externo en una solicitud expresada en términos del esquema conceptual, y luego en una solicitud en el esquema interno que se procesará sobre la base de datos almacenada. Si la solicitud es una obtención de datos, será preciso modificar el formato de la información extraída para que coincida con la vista externa del usuario, este proceso se denomina correspondencia o transformación (mapping) [2] [3].

Módulos que componen un SGBD

Los SGBD son sistemas de software complejos; se constituyen por varios componentes, que juntos se complementan para darle vida a un SGBD.

A continuación se presenta un representación genérica de cómo funciona una base de datos, no obstante, esta representación puede variar dependiendo del SGBD y del avance o evolución de la tecnología informática. Algunos módulos son [2] [3]:

- La base de datos y el catálogo del SGBD, en general, se almacenan en disco. El acceso a disco suele ser principalmente controlado por el sistema operativo, el cual planifica la entrada/salida del disco. Un módulo del SGBD de alto nivel controla el acceso a la información del SGBD almacenada en disco.

- El compilador de LDD (Leguaje de Definición de Datos) procesa las definiciones de esquemas especificados en el LDD, y almacena las descripciones de los esquemas en el catalogo del SGBD. El catálogo contiene información como los nombres de los archivos y de los elementos de datos, los detalles de almacenamiento de cada archivo, la información de correspondencia entre los esquemas y las restricciones, además de otros tipos de información que es necesaria para los módulos del SGBD.

- El procesador de base de datos en tiempo de ejecución se encarga de los accesos a la misma; recibe operaciones de obtención o actualización de datos y las ejecuta sobre la base de datos.

- El compilador de consultas administra las consultas de alto nivel que se introducen interactivamente. Analiza la sintaxis y compila la consulta o la interpreta creando el código de acceso a la base de datos, y luego genera llamadas al procesador en tiempo de ejecución para ejecutar dicho código.

2.3 Organización Física de una Base de Datos

La colección de datos que constituye una base de datos debe estar almacenada físicamente en algún medio de almacenamiento de la computadora. El SGBD podrá recuperar, actualizar y procesar estos datos cuando sea necesario. Existen dos medios de almacenamiento:

- ***Almacenamiento primario***: medio de almacenamiento sobre el cual la *Unidad Central de Proceso* (UCP) opera directamente. En esta clasificación se encuentran la memoria principal y las memorias caché, ambas son de tamaño relativamente reducido, volátiles, pero muy rápidas. Por lo general, el almacenamiento primario ofrece acceso muy rápido a los datos, aunque su capacidad de almacenamiento es limitada.

- ***Almacenamiento secundario***: en esta categoría se encuentran los discos magnéticos, discos ópticos, memorias USB (Universal Serial Bus), entre otros dispositivos. Los dispositivos de almacenamiento secundario tienen mayor capacidad, son de menor costo y ofrecen acceso más lento a los datos que los dispositivos de almacenamiento primario. La UCP no puede procesar directamente los datos en almacenamiento secundario; deben copiarse previamente en el almacenamiento primario.

En la computadora los datos residen y se transportan a través de una jerarquía de medios de almacenamiento. La memoria más rápida es la más cara y, por lo tanto, la de menor capacidad. La memoria más lenta es el almacenamiento secundario, mucho más económica y dispone de una capacidad mucho mayor que la memoria primaria.

La principal desventaja del almacenamiento primario es su volatilidad, esto significa que pierde su contenido en caso de perder su fuente de energía. Por lo tanto, la memoria primaria no puede lograr persistencia luego de perder su fuente de energía.

Las BBDD siempre residen en el almacenamiento secundario. Esto se debe a que almacenan una gran cantidad de información que debe perdurar con el tiempo. Se necesitan dispositivos de almacenamientos muy grandes.

Es importante estudiar y comprender las propiedades y características del almacenamiento secundario, y la forma en la que se pueden organizar los archivos de datos con el fin de diseñar BBDD eficaces con un rendimiento aceptable.

Las técnicas usadas para almacenar grandes cantidades de datos estructurados en un disco, son importantes para los diseñadores de BBDD y para quienes implementan los SGBD, deben conocer las ventajas y desventajas de cada técnica para diseñar, implementar y operar con una base de datos o con un SGBD específico.

En general, los SGBD ofrecen varias opciones para organizar los datos, y el proceso de diseño físico de BBDD implica elegir, entre las opciones disponibles, las técnicas de organización de datos adecuadas para los requerimientos de la aplicación.

Las aplicaciones que utilizan una Base de Datos sólo requieren una pequeña porción de la misma en un momento dado, a fin de procesarla. Siempre que se requiera cierto dato, el mismo debe ser localizado en el disco, transferirlo a memoria principal para procesarlo y luego escribirlo en disco si es que sufrió alguna modificación. Los datos almacenados en disco son organizados en archivos de registro. Cada registro es una colección de valores de datos. Los registros deben almacenarse en el disco de manera tal que sea posible localizarlos de manera eficiente.

Aunque los usuarios no deben tener conocimiento de los detalles del diseño físico de la base de datos, es importante saber que éstos afectan al rendimiento, un factor de gran importancia en la satisfacción del usuario con el sistema de base de datos.

Una pregunta que surge aquí es *¿Podría el usuario obtener la información deseada en el formato apropiado y en un tiempo conveniente?*

La frase, "tiempo conveniente", puede expresarse generalmente como tiempo de respuesta aceptable. La "información deseada" y el "formato apropiado" no se afectan mucho por la organización física de la base de datos, pero el tiempo de respuesta sí. El tiempo de respuesta es el tiempo transcurrido entre la iniciación de una operación sobre la base de datos y la disponibilidad del resultado.

Un buen diseño físico de la base de datos almacenaría datos de forma que puedan recuperarse, actualizarse y manipularse en el mínimo tiempo posible.

En la Figura 2.2, se presenta el sistema para el acceso físico a la base de datos. Se puede observar la interacción del usuario con el sistema de base de datos al iniciar una consulta. Usualmente se transforma la consulta del usuario (demanda) en una forma efectiva para su posterior ejecución. El pedido o demanda, llevado a un lenguaje eficiente, activa entonces al administrador de buffers[1], que controla el movimiento de datos entre la memoria principal y el almacenamiento de la base de datos en disco. El gestor de archivos brinda soporte al administrador de buffer administrando la reserva de localizaciones de almacenamiento en disco y las estructuras de datos asociadas. Además de los datos del usuario, el disco contiene el diccionario de datos, que define la estructura de los datos del usuario y cómo éstos pueden utilizarse [3] [13].

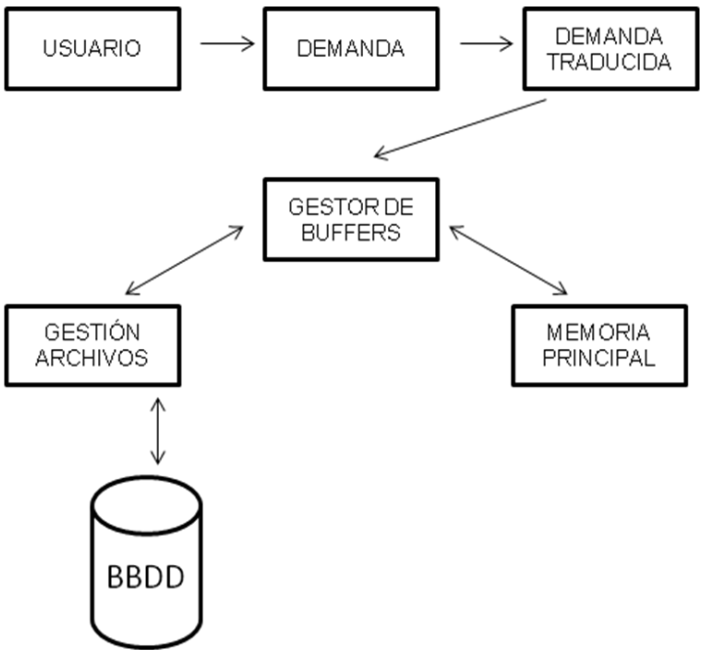


Figura 2.2. Acceso físico a una Base de Datos

[1] En informática, un buffer de datos es una ubicación de la memoria en un disco o en un instrumento digital reservada para el almacenamiento temporal de información digital, mientras que está esperando ser procesada [21].

Organización de los archivos

Los datos se almacenan generalmente en forma de registros. Cada registro consta de una colección de valores o elementos de datos relacionados, donde cada valor está formado de uno o más bytes y corresponde a un determinado campo del registro. Una colección de nombres de campos y sus tipos de datos correspondiente constituye una definición de tipo de registro o formato de registro. El tipo de dato asociado a cada campo, especifica el tipo de valores que el campo puede tomar. El tipo de dato de un campo es casi siempre uno de los tipos de datos estándar empleados por los lenguajes de programación. Entre ellos se encuentran los tipos de datos numéricos (entero, entero largo, real, etc.), de cadena de caracteres (longitud fija o variable), booleanos (0 y 1, o Falso y Verdadero).

Actualmente existe la necesidad de almacenar elementos de datos que consisten en objetos grandes no estructurados, que representan imágenes, videos, audio e incluso texto libre. Para ello, la mayoría de las BBDD proveen un tipo de dato denominado BLOB (binary large objects).

Un archivo es una secuencia de registros. En muchos casos, todos los registros de un archivo son del mismo tipo. Si todos los registros del archivo tienen el mismo tamaño (en bytes), se dice el archivo se compone de registros de longitud fija. Si diferentes registros del archivo tienen tamaños distintos, se dice que el archivo está constituido por registros de longitud variable.

Los registros de longitud fija poseen todos el mismo tamaño. Todos tienen los mismos campos, y las longitudes de cada campo son fijas, por lo que el sistema podrá identificar la posición inicial de cada campo en relación con la posición inicial del registro. Esto facilita la localización de los valores de los campos con los programas que tienen acceso a tales archivos.

Es posible representar como un archivo de registros de longitud fija a un archivo que lógicamente debería tener registros de longitud variable. Por ejemplo, en el caso de los campos opcionales se puede incluir todos los campos en todos los registros del archivo, pero se debe almacenar un valor nulo especial si no existe valor para determinado campo. En el caso de un campo repetitivo, se puede asignar a cada registro tantos espacios como el máximo número de valores que puede tomar ese campo. En ambos casos se

desperdicia espacio si algunos registros no tienen valores para todos los espacios físicos disponibles en cada registro [3].

En el caso de los campos de longitud variable, todos los registros tienen un valor en cada campo, pero no se conoce la longitud exacta de los valores de algunos campos. Para determinar los bytes que representan cada campo dentro de un registro en particular, se pueden utilizar caracteres separadores especiales para poner fin a los campos de longitud variables, estos valores especiales no deben ser válidos para algún campo determinado. Otra posibilidad es almacenar la longitud en bytes de cada campo del registro, delante del valor del campo.

Los algoritmos que procesan archivos de registros de longitud variable generalmente son más complejos que los que manejan registros de longitud fija, donde la posición inicial y el tamaño de cada campo son conocidos y fijos.

Almacenamiento de registros a memoria secundaria

Los registros de un archivo se deben asignar a bloques del disco, pues, el bloque es la unidad de transferencia de datos entre el disco y la memoria. Si el tamaño del bloque es mayor que el del registro, cada bloque contendrá numerosos registros, aunque inusualmente algunos archivos pueden tener registros grandes que no caben en un bloque. Si se tiene un tamaño de bloque de B bytes, para un archivo de registros de longitud fija de tamaño R, con B >= R, se podrá colocar B/R registros por bloque. El cociente B/R se denomina factor de bloques del archivo. En general, R no será divisor exacto de B, de modo que se tendrá cierto espacio desocupado en cada bloque.

Para aprovechar el espacio no utilizado, se puede almacenar parte de un registro en un bloque y el resto en otro bloque. Un puntero al final del primer bloque apuntará al bloque que contiene el resto del registro en caso de que no sea el siguiente bloque consecutivo del disco. Esta organización se llama extendida, porque los registros pueden extenderse más allá del final de un bloque. Siempre que un registro sea mayor que un bloque, es necesario emplear una organización extendida. Si no se permite que los registros crucen los límites de los bloques, se dice que la organización es no extendida. Ésta se utiliza con registros de longitud fija en los que B > R, pues, hace que cada registro se inicie en una posición conocida del bloque, lo cual simplifica el procesamiento de registros.

En el caso de registros de longitud variable, se puede utilizar tanto la organización extendida como la no extendida. Si el tamaño medio de los registros es grande, resultará ventajoso emplear la organización extendida para reducir el espacio perdido en cada bloque. En el caso de registros de longitud variable que utilizan la organización extendida, cada bloque puede almacenar un número diferente de registros. En este caso, el factor de bloques representa el número medio de registros por bloque para el archivo.

Existen varias técnicas para asignar los bloques de un archivo en disco. Una de ellas es la asignación contigua, donde los bloques del archivo se asignan a bloques consecutivos del disco. Esto agiliza notablemente la lectura de todo el archivo si se emplea doble buffer, pero dificulta la expansión del archivo. Otra técnica es la asignación enlazada donde cada bloque del archivo contiene un puntero al siguiente bloque de ese archivo. Esto facilita la expansión del archivo pero vuelve más lenta su lectura. Una combinación de las dos técnicas descriptas anteriormente asigna grupos de bloques de disco consecutivos, y luego enlaza los grupos. A estos grupos se los denomina segmentos de archivo o extensiones. Otra posibilidad es utilizar la asignación indexada, donde uno o más bloques de índice contienen punteros a los bloques de los archivos actuales [3] [6].

El “viaje” de un byte

Existen diversos elementos de hardware y software que intervienen cuando se necesita grabar cierta información en el disco. Desde el punto de vista de un programa de usuario, grabar un dato en disco puede verse como una simple instrucción, pero el “viaje” es largo y requiere de la intervención de varios componentes.

Un byte (unidad grabable más pequeña) se inicia en la memoria principal como representación de contenido de alguna variable de algún programa de usuario. Ante la necesidad de querer llevar dicho byte a memoria secundaria interviene el sistema operativo a través de su “*administrador de archivos”*. El administrador luego cerciora las características lógicas del archivo, compatibilidad, apertura, tipo del archivo, permisos, entre otros.

Una vez realizadas las comprobaciones el administrador de archivos determina donde se depositará el byte correspondiente. Se localiza la unidad, el cilindro, la pista y el sector correspondiente en disco y se determina si el sector correspondiente está cargado en memoria principal. En caso de que el sector necesite ser transferido a memoria se debe

encontrar un buffer de entrada / salida (E/S) disponible, para luego traer dicho sector a la memoria principal donde se escribirá el byte correspondiente.

El trabajo de llevar y controlar que el byte sea grabado efectivamente en el disco lo realiza otro dispositivo denominado "*controlador del disco*". Lo que sucede a partir de este momento es lo que más tiempo insume. Ubicar la cabeza de lectura y escritura en la pista y el sector de la unidad donde el byte será almacenado lleva un tiempo sumamente alto para una computadora, en comparación a los tiempos insumidos en memoria principal [1].

Operaciones con archivos

Las operaciones con archivos suelen agruparse en operaciones de recuperación y de actualización. Las primeras no alteran los datos del archivo, pues sólo localizan ciertos registros. Las segundas modifican el archivo por la inserción o eliminación de registros, o por la modificación de los valores de los campos.

Algunas de las operaciones más representativas para localizar y leer los registros de un archivo son: *abrir, volver al principio, buscar o localizar, leer, eliminar, modificar, insertar y cerrar*.

Es importante señalar la diferencia entre los términos organización del archivo y método de acceso. La organización del archivo se refiere a la organización de los datos de un archivo en registros, bloques y estructuras de acceso; esto incluye la forma en que los registros y los bloques se colocan en el medio de almacenamiento y se interconectan. El método de acceso, en cambio, proporciona un conjunto de operaciones que se pueden aplicar al archivo. En general, es posible aplicar varios métodos de acceso diferentes a una organización de archivo.

Algunos archivos pueden ser estáticos, lo que significa que pocas veces se efectúan operaciones de actualización; otros archivos son dinámicos, dado que pueden cambiar con frecuencia, lo que significa que se les aplican constantemente operaciones de actualización. Una organización de archivo deberá ser adecuada de modo de realizar lo más eficiente posible las operaciones que se esperan aplicar a menudo sobre el archivo [1].

Capítulo 3

Búsqueda de Información. Indización

3.1 Búsqueda de información

Las operaciones posibles sobre una base de datos son inserción, eliminación, actualización y búsqueda de información. Generalmente, el 80% de estas operaciones son de consulta, lo que implica buscar información y motiva a reducir el tiempo máximo posible que requiere encontrar un elemento en una base de datos.

Una de las alternativas para mejorar el tiempo de respuesta ante una solicitud es tener toda la información contenida en memoria principal (*RAM - Random Access Memory)* que permite lograr un rápido acceso. Esto es imposible debido a que la memoria principal no tiene el tamaño suficiente.

Según las propiedades de la memoria secundaria, el acceso a la información es costoso, por lo tanto, será necesaria la utilización de diferentes técnicas para mitigar este problema y lograr la mayor eficiencia posible en la recuperación de información.

El caso más simple consiste en disponer de un archivo serie, es decir, sin ningún orden preestablecido, donde para acceder a un registro determinado, se deben visitar todos los registros previos en el orden que estos fueron almacenados físicamente. Aquí, la búsqueda de un dato específico se detiene cuando se localiza el registro que contiene dicho dato (previo acceso a todos los registros previos), o en el final del archivo si el resultado no es exitoso (el dato buscado no se encuentra). Por lo tanto, el mejor caso es ubicar el dato deseado en el primer registro (1 lectura), y el peor caso en el último registro (*N* lecturas, siendo *N* la cantidad de registros almacenados en el archivo). El caso promedio consiste entonces en realizar *N/2* lecturas. Es decir, la performance depende de la cantidad de registros que contiene el archivo, en este caso se obtiene una performance de orden N.

Si el archivo estuviese físicamente ordenado y el argumento de búsqueda coincidiera con el criterio de ordenación, la variación en relación al caso anterior, se produce para el caso que el dato buscado no se encuentre en dicho archivo. En ese caso, el proceso de búsqueda se detiene en el registro cuyo dato es “mayor” al buscado. De ese modo, en

ese caso no existe la necesidad de recorrer todo el archivo. No obstante, la performance sigue siendo de orden *N*, y a la estrategia de búsqueda utilizada en ambos casos se la denomina "*búsqueda secuencial*".

Si se dispone de un archivo con registros de longitud fija y además físicamente ordenado, es posible mejorar esta performance de acceso si la búsqueda se realiza con el mismo argumento que el utilizado para ordenar tal archivo. En este caso, la estrategia de búsqueda puede ser mejorada.

La primera comparación del dato que se pretende localizar es contra el registro medio del archivo, es decir, el que tiene *NRR (Número Relativo de Registro)*[2] *= N/2*. Cada registro tiene un número relativo dentro del archivo. Dicho número se denomina *NRR* y permite calcular la distancia en bytes desde el principio del archivo hasta cualquier registro [1].

Si el registro no contiene ese dato se descarta la mitad mayor o menor, según corresponda, reduciendo así, el espacio de búsqueda a los *N/2* registros restantes (mitad restante). Nuevamente se realiza la comparación pero con el registro medio de la mitad restante, repitiendo así este proceso hasta reducir el espacio de búsqueda a un registro.

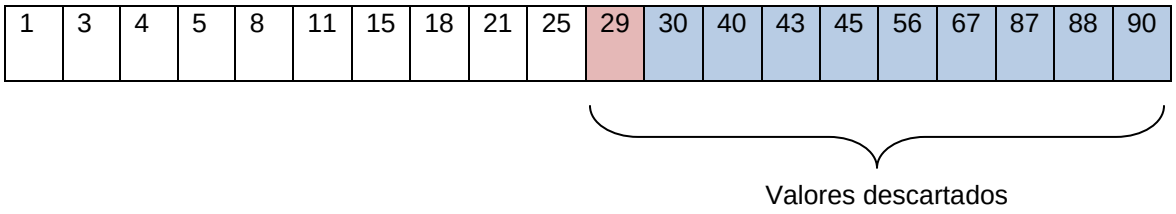
En el siguiente ejemplo, se busca el elemento 3, como el archivo contienen 20 registros el primer *NNR* accedido es el 10.

1	3	4	5	8	11	15	18	21	25	29	30	40	43	45	56	67	87	88	90

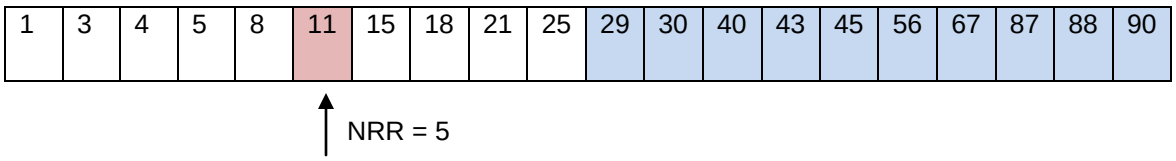
NRR = 10

En esa posición se encuentra el elemento 29. Los registros posteriores al registro del décimo lugar se descartan, pues, se puede asegurar que son valores superiores al buscado. Nuevamente se aplica el mismo procedimiento a la mitad restante de menor valor.

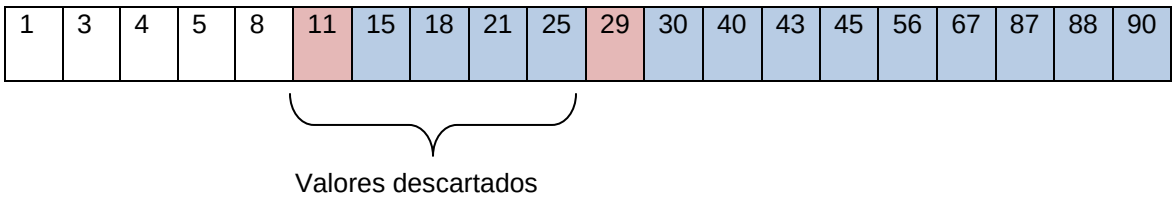
[2] El NRR es un concepto importante que surge de la noción de un archivo como un conjunto de registros y no como un conjunto de bytes. El NRR de un registro proporciona su posición relativa con respecto al principio del archivo. El primer registro de un archivo tiene un NRR de 0, el siguiente tiene un NRR de 1, y así sucesivamente [1].



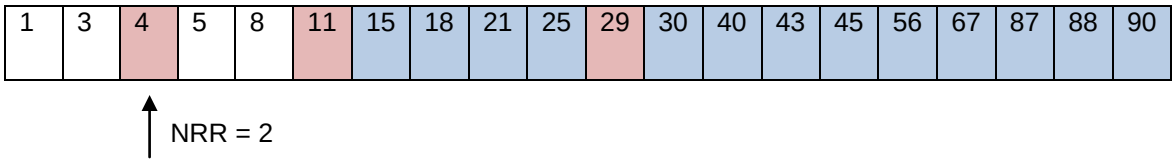
Se calcula el nuevo punto medio, que será la posición 5 del archivo.

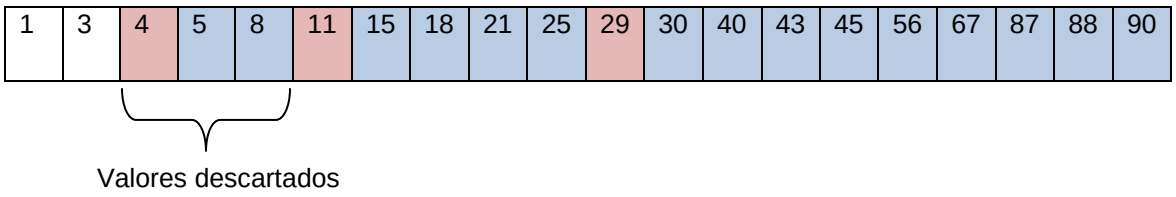


Sucede exactamente lo mismo que el primer caso. Como en esta posición tampoco está el elemento buscado se descartan los mayores al elemento 11, y se vuelve a aplicar el mismo procedimiento.

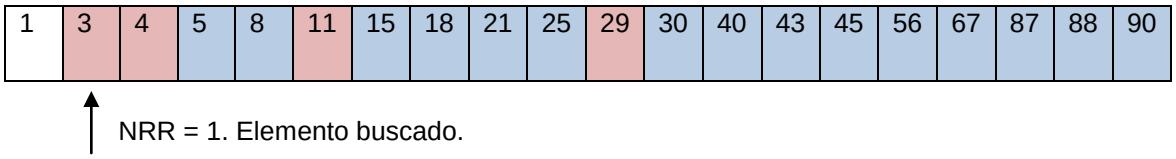


El proceso calcula nuevamente el punto medio. En este caso da como resultado 2.





Al aplicar nuevamente el procedimiento, se accede a la posición 1 y el elemento efectivamente es encontrado.



El criterio de búsqueda explicado anteriormente, donde la mitad de los registros restantes se descartan en cada comparación, se denomina "*búsqueda binaria"*.

En la búsqueda binaria, si *N* es el número de registros almacenados en un archivo, el orden de búsqueda será de $Log_2(N)$, teniendo en cuenta los accesos. Se puede notar que utilizando esta técnica se mejora el orden de búsqueda con respecto a la búsqueda secuencial, la cual tiene un orden lineal.

El principal inconveniente de la búsqueda binaria es la necesidad de que el archivo se encuentre físicamente ordenado. El proceso de ordenación física del archivo es un proceso costoso y bastante complejo. No es objetivo de este apartado describir las diferentes técnicas de ordenación existentes, ordenar los elementos de un archivo posee un estudio particular y existen diferentes técnicas para llevar a cabo esta tarea.

Si bien la búsqueda binaria mejora la performance con respecto a la secuencial, su costo de implantación sigue siendo demasiado elevado, es por ello que se plantea una nueva alternativa, cuyo concepto consiste en poder manejar el archivo como si estuviese ordenado físicamente sin que realmente lo esté. Dicho de otra forma, simular el ordenamiento físico de un archivo, salvando de esta manera, el costo que conlleva tal ordenación. Para llevar a cabo esta idea se utilizan índices.

Indización

El objetivo de tener ordenado un archivo se fundamenta en tratar de minimizar los accesos a memoria secundaria durante la búsqueda de información. Sin embargo, la performance es del orden *logarítmico*, y puede ser mejorada.

La mayoría de la bibliografía consultada, e inclusive este informe, poseen un índice, es decir una tabla que contiene un conjunto de temas y un número de página donde puede encontrarse dicho tema. La lista de temas que contiene un índice se los denomina "*claves* o *llaves"*, y los números de páginas son los campos de referencia al contenido. Todos los índices están basados en este concepto básico de claves y campos de referencia.

El índice de un libro permite localizar determinada información rápidamente sin tener que examinar todo lo anterior. La utilización del índice mejora la performance de búsqueda en comparación a una búsqueda hoja por hoja o secuencial. Por lo tanto un índice es un recurso muy valioso al momento de realizar búsquedas.

Una ventaja de la indización es que posibilita mantener un orden en el archivo sin que el mismo se encuentre físicamente ordenado, evitando tal costo.

Otra ventaja de los índices, es la posibilidad de tener varios caminos de búsqueda. Por ejemplo, para ordenar los libros de una biblioteca, de manera tal de poder localizarlos rápidamente, se podría establecer diversos órdenes, por autor, titulo, editorial, entre otros.

Cuando se realiza la búsqueda de un dato se debe considerar la cantidad de accesos a la memoria secundaria en pos de encontrar esa información, y en cada acceso, la verificación de si el dato obtenido es el buscado (comparación). Surgen dos parámetros para analizar:

1. *cantidad de accesos,* y
2. *cantidad de comparaciones*.

El primero de ellos se realiza sobre la memoria secundaria, lo que implica un costo relativamente alto; mientras que el segundo es sobre la memoria principal, por lo que el costo es relativamente bajo. Para entender los tiempos computacionales, se puede decir que si un acceso a memoria principal se lleva a cabo en unos pocos nanosegundos[3], entonces un acceso a memoria secundaria representaría milisegundos[4]. Por lo tanto, en el acceso a memoria secundaria se pone mayor atención en función de mejorar los tiempos de acceso.

Para comprender mejor el funcionamiento de los índices se presenta en la Tabla 3.1 el siguiente archivo de datos a modo de ejemplo:

[3] Un nanosegundo es la milmillonésima parte de un segundo. Este tiempo tan corto no se usa en la vida diaria, pero es de interés en ciertas áreas de la física, la química, la electrónica y en la informática [24].

[4] Un milisegundo es el período que corresponde a la milésima fracción de un segundo [25].

Dir. Reg.	Cía.	Código	Título	Grupo	Interprete
15	GAT	24	Imágenes paganas	Virus	Virus
36	GAT	13	Dame un señal	Virus	Virus
58	EMI	11	Signos	Soda Stereo	Soda Stereo
118	GPS	3452	Cosas peligrosas	Los Piojos	Los Piojos
161	EMI	2690	Perdiendo el control	Miguel Mateos	Miguel Mateos
203	GPS	345	Gracias	Guasones	Guasones
598	SON	114	Esquivando Puentes	La Renga	La Renga

Tabla 3.1. Archivo de datos.

Al crearse el archivo, con clave primaria formada por los atributos Cía. + Código, se crea el índice primario asociado. El índice y el archivo se presentan en la Tabla 3.2.

Clave	Ref.	Dir.	Registro de Datos
GAT24	15	15	GAT|24| Imágenes paganas | Virus | Virus
GAT13	36	36	CAT|13| Dame un señal | Virus | Virus
EMI11	58	58	EMI|11|Signos |Soda Stereo |Soda Stereo
GPS3452	118	118	GPS|3452|Cosas Peligrosas |Los Piojos |Los Piojos
EMI2690	161	161	EMI|2690| Perdiendo el control | Miguel Mateos | Miguel Mateos
GPS345	203	203	GPS|345|Gracía |Guasones|Guasones
SON114	598	598	SON| 114 | Esquivando Puentes | La Renga | La Renga

Tabla 3.2. Representación de un índice y el archivo correspondiente.

Las columnas de la izquierda representan el índice generado y ordenado por la *clave primaria* del archivo de datos, en tanto que la columna de la derecha representa el archivo de datos propiamente dicho. Se puede notar que el archivo de datos no se encuentra ordenado por criterio alguno.

Para realizar cualquier búsqueda de datos, por ejemplo si se desea buscar el compositor del tema “Signos”, en primer lugar se busca la clave primaria (EMI11) en el índice. Una vez hallada la referencia (58), correspondiente a la dirección del primer byte del registro buscado, se accede directamente al archivo de datos según lo indicado por dicha referencia.

Un detalle importante es que raramente se solicite un dato por su clave primaria, dado que la misma generalmente es desconocida por el usuario tradicional. Una búsqueda más común resulta ser por el nombre de la canción o eventualmente por autor, que son atributos más conocidos y fáciles de recordar. Tales atributos, nombre de canción o autor, seguramente contengan valores repetidos para diversos registros del archivo original. Por este motivo no es posible pensarlos como parte de una clave primaria. La clave que soporta valores repetidos se denomina *clave secundaria*.

Surge la necesidad de crear otro tipo de índice mediante el cual se pueda acceder a la información de un archivo, pero con datos fáciles de recordar. De esta manera surge el uso de índices secundarios.

Un índice secundario es una estructura adicional que permite relacionar una clave secundaria con una o más claves primarias, dado que como ya se ha mencionado previamente, varios registros pueden contener el mismo valor repetido. Luego, para acceder al dato deseado, primero se accede al índice secundario por clave secundaria, allí se obtiene la clave primaria y luego se accede con dicha clave al índice primario para obtener finalmente la dirección efectiva del registro que contiene la información buscada.

El índice secundario no posee directamente la dirección física de elemento de dato, esto se debe a que, al tener la dirección física solamente definida para la clave primaria, si el registro cambia de lugar en el archivo, solo debe actualizarse la información asociada clave primaria.

Si se supone que para un archivo cualquiera se tiene definido un índice primario y cuatro secundarios, si se modifica la posición física de un registro en el archivo de datos, los cinco índices deberían modificarse. Para mejorar esta situación, solo el índice primario tiene la dirección física y es el único que debe alterarse, mientras que todos los índices secundarios permanecen sin cambios.

En la Tabla 3.3 se presenta un índice secundario (índice de *Grupos de Música*)

Clave Secundaria	Clave Primaria
Miguel Mateos	EMI11
Soda Stereo	GPS1323
La Renga	AGP23
Virus	SON22
Virus	CAT13
Los Piojos	CAT15

Tabla 3.3. Archivo de índice secundario.

En la Tabla 3.3 puede verse claramente que cuando la clave secundaria se repite, existe un segundo criterio ordenación, establecido por la clave primaria.

Si se desea buscar un tema del intérprete "*Miguel Mateos*", primero se busca la clave "*Miguel Mateos*" en el índice secundario, se obtiene allí la clave primaria EMI11 y finalmente se accede al índice primario para obtener luego la dirección física del archivo.

Los índices son una porción reducida de información con respecto a la información total almacenada en los registros de un archivo. Esta información es necesario gestionarla de manera adecuada para lograr la mejor performance posible.

El espacio en disco requerido para almacenar un índice es típicamente mucho menor que el espacio de almacenamiento del archivo completo (puesto que los índices generalmente contienen sólo los campos clave de acuerdo con los que la tabla será ordenada, y excluyen el resto de los detalles de la tabla). En una base de datos relacional un índice es una copia de parte de una tabla.

Frente a lo presentado hasta el momento, el diseñador de una base de datos se puede ver tentado a la creación de muchos índices. Ahora bien, es necesario saber que el mantenimiento de los índices conlleva una carga adicional de procesamiento. De manera tal, que cada vez que se haga una nueva inserción o actualización sobre una tabla de una base de datos, los índices también deberán ser actualizados, razón por la cual, se debe lograr el equilibrio justo, para indexar aquellos campos que realmente sean necesarios en el momento de optimizar la recuperación de información [10].

Si el tamaño de los índices es lo suficientemente pequeño como para caber en memoria *RAM*, el manejo de los mismos se facilita, debido a que el ordenamiento de los índices en *RAM* es un proceso poco costoso que permitirá realizar búsquedas binarias accediendo de manera eficiente a los datos de un archivo sin necesidad de que los elementos de dicho archivo deban estar ordenados.

En la mayoría de los casos, los índices no pueden ser manejados en memoria *RAM*, razón por la cual es necesario almacenarlos en un archivo en memoria secundaria. Sin bien el ordenamiento de un archivo en memoria secundaria es un proceso costoso, el ordenamiento de índices sigue siendo conveniente respecto a la posibilidad de ordenar el archivo de datos, debido a que el tamaño del archivo de índices es menor. Sin embargo, el número de desplazamientos en memoria secundaria sigue siendo inaceptable, razón por la cual, es necesario buscar una alternativa que permita la organización correcta de los índices en pos de lograr una búsqueda eficiente.

Esta nueva alternativa consiste en organizar los índices utilizando la familia de árboles B. Esta clase de árboles balanceados son estructuras muy potentes a la hora de administrar índices asociados a archivos de datos, con un nivel de acceso rápido muy importante [1] [2].

Familia de Árboles B

La mejor estructura para un archivo va a depender del archivo en sí. Si el archivo de datos contiene unos pocos registros que puedan estar en un solo nodo, utilizar un árbol no es necesario, dado que agrega un nivel de direccionamiento por cada índice generado. Tampoco es conveniente si el archivo no requiere consultarse con tanta frecuencia.

Los árboles B son árboles multicamino con una construcción especial que permite mantenerlos balanceados a bajo costo.

Las estructuras de árboles surgen por la necesidad de descubrir un método para el almacenamiento y extracción de la información en grandes y voluminosos sistemas de archivos, que proporcione un acceso rápido a los datos con costos mínimos.

Utilizar un árbol B para indizar un archivo con un millón de registros permite encontrar la clave de cualquier registro con no más de cuatro desplazamientos a disco. Una búsqueda binaria con el mismo archivo puede requerir más de veinte desplazamientos [1] [2].

Árboles B

Muchas de las ideas en informática han surgido al cambiar el enfoque para examinar un problema. Los árboles B son un ejemplo de cambiar un punto de vista.

En los árboles B el enfoque propuesto es que se elige construir el árbol hacia arriba a partir de la base, en lugar de hacerlo hacia abajo desde la raíz. Lo interesante de la propuesta ascendente radica en que en lugar de intentar buscar formas de resolver una mala situación, se trata de evitar el problema. La familia de árboles B permiten que la raíz emerja, en vez de colocarla y después encontrar la manera de cambiarla, para lograr balancearlo.

Un árbol B se compone de nodos, donde cada nodo contiene una secuencia de ordenada de claves y un conjunto de punteros. El número de punteros siempre excede en uno al número de claves. Al número máximo de punteros que pueden almacenarse en un nodo se le denomina "*orden del árbol"*. Existen, por definición, los "*nodos hojas"*, son aquellos nodos que no poseen nodos sucesores, en estos casos los punteros contienen algún valor considerado inválido. En la aplicación práctica, también existe otra información almacenada junto a la clave, la referencia a un registro que contiene el resto de la información asociada con la clave.

La Figura 3.1 presenta el formato gráfico de un nodo para un árbol B de orden M.

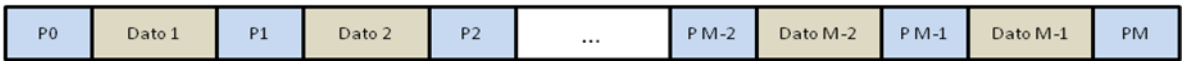


Figura 3.1. Estructura grafica de un nodo para un árbol B. Puede contener como máximo M punteros y M – 1 elementos.

La Figura 3.1 se puede representar mediante la siguiente definición de estructura de datos [2]:

```
Const orden = 255;
Type reg_arbol_b = record;
       hijos : array [0..orden] of integer;
       claves: array [1..orden] of tipo_de_dato;
       nro_registros: integer;
     End;
```

Donde “*hijos”* es un arreglo que contiene la dirección de los nodos que son descendientes directos, “*claves*” es un arreglo que contiene las claves que forman el índice del árbol y “*nro_registros”* indica la dimensión efectiva en uso de cada nodo, es decir, la cantidad de elementos del nodo. Notar que el vector de “*hijos*” es un elemento más grande que el vector de “*claves*”, además, el orden del árbol en este caso es 256.

En el momento inicial, el único nodo existente en el árbol será el nodo raíz, el cual no contendrá ningún elemento. La Figura 3.2 presenta la representación gráfica de un nodo raíz, de un árbol B de orden 4, sin elemento alguno.

Figura 3.2. Nodo raíz de orden 4 inicialmente vacío.

Los elementos comenzarán el proceso de inserción a partir del nodo raíz. Si el nodo raíz tiene lugar, se procederá a insertarlos en este nodo teniendo en cuenta que los elementos de datos deben quedar ordenados dentro del nodo. Este proceso no resulta costoso, dado que el registro que contiene al nodo raíz está almacenado en memoria, por lo tanto su ordenación sólo debe contabilizar accesos a memoria RAM.

Cuando el nodo se completa, la llegada de un nuevo elemento provocará desborde u overflow. Esto significa que en el nodo no hay capacidad disponible para almacenar un nuevo elemento, situación que debe ser resuelta.

El nodo es dividido en dos nodos y se distribuyen las claves entre ambos los más equitativamente posible. Puesto que ahora se tienen dos nodos hojas, es necesario crear un nivel superior en el árbol para que, cuando se realice una búsqueda se pueda elegir entre ambas hojas. Es necesario crear una nueva raíz, para ello, se promueve una nueva clave que separe ambas hojas. Este proceso se denomina inserción, división y promoción [2].

La Figura 3.3 representa la situación descripta anteriormente.

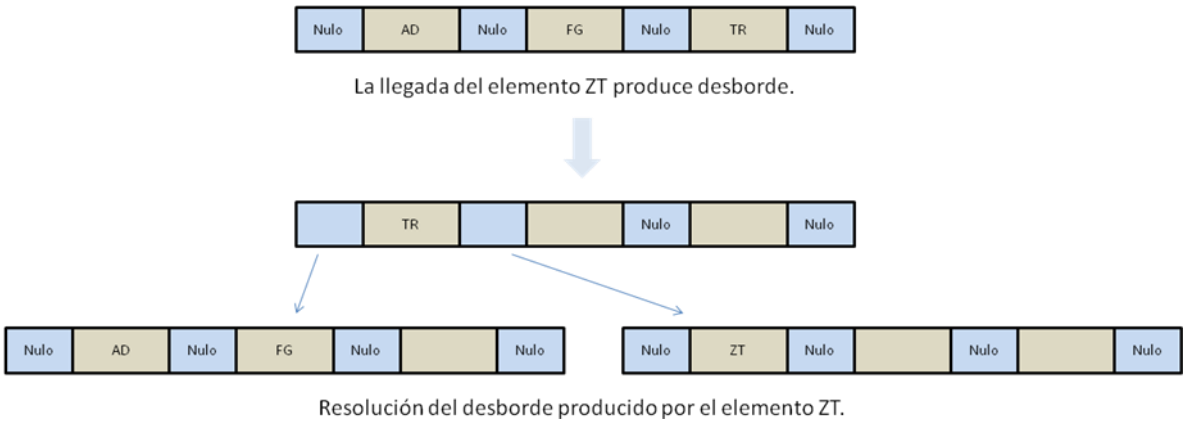


Figura 3.3. Proceso de inserción con desborde, división y promoción.

Cada vez que se intenta agregar una nueva clave se comienza con una búsqueda que llega hasta el nivel hoja y después se busca el lugar para la nueva clave, se continúa con el proceso de inserción, división y promoción en forma ascendente desde abajo [1].

El proceso continúa con la inserción de elementos en el árbol de la misma forma; cuando el árbol crece en altura es la raíz la que se aleja de los nodos terminales, de manera tal que siempre el árbol crezca de manera balanceada [2].

Las definiciones de términos tales como *orden* y *hojas* permiten establecer en forma precisa las propiedades que deben estar presentes en una estructura de datos para cumplir los requisitos de un árbol B.

Es posible formular una definición precisa de las propiedades de un árbol B de orden M [10]:

- Cada nodo tiene a lo sumo M descendientes.
- Cada nodo, excepto la raíz y las hojas, tiene por lo menos [M/2] descendientes.
- La raíz tiene por lo menos dos descendientes (a menos que sea una hoja). Cuando el árbol contiene un solo nodo.
- Todas las hojas aparecen en el mismo nivel.
- Un nodo que no es hoja con K descendientes contiene K-1 claves.

- Un nodo hoja contiene por lo menos [M/2]-1 claves y no más de M-1 claves.

El proceso algorítmico de búsqueda de un elemento en un árbol balanceado no difiere demasiado del mismo proceso en un árbol binario común. Comienza la búsqueda en el nodo raíz y se va bifurcando hacia los nodos terminales en la medida que el elemento no sea localizado.

Un árbol balanceado tiene como principal característica que todos los nodos terminales se encuentran a la misma distancia del nodo raíz. De esta forma, se puede asegurar que la performance de búsqueda dentro de esta estructura, siempre se encuentra acotada por el orden de búsqueda logarítmico con base equivalente al orden del árbol.

Si se tratara de un árbol de balanceado de orden 256, con un millón de elementos que lo conformen, en el peor de los casos, la altura del árbol (y por ende la cantidad de accesos máxima necesaria para recuperar la información) está acotado a 4 accesos [1] [10].

Se han considerado hasta el momento las operaciones de búsqueda e inserción sobre un árbol B. Estas dos operaciones, si bien pueden ser consideradas las más importantes desde un punto de vista de utilización (recordar que en promedio el 80% de las operaciones sobre un archivo son de consultas y que, del 20% restante, la mayoría la representan operaciones de inserción), no son las únicas operaciones posibles. Otra operación a tener en cuenta consiste el proceso de baja de información.

Para el caso de las eliminaciones de claves en un árbol B, se pueden resumir en los siguientes pasos [1] [2]:

1. Si la clave a eliminar no está en una hoja, se debe intercambiar con su sucesor inmediato, el cual está en una hoja.

2. Se elimina la clave.

3. Si la hoja ahora contiene por lo menos el número mínimo de claves, no se requiere ninguna acción adicional, pero si la hoja queda con menos elementos que el mínimo de claves permitidas, se deben examinar los hermanos izquierdo y derecho, en ese orden preferentemente.

a. Si un hermano tiene más elementos que el número mínimo de claves, se debe realizar una redistribución de elementos entre el nodo hermano y el nodo con la clave a eliminar.

b. Si ninguno de los hermanos tiene más elementos que el mínimo de claves, se concatenan las dos hojas (el nodo hoja con la clave a eliminar y un nodo hoja hermano, denominado hermano adyacente) y una clave del padre, la que se encuentre en la mitad.

4. Si las hojas se concatenaron, aplicar los pasos 3 al 5 al padre.

Algunas conclusiones

Los árboles balanceados representan una buena solución como estructura de datos, para implementar el manejo de índices asociados a archivos de datos.

Se debe notar que cualquier operación (consulta, alta, baja o modificación) se realiza en términos aceptables de performance. Se puede considerar entonces, que este tipo de estructura representa una solución viable para almacenar índices.

Siempre se debe tener en cuenta que las estructuras de árboles B serán utilizadas para administrar los índices asociados a claves primarias, candidatas o secundarias. Dichos archivos de índices contendrán la estructura presentada, clave, hijos y referencia del resto del registro en el archivo de datos original. El archivo de datos original se plantea como un archivo serie donde cada elemento se inserta siempre al final.

Árboles B* y B+

Como desprendimiento de los árboles B, aparecen tanto los árboles denominados B* y B+. Los primeros son árboles B de construcción especial donde cada nodo se completa hasta *2/3* de capacidad como mínimo, aumentando el porcentaje de ocupación de cada nodo. De esta forma, los árboles B* plantean una alternativa a los árboles B generando una estructura más eficiente en cuanto a la utilización de espacio.

En los arboles B* mejoran la eficiencia de los arboles B, aumentando la cantidad mínima de elementos en cada nodo, por lo tanto ayuda a disminuir la altura final del árbol. Como contraparte, en esta clase de árboles las operaciones de inserción resultan un poco más lentas.

Los árboles B* son árboles que al ocupar cada nodo con más elementos, crecen más lentamente. La ventaja de este tipo de estructuras radica en que la altura es menor y por consiguiente la performance de búsqueda de un elemento es mejor [1] [2] [10].

Un árbol B* de orden M tiene las siguientes propiedades:

- Cada nodo tiene un máximo de M descendientes.
- Cada nodo, excepto la raíz y las hojas, tiene al menos [(2M-1)/3] descendientes.
- La raíz tiene al menos dos descendientes, a menos que sea una hoja.
- Todas las hojas aparecen en el mismo nivel.
- Un nodo que no sea hoja con K descendientes contiene K-1 claves.
- Un nodo hoja contiene por lo menos [(2M-1)/3] claves y no más de M-1.

Por último, se presentan los árboles B+. Este tipo de estructuras presenta dos ventajas: la búsqueda eficiente de información (a través de un árbol balanceado) y el acceso secuencial a bajo costo (a través de un conjunto hojas de nodos enlazados, que constituyen una lista). De esta forma es posible asegurar el cumplimiento de los dos preceptos básicos que definen los índices, la búsqueda aleatoria eficiente de información y poder acceder a dicha información en forma ordenada. Los árboles B+ son una clase particular de árboles B donde se cumplen las siguientes propiedades [10]:

- Cada nodo del árbol puede contener como máximo M descendientes y M-1 elementos.
- La raíz no posee descendientes o tiene al menos dos.
- Un nodo con k descendientes contiene k-1 elementos.
- Los nodos terminales tiene como mínimo [M/2] - 1 elementos y como máximo M-1 elementos.
- Los nodos que no son terminales ni raíz tienen como mínimo [M/2] descendientes.
- Todos los nodos terminales se encuentran al mismo nivel.

- Los nodos terminales representan un conjunto de datos y están vinculados constituyendo al mismo tiempo una estructura lineal (lista enlazada).

Capítulo 4

Dispersión (HASHING)

4.1 Introducción

La velocidad de respuesta, la eficiencia en la búsqueda de información, la cantidad de accesos a memoria secundaria y otros conceptos, fueron y son motivos de investigación con el objetivo de obtener mejores prestaciones. Obviamente, las bases de datos no quedan fuera de estos términos al momento de persistir, organizar y gestionar la información.

El acceso a la información en memoria secundaria es considerado un proceso extremadamente lento (en términos relativos) comparado con las altas velocidades de acceso a memoria principal o primaria; por lo tanto, cuantos menos accesos para encontrar un elemento se realicen a memoria secundaria, más velozmente se tendrá la respuesta deseada.

Existen tres modos principales de acceder a la información en un sistema de archivos que integra una base de datos: en forma *secuencial*, a través de un *índice (árboles)*, y de manera *directa*.

En general, al utilizar índice los costos adicionales en el mantenimiento de archivos pueden reducirse lo suficiente como para que sean aceptables, pero hay ocasiones en que la demanda de un sistema de archivos es tan extrema que dichos costos se vuelven intolerables, es decir, resulta absolutamente necesaria una forma de acceso que requiera, en promedio, menos de dos accesos a memoria secundaria por cada operación.

Cuando la consideración más importante del diseño es obtener un acceso rápido para obtener un elemento de un archivo, el acceso directo utilizando una clave, si pudiera llevarse a cabo, es el objetivo deseado. La dispersión es el método más efectivo utilizado para tal fin.

En teoría, la dispersión hace posible encontrar cualquier elemento de un archivo con sólo un acceso a disco.

En la práctica real, se utilizan, en promedio, menos de dos accesos a memoria secundaria para obtener el elemento buscado, es decir, gran parte de las operaciones se logran con un solo acceso, mientras algunas búsquedas necesitan más de un acceso [1] [2].

Dispersar un archivo

Dispersar consiste en aplicar una función *F(k)*, donde *k* es una cadena o valor numérico, generalmente denominado clave. Esta función se denomina *Función de Dispersión o Función de Hash*, y retorna como resultado una dirección válida para un rango de direcciones determinadas, es decir, *0* <= *F(k)* <= *M* (siendo M es el tamaño máximo del espacio de direcciones permitido). La dirección generada por la función de dispersión se utiliza como base para la búsqueda, inserción y eliminación de elementos en un archivo. Muchas veces al término dispersión se lo denomina "*aleatorización"* debido que no existe una relación obvia inmediata entre la clave que se está dispersando y la dirección resultante luego de haberle aplicado dicha función.

Uno de los primeros problemas que surgen cuando se utiliza dispersión es que dos o más claves diferentes, aplicando la misma función de dispersión, pueden tener como resultado la misma dirección, es decir, que los elementos potencialmente serán enviados a la misma posición en el archivo. A esta situación se la conoce como "*colisión"* y se debe encontrar la forma de resolverla [1].

Cuando *F(k) = F(y)*, donde k<>*y*, se dice que k e y son claves "*sinónimas"*.

Tipos de dispersión

La dispersión presenta dos alternativas para su implantación en la memoria secundaria, el tratamiento de espacio en forma "*estática"* y el tratamiento de espacio en forma "*dinámica"*.

Cuando se utiliza espacio de direccionamiento estático el espacio disponible, para dispersar los elementos del archivo, está fijado previamente. La función de dispersión retorna como resultado una dirección válida para dicho espacio.

Cuando se utiliza espacio de direccionamiento dinámico el espacio para dispersar los elementos del archivo aumenta o disminuye en función de las necesidades de espacio que en cada momento tiene el archivo. En esta implantación, la función de dispersión

genera un valor intermedio que será utilizado para obtener una dirección física posible para el archivo. Estas direcciones físicas no están establecidas a priori, sino que son generadas de manera dinámica [2].

4.2 Parámetros que afectan a la eficiencia de la dispersión

Cuando se utiliza dispersión en un espacio de direcciones estático, existen un conjunto de parámetros esenciales que afectan la eficiencia y definen su comportamiento. Ellos son: la *"función de dispersión"*, el *"capacidad de cada dirección de memoria"*, la *"densidad de empaquetamiento"* y el *"tratamiento de las colisiones"*.

4.2.1 Función de dispersión

El objetivo de una función de dispersión es esparcir las claves lo más uniformemente posible en el intervalo de direcciones disponibles. Hay tres modos de posibles de distribución para una función de dispersión.

Idealmente, existe una función de dispersión que distribuye las claves de manera que no exista colisión alguna; esta distribución se denomina "*uniforme" o "función de dispersión perfecta"*. Este tipo de funciones son muy difíciles de encontrar cuando se van a dispersar una gran cantidad de elementos.

Otra alternativa la constituye aquella función de dispersión que otorgue a todas las claves una misma dirección, dando como resultado el máximo número de colisiones. Este es el peor caso de distribución.

En la práctica se puede encontrar un punto medio con aquellas funciones de dispersión que dan como resultado una distribución "*aleatoria"* de las claves, es decir, que las claves se encuentran esparcidas y con pocas colisiones.

La Figura 4.1 presenta gráficamente el resultado de las tres posibles distribuciones explicadas anteriormente [1].

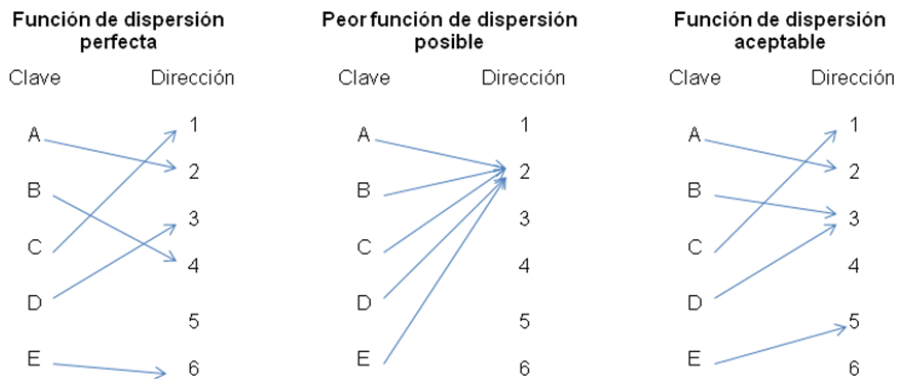


Figura 4.1. Distribuciones posibles.

Las funciones de dispersión utilizadas más comunes son aquellas que producen una distribución aleatoria con muy pocas colisiones, aún cuando trabajan bajo un acotado espacio de direcciones. Algunas de ellas son:

- **Módulo tamaño de la memoria**: esta función divide la clave por el tamaño del espacio de direcciones disponible, y luego utiliza el resto resultante de dicha división como dirección física para la clave en cuestión. Suele ocurrir, en casi la mayoría de las funciones de dispersión, que generalmente se utiliza el resto en alguna parte para producir una dirección en el intervalo de direcciones correspondiente. Para obtener mejores resultados, es conveniente que la división se realice con un número primo, ya que genera mejores resultados [1] [2].

- **Centros cuadrados**: esta función implica elevar la clave al cuadrado y luego extraer los dígitos centrales para ser ajustados al espacio de memoria disponible mediante la división y el uso del resto como dirección física en donde se almacenará la clave correspondiente. Es importante aclarar que el método original de centros cuadrados, utiliza solamente los dígitos centrales como dirección física donde será almacenada la clave, es decir, que no aplica la división, ni la utilización del resto. Sin embargo, cuando se utiliza un espacio de direcciones pequeño se debe aplicar tal división para generar una dirección válida [1].

- **Transformación de la base**: esta función convierte la clave a alguna otra base numérica que no sea con la que se está trabajando, luego se debe dividir la clave, ya transformada, por el total de direcciones disponibles y utilizar el resto como dirección física para almacenarla.

A modo de ejemplo, a continuación se presenta el pseudocódigo[5] de una de las funciones de dispersión:

```
Función hash ("clave")
  Si (la "clave" es alfanumérica) entonces
    Transformar la "clave" en una "clave" numérica
  Fin del Si.
  Dividir la "clave" por el tamaño de la memoria y obtener el
resto de la división.
  Retornar el resto de la división como resultado final.
Fin de la función.
```

Esta función es muy simple y representa la función de dispersión denominada "*módulo tamaño de memoria*". Como toda función de dispersión, trabaja con una clave numérica, por lo tanto si la clave que recibe no es numérica se transforma previamente en una clave numérica, y finalmente se retorna el resto del cociente entre la clave y el tamaño del espacio memoria. Dicho resultado representa una dirección válida del espacio de direcciones utilizado.

4.2.2 Almacenar más de una clave por dirección

Cuando la computadora recibe información del disco, para el sistema de E/S (Entrada/Salida – Sistema que traslada un bloque de información del almacenamiento físico a alguna de las memorias primarias [22]), es igualmente fácil transferir un conjunto de claves que transferir una clave. Si bien el tamaño de transferencia es limitado, se puede lograr trasladar un conjunto de claves en el mismo instante de tiempo.

Si se solicita una clave, la dirección base se determina por dispersión. Un conjunto de claves perteneciente a dicha dirección será trasladado a memoria principal, para luego intentar encontrar la clave solicitada.

Una de las ventajas más importantes de poder almacenar más de una clave por dirección, es que permite soportar una o más colisiones (claves distintas que por función de dispersión van a la misma dirección física del archivo), dependiendo de la cantidad de claves que se permitan almacenar en una dirección.

[5] En ciencias de la computación, y análisis numérico el pseudocódigo (o falso lenguaje) es una descripción de un algoritmo de programación informático de alto nivel compacto e informal que utiliza las convenciones estructurales de un lenguaje de programación verdadero, pero que está diseñado para la lectura humana en lugar de la lectura en máquina, y con independencia de cualquier otro lenguaje de programación [23].

Una vez que la dirección se encuentra completa, puede ocurrir lo que se denomina saturación o desborde. Esto sucede con menor frecuencia si se permite almacenar más de una clave por dirección.

Lograr tener direcciones con capacidad para más de una clave afecta el concepto denominado *densidad de empaquetamiento*. La relación tiene su fundamento en que para calcular la densidad de empaquetamiento de un archivo se debe tener en cuenta la cantidad de direcciones así como también cuantas claves puede albergar cada una de ellas.

Elegir el tamaño de dirección adecuado no es una tarea sencilla, por un lado se tiene la limitación del tamaño de transferencia desde la memoria principal hacia la memoria secundaria y viceversa. Por otro lado, se deben estudiar las diversas organizaciones del archivo, es decir por ejemplo, si es mejor tener 1000 direcciones con capacidad para una sola clave o 500 direcciones con capacidad para dos claves [1].

4.2.3 Densidad de Empaquetamiento (DE)

La DE es la proporción entre el número de registros (*R*) por almacenar y el número de lugares disponibles (*N*) [1], [2].

Si cada dirección puede almacenar una sola clave, entonces:

$$DE = R / N$$

La fórmula de la DE se ve afectada si cada dirección puede almacenar más de una clave por dirección. En este caso el espacio disponible se debe definir como la cantidad de direcciones *N*, multiplicada por la cantidad de claves que puede almacenar cada una de éstas direcciones *CCD (Cantidad de Claves por Dirección)*. En este caso la DE se calcula como:

$$DE = R / (CCD \times N)$$

La DE es una medida de la cantidad del espacio que se usa en realidad en un archivo. Es un indicador importante para evaluar un ambiente de dispersión.

Si se tienen 10 claves esparcidas en un rango de 19 direcciones habrá una probabilidad menor de que suceda una colisión que si se tienen esas mismas 10 claves esparcidas en un rango de 29 direcciones [1].

Se puede concluir, entonces, que cuanto menor sea la DE menor será la probabilidad de que ocurran colisiones, pero hay que tener en cuenta que habrá mayor cantidad de direcciones sin ocupar.

Se debe decidir cuánto espacio extra se está dispuesto a utilizar para lograr reducir el número de colisiones. Si bien es deseable minimizar las colisiones, no se debería asignar al archivo un número de direcciones excesivo que deje mucho lugar sin ocupar.

4.2.4 Tratamiento de colisiones

Como primera medida se debe tratar de buscar una función de dispersión que produzca el menor número de colisiones posible. Anteriormente se menciono que un ideal es encontrar una función de dispersión que evite por completo las colisiones, función que se denomina "*función de dispersión perfecta"*. El problema es que encontrar esta función es casi técnicamente imposible cuando se desea dispersar una cantidad razonable de claves. Se sabe que de $10^{120.000}$ funciones de dispersión sólo una de ellas podría ser la perfecta buscada, por ello, no vale la pena el esfuerzo [1].

Se debe elegir y utilizar una función de dispersión que distribuya las claves los más aleatoriamente posible para evitar tener un gran número de colisiones, reduciendo a éstas a un número aceptable.

Acompañando a la elección de una función de dispersión aceptable, se puede utilizar memoria adicional, debido a que seguramente se produzcan menos colisiones cuando se cuenta con muchas más direcciones que claves a dispersar.

Más allá de las medidas que se tomen para evitar y reducir las colisiones, seguramente se presentará el problema de que a una dirección que se encuentre completa le ocurra una colisión, tal situación se la denomina "*Saturación*" o "*Desborde*". La clave no podrá ser almacenada en su correspondiente dirección base y deberá ser reubicada en otra dirección. Para resolver esta situación existen diversos métodos que se explicarán más adelante.

4.3 Predicción de la cantidad de claves en saturación o desborde.

Cuando se utiliza una función de dispersión aleatoria, cualquier dirección del espacio disponible tiene la misma probabilidad de ser elegida que las demás. Si una cierta dirección es otorgada a una clave, no se incrementa ni se disminuye la probabilidad de que sea nuevamente asignada a cualquier otra clave. Es de esperar que algunas direcciones sean elegidas más de una vez y otras nunca sean elegidas.

Lograr una distribución uniforme (sin colisiones) es prácticamente imposible, en la mayoría de los casos algunas claves no estarán en su dirección base y estarán ocupando otra dirección en el archivo. Existe una forma de poder estimar cuántas claves pueden ser asignadas a una dirección determinada, es decir, cuántas colisiones pueden llegar a ocurrir, y por lo tanto, un valor aproximado de cuantas claves en saturación se tendrá.

En un espacio de *N* direcciones, cuando se dispersa una clave existen dos posibles resultados respecto a cualquier dirección dada:

1. que la dirección sea elegida (*A*), o
2. que no sea elegida (*B*).

En términos de probabilidades, se puede decir que la probabilidad de que una dirección sea elegida entre *N* direcciones es:

$$P(A) = a = 1 / N$$

La dirección tiene una oportunidad entre N de ser elegida.

De manera similar, la probabilidad de que la dirección no sea elegida es:

$$P(B) = 1 – P(A) = b = 1 – 1/N$$

Si *N* = 10 y se dispersa una clave, la probabilidad de que la dirección "*y*" sea elegida es *P(A) = 1 / 10 = 0,1* y la probabilidad de que la dirección "*y*" no sea elegida es P*(B) = 9 / 10 = 0.9*. Entonces, a = 0,1 y b = 0,9.

Al dispersar dos claves, la probabilidad de que ambas claves vayan a la misma dirección resulta en el producto matemático de ambas probabilidades individuales (por tratarse de eventos independientes). Por lo tanto, tal probabilidad es:

$$P(AA) = P(A) \times P(A) = (1 / N) \times (1 / N) = 1 / N^2$$

Para complementar al resultado anterior, existen otras combinaciones posibles. Que la primer clave sea asignada a una dirección determinada y la segunda clave no, o que la primer clave no sea asignada a una dirección determinada y la segunda si, o que ambas sean asignadas a una dirección determinada. Estos eventos son: *P(AB)*, *P(BA)* y *P(BB)* respectivamente.

En general, se puede armar la secuencia de combinaciones de los eventos *A* y *B* que se desee, sin importar el orden en que sucedan.

Dispersar tres claves y saber cuál es la probabilidad de que sean asignadas sólo dos claves a una dirección determinada es: *P(AAB)*, *P(ABA)* y *P(BAA)*. Estas tres secuencias son independientes entre sí, la probabilidad de que sucedan dos eventos *A* y un evento *B* resulta en la suma de las probabilidades de cada resultado individual.

"*2A y 1B = P(AAB) + P(ABA) + P(BAA)*", se suman todas las combinaciones en que dos A y un B pueden distribuirse en tres lugares.

En base a lo indicado anteriormente, se puede expresar de manera genérica la probabilidad de cada combinación que se desee encontrar. La probabilidad queda conformada de la siguiente manera:

$$a^k \times b^{(r-k)}$$

Donde "*r*" representa la cantidad de claves a dispersar y "*k*" la cantidad de veces que una dirección determinada sea asignada (cantidad de eventos *A*) luego de dispersar las "*r*" claves.

Se utiliza la fórmula del número de formas posibles para seleccionar "*k*" cosas entre un conjunto de "*r*" cosas:

$$C = r! \ / ((r-k)! \times k!)$$

Se puede expresar la probabilidad de que se elija una dirección "*k*" veces y no sea elegida "*r-k*" veces, cuando se tienen "*r claves*", como:

$$P(k) = C \times (a^k \times b^{(r-k)})$$

Para *k = 0*, se calcula la probabilidad de que una dirección no tenga claves asignadas luego de dispersar todas las claves.

Para *N = 10* y *r = 7*, entonces, la probabilidad de que una dirección no tenga claves asignadas es:

$P(0) = (7! / ((7 - 0)! \times 0!)) \times ((1 / 10)^0 \times (9 / 10)^{(7-0)}) = (0,9)^{(7)} = 0,478.$

La desventaja de ésta fórmula, es que resulta casi imposible calcularla cuando se tienen muchas direcciones y muchas claves a dispersar.

Cuando se tienen valores grandes de direcciones y de claves a dispersar, existe una función que da una muy buena aproximación a *P(k)* y que resulta mucho más fácil de calcular. Esta función se llama "*Función de Poisson*" y se expresa de la siguiente manera:

$$P(k) = ((r / N)^k \times e^{-(r/N)}) / k!$$

N, "*r*", "*k*" y *P(k)* significan lo mismo que en los planteos anteriores.

Para *N = 5000* y *r = 3500*, se calcula *P(0)*, *P(1)* y *P(2)*:

$P(0) = ((3500 / 5000)^0 \times e^{-(3500/5000)}) / 0! = e^{-(0,7)} = 0,497$. Aproximadamente el 50% de las direcciones no tendrán claves almacenadas.

$P(1) = ((3500 / 5000)^1 \times e^{-(3500/5000)}) / 1! = 0,7 \times e^{-(0,7)} = 0,3479$. Aproximadamente el 35% de las direcciones tendrán exactamente una sola clave asignada, es decir, que esas direcciones no tendrán sinónimos.

$P(2) = ((3500 / 5000)^2 \times e^{-(3500/5000)}) / 2! = (0,49 \times e^{-(0,7)}) / 2 = 0,121$. Aproximadamente el 12% de las direcciones tendrán exactamente dos claves asignadas, es decir, que esas direcciones tendrán exactamente un sinónimo.

Cuanto más grande es la cantidad de claves asignadas a una dirección determinada, menor será el valor de probabilidad de acceder a esa dirección. Generalmente, para "*k*" > 5 las probabilidades son realmente muy bajas, y no tiene sentido analizarlas.

Utilizando la *Función de Poisson* se puede predecir el número de direcciones que tendrán un número dado de claves asignadas. En general, si hay *N* direcciones, entonces, el número esperado de direcciones con "k" cantidad de claves asignadas a ella es:

N x P(k)

P(k) da la proporción de direcciones que tienen *k* claves asignadas a ellas cuando se ha aplicado la función de dispersión a todas las claves.

Poder predecir la proporción de direcciones que tendrán una cierta cantidad de claves asignadas cuando se utiliza una función de dispersión aleatoria, permite estimar el número de colisiones que pueden ocurrir y por lo tanto, es posible determinar cuando éste número de colisiones comienza a afectar la performance de un ambiente de dispersión determinado.

Estimar de antemano cuantas claves podrán ser asignadas en una dirección brinda la posibilidad de conocer cuántas colisiones se pueden tolerar antes de que suceda una *saturación* o *desborde* (*overflow),* situación que debe ser solucionada y que afecta el desempeño global de la dispersión [1] [2].

4.4 Tratamiento de colisiones con saturación o desborde (overflow).

Cuando el número de colisiones en una determinada dirección iguala a la cantidad de claves que la misma puede almacenar, ocurre un problema denominado *saturació*n o *desborde* (*overflow*). La clave a insertar en el archivo no puede almacenarse en su dirección base y se debe decidir donde realmente se almacenará.

Si cada dirección tiene capacidad para almacenar una sola clave, cada vez que ocurra una colisión sobre esa dirección se producirá saturación. Si cada dirección tiene capacidad para dos claves, se necesita de dos colisiones para que ocurra saturación. Si cada dirección tiene capacidad para tres claves se necesita de tres colisiones para que ocurra saturación. Entonces, si "*c*" es la capacidad de una dirección, "*c-1*" es la cantidad de colisiones que puede tolerar esa dirección antes de que ocurra una saturación.

El concepto de colisión esta linealmente relacionado con el concepto de saturación o desborde. Si se logra reducir la probabilidad de que ocurran colisiones, se logrará reducir la probabilidad de tener claves en saturación.

Una clave en saturación es una clave fuera de su dirección base. Se encuentra almacenada en una dirección distinta a la que se obtuvo al aplicarle la función de dispersión.

Existen diversas técnicas para ubicar y almacenar estas claves que se encuentran fuera de su dirección base, estás técnicas se denominan: *"Técnicas de resolución de colisiones con saturación"* o simplemente *"Técnicas de resolución de colisiones"*. Ambas denominaciones son aceptadas y dependen de la bibliografía consultada.

Aunque se cuente con un amplio rango de direcciones y se tengan que dispersar muy pocas claves, un ambiente de dispersión debe estar preparado para solucionar problemas de saturación, para lo cual, se debe decidir que técnica de resolución de colisiones se utilizará para cuando ocurra tal problema.

Se detallan cuatro métodos para un espacio de direccionamiento estático: *saturación progresiva, saturación progresiva encadenada, saturación progresiva encadenada con área de desborde por separado y doble dispersión*.

4.4.1 Saturación progresiva

Inserción

El método de saturación progresiva es el método más sencillo para reubicar una clave en saturación y consiste en almacenar dicha clave en la dirección siguiente más cercana a su dirección base.

Si al insertar una clave se produce saturación se busca secuencialmente en las direcciones consecutivas hasta encontrar alguna con lugar libre. La clave se almacena en esta nueva dirección. Para aquellos casos en el que la búsqueda llegue al final del espacio de direcciones disponible, se continúa por la primera dirección, dando la vuelta a dicho espacio de direcciones.

La Figura 4.2 presenta el resultado de dispersar 5 claves. El tamaño del espacio de disponible es de 11 direcciones, la capacidad de cada dirección es para una clave y la función de dispersión es el módulo 11 (tamaño del espacio de direcciones).

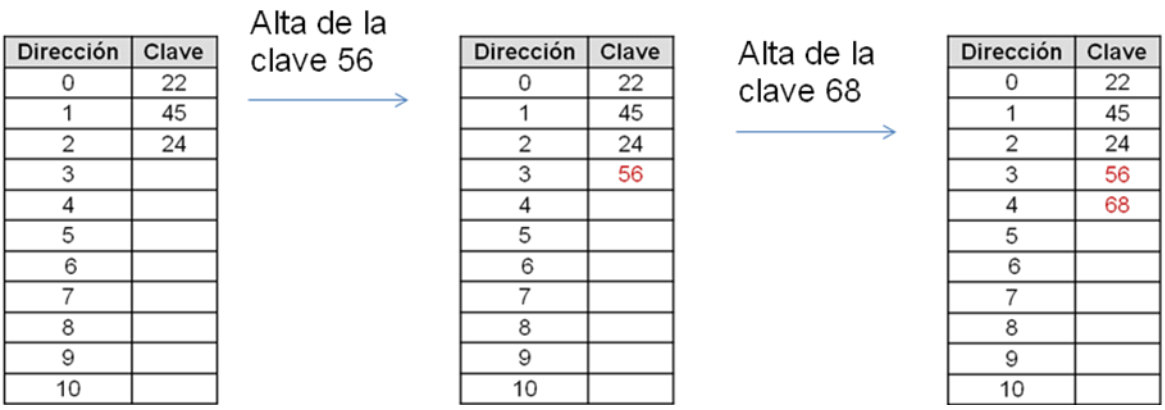


Figura 4.2. Resultado de dispersar las claves 45, 24, 22, 56 y 68. Las claves en color rojo son claves en saturación, están fuera de su dirección base.

En dicho ejemplo, las claves 45, 24 y 22 no producen saturación y son almacenadas en sus respectivas direcciones base. Las claves 56 y 68 producen saturación y se almacenan fuera de sus direcciones base, siguiendo la técnica de saturación progresiva.

La dirección base de la clave 56 es la dirección 1, dirección que se encuentra completa, entonces, se busca secuencialmente un lugar libre pasando por la dirección 2 que también está completa y luego por la dirección 3 que tiene un lugar libre y efectivamente la clave 56 es almacenada en dicha dirección. Lo mismo ocurre con la clave 68 pero partiendo desde su dirección base 2.

Si el buscar un lugar libre provoca que se llegue al final del archivo, entonces, se continúa por la dirección inicial del espacio de direcciones correspondiente.

Búsqueda

La búsqueda de una clave en la saturación progresiva procede de la misma manera que la inserción. Se inicia en la dirección base de dicha clave, si no es encontrada, se busca en las direcciones consecutivas hasta que sucedan algunas de las siguientes condiciones [1] [2]:

- La clave es encontrada. Es una clave en saturación.
- Se encuentra una dirección vacía. La clave no fue encontrada.

- Se retorna a la dirección base (dirección inicial). Esto quiere decir que se reviso todo espacio de direcciones. La clave no fue encontrada.

Eliminación

La eliminación de una clave, independientemente de la técnica de resolución de colisiones que se utilice, implica considerar dos aspectos:

1. que el lugar que se libere no sea de obstáculo para futuras búsquedas, y
2. que el espacio que se libere pueda ser reutilizado por futuras inserciones.

En particular, en saturación progresiva, si en el proceso de eliminación no se toman ciertas precauciones, puede acarrear problemas para encontrar posteriormente algunas claves.

La Figura 4.3 describe el caso en que la baja de una clave puede hacer que se pierdan otras claves.

Las clave 56, es una clave en saturación, su dirección base es la 1. El problema ocurre cuando al eliminar la clave 24, se libera la dirección 2, dejando dicho lugar vacío. Al intentar buscar la clave 56, nunca podrá ser encontrada, debido a que la búsqueda se detendrá en la dirección 2, asumiendo que la clave no existe en el archivo.

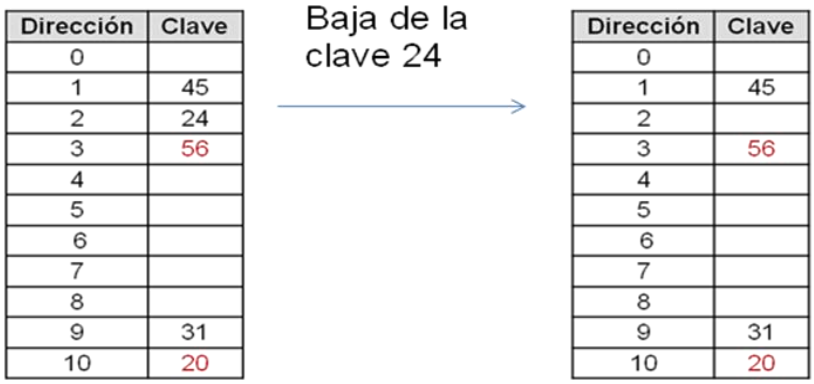


Figura 4.3. Eliminación de la clave 24. Pérdida de la clave 56.

Una técnica sencilla para evitar el problema descripto y perder claves en saturación es insertar marcas denominadas: *marcas de inutilización o marcas de eliminación (####)*.

Estas marcas de eliminación no solo indican que hay un lugar disponible para futuras inserciones, sino que además indican que la dirección alguna vez estuvo con su capacidad al máximo, pudiendo existir claves en saturación para dicha dirección. La marca de eliminación no indica desborde sino que indica que la dirección alguna vez se completó y que puede existir alguna clave en saturación mas adelante en el archivo [2].

La Figura 4.4 presenta como queda el archivo luego de eliminar la clave 24 cuando se utilizan marcas de eliminación. Ahora, al buscar la clave 56, la misma podrá ser localizada en la dirección 3 del archivo.

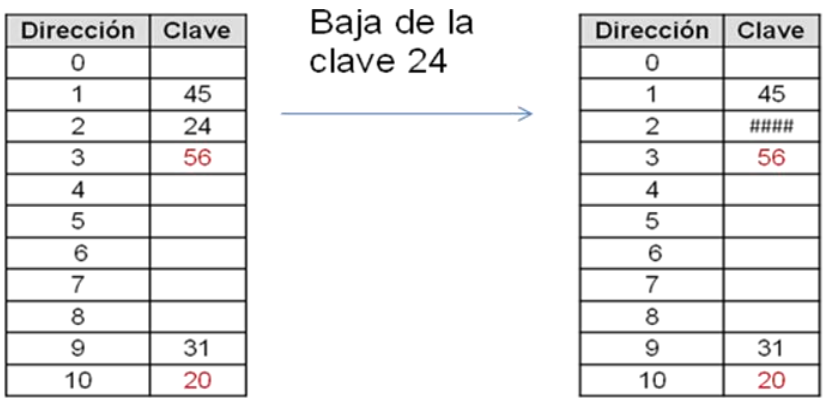


Figura 4.4. Resultado correcto para la eliminación de la clave 24. No se produce la pérdida de clave alguna.

Si alguna clave se asigna a la dirección 2, es almacenada sin ningún inconveniente. Si bien la dirección 2 contiene una marca de eliminación, esto no impide su reutilización como lugar disponible.

No siempre que ocurre el borrado de una clave es necesario introducir una marca de eliminación.

En la Figura 4.4, si se elimina la clave 56, no es necesario establecer una marca, pues, aunque la dirección 3 está completa, como la dirección siguiente, dirección 4, está vacía y no posee marca de eliminación, no se pierde nada con dejar el lugar vacío sin marca de eliminación, es inapropiado colocar una marca de eliminación cuando la misma no es necesaria.

Conclusión

La saturación progresiva es una técnica muy sencilla y rápida de implementar, pero presenta un problema. Si se generan muchas colisiones, habrá muchas claves en saturación ocupando espacios que no deben. Esto provoca cúmulos o conjunto de claves en un sector del archivo. A medida que se producen colisiones, las claves se van almacenando a mayor distancia de su dirección base, de tal manera que se requieren varios accesos a disco para encontrar alguna de dichas claves en saturación.

Si en cada acceso a la memoria secundaria se puede traer una sola dirección, entonces, en la Figura 4.1 encontrar la clave 56 va a requerir tres accesos. Es decir, que cuanto más lejos se encuentre una clave de su dirección base más accesos se requerirán para encontrarla.

A continuación se plantean otras técnicas de resolución de colisiones que tratan de evitar los problemas de que claves sinónimas se encuentren a gran distancia de sus respectivas direcciones base.

4.4.2 Saturación Progresiva Encadenada

El método de saturación progresiva encadenada es una variante de la saturación progresiva, diseñada para evitar los problemas causados por la acumulación de claves.

Inserción

Trabaja igual que el método anterior, cuando la función de dispersión, aplicada a una clave, genera una dirección sin espacio disponible, la misma es ubicada en la dirección inmediata siguiente con espacio disponible. La diferencia radica en que, una vez localizada la nueva dirección y almacenada la clave, ésta se encadena o enlaza con la dirección base inicial, generando una cadena o lista de claves que son sinónimas.

Cada dirección de una cadena de sinónimos contiene la dirección de la siguiente clave que por función de dispersión ha generado la misma dirección base en el archivo. El efecto neto de tener una lista ligada de sinónimos es que ante un requerimiento de búsqueda solo se necesita acceder a las claves enlazadas por los punteros, reduciendo así, los accesos a la memoria secundaria.

El método de saturación progresiva encadenada debe tener en cuenta dos aspectos:

1. debe agregarse en cada dirección un enlace, por lo que requiere almacenamiento adicional, y

2. debe poder garantizar llegar a cualquier sinónimo partiendo desde su dirección base.

La Figura 4.5 muestra la estructura de un archivo para la saturación progresiva encadenada en donde se han dispersado inicialmente las claves 34, 57 y 26, las cuales no han producido desborde. Luego, llega la clave 23 que produce desborde en la dirección 1. La clave 23 es ubicada en la dirección 3 y se actualiza el enlace de la dirección 1. Cuando llega la clave 56, se vuelve a producir desborde en la dirección 1 y la misma es almacenada en la dirección 5 actualizando nuevamente los enlaces involucrados. Hay que recordar que la búsqueda de una dirección que posea lugar libre transcurre de la misma manera que la saturación progresiva.

Dirección	Clave	Enlace
0		-1
1	34	-1
2	57	-1
3		-1
4	26	-1
5		-1
6		-1
7		-1
8		-1
9		-1
10		-1

Dirección	Clave	Enlace
0		-1
1	34	3
2	57	-1
3	23	-1
4	26	-1
5		-1
6		-1
7		-1
8		-1
9		-1
10		-1

Dirección	Clave	Enlace
0		-1
1	34	5
2	57	-1
3	23	-1
4	26	-1
5	56	3
6		-1
7		-1
8		-1
9		-1
10		-1

Figura 4.5. Estructura de archivo para saturación progresiva encadenada.

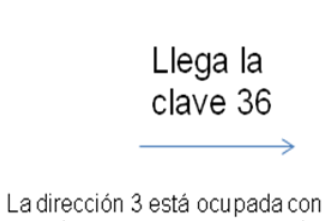
El método de saturación progresiva encadenada permite encontrar una clave con menos accesos que la saturación progresiva tradicional, debido a que sólo busca en cadena de sinónimos.

Mantener esta propiedad requiere que cada cadena de sinónimos se inicie en su respectiva dirección base, este es un detalle importante teniendo en cuenta que hay claves en saturación que ocupan direcciones base que no les corresponde.

La Figura 4.6 ilustra un problema común en saturación progresiva encadenada. Cuando llega la clave 36 se encuentra con que otra clave, la 23, que no pertenece a su cadena de

sinónimos, está ocupando su dirección base. En este caso se dice que la clave 23 es una "*clave intrusa*", pues, está ocupando un lugar que no le corresponde.

La Figura 4.7 ilustra la solución a este problema. Se debe reubicar la clave intrusa en algún otro lugar y reacomodar los enlaces correspondientes. Para reubicar la clave intrusa se utiliza la misma técnica que para una nueva clave, es decir, se busca secuencialmente un lugar libre. Una vez reubicada la clave intrusa se debe buscar la clave anterior en la cadena de sinónimos, es decir, que clave estaba apuntando a dicha clave intrusa, luego se actualiza el puntero con la nueva ubicación. Así, se genera el espacio libre en la dirección base 3 para la clave 36, logrando iniciar una posible cadena de futuras claves sinónimas.



Dirección	Clave	Enlace
0		-1
1	34	5
2	57	-1
3	23	-1
4	26	-1
5	56	3
6		-1
7		-1
8		-1
9		-1
10		-1

Figura 4.6. El problema de la clave intrusa. Se debe buscar un nuevo lugar para la clave 23, dejando el lugar disponible para la clave 36.

Resultado del alta de la clave 36.

Dirección	Clave	Enlace
0		-1
1	34	5
2	57	-1
3	36	-1
4	26	-1
5	56	6
6	23	-1
7		-1
8		-1
9		-1
10		-1

Figura 4.7. Solución al problema de la clave intrusa. Al reubicar la clave 23 se actualiza el enlace de la dirección 5, el cual ahora apunta a la dirección 6, donde reside la clave 23.

Búsqueda

Cuando se requiere una clave que no se encuentra en su dirección base, se sigue la cadena de sinónimos correspondiente, evitando así, el cúmulo de claves formado por el mismo funcionamiento de la técnica. En general, se logra encontrar una clave accediendo a menos direcciones que la saturación progresiva tradicional.

La Figura 4.8 presenta un ejemplo que incluye tanto saturación progresiva como en saturación progresiva encadenada, ante el mismo requerimiento de búsqueda. Las flechas de color verde indican el acceso de lectura en la dirección del archivo. Con saturación progresiva, la clave 23 se obtiene luego de 6 lecturas, mientras que con saturación progresiva encadenada sólo se necesitan 3 lecturas.

Direcciones leídas para la búsqueda de la clave 23 en Saturación Progresiva.

Dirección	Clave	Enlace
0		-1
1	34	5
2	57	-1
3	36	-1
4	26	-1
5	56	6
6	23	-1
7		-1
8		-1
9		-1
10		-1

Direcciones leídas para la búsqueda de la clave 23 en Saturación Progresiva Encadenada.

Dirección	Clave	Enlace
0		-1
1	34	5
2	57	-1
3	36	-1
4	26	-1
5	56	6
6	23	-1
7		-1
8		-1
9		-1
10		-1

Figura 4.8. Comparación de cantidad de lecturas para saturación progresiva y saturación progresiva encadenada.

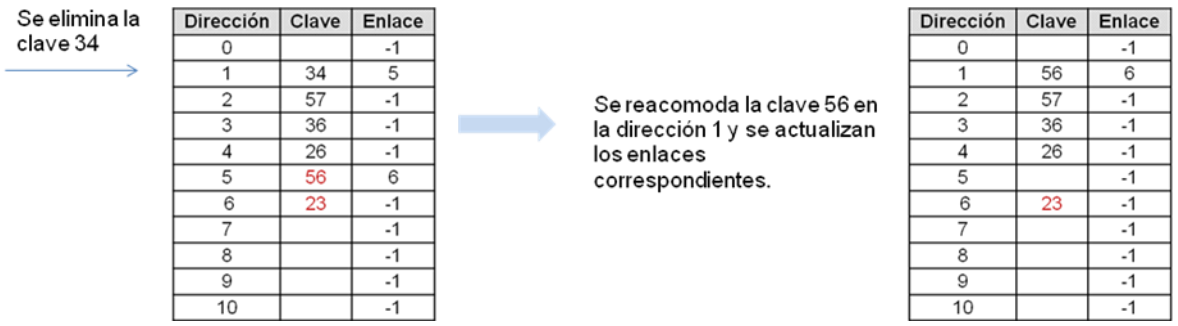
La ventaja de destinar un poco más de almacenamiento y tener un enlace en cada dirección es importante. En la mayoría de los requerimientos de búsqueda, la clave es encontrada con menos accesos a memoria secundaria que en la técnica de saturación progresiva tradicional.

Eliminación

En la operación de eliminación hay algunos detalles que atender. Cuando la clave a eliminar reside en su dirección base correspondiente, se genera un lugar disponible que puede ser aprovechado para almacenar, en caso de que exista, alguna otra clave en saturación que pertenezca a dicha dirección base, liberando así, una dirección de memoria con clave intrusa y evitando movimientos futuros.

Si la clave a eliminar es una clave en saturación, es decir, que no reside en su dirección base, se elimina y se actualizan los enlaces correspondientes sin realizar reacomodamiento alguno.

La Figura 4.9 presenta la eliminación de la clave 34, que reside en su dirección base 1 y representa el inicio de la cadena de sinónimos de tal dirección. Hay un reacomodamiento de la clave 56, que es la siguiente en dicha cadena, liberando así la dirección 5 y dejando de ser una clave intrusa.



Dirección	Clave	Enlace
0		-1
1	34	5
2	57	-1
3	36	-1
4	26	-1
5	56	6
6	23	-1
7		-1
8		-1
9		-1
10		-1

Dirección	Clave	Enlace
0		-1
1	56	6
2	57	-1
3	36	-1
4	26	-1
5		-1
6	23	-1
7		-1
8		-1
9		-1
10		-1

Figura 4.9. Eliminación de una clave que inicia su cadena de claves sinónimas.

Conclusión

Tanto en saturación progresiva encadenada como en saturación progresiva, las claves en saturación y las claves que residen en sus respectivas direcciones base comparten el mismo espacio de memoria.

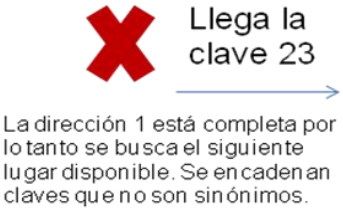
Esto tiene un impacto negativo sobre saturación progresiva encadenada pues, cada dirección debe ser ocupada por su respectiva clave base para poder iniciar la cadena de sinónimos. Aquellas direcciones que no se ocupen pueden albergar claves en saturación (claves intrusas) que luego deberán ser reubicadas en caso de que llegue la clave base correspondiente a dicha dirección.

El costo de reacomodar estas claves intrusas es alto y perjudica de manera notoria a la performance de la técnica. Aunque estas situaciones son excepcionales y ocurren con poca frecuencia, cuando se cuenta con una función de dispersión que distribuya las claves de manera aleatoria.

Para concluir, la capacidad de cada dirección es únicamente para una sola clave. Permitir almacenar más de una clave por dirección conduciría al error de encadenar claves que no son sinónimos.

La Figura 4.10 presenta un archivo dispersado con el fin de describir el problema que ocasiona tener más de una clave en una dirección. Al insertar la clave 23 se encuentra que la dirección base 1 está completa, se busca secuencialmente en las siguientes direcciones un lugar libre (no una dirección libre), llegando hasta la dirección 3 y luego se actualiza el enlace correspondiente.

La dirección 3 queda compartida por dos claves que no son sinónimos, 36 y 23, por lo tanto en la cadena de sinónimos de la dirección 1 queda con claves incorrectas, la clave 36, que residía con anterioridad en la dirección 3, no pertenece a dicha cadena.



Dirección	Clave	Clave	Enlace
0			-1
1	34	56	3
2	57	35	-1
3	36	23	-1
4			-1
5			-1
6			-1
7			-1
8			-1
9			-1
10			-1

Figura 4.10. Organización incorrecta de un archivo que utiliza como técnica de resolución de colisiones la saturación progresiva encadenada.

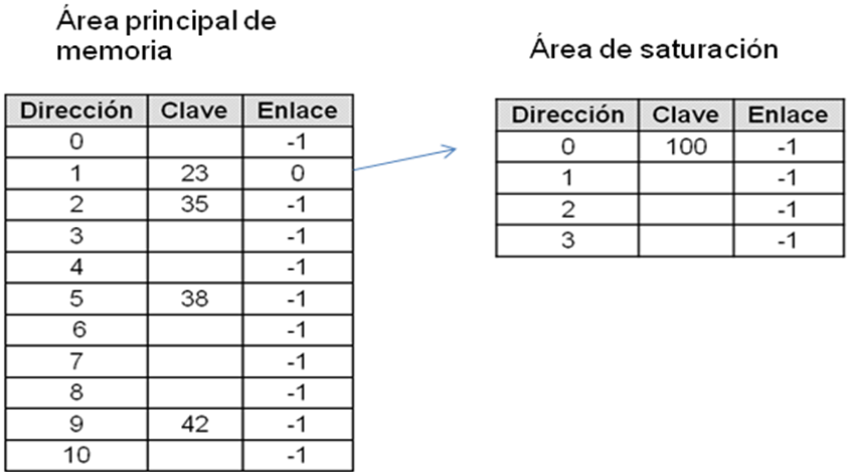
4.4.3 Saturación Progresiva Encadenada con Área de Desborde por Separado

La técnica de saturación progresiva encadenada con área de desborde por separado es una variante a la técnica de saturación progresiva encadenada, para evitar el problema de tener las potenciales direcciones base ocupadas con claves en saturación.

En esta organización se mantiene el enlace en cada una de las direcciones pero además, existe un área de memoria separada en donde residirán exclusivamente las claves en saturación. Así, se tiene un conjunto de direcciones base o área principal de memoria, alcanzables por función de dispersión, y un conjunto de direcciones de saturación denominado "*área de desborde o saturación*".

La principal ventaja de esta organización de archivos es que se mantienen libres aquellas potenciales direcciones base que aún no han sido ocupadas, logrando evitar el problema de tener que reacomodar claves en saturación.

La Figura 4.11 presenta la estructura de archivo para esta técnica con algunas claves ya dispersadas. Se utiliza como función de dispersión la clave módulo 11 (tamaño de la memoria principal).



Dirección	Clave	Enlace
0		-1
1	23	0
2	35	-1
3		-1
4		-1
5	38	-1
6		-1
7		-1
8		-1
9	42	-1
10		-1

Dirección	Clave	Enlace
0	100	-1
1		-1
2		-1
3		-1

Figura 4.11. Estructura de archivo para la técnica de saturación progresiva encadenada con área de desborde por separado.

Si el área de memoria principal es lo suficientemente grande y se cuenta con una función de dispersión aleatoria, el área de saturación puede ser mucho más acotada que el área

principal y almacenar sólo una clave por dirección. Los desbordes, bajo los parámetros adecuados, son situaciones excepcionales que ocurren esporádicamente.

El cálculo del tamaño del área de saturación depende de la configuración de los parámetros para área principal de memoria. Previamente se explico cómo predecir la cantidad posible de claves en saturación en base a ciertos valores elegidos para un ambiente de dispersión.

Esta técnica trabaja de la misma manera que la saturación progresiva encadenada, la única diferencia es que el encadenamiento de sinónimos, si es que existe, se inicia en el área principal de memoria y se continúa en el área de desborde o saturación.

Inserción

En la inserción se verifica la existencia de lugar en la dirección base correspondiente, si la misma esta completa, es decir, hay saturación, entonces se accede al siguiente espacio libre en el área de saturación, se almacena la clave y se actualizan los enlaces correspondientes.

La Figura 4.12 y 4.13 muestran el proceso de inserción de la clave 56 y luego de la clave 90. En el archivo ya existen claves dispersadas.

Cuando llega la clave 56 se produce saturación, su dirección base está completa y por lo tanto, tiene que residir en el siguiente lugar libre del área de saturación. Una vez almacenada la clave se actualiza el enlace de la dirección 1 (dirección base de la clave 56) para que apunte a la dirección correspondiente del área de saturación.

Algo similar a lo anterior sucede cuando llega la clave 90 a su dirección base 2, la diferencia en este caso es que para la dirección 2 ya existe una cadena de sinónimos iniciada. Esto último no modifica el proceso de inserción. Se almacena la clave 90 en el siguiente lugar libre del área de saturación y luego se actualiza el enlace de la dirección 2 para que apunte a esta nueva clave.

Notar además, que se actualiza el enlace de la dirección 2 del área de saturación para mantener la cadena de sinónimos.

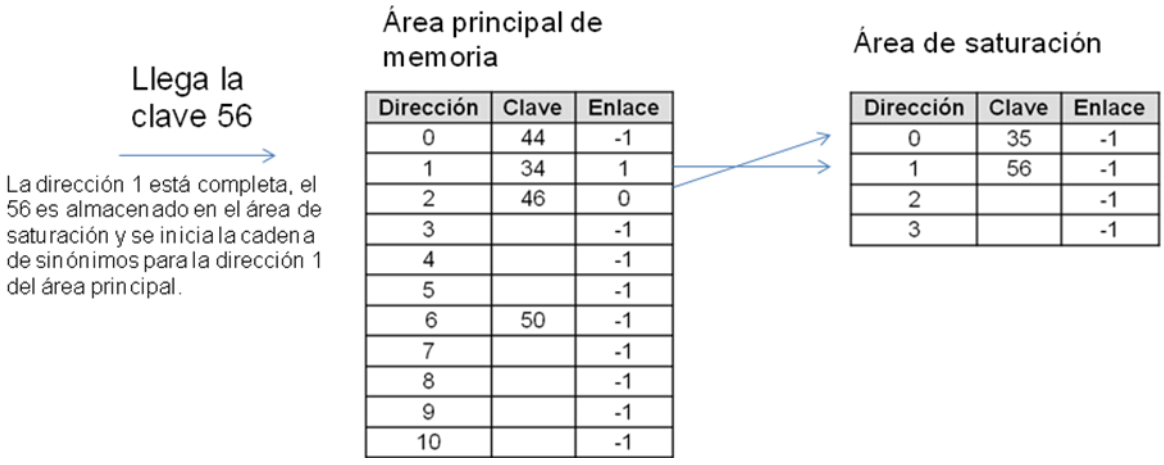


Figura 4.12. Alta de la clave 56. Se produce saturación y es almacenada en la dirección 1 del área de desborde.

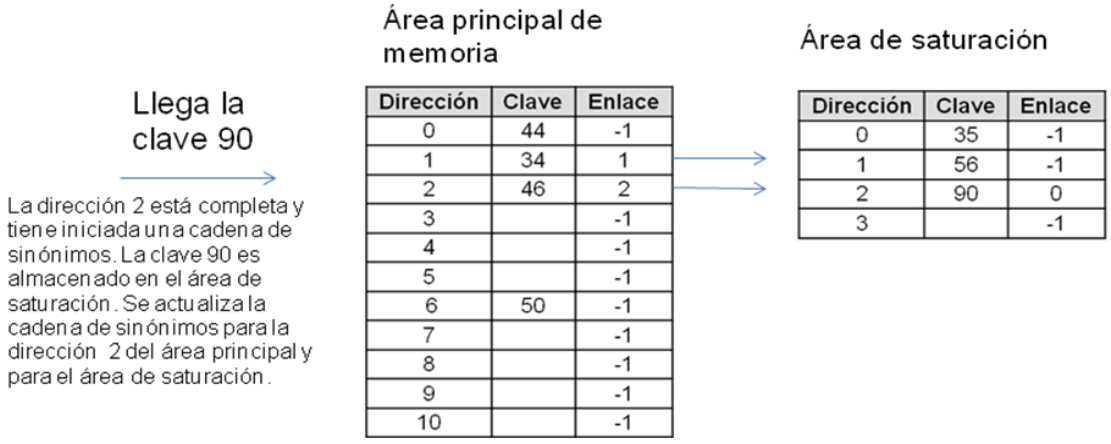


Figura 4.13. Alta de la clave 90. Se produce saturación y es almacenada en la dirección 2 del área de desborde.

Búsquedas

Para las operaciones de búsquedas se procede de la misma manera que en la técnica de saturación progresiva encadenada. Se comienza desde la dirección base correspondiente y luego continua, en caso de ser necesario, por la cadena de sinónimos. La diferencia radica en el área en donde se realiza la búsqueda.

Ante un requerimiento de búsqueda, puede ocurrir que:

- La clave es encontrada en su dirección base correspondiente.

- La clave no se encuentra en su dirección base correspondiente y el enlace de dicha dirección contiene el valor invalido (-1) de encadenamiento, entonces, no hay cadena de sinónimos en la cual buscar. Se puede asegurar que la clave no existe en el archivo.

- La clave no se encuentra en su dirección base correspondiente y el enlace de dicha dirección contiene un valor válido de encadenamiento, entonces, existe una cadena de sinónimos en el área de saturación. Se debe seguir tal cadena hasta encontrar la clave buscada o hasta llegar a un valor inválido de encadenamiento, en este último caso, se puede asegurar que la clave no existe en el archivo.

Eliminación

La operación de eliminación posee una ventaja muy importante respecto a la eliminación en saturación progresiva encadenada. Al eliminar una clave que reside en su dirección base o en el área de saturación, no hay necesidad de reacomodar a ninguna otra clave. Esto se debe a que en esta técnica cada clave ocupa el lugar que le pertenece y no existe el concepto de clave intrusa. Sólo se debe atender la actualización de los enlaces correspondientes para mantener la cadena de sinónimos.

En esta técnica tampoco existe el concepto de marcas de eliminación o inutilización, como sucede en la saturación progresiva. Al buscar una clave para eliminar se examinan sólo aquellas direcciones que pertenecen a la cadena de sinónimos de la dirección base de la clave a eliminar.

Si la clave se elimina del área principal se deja ese lugar disponible para cualquier otra nueva clave.

Si la clave se elimina del área de saturación, ese lugar quedará desperdiciado. Esto se debe a que las direcciones del área de saturación nunca son reutilizadas.

Conclusión

Si bien esta técnica evita el problema de tener todas las claves reunidas en un mismo espacio de memoria, simplificando el proceso en las inserciones y eliminaciones, no siempre mejora el desempeño en comparación con la técnica de saturación progresiva encadenada. Si el archivo a dispersar es lo suficientemente grande, puede suceder que el

área de saturación se encuentre en un cilindro de disco diferente al del área principal. En este caso, la búsqueda de un registro en saturación implicará movimientos muy costosos, bajando el rendimiento de la técnica para cada operación que se realice.

4.4.4 Dispersión doble

La dispersión doble representa la última técnica que intenta evitar el problema de tener cúmulos de claves.

La dispersión doble plantea almacenar las claves en saturación a cierta distancia de sus direcciones base, de manera que las direcciones más cercanas a una dirección saturada permanezcan, en lo posible, libres para futuras inserciones. Evitando así tener los cúmulos de claves en un sector del archivo.

Esta técnica incorpora una segunda función de dispersión. La cual es una función similar a la primera pero su resultado es utilizado como desplazamiento. La segunda función es solamente utilizada cuando existe un desborde.

Inserción

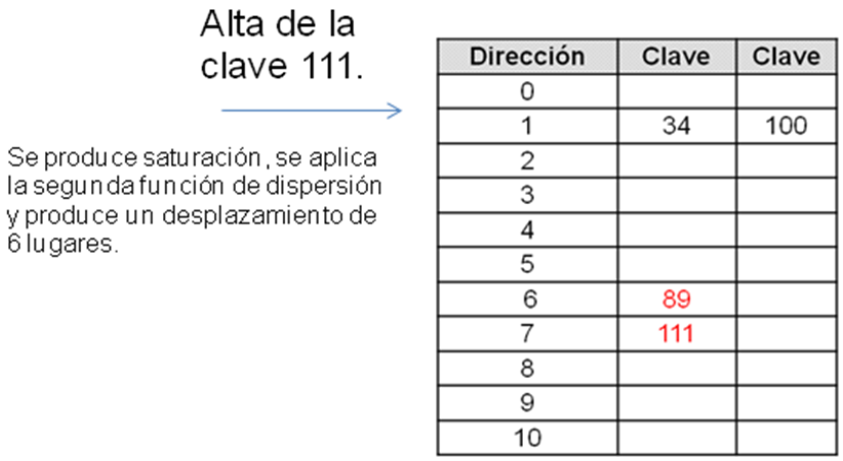
Si al agregar una nueva clave en el archivo produce saturación se procede a aplicar la segunda función de dispersión a la clave. La segunda función de dispersión representa un desplazamiento a partir de la dirección base de la clave que se está dispersando. Si está nueva dirección (base + desplazamiento) se encuentra completa, se vuelve a aplicar nuevamente el mismo desplazamiento generando otra nueva dirección. Este proceso se repite hasta encontrar una dirección que posea algún lugar disponible.

Si el desplazamiento sobrepasa al final del espacio de direcciones se debe continuar desde la primera dirección, dando la vuelta al espacio de direcciones.

Generalmente, como segunda función de dispersión se utiliza el resto de la división por un valor primo relativo al tamaño del espacio de direcciones, garantizando que se puedan analizar todas las direcciones del espacio de memoria disponible.

En la Figura 4.14 la clave 111, por función de dispersión produce la dirección base 1 (111 módulo 11). Tal dirección se encuentra completa, se produce saturación. Se aplica la segunda función de dispersión (111 módulo 7) para obtener el desplazamiento

correspondiente, en este caso es de 6 lugares. La dirección nueva a examinar es la dirección 7, como posee lugar disponible, la clave es almacenada en dicha dirección.



Dirección	Clave	Clave
0		
1	34	100
2		
3		
4		
5		
6	89	
7	111	
8		
9		
10		

Figura 4.14. Alta de la clave 111. Se produce saturación y es almacenada en la dirección 7.

Observando el ejemplo de la Figura 4.14 se puede corroborar que aplicando el desplazamiento obtenido se puede llegar a examinar todas las direcciones del archivo. Las direcciones serían examinadas en el siguiente orden: 7, 2, 8, 3, 9, 4, 10, 5, 0, 6 y se llega a la dirección 1, donde se produjo el desborde. Llegar a examinar todas las direcciones implica que no hay lugar en el archivo para la nueva clave.

Existen dos casos particulares en donde la segunda función incrementa en uno el desplazamiento retornado:

1. cuando el resto de la división es cero, o
2. cuando el resto de la división es igual al tamaño de la memoria.

En ambos casos, si no se incrementa en uno el resultado de la segunda función, se tendría el efecto de examinar siempre la misma dirección. En el primer caso, debido a que el desplazamiento obtenido es nulo y en el segundo caso se da una vuelta completa al espacio de direcciones quedando siempre en la dirección que ha saturado.

Búsqueda

Cuando se realiza la búsqueda de una clave y la misma no es localizada en su dirección base, se debe utilizar la segunda función de dispersión para obtener el desplazamiento que le corresponde y comenzar a examinar el resto de las direcciones del archivo.

La búsqueda se detiene cuando:

- La clave ha sido localizada en alguna de las direcciones examinadas.
- Se examina una dirección que posee un lugar disponible sin marca de eliminación. Se puede asegurar que la clave no existe en el archivo.
- Se llega a la dirección base inicial. Se examinaron todas las direcciones posibles. Se pude asegurar que la clave no existe en el archivo.

En esta técnica aparece nuevamente el concepto de marca de eliminación. Se recuerda que estas marcas aseguran poder encontrar una clave en saturación, independientemente de las eliminaciones que se realicen.

Eliminación

Para la eliminación se utilizan las conocidas marcas de eliminación descriptas en saturación progresiva para no perder claves en saturación.

La marca de eliminación se coloca cuando la dirección en la cual se encuentra la clave a eliminar está completa, pudiendo tratarse de una dirección saturada.

Conclusión

La dispersión doble reduce el acumulamiento local y tiende a esparcir las claves, pero adolece de un problema importante, ya que viola el principio de localidad por la posibilidad de trasladar una clave en saturación a gran distancia de su dirección base, incrementando así la probabilidad de que se necesite tiempo adicional para obtener la dirección de la clave en saturación cuando el archivo es muy grande y ocupa más de un cilindro del disco.

4.5 Dispersión con espacio de direccionamiento dinámico

Hasta el momento, se ha presentado al método de dispersión como una alternativa de almacenamiento que distribuye las claves en un rango de direcciones disponibles y establecidas de antemano. En las situaciones planteadas se ha determinado el número de direcciones y la capacidad de cada una de ellas, sin analizar esta precondición en detalle.

Es importante realizar un análisis más profundo con respecto a la cantidad de direcciones disponibles, en el momento en que se empieza a trabajar con un archivo utilizando dispersión.

Los problemas empiezan cuando la densidad de empaquetamiento tiende a 1. Cuando el archivo se completa, es necesario obtener mayor cantidad de direcciones y también es necesario que la función de dispersión pueda alcanzar a direccionar las nuevas direcciones generadas.

Aumentar el espacio de direcciones no es algo sencillo. Se debe modificar la función de dispersión y eso traerá aparejado que todo el archivo deberá ser nuevamente dispersado, dado que la función original ha cambiado.

El costo de dispersar el archivo nuevamente es muy alto. El tiempo utilizado es importante, y mientras se realiza esta operatoria no es posible llevar a cabo un requerimiento de acceso por parte de algún usuario final.

Cuando se crea un archivo, es importante analizar su posible tasa de crecimiento y determinar la cantidad de direcciones que optimice el uso del espacio con respecto a la periodicidad de una nueva dispersión.

Una alternativa posible es trabajar con archivos que administren el espacio de direccionamiento de manera dinámica. En este caso no se establece de antemano la cantidad de direcciones disponibles, sino que el espacio disponible crece o disminuye conforme crece o disminuye el archivo [2].

4.5.1 Dispersión extensible

La dispersión extensible es una de las alternativas de implementación para la dispersión con espacio de direccionamiento dinámico. El método consiste en aumentar el número de direcciones disponibles a medida que realmente se lo necesita.

Este método no utiliza el concepto de densidad de empaquetamiento. Esto se debe a que el espacio físico aumenta en función de la cantidad de registros que dispone el archivo en cada momento.

Los métodos dinámicos cambian la política de trabajo con respecto a la función de dispersión. Las direcciones no están prefijadas de antemano, por lo tanto, una función de dispersión no puede asegurar retornar, para una clave determinada, una dirección física válida.

Para dispersión extensible, se implementa una única función de dispersión que a partir de una clave, retorne una cadena de bits (sólo ceros y unos). Mediante esta cadena de bits se podrá posteriormente obtener la dirección física del archivo para la clave en cuestión.

Esta técnica, necesita para su implementación, una estructura auxiliar. Esta estructura es la encargada de contener la dirección física de las claves dispersadas. La cadena de bits que retorna la función de dispersión, servirá de punto de entrada a esta tabla para luego obtener la dirección física correspondiente. La tabla reside en memoria secundaria y se traslada a memoria principal para su utilización. Generalmente se necesitan pocos accesos para mantener su organización [2].

Inserción

El método comienza con una sola dirección en la memoria secundaria y una tabla que solamente contiene esa única dirección.

La Figura 4.15 presenta el estado inicial del archivo y la tabla correspondiente a esta técnica. La tabla posee un valor asociado que indica cual es la cantidad de bits necesarios para acceder a la misma y obtener una dirección física. Estos bits pertenecen a la secuencia obtenida por la función de dispersión. Inicialmente el valor asociado de la tabla es cero, lo que indica que no es necesario bit alguno.

Además, cada dirección física en disco contiene un valor asociado que indica cuantos bits fueron necesarios utilizar de la cadena de bits retornados por la función de dispersión para poder ubicar las claves existentes en dicha dirección.

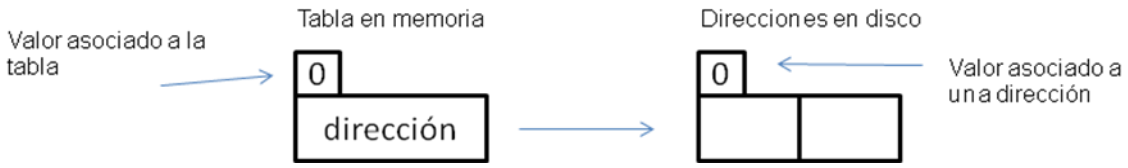


Figura 4.15. Estado inicial del archivo para dispersión extensible. Cada dirección en la memoria secundaria tiene capacidad para almacenar dos claves.

Para insertar una clave se calcula la función de dispersión y se toman tantos bits como indica el valor asociado que posee la tabla en memoria. En un principio el valor es cero, es decir, que la clave se debe ubicar en la única dirección física disponible.

Cuando una dirección se completa, la inserción de cualquier otra clave produce saturación o desborde. Para resolver una saturación se deben seguir un conjunto de acciones.

Primero se aumenta en uno el valor asociado de la dirección que se ha saturado. Luego, se genera una nueva dirección vacía con el mismo valor asociado.

Se compara el valor asociado de la dirección saturada con el valor asociado de la tabla existente en memoria. Si el valor asociado de la dirección es mayor que el valor asociado de la tabla, significa que la tabla no dispone de entradas suficientes para direccionar la nueva dirección generada. En esta situación, la cantidad de entradas de la tabla se duplica y el valor asociado de la tabla se incrementa en uno.

Las claves de la dirección saturada más la nueva clave se deben dispersar nuevamente entre la dirección saturada y la nueva dirección generada. Esto se debe a que el valor asociado de la tabla se ha incrementado en uno, modificando la cantidad de bits que se deben tener en cuenta ante un resultado devuelto por la función de dispersión. La Figura 4.16 muestra la secuencia de pasos ante una saturación. Se obtiene que el bit menos significativo del 56 es el 0 y para el 23 y 45 es el 1.

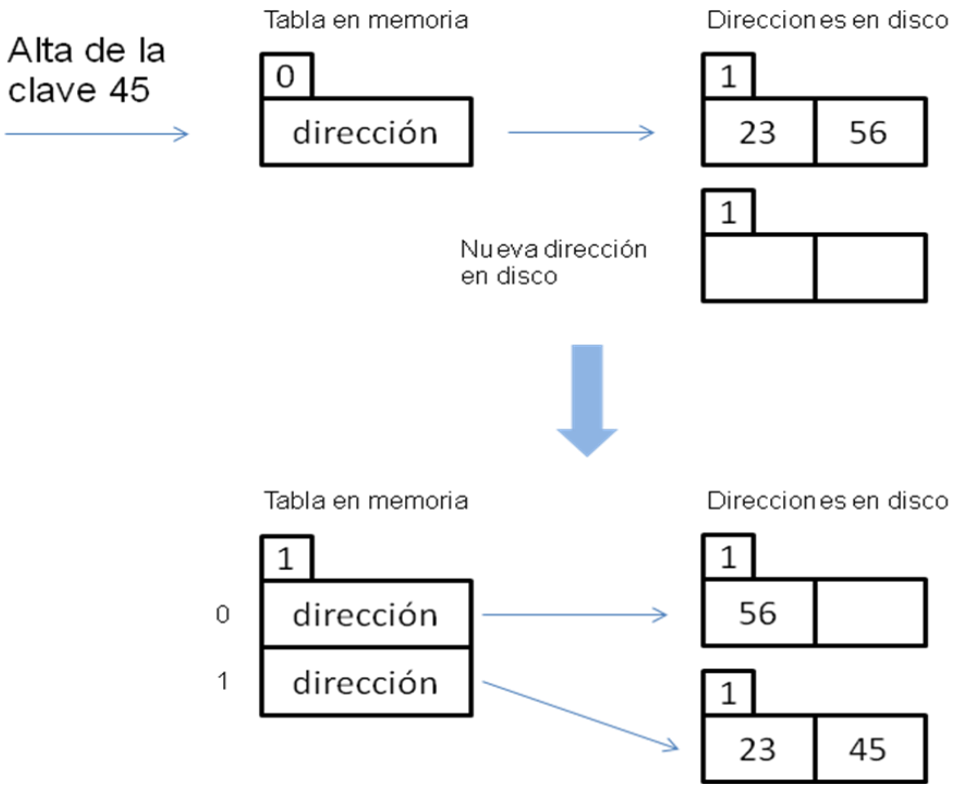


Figura 4.16. Alta de la clave 45. Se produce desborde, se genera una nueva dirección y se duplica la tabla.

La tabla resultante de la Figura 4.16 posee dos entradas (0 y 1). Ahora, para cualquier clave nueva, se debe tomar el bit menos significativo del resultado de la función de dispersión, pues así lo indica el valor asociado de la tabla.

La Figura 4.17 presenta la llegada de la clave 39. Se accede a la posición 1 de la tabla y la dirección física correspondiente se encuentra completa, por lo tanto se produce saturación.

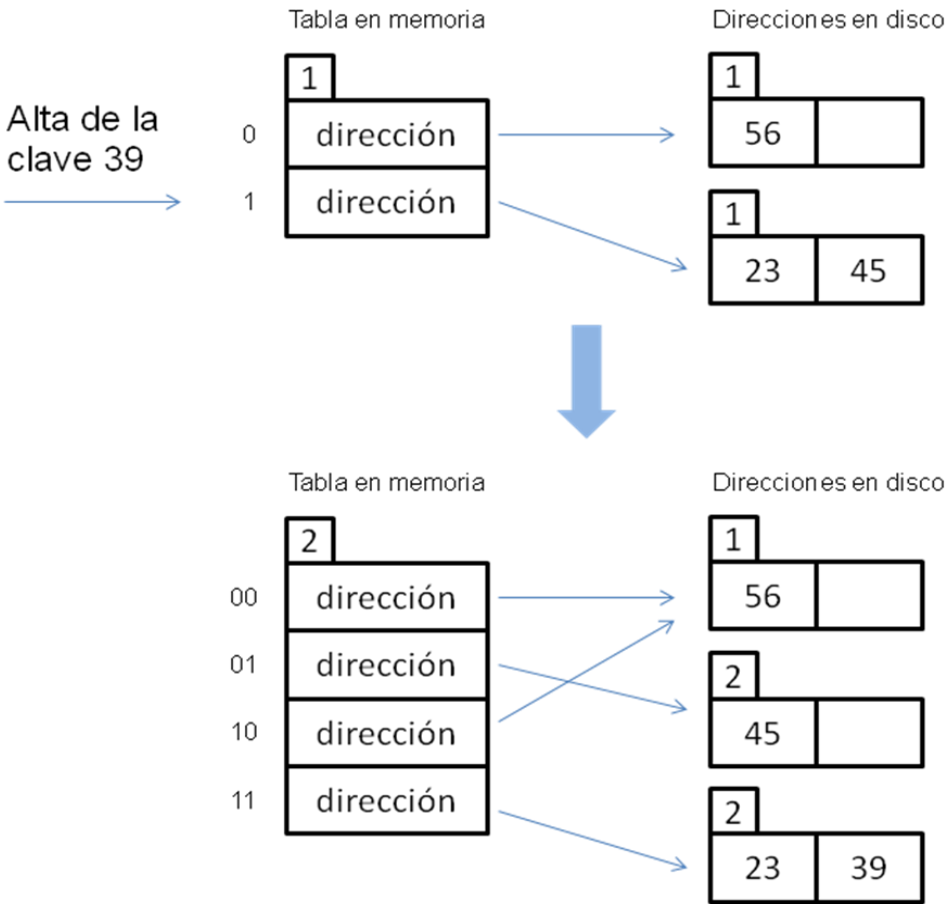


Figura 4.17. Alta de la clave 39. Se produce desborde, se genera una nueva dirección y se duplica la tabla.

No siempre es necesario duplicar la tabla. Si el valor asociado a la dirección que se ha saturado es menor o coincide con el valor asociado que posee la tabla en memoria significa que la tabla posee suficientes entradas para referenciar a la nueva dirección creada, en ese caso la tabla no se duplica.

En resumen, el método de dispersión extensible debe cumplir con las siguientes pautas:

- Se utilizan solo los bits necesarios de acuerdo a cada instancia del archivo.
- Los bits utilizados forman la dirección de memoria que indica la dirección física del nodo que se utilizará.

- Cuando sucede una saturación, se deben dispersar nuevamente las claves de la dirección saturada y la nueva dirección generada. Para esto se considera un bit más.

- La tabla tendrá tantas entradas (dirección al disco) como 2^i, siendo *i* el número de bits actuales a considerar ante cualquier nueva inserción.

Búsqueda

La función de dispersión retorna la secuencia de bits para la clave solicitada. Luego, se utilizan la cantidad de bits necesarios según el valor asociado a la tabla en memoria y se accede a la misma en busca de la dirección física en donde supuestamente radica la clave buscada.

El proceso de búsqueda asegura encontrar cada clave en un solo acceso. En caso de que la clave no se encuentre en la dirección buscada, se puede asegurar que no existe en el archivo. No es necesario evaluar otras direcciones.

Eliminación

Para la eliminación no se toman medidas adicionales. Se busca la clave a eliminar y si se encuentra, se libera dicha ubicación.

4.6 Conclusión

Cada variante de implementación para dispersar un archivo presenta aspectos positivos y negativos. Cuando se cuenta con la posibilidad de acceder a un elemento por su clave primaria, la organización a través de dispersión siempre resulta ser más eficiente en comparación con la indización.

Como se analizó anteriormente, la dispersión asegura encontrar un elemento del archivo, en la mayoría de los casos, con un solo acceso a la memoria secundaria.

Si se desea asegurar siempre un acceso a disco para recuperar un elemento, la variante de dispersión extensible, que utiliza espacio de direccionamiento dinámico, lo logra [2].

Capítulo 5

E-HASH – Herramienta de Software para Dispersión de Archivos.

5.1 Introducción y marco conceptual

En este capítulo se describe *E-HASH*, una herramienta de software para la enseñanza de técnicas de dispersión de archivos. El propósito fundamental de *E-HASH* es el de asistir a los alumnos en el aprendizaje de "*Dispersión de Archivos*", con el marco conceptual impuesto por la asignatura *Introducción a las Bases de Datos* de la *Facultad de Informática* de la *UNLP*.

En la experiencia como docentes de la cátedra se ha observado que si bien, las guías prácticas alcanzan para la comprensión general del tema en cuestión, es muy conveniente y beneficioso disponer de un software asistente que actúe como una herramienta complementaria en el proceso enseñanza y aprendizaje.

Actualmente, los alumnos resuelven las guías prácticas con el método tradicional de lápiz y papel para luego evacuar sus dudas con los auxiliares docentes, con quienes corroboran la precisión de la interpretación y resolución de sus ejercicios. La utilización de *E-HASH* no pretende reemplazar la práctica tradicional sino que tiene como propósito actuar como complemento, fortaleciendo la actividad de enseñanza y aprendizaje del tema tratado. Mediante la herramienta generada el alumno podrá analizar la resolución un problema, generando el caso de uso y comprobando su resolución paso a paso.

La asignatura de *Introducción a las Bases de Datos* pretende generar herramientas de software educativas que sirvan de complemento durante el proceso de aprendizaje de un tema determinado. Al respecto ya se han generado dos herramientas: *CASER* para el modelado de BBDD, y *HEA* para la simulación de implementación de índices utilizando como estructura de datos la familia de árboles B.

E-HASH es un aporte más que complementa este conjunto de herramientas de software educativas. En este caso se pretende brindar una herramienta que actúe como

complemento para el proceso de enseñanza y aprendizaje del tema "*dispersión de archivos*".

E-HASH está desarrollada con software de uso libre y es totalmente portable. Utiliza tecnologías *HTML* y *Javascript*, brindando la posibilidad de ser utilizado bajo cualquier navegador web de cualquier sistema operativo. No requiere la instalación de ningún software adicional ni requiere configuración alguna.

El objetivo de *E-HASH* es brindar al alumno una herramienta que le sirva de complemento al momento de resolver las guías prácticas referentes al tema "*dispersión de archivos"*. Si bien, el alumno posee numerosas consultas prácticas en donde puede resolver sus ejercicios y evacuar sus dudas, se considera útil tener un asistente que pueda en cualquier momento resolver y aclarar una situación determinada.

Utilizar una herramienta de software motiva al alumno a realizar pruebas y variantes con los ejercicios que debe resolver, brindando la posibilidad de descubrir nuevas alternativas y abarcar un espectro conceptual mucho más amplio con el tema en cuestión.

Si bien la práctica tradicional en lápiz y papel representa la principal metodología para desarrollar los temas prácticos, es muy común y notorio, en los tiempos actuales, ver a una gran cantidad de alumnos que utilizan su *notebook* o *netbook* como medio para guardar sus apuntes, desarrollar sus prácticas y demás tareas que demanda una carrera universitaria. El avance de la tecnología y el acceso a estas herramientas informáticas por parte del alumno permiten explotar diversos recursos informáticos, plataformas de comunicación web, mensajerías, grupos, foros, herramientas de simulación, herramientas de desarrollo, entre otras. Todas ellas forman parte del software educativo que actualmente rodea al alumno y que son el puente para despertar la curiosidad y el entusiasmo en el abordaje de algún tema determinado.

5.2 Descripción funcional

E-HASH es una herramienta de software pensada de manera evolutiva e intuitiva para que el alumno pueda ir paso a paso configurando un ambiente de dispersión.

La configuración de un ambiente de dispersión consta de varios parámetros, que determinan diferentes resultados y performance. La herramienta presenta una interface organizada en solapas, donde se definen un conjunto de parámetros relacionados y

agrupados de manera estratégica para que el alumno pueda comprender y relacionar los valores elegidos.

Las solapas que organizan los parámetros son seis, y su orden respectivo es:

1. *Inicio.*
2. *Tipo de Dispersión.*
3. *Función de Dispersión.*
4. *Espacio de Memoria.*
5. *Resolución de Colisiones.*
6. *Simulación.*

La Figura 5.1 presenta la interfaz de la herramienta cuando se accede a través del navegador web. Inicialmente se encuentra en la solapa de "*inicio*", el resto de las solapas aún no se encuentran habilitadas, las mismas serán habilitadas de acuerdo a las opciones elegidas.

Figura 5.1. Pantalla inicial de *E-HASH*. Solapa de Inicio.

La última solapa, denominada "*Simulación*" permite ingresar a un modelo de simulación y demostración en donde el alumno podrá dispersar los valores de clave que desee. Esta solapa está habilitada sólo cuando se cumplan los requisitos necesarios para un ambiente de dispersión determinado.

Sobre el margen derecho se brinda un menú de ayuda contextual con los temas más relevantes sobre dispersión de archivos. Es importante, antes de configurar un ambiente de dispersión, que el alumno tenga presente ciertos conocimientos básicos que influyen en el desempeño de un ambiente de dispersión.

5.2.1 Configuración de parámetros

Para configurar un ambiente de dispersión la primer decision tiene que ver con el tipo de espacio de direccionamiento a utilizar. Se debe elegir entre un espacio de direcciones estático, prefijo de antemano, o un espacio de direccionamiento dinámico.

E-HASH brinda ambas posibilidades de configuración para el espacio de direccionamiento. A través de la opción "*Comenzar a Operar*" se comienza con la configuración de los parametros necesarios en un ambiente de dispersión. Se habilita la solapa denominada "*Tipo de Dispersión*", desde donde se podrá seleccionar cualquiera de los dos tipos de dispersión posible, estática o dinámica.

La Figura 5.2 presenta la solapa "*Tipo de Dispersión*" junto con las dos opciones posibles para un espacio de direccionamiento.

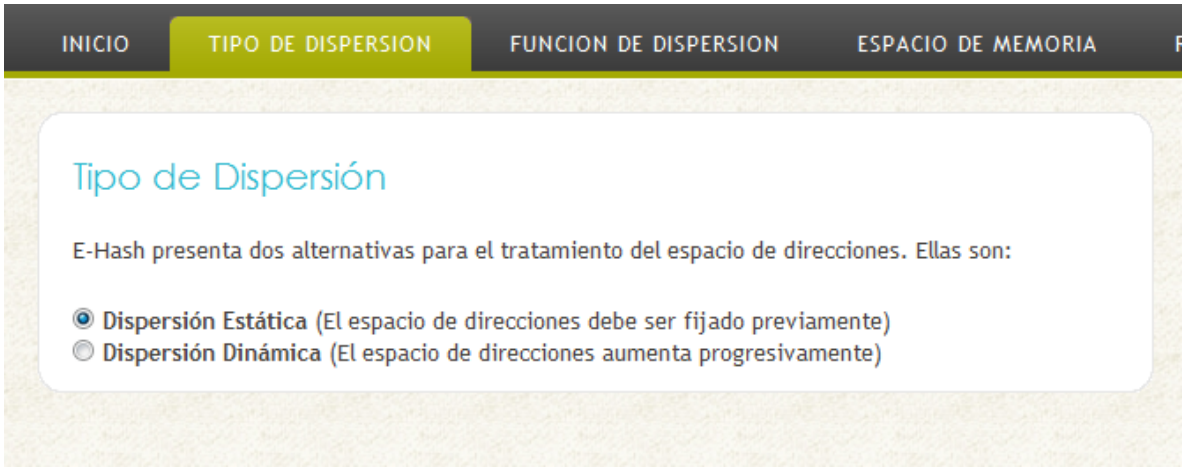


Figura 5.2. Selección del tipo de espacio de direccionamiento. Solapa Tipo de Dispersión.

Una vez seleccionado el tipo de espacio de direccionamiento que será utilizado, se habilitará el subconjunto de las solapas restantes. Tal subconjunto de solapas está determinado por el tipo de espacio de direcciones que se ha seleccionado.

La Tabla 5.1 presenta los valores que son necesarios seleccionar de acuerdo al tipo de espacio de direccionamiento que se ha elegido.

Direccionamiento Estático	Direccionamiento Dinámico
Función de dispersión	Capacidad de una dirección
Cantidad de direcciones válidas o Densidad de Empaquetamiento	Cantidad de claves a dispersar
Capacidad de una dirección	
Cantidad de claves a dispersar	
Técnica de resolución de colisiones con saturación	

Tabla 5.1. Parámetros que son necesarios seleccionar de acuerdo al tipo de espacio de direccionamiento.

Cuando el espacio de direccionamiento se administra dinámicamente, la función de dispersión es única y la misma retorna una cadena de bits para la clave que se va a dispersar, motivo por el cual no es una opción posible de modificar.

Algo similar sucede con la técnica de resolución de colisiones. Para resolver una situación de saturación en un espacio de direccionamiento dinámico se utiliza solamente la técnica de "*Dispersión Extensible*", por lo tanto, la resolución de saturación queda determinada por haber elegido el tipo de espacio de direccionamiento dinámico.

Espacio de direccionamiento estático

Cuando la opción elegida para un ambiente de dispersión es el direccionamiento estático se debe previamente seleccionar un conjunto de parámetros necesarios, los cuales determinan el desempeño del correspondiente ambiente de dispersión. Ellos son:

- *"Función de Dispersión"*: *E-HASH* posee tres opciones. Todas las funciones de dispersión disponibles producen una distribución aleatoria de las claves.

- *"Cantidad de direcciones válidas o densidad de empaquetamiento*": se debe indicar el tamaño que tendrá el espacio de direcciones disponible. Se puede especificar directamente indicando cuantas direcciones estarán disponibles o indirectamente a través de la densidad de empaquetamiento. Ambas alternativas son excluyentes entre sí.

- "*Capacidad de una dirección"*: se debe decidir que tamaño tendrá cada dirección de memoria, es decir, cuantas claves podrá almacenar cada dirección.

- "*Cantidad de claves a dispersar"*: se indica la cantidad de claves a dispersar. Este valor representa la cantidad de elementos que contiene el archivo.

- "*Técnica de resolución de colisiones"*: cuando sucede una saturación se debe emplear alguna técnica de resolución de colisiones para ubicar la clave que ha producido tal saturación. *E-HASH* cuenta con cuatro técnicas estáticas para tratar una saturación.

Función de dispersión

La elección de una función de dispersión aceptable es muy importante para un ambiente de dispersión, dado que su comportamiento determinará como se dispersarán las claves de archivo. El ambiente de simulación describe el comportamiento extremo del archivo ante muchas colisiones, para que el alumno comprenda cada método. Las tres funciones que se pueden elegir con *E-HASH* se comportan de manera eficaz aunque se posean pocas direcciones y muchas claves a dispersar. Se presentan sólo funciones que realizan distribuciones aleatorias.

La elección entre las diferentes funciones le permite al alumno poder comparar el comportamiento de cada una de ellas de acuerdo al conjunto de claves que son dispersadas.

Existe un paso que es común a todas las funciones de dispersión implementadas por *E-HASH*. Cuando la clave a dispersar es alfanumérica se le aplica un proceso de transformación que la convierte a un valor numérico, para luego poder aplicarle la función de dispersión propiamente dicha. Este proceso es conocido como "*desglosar y sumar*" y consiste en tomar cada digito alfanumérico perteneciente a la clave y obtener su valor correspondiente *ASCII*[6] (American Standard Code for Information Interchange — Código Estándar Estadounidense para el Intercambio de Información). Los valores obtenidos de la conversión de cada digito alfanumérico son sumados para lograr un valor único, al cual se le aplicará la función correspondiente para obtener una dirección [19].

[6] Debido a que las computadoras solamente entienden números, el código ASCII es una representación numérica de un carácter como 'a' o '@' [19].

Las funciones de dispersión que presenta *E-HASH* fueron presentadas de forma teórica en el Capítulo 4. Se detallará a continuación, aspectos de su implementación para llevarlas a la práctica:

- *Módulo tamaño de memoria*: la dirección retornada por la función es el resto de la división entre la clave a dispersar y el tamaño del espacio de direcciones. Por ejemplo, si se cuenta con un rango de 11 direcciones, la clave 20 se dispersa a la dirección 9 (20 módulo 11 = 9). Es conveniente que el tamaño del espacio de direcciones sea un valor primo para que la función pueda producir una distribución más aleatoria. Generalmente, la división con un valor primo genera resultados diferentes (a partir de diferentes secuencias consecutivas) que los obtenidos con la división entre un número que no sea primo [1].

- *Centros cuadrados*: para formar la dirección se eleva la clave al cuadrado y luego se extraen, de la parte central del número, un conjunto de dígitos para ser ajustados al espacio de direcciones correspondiente. Generalmente, los dígitos centrales extraídos caen fuera del espacio de direcciones permitido, por lo tanto la función los ajusta a dicho espacio realizando la división por el tamaño del espacio de direcciones y tomando el resto como dirección real. Por ejemplo, si se cuenta con un rango de 11 direcciones, la clave 120 se dispersa a la dirección 0, dado que 120^2 = 14400, en este caso los dígitos centrales extraído son 440, entonces, 440 módulo 11 = 0.

- *Transformación de la base*: este método convierte la clave a una base numérica distinta al decimal, que es la base con la que trabaja *E-HASH*, luego divide dicho resultado por el tamaño de la memoria utilizando el resto como resultado final y dirección física del archivo. En particular, cuando se selecciona esta función, se habilita una lista con tres posibilidades distintas para elegir la base a la que se desea convertir la clave. Ellas son: base 11 (Undecimal), base 12 (Duodecimal) y base 16 (Hexadecimal). La Figura 5.3 presenta la lista de bases mencionadas.

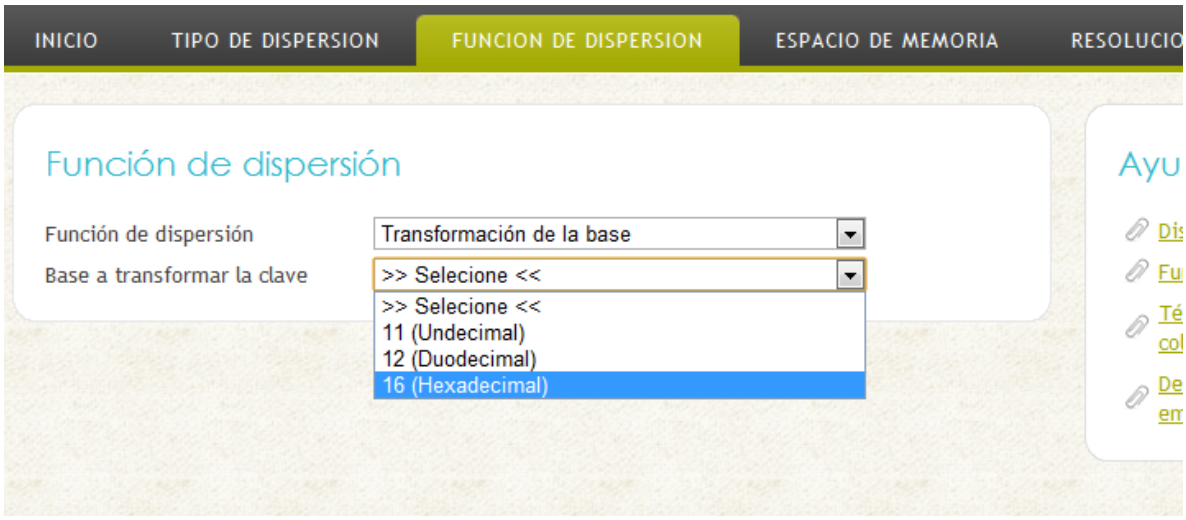


Figura 5.3. Bases posibles para transformar la clave. Opción habilitada para la función de dispersión "*Transformación de la base*". Solapa Función de Dispersión.

La dispersión de las claves en el espacio de memoria disponible es, a priori, impredecible. No existe relación alguna entre las claves que se dispersan, por lo tanto, sin importar la función elegida, el comportamiento de cualquiera de ellas varía de acuerdo al conjunto de claves elegidas para dispersar.

Cantidad de direcciones o densidad de empaquetamiento

Cuando se decide trabajar con un espacio de direccionamiento estático es necesario indicar la cantidad de direcciones que son posibles de direccionar a través de la función de dispersión.

Las configuraciones para el espacio de memoria se realizan desde la solapa "*Espacio de Memoria*". *E-HASH* posee dos formas de seleccionar el tamaño del espacio de direcciones. Una de ellas es especificar explícitamente dicho valor a través de la lista que posee el combo desplegable denominado "*Cantidad de direcciones*", las opciones presentadas aquí son cinco y todas representan un valor primo, *11, 13, 17, 19* y *23* direcciones.

La otra posibilidad es determinar la cantidad de direcciones a través de la elección de la densidad de empaquetamiento. Si bien la densidad de empaquetamiento representa el porcentaje en que será ocupado en el espacio de direcciones, es posible calcular la

cantidad de direcciones necesarias para cubrir el porcentaje deseado; esto es posible dado que posteriormente se deberá elegir la cantidad de claves a ser dispersadas.

Ambas posibilidades son excluyentes entre sí. En la Figura 5.4 se presenta uno de los combos de selección con sus opciones posibles.

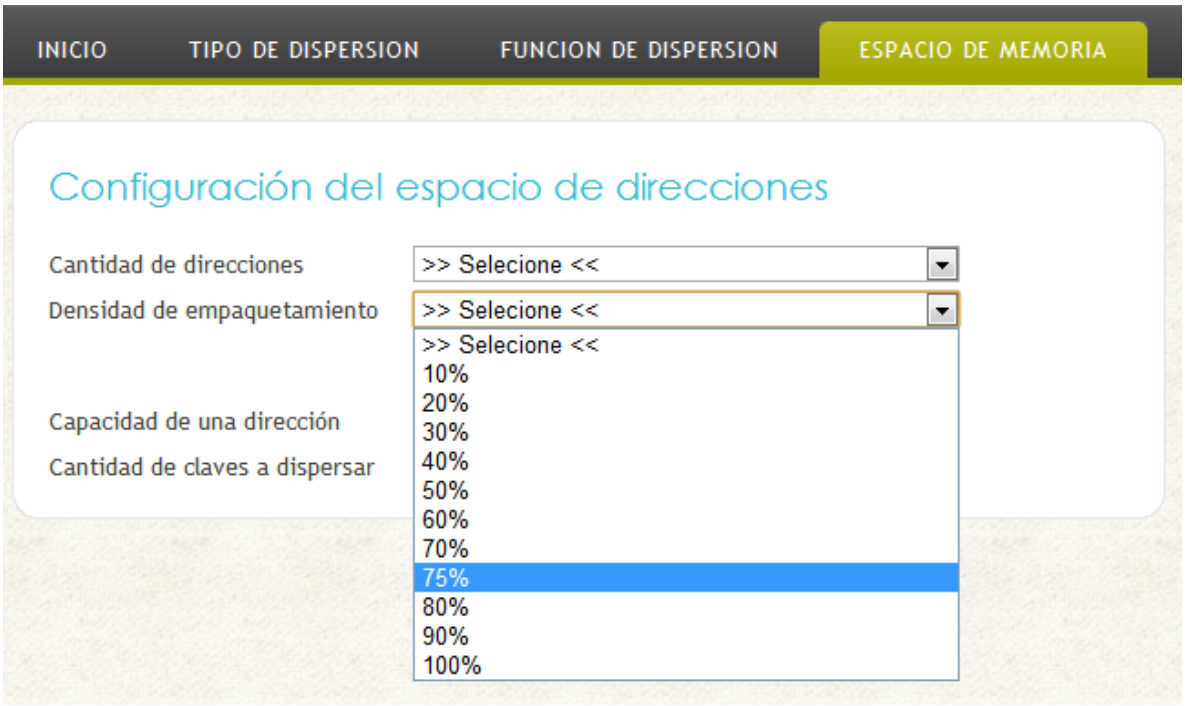


Figura 5.4. Selección de la densidad de empaquetamiento. Una forma de establecer la cantidad de direcciones que contendrá el espacio de direccionamiento. Solapa Espacio de Memoria.

En el rango de porcentajes para la densidad de empaquetamiento aparece un valor discontinuo con respecto a la escala de los demás valores, 75%. Este valor de porcentaje especial, marca el umbral deseado para la densidad de empaquetamiento, es decir, mientras tal densidad se encuentre en un valor inferior a ese tope, la probabilidad de que ocurra un desborde disminuye considerablemente y tiende a cero [2].

Capacidad de una dirección

Además de determinar el rango de direcciones posibles para almacenar claves, se debe indicar cuantas claves podrán ser almacenadas en cada una de estas direcciones. Este valor representa cuantas claves son aceptadas en una dirección antes de que se produzca saturación.

La elección del tamaño de una dirección se realiza mediante la lista presenta a través del combo desplegable denominado "*Capacidad de una dirección*" y presenta las opciones de *1, 2, 3* y *4* claves.

Cuanto mayor es la capacidad de una dirección, hay menor probabilidad que ocurra un desborde o saturación. Esto se debe a que las funciones de dispersión que presenta *E-HASH* producen una distribución aleatoria de las claves y, por lo tanto, para valores de capacidad superior a dos claves por dirección es muy difícil que produzca desborde con un conjunto de claves ingresadas al azar.

También es posible elegir estratégicamente un conjunto de claves con el objetivo de saturar una determinada dirección, a fin de comprobar algún comportamiento determinado.

Cantidad de claves a dispersar

Este valor representa la cantidad de elementos que contendrá el archivo. Con la elección de la cantidad de claves, se puede determinar si la cantidad de direcciones seleccionadas anteriormente son suficientes para almacenar todas las claves, o también determinar el tamaño del espacio de direcciones a través de la densidad de empaquetamiento, calculando un espacio de direcciones que sea lo necesariamente grande como para albergar la cantidad de claves que se desea dispersar.

La opción para seleccionar la cantidad de claves a dispersar se denomina "*Cantidad de claves a dispersar*" y presenta un combo desplegable con los valores *5, 10, 15, 20, 25* y *30* claves posibles. La Figura 5.5 presenta el combo desplegable con las opciones anteriormente comentadas.

El rango de valores que posee esta opción le permite al alumno abarcar la diversidad de ejercicios que presentan las guías prácticas, y evaluar la performance de cada ambiente de dispersión en base a diferentes cantidades de claves dispersadas.

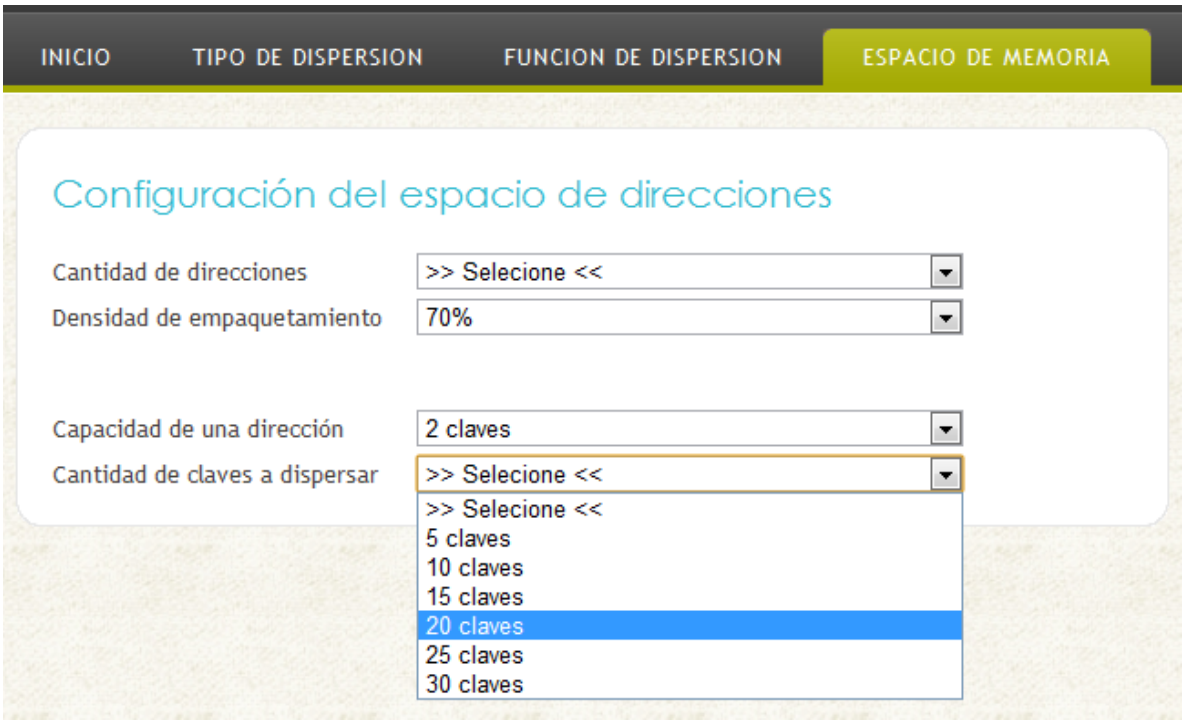


Figura 5.5. Selección de la cantidad de claves que se van a dispersar. Solapa Espacio de Memoria.

Técnica de resolución de colisiones

La elección de la técnica de resolución de colisiones con saturación se realiza desde la solapa "*Resolución de Colisiones*". La selección de este parámetro indica que estrategia se deberá seguir en caso de que una clave provoque la saturación de una dirección.

E-HASH posee cuatro técnicas distintas para resolver una saturación, ellas son:

1. *Saturación progresiva.*
2. *Saturación Progresiva Encadenada.*
3. *Saturación Progresiva con Área de Desborde por Separado.*
4. *Dispersión Doble*.

De acuerdo a la técnica seleccionada se obtendrá la animación correspondiente al momento de resolver una saturación. El alumno podrá seleccionar diferentes técnicas para observar su comportamiento en la resolución de una saturación.

En el caso particular de la dispersión doble, es necesario definir la segunda función de dispersión que indica el desplazamiento que se utilizará para ubicar una clave en saturación. La selección de la segunda función de dispersión se realiza a través del combo desplegable denominado "*Segunda Función de Dispersión*" y presenta tres posibles funciones: *clave módulo 3, clave módulo 5 y clave módulo 7*.

La Figura 5.6 presenta el combo desplegable para elegir la técnica de resolución de colisiones.

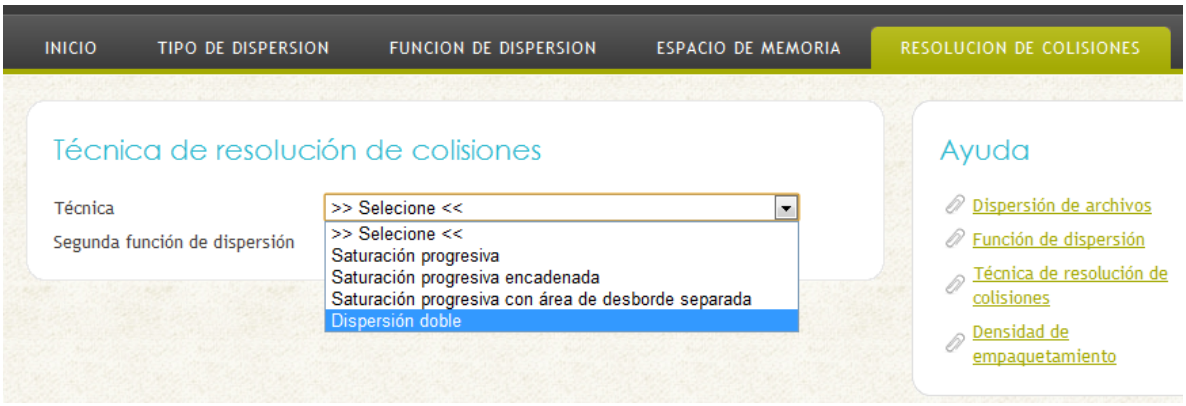


Figura 5.6. Selección de la técnica de resolución de colisiones. Solapa Resolución de Colisiones.

Espacio de direccionamiento dinámico

Cuando se selecciona un espacio de direccionamiento dinámico la cantidad de parámetros que se deben definir es menor. Sólo se habilita la solapa de "*espacio de memoria*", desde donde se configura la "capacidad *de una dirección*" y la "*cantidad de claves a dispersar"*.

La "*cantidad de direcciones*" y la "*densidad de empaquetamiento*" no son conceptos configurables en un entorno dinámico de dispersión. Esto se debe a que el espacio de memoria crece según la necesidad del archivo que se esté dispersando, por ende, no es posible predecir una cantidad de direcciones o una densidad de empaquetamiento.

Para un espacio de direccionamiento dinámico, no es posible seleccionar una función de dispersión, debido a que sólo se implementa una función de dispersión que retorna una cadena de bits que representa a la clave ingresada. La cadena de bits es utilizada para obtener la dirección física en donde debe residir la clave.

Lo mismo ocurre con la técnica de resolución de colisiones. Se implementa el direccionamiento dinámico a través de "*dispersión extensible*" como técnica dinámica para dispersión de archivos.

Validaciones durante la configuración del ambiente de dispersión

E-HASH realiza validaciones a medida que se seleccionan los parámetros necesarios para configurar un ambiente de dispersión. El objetivo de estas validaciones es prevenir que se generen combinaciones de valores que no se correspondan con los permitidos para configurar un ambiente de dispersión razonable.

Inicialmente, se realiza un testeo cuando se selecciona el tipo de espacio de direccionamiento con el que se desea trabajar (estático o dinámico), habilitando y deshabilitando las solapas y las listas de opciones correspondientes, evitando así la elección de parámetros innecesarios para uno u otro tipo de espacio de direccionamiento. No es necesario emitir mensaje alguno que indique que ciertos valores no son posibles de seleccionar para el tipo de direccionamiento elegido, ya que directamente no se permite el acceso a tales valores.

Cuando se configuran los valores de la solapa "*Espacio de Memoria*", se indica explícitamente la cantidad de direcciones, siendo entonces necesario corroborar que la cantidad de claves a dispersar sea menor que la cantidad de lugares en el espacio de memoria seleccionado, es decir, que cada clave tenga al menos un lugar donde pueda ser almacenada. Esto último implica que no se permite tener una densidad de empaquetamiento mayor al 100%.

Cuando la combinación de estos valores da como resultado que la cantidad de claves a dispersar supera a la cantidad de lugares de memoria, aparece un mensaje con la leyenda de "*La configuración elegida no es válida*". La Figura 5.7 presenta el mensaje luego de la elección de valores que provocan una configuración inválida.

Figura 5.7. Mensaje de configuración inválida. Hay una configuración de valores seleccionados que no son aceptados.

La técnica de resolución de colisiones "*Saturación Progresiva Encadenada*" sólo acepta una clave por dirección. Para el caso de haber seleccionado la técnica anteriormente mencionada y haber definido una capacidad para cada dirección mayor a una clave, aparecerá un mensaje de configuración inválida, idéntico al de la Figura 5.7.

Por último, se debe habilitar la simulación del ambiente de dispersión correspondiente. Esta validación transcurre de manera implícita, no existe mensaje alguno, sólo se habilita la solapa correspondiente cuando la selección de parámetros se ha establecido en el marco conceptual permitido para poder tener un ambiente dispersión válido.

5.2.2 Operaciones básicas

E-HASH simula tres tipos de operaciones sobre un archivo de datos. Permite realizar inserciones, eliminaciones y búsquedas de claves. Estas operaciones pueden llevarse a cabo en cualquier orden.

En la práctica real hay operaciones que ocurren mucho más frecuentemente que otras. En el ámbito de las BD, se estima que las búsquedas representan el 80% de las operaciones que se realizan sobre un archivo de datos. El 20% restante se reparte entre las inserciones y las eliminaciones, siendo estás últimas menos frecuentes. Sin embargo,

esto no es más que un dato estadístico, *E-HASH* permite realizar la cantidad de operaciones en cualquier porcentaje.

E-HASH muestra el comportamiento y el resultado final del archivo de datos que se esté dispersando mientras se resuelven estas operaciones, describiendo tanto los procesos como los resultados intermedios.

5.2.3 Modelo de Simulación

El entorno de simulación contiene una descripción de información que permite seguir, y eventualmente, comprender del proceso de dispersión del archivo. El centro de atención de la simulación es describir gráficamente como queda el archivo en memoria secundaria.

La parte central de la interfaz representa la memoria secundaria y se muestra la estructura del archivo de acuerdo a los parámetros seleccionados previamente. La estructura del archivo se simula mediante una tabla, donde la primera columna representa la dirección física de cada posición de memoria asignable por la función de dispersión. Siempre se inicia desde la dirección 0 (cero) y continua secuencialmente hasta alcanzar la cantidad de direcciones seleccionadas cuando se configuró el espacio de memoria. El resto de las columnas representan la capacidad de almacenamiento de cada dirección en particular. Una celda con cuatro guiones (- - - -) representa un lugar libre, es decir, que se encuentra vacío y puede albergar una clave.

La Figura 5.8 muestra la representación que hace *E-HASH* de una posible configuración del archivo en memoria secundaria. En este caso se cuenta con 11 direcciones, numeradas del 0 al 10 y con capacidad para 2 claves cada una.

Figura 5.8. Simulación de la organización de un archivo en la memoria secundaria y panel de lecturas y escrituras sobre el mismo.

Se describe la información sobre los accesos a memoria secundaria, ya sean estos para leer o para escribir. Si bien esta información no es almacenada en disco, es importante para el alumno poder disponer de estos valores para comparar la eficiencia de cada implementación de acuerdo a los parámetros elegidos.

Se describe la cantidad de lecturas y escrituras de la última operación realizada, ya sea una inserción, una eliminación o una búsqueda. Con estos datos se puede notar cuán

costoso fue realizar la operación correspondiente. Unas líneas más abajo se encuentran los totales que representan la suma total de lecturas y escrituras que se han realizado sobre memoria secundaria.

Para poder determinar la cantidad de lecturas y escrituras necesarias ante una operación, se necesita saber cuál es la unidad de transferencia entre la memoria secundaria y la memoria principal, es en esta última donde se realizan realmente las operaciones. *E-HASH* asume que la unidad de transferencia es de una dirección, es decir, que en un acceso a memoria secundaria se puede leer o escribir una sola dirección, sin importar la capacidad que la misma contenga. En la práctica real esto queda determinado por la unidad transferencia del bus de datos entre la memoria primaria y la memoria secundaria y generalmente puede abarcar varias posiciones de memoria.

Sobre la izquierda de la pantalla de simulación se pueden observar dos paneles. El primero es donde se representa la función de hash o función de dispersión. Ante una operación, se debe aplicar la función de dispersión correspondiente y es en este panel donde se podrá apreciar el resultado de aplicar dicha función.

No se visualiza ningún detalle de implementación interna de la función, sino que la misma se muestra como una caja negra, donde se tiene la clave como entrada y en la salida se produce un valor, que representa la dirección asignada.

Cuando se ha seleccionado la dispersión doble como técnica de resolución de colisiones, se debe también seleccionar una segunda función de dispersión. En principio la segunda función no se encuentra visible. La misma aparece por debajo de la primera función de dispersión cuando sucede una saturación.

La Figura 5.9.a presenta el panel para la función de dispersión y la Figura 5.9.b presenta el mismo panel pero con la segunda función de dispersión visible.

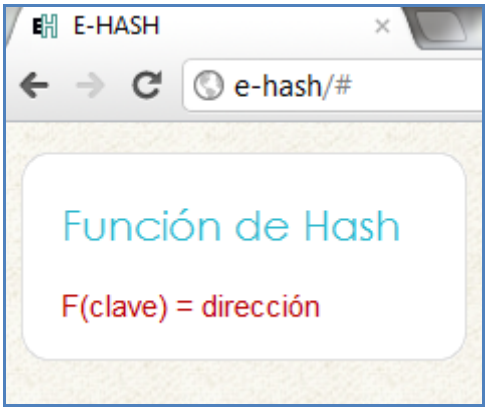


Figura 5.9.a. Panel que representa la animación de la Función de Dispersión.

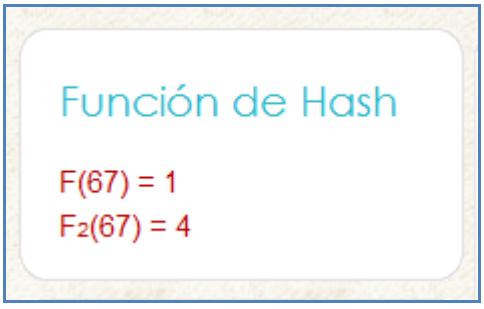


Figura 5.9.b. Panel que representa la animación de la Segunda Función de Dispersión.

Por debajo del panel para la función de dispersión se presenta el panel de los parámetros seleccionados. En este panel se describe la información de todos los parámetros seleccionados durante la configuración del ambiente de dispersión. Es muy importante medida que se está dispersando un archivo que el alumno recuerde con que parámetros está trabajando, a fin de poder comparar las distintas configuraciones y evaluar los distintos comportamientos (Figura 5.10).

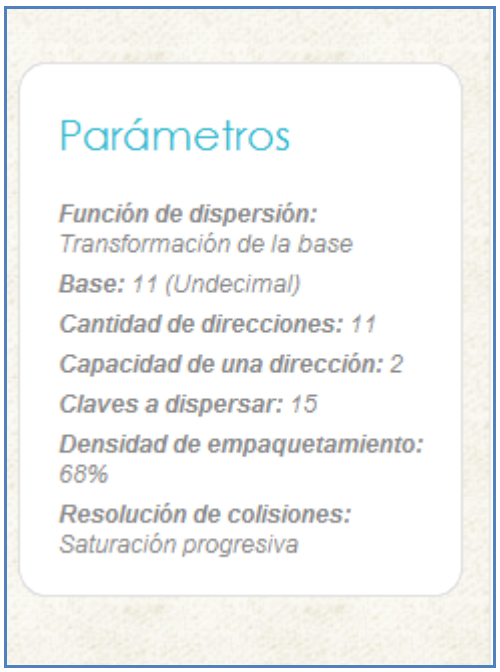


Figura 5.10. Panel en donde se visualizan los parámetros previamente seleccionados.

Sobre la columna de la derecha de la pantalla, junto a la parte central, también se cuenta con dos paneles. El primero registra el historial de operaciones realizadas.

Registrar el historial de operaciones es importante para saber el orden en que transcurrieron dichas operaciones, y el resultado producido por cada una.

La Figura 5.11 presenta el panel del historial de operaciones con la descripción de algunas operaciones realizadas.

Figura 5.11. Panel que muestra el historial de operaciones realizadas.

Debajo del historial de operaciones, se encuentra un breve menú de ayuda. El menú de ayuda siempre se encuentra visible, desde la configuración de los parámetros hasta la simulación del ambiente de dispersión. La ayuda cuenta con la posibilidad de obtener breves explicaciones sobre temas puntuales de dispersión de archivos. El alumno puede en cualquier momento "refrescar" algunos temas, entre ellos, "*función de dispersión", "resolución de colisiones con saturación", "densidad de empaquetamiento"*, entre otros. El objetivo es que a través de la herramienta, no sólo pueda armar y probar ejemplos prácticos, sino que también pueda afianzar conceptos teóricos de los temas en cuestión.

La Figura 5.12 presenta el menú de ayuda con las opciones posibles.

Figura 5.12. Menú de Ayuda.

Por último, sobre el margen superior derecho se encuentra el menú de operaciones posibles a realizar por *E-HASH*. Cada operación se encuentra representada por un botón gráfico y la clave a dispersar debe ser ingresada en el campo de texto correspondiente.

La Tabla 5.2 presenta las operaciones posibles a realizar junto a la simbología correspondiente a cada de ellas.

Simbología	Significado
	Insertar una clave.
	Eliminar una clave.
	Buscar una clave.
	Finalizar simulación.
	Teclado virtual.

Tabla 5.2.Simbología, en tamaño real, utilizada para las operaciones posibles.

La Figura 5.13 presenta el panel que contiene el menú de operaciones y el campo de texto para ingresar las claves a dispersar.

Tabla 5.13. Panel de operaciones.

Cuando se ingresa una clave en el campo de texto, la misma puede ser insertada en el archivo presionando el botón verde con el signo más (), se puede intentar eliminar presionando el botón rojo con el signo menos () o se puede buscar presionando la lupa (). En todos los casos se procede con la animación correspondiente que muestra como se resuelve la operación correspondiente en base a los parámetros seleccionados.

Existe la posibilidad de finalizar una simulación. Mediante la opción, gráficamente descripta con una cruz roja () se cierra la pantalla de simulación y se vuelven a visualizar todas las solapas desde el inicio.

Si se utiliza una pantalla táctil, o si alguna funcionalidad del teclado no responde, o por simple gusto, existe la posibilidad de utilizar un teclado virtual. Mediante la opción cuyo gráfico está representado por un teclado (), a la izquierda del cuadro de texto donde se ingresan los valores de las claves, se puede desplegar dicho teclado virtual. La Figura 5.14 presenta el teclado virtual incorporado en *E-HASH.*

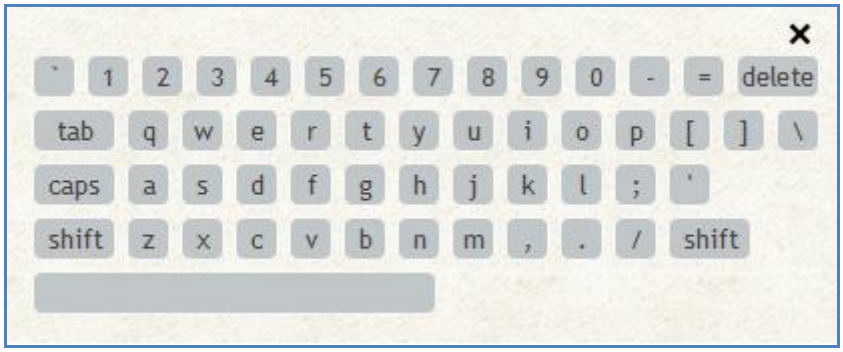


Figura 5.14. Teclado Virtual.

Una de las particularidades del ambiente de simulación es que la mayoría de los paneles descriptos pueden ser trasladados a cualquier lugar de la pantalla. Excepto el historial de operaciones y el menú de ayuda, todos los demás paneles pueden ser llevados al lugar de la pantalla que más cómodo resulte para el alumno que está utilizando la herramienta. Contar con esta posibilidad es cómodo en el uso de la herramienta.

Esta última característica cobra fuerza debido a la diversidad de tamaños de los dispositivos informáticos del mercado, y a la gran variedad de resoluciones de pantalla con la cual estos dispositivos funcionan.

Animación de las operaciones

Cada operación realizada se representa mediante una animación que explica detalladamente cuales son los pasos que se realizan. La resolución de una operación, paso a paso, animada, le permite al alumno comprender mejor cual es el mecanismo utilizado para resolver el problema y obtener el resultado final de la misma.

La animación contiene una simbología gráfica específica, donde cada símbolo, de acuerdo a la operación que se esté realizando, indica un resultado único.

La Tabla 5.3 muestra los distintos gráficos utilizados para la animación.

Simbología	Significado
	Acceso a la memoria secundaria.
	Éxito de una operación.
	En una inserción representa una saturación. Para una eliminación o búsqueda representa que la clave no existe en el archivo.
	Presencia de una clave intrusa. Sólo se utiliza cuando la técnica de resolución de colisiones con saturación es la "Saturación Progresiva Encadenada".
	Mensaje indicador de una inserción en el historial de operaciones.
	Mensaje indicador de una eliminación en el historial de operaciones.
	Mensaje indicador de una búsqueda en el historial de operaciones.
	Bandera que indica alguna información en el historial de operaciones cuando se realiza una operación en dispersion extensible.
	Acceso a la memoria principal. Sólo para dispersión extensible.

Tabla 5.3.Simbología, en tamaño real, utilizada para realizar las animaciones.

Cuando se realiza una operación, lo primero que se anima es el resultado de aplicar la función de dispersión para la clave ingresada. Dicho resultado se muestra mediante un breve parpadeo.

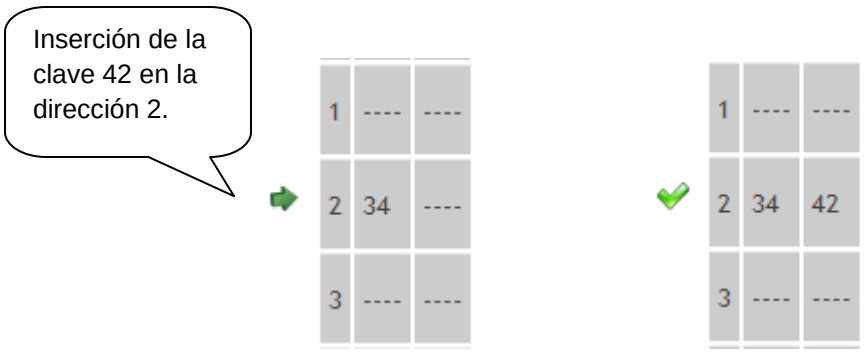
Las animaciones siguientes, contienen muchos pasos en común más allá del tipo de operación realizada, pero para una mejor explicación y comprensión es conveniente separarlas por tipo de operación.

Animación de la operación de inserción

Realizar la animación de una inserción consiste en analizar la técnica de resolución de colisiones que se ha seleccionado.

Cada técnica, al momento de enfrentar una saturación, acciona de manera diferente, por lo tanto, la animación de cada técnica deberá corresponderse con las acciones necesarias en función del estado del archivo.

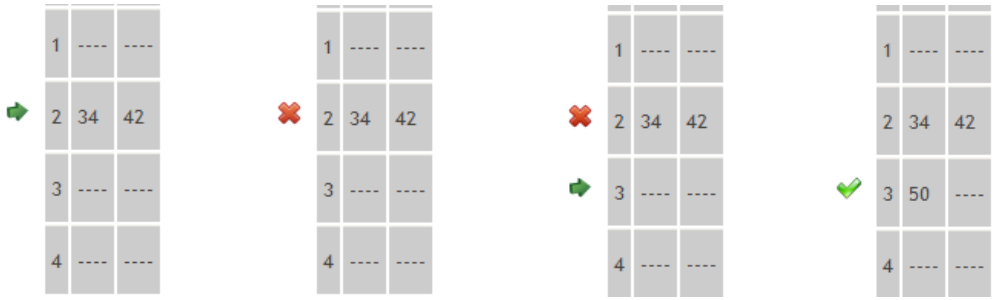
En principio, si no ocurre una saturación, todas las técnicas se comportan de la misma manera, por lo tanto, la animación procede con el mismo criterio. Después que finaliza la animación de la función de dispersión, se describe dinámicamente el acceso a la dirección en memoria secundaria indicada. El acceso a la dirección se presenta de manera animada con el símbolo de acceso a memoria secundaria (➡), parpadeando unos segundos. Si la dirección correspondiente contiene lugar, la clave se almacena en dicha dirección y se detalla el símbolo de operación exitosa (✔). La Figura 5.15 presenta la situación anteriormente descripta.



La Figura 5.15. Secuencia de una inserción sin saturación. Llega la clave 42 y es almacenada en la dirección 2.

Si la dirección base de clave se encuentra completa, se produce saturación y la nueva clave debe ser reubicada. En este caso se muestra el símbolo de inserción con saturación (✖), indicando que la nueva clave no puede ser almacenada en la dirección correspondiente. A partir de aquí, se inicia el proceso de buscar un nuevo lugar para la clave de acuerdo a la técnica de resolución de colisiones seleccionada. Para ello se utilizan los mismos símbolos que para una inserción sin saturación. A medida que se va buscando un lugar disponible para la nueva clave, se indica los accesos a la memoria secundaria (➡), y una vez que se llega a una dirección que contiene espacio disponible se almacena la clave en dicha dirección marcando la operación como exitosa (✔).

La Figura 5.16 presenta la situación descripta ante la inserción de la clave 50 que produce saturación en la dirección 2 del archivo.



La Figura 5.16. Secuencia de una inserción en la dirección 2, se produce saturación y la técnica de resolución de colisiones es "*Saturación Progresiva*".

Existe un caso especial cuando la técnica de resolución de colisiones seleccionada es Saturación Progresiva Encadenada. En este caso, puede suceder que cuando se realice una inserción, la ocurrencia de saturación se deba a que hay una clave intrusa, es decir, una clave que no pertenece a la cadena de sinónimos correspondiente. Para marcar esta situación se muestra en la dirección correspondiente el símbolo de presencia de clave intrusa (✖).

La Figura 5.17 presenta la secuencia de pasos a seguir ante una inserción con presencia de clave intrusa. En este caso cuando llega la clave 36 a su dirección base 3, se encuentra con que la misma está ocupada por la clave 23, clave intrusa. Para resolver tal situación, se debe buscar un nuevo lugar para la clave 23 liberando la dirección 3 para la clave 36. Una vez ubicada la clave 23 se debe actualizar el enlace correspondiente para no perder la cadena de claves sinónimas, en este caso el enlace de la dirección 1. Los tres tildes finales representan las tres direcciones que fueron modificadas por la operatoria, es decir, las direcciones que deben ser escritas en memoria secundaria.

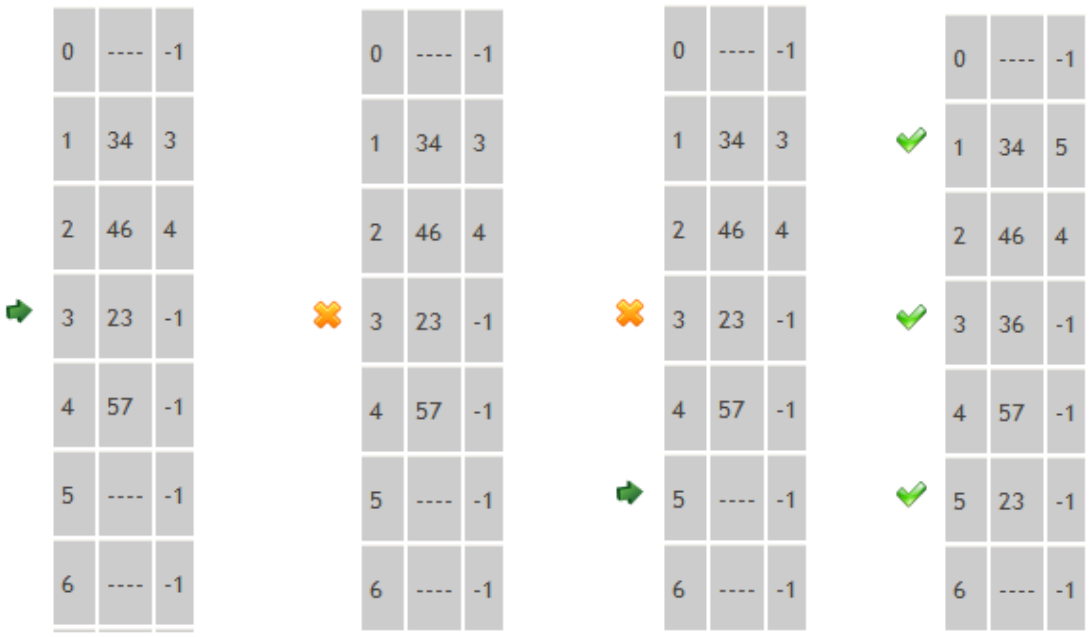


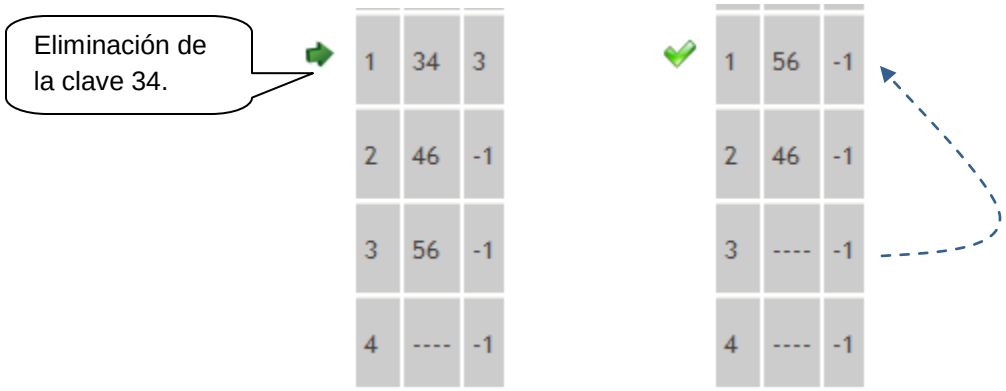
Figura 5.17. Secuencia de inserción para la clave 36, presencia de clave intrusa.

Animación de la eliminación y búsqueda de una clave

La eliminación de una clave implica primero realizar la búsqueda sobre el archivo dispersado. Esto requiere saber en qué dirección reside o debe residir la clave a eliminar. Para iniciar la búsqueda de una clave, se analiza la dirección base de dicha clave. Para ello, se muestra el acceso a la dirección correspondiente mediante el símbolo de acceso a memoria secundaria ( ).

Si la clave reside en su dirección base se muestra en dicha dirección el símbolo de que la operación fue exitosa ( ). Si se trata de una búsqueda no hay más acciones a realizar, pero si la operación es una eliminación pueden existir algunas tareas adicionales para mantener la integridad del archivo dispersado. Las tareas a realizar, luego de una eliminación, dependen exclusivamente de la técnica de resolución de colisiones que se ha seleccionado, la cual establece una forma determinada de organización física del archivo. Estas tareas pueden consistir en *reacomodar enlaces*, *reacomodar claves o establecer marcas de eliminación.* Todas ellas tienen un costo en cantidad de lecturas y escrituras en disco.

La Figura 5.18 presenta la animación de la eliminación de la clave 34, la misma se encuentra en su dirección base.



La Figura 5.18. Secuencia de una eliminación en la dirección 1 con la técnica de resolución de colisiones "*Saturación Progresiva Encadenada*". Se reubica la clave 56 que pertenece a la cadena de sinónimos de la dirección 1.

Si la clave no reside en su dirección base, puede ser que se trate de una clave en saturación o de que la clave no exista en el archivo, por lo tanto, se debe efectuar la búsqueda de la misma. La búsqueda de una clave fuera de su dirección base, depende exclusivamente de la técnica de resolución de colisiones seleccionada, dado que se debe seguir el mismo método que fue utilizado para almacenarla. Las acciones a seguir son variadas y requieren lecturas en memoria secundaria, hasta llegar al estado donde se pueda comprobar si la clave existe o no en el archivo. Las acciones a seguir pueden consistir en *búsquedas secuenciales*, *búsquedas siguiendo una cadena de enlaces*, *búsquedas en un archivo separado* o *búsquedas utilizando desplazamientos*.

En el caso de la clave no se encuentre en el archivo, luego de realizar la búsqueda correspondiente, se muestra en la última dirección inspeccionada el símbolo correspondiente (✖).

Lecturas y escrituras en disco

Los accesos a la memoria secundaria son indicados para todas las técnicas estáticas. Esta información permite evaluar el costo de cada implementación y organización de archivos para el conjunto de claves a dispersar.

Una vez que finaliza la animación de la operación, se actualiza la información de las lecturas y escrituras en base al costo de dicha operación.

La Figura 5.19 muestra el panel de accesos a disco después de haber realizado algunas operaciones.

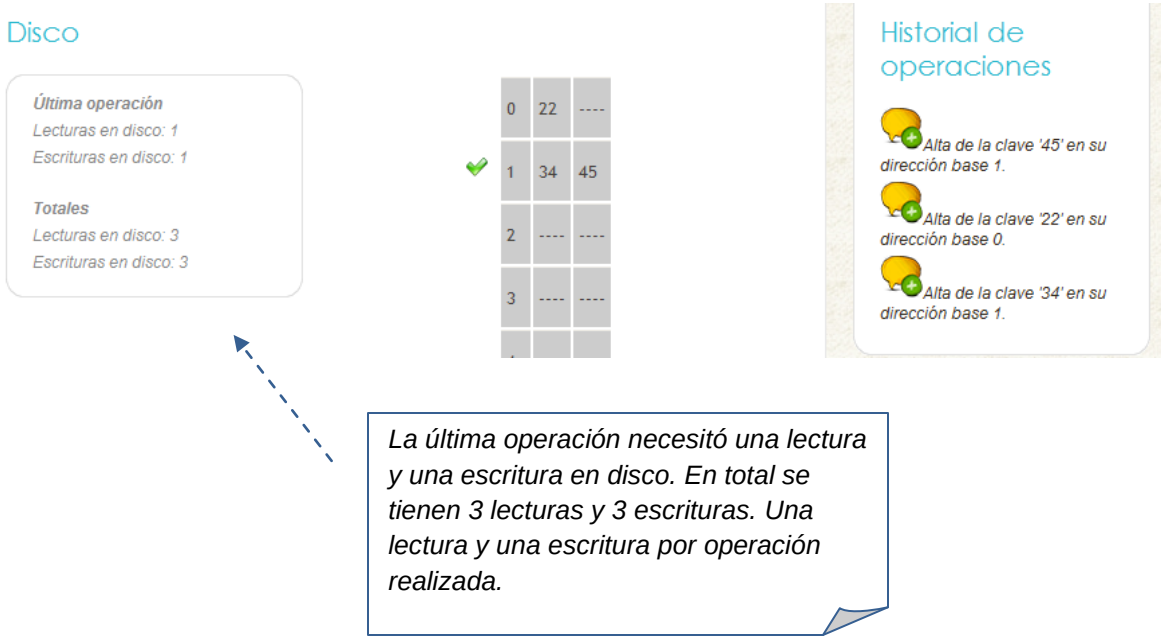


Figura 5.19. Panel de accesos a memoria secundaria que muestra la cantidad de lecturas y escrituras luego de haber realizado un conjunto de operaciones.

Animación de la dispersión extensible

En *E-HASH* la dispersión extensible es la única técnica implementada para un espacio de direccionamiento dinámico.

En la implementación de la dispersión extensible existe una estructura auxiliar almacenada en memoria secundaria pero manipulada en memoria principal mientras se dispersa el archivo. Esto es un detalle muy importante y no debe quedar fuera del proceso de animación, dado que es la estructura la que sirve de intermediaria para llegar a una dirección física en memoria secundaria.

La animación de la dispersión extensible consiste en describir la evolución del archivo a medida que se van dispersando las claves. El archivo y la tabla auxiliar crecen a medida

que se van dispersando las claves, la animación para una operación determinada abarca aspectos de su resolución y de cómo evoluciona el archivo y la tabla correspondiente, y se torna algo más extensa que las animaciones de las técnicas para un espacio de direccionamiento estático. De todas maneras se trata de buscar un punto medio que logra que se pueda entender el proceso de una operación en un lapso de tiempo razonable de animación.

Inicialmente, para simular la organización inicial, se muestra una sola dirección en memoria secundaria y una sola entrada en la estructura auxiliar.

La Figura 5.20 presenta la simulación que realiza *E-HASH* para la organización inicial de las estructuras en dispersión extensible. En el modelo de simulación propuesto, el archivo en disco contiene espacio para dos claves por cada dirección.

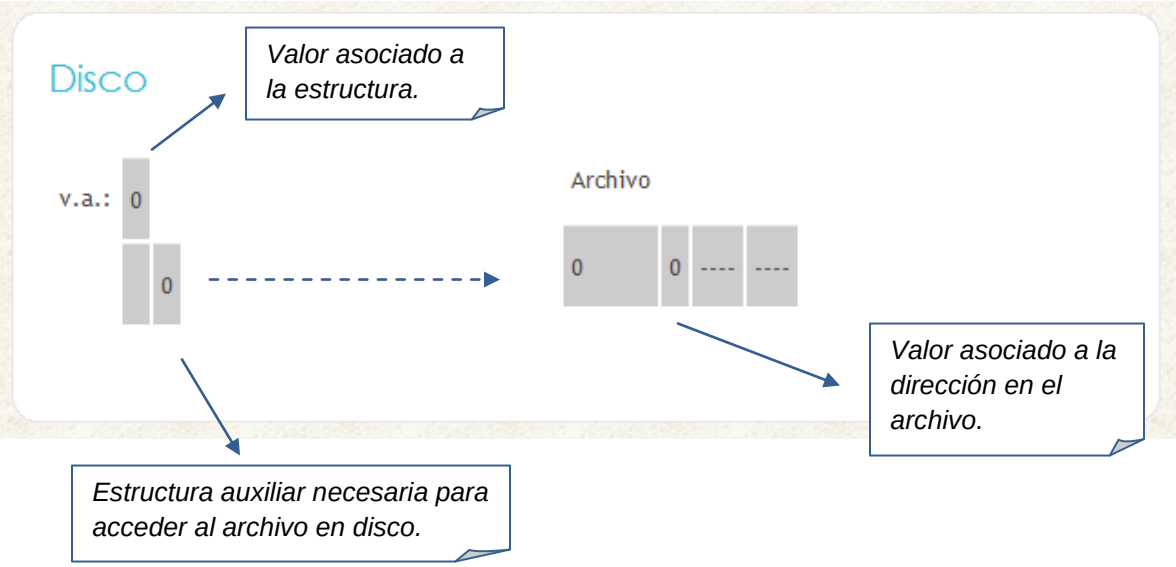


Figura 5.20. Estado inicial de la estructura auxiliar en memoria principal y el archivo en disco utilizados en la dispersión extensible.

Ahora el resultado de aplicar la función de dispersión a una clave es una cadena de bits y no una dirección física. Esta cadena de bits sirve para acceder a la tabla que se encuentra en memoria principal y obtener la dirección física para la clave se está dispersando.

La Figura 5.21.a y 5.21.b presentan el resultado de la función de dispersión en dos instancias distintas para la clave 120. En el primer caso, el valor asociado de la tabla indica que no se deben utilizar ningún bit, existe una sola dirección en memoria secundaria. En el segundo caso, se indica que se deben tomar 2 bits como entrada a la tabla para obtener la dirección física de la clave, motivo por el cual ambos bits se muestran con formato de texto en negrita.

Figura 5.21.a. Resultado de la función de dispersión para la clave 120. En este caso el valor asociado de la tabla es 0. No se utiliza ningún bit.

Figura 5.21.b. Resultado de la función de dispersión para la clave 120. En este caso el valor asociado de la tabla es 2 y se deben utilizar dicha cantidad de bits como entrada a la tabla.

Luego de resolver la animación de la función de dispersión, se describe el acceso a la tabla en memoria principal. Para acceder a la tabla se utilizan un subconjunto de los bits resultantes de la función de dispersión. Entonces, se busca en la tabla una entrada que coincida con este subconjunto de bits. Esto se indica a través del símbolo de acceso memoria principal (➧) mediante un breve parpadeo en la posición de la tabla correspondiente.

La entrada en la tabla en memoria principal contiene una dirección que se corresponde con un lugar en disco, es a ese lugar adonde se debe acceder para realizar la operación correspondiente.

La animación del acceso a la dirección en memoria secundaria se realiza de manera similar que la realizada anteriormente, pero en este caso se muestra el símbolo de acceso a la memoria secundaria (➡) mediante un breve parpadeo.

Cuando el nodo accedido en memoria secundaria posee lugar disponible para guardar la clave que se está dispersando, se almacena la clave en la ubicación de memoria correspondiente, y se muestra el símbolo de que la operación fue exitosa (✔). Lo mismo sucede si la operación es una búsqueda o una eliminación y la clave existe en el archivo.

La Figura 5.22 muestra la estructura auxiliar y el archivo en disco con una secuencia de inserciones, pero sin que aún suceda saturación.

Figura 5.22. Inserción sin desborde. La llegada de las claves *alfa* y *beta* van al único lugar de memoria disponible, completando la misma en su capacidad total.

Cuando al insertar una clave se produce saturación, se muestra en la ubicación correspondiente el símbolo de inserción con saturación (✖).

Lo mismo sucede si la operación se trata de una eliminación o una búsqueda no exitosa, en ese caso finaliza la operación con un resultado no satisfactorio. No se debe continuar buscando, ya se puede asegurar que la clave no existe en el archivo.

Una saturación siempre debe ser resuelta y la clave debe residir en algún lugar de memoria secundaria. La animación de la resolución de una saturación en dispersión extensible involucra varios pasos; no obstante, en *E-HASH* se presenta una animación explicativa en pocos estadios:

- Se anima el incremento en uno del valor asociado de la dirección que se ha saturado y su correspondiente comparación con el valor asociado de la tabla.
- Se procede con la creación de una dirección nueva para el archivo en memoria secundaria, que junto con la dirección que se ha saturado contendrán las claves involucradas, las que están en la dirección saturada más la nueva clave que ha provocado la correspondiente saturación. Ambos lugares de memoria presentan el mismo valor asociado.
- En caso necesario se duplica la tabla asociada al archivo de datos.

En esta instancia se debe decidir si es necesario duplicar la cantidad de direcciones de la tabla con el fin de generar suficientes entradas para satisfacer la cantidad de ubicaciones en disco. Se comparan los valores asociados a ambas estructuras, el valor asociado del nodo saturado y el valor asociado de la tabla en memoria principal. Ambos valores parpadean al mismo tiempo mostrando la comparación entre los mismos.

Siempre que ocurre saturación se debe añadir una dirección más al archivo de datos pero no siempre la tabla es duplicada. La secuencia de Figuras 5.23.a, 5.23.b y 5.23.c presentan la secuencia de inserciones de las claves presentadas en la Tabla 5.4, provocando saturación y duplicando la tabla en memoria correspondiente.

Clave a dispersar	Cadena de 8 bits retornados por la función de dispersión.
Alfa	10010100
Beta	10011100
gamma	00000011
delta	00001010

Tabla 5.4. Claves a dispersar junto con los valores retornados por la función de dispersión extensible.

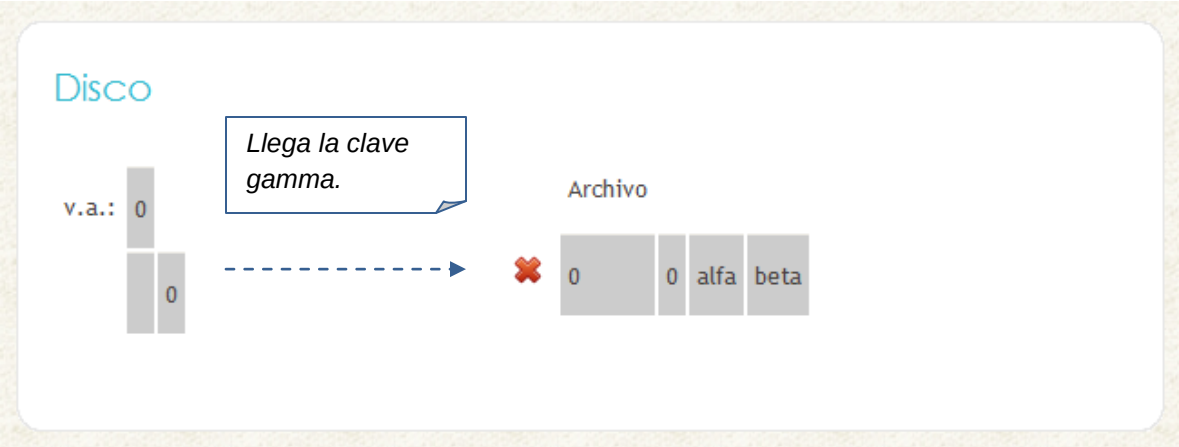


Figura 5.23.a. La llegada de gamma produce la saturación de la dirección 0 del archivo de datos.

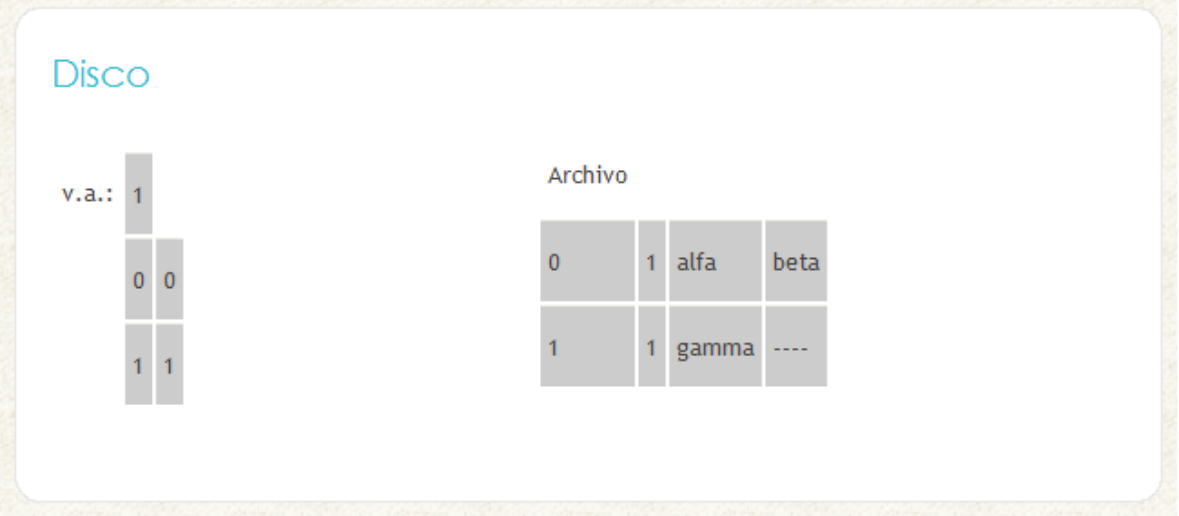


Figura 5.23.b. Resolución de la saturación provocada por la clave gamma. Se vuelven a dispersar nuevamente las claves involucradas, debido al cambio del valor asociado de la tabla auxiliar en memoria principal.

Cuando llega la clave *delta*, vuelve a producir una saturación en la dirección 0 del archivo de datos. La dirección 1 no se afectada por la saturación correspondiente, la misma queda intacta.

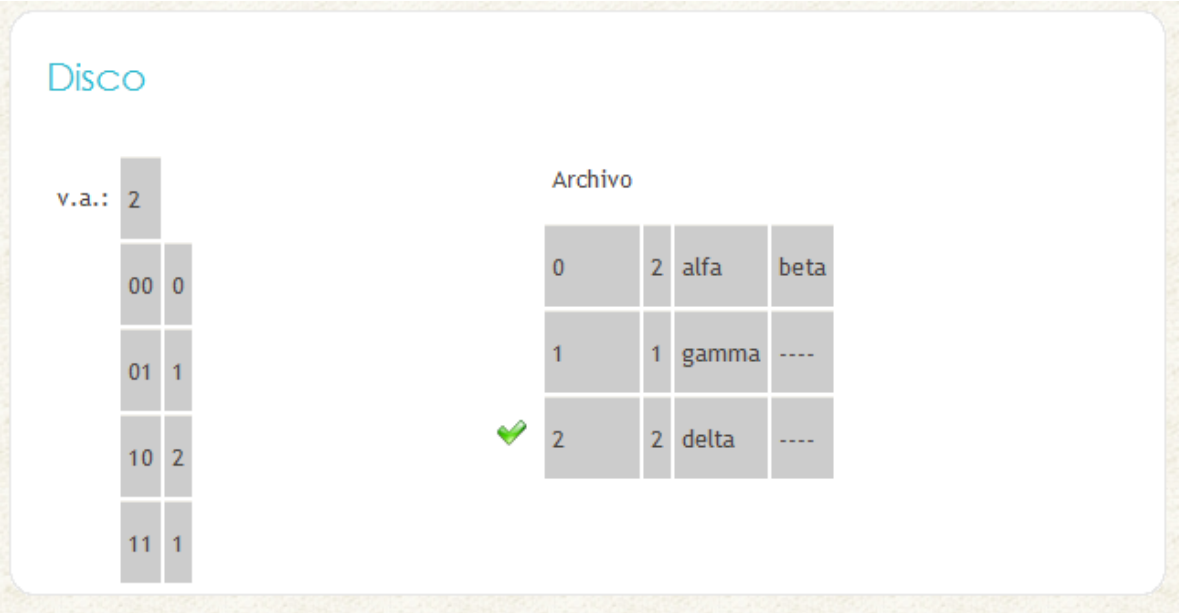


Figura 5.23.c. Estructura de la tabla auxiliar en memoria principal y el archivo de datos en disco luego de haber dispersado las claves, *alfa*, *beta*, *gamma* y *delta*.

Sólo resta volver a dispersar las claves de la dirección que se ha saturado y la nueva clave que ha provocado la saturación. Se debe recordar que el incremento de los valores asociados se utiliza para la redistribución de dichas claves.

Para realizar la redistribución de las claves involucradas se toman una a una, comenzando por las que están en la dirección saturada hasta finalizar con la nueva clave, y se les aplica nuevamente la función de dispersión para volver a ubicarlas en alguna de ambas direcciones. Para cada clave involucrada del resultado de la función de dispersión se toma un bit más, esto se debe al incremento de los valores asociados respectivos.

La animación corresponde en volver a mostrar el resultado de la función de dispersión, junto con la cantidad de bits necesarios, mostrar el acceso a la posición correspondiente de la tabla en memoria principal mediante su respectivo símbolo (➡), y posteriormente el acceso a la memoria secundaria (➡) en donde debe ser ubicada la clave.

Para cada clave que es nuevamente dispersada se almacena en su dirección correspondiente y se muestra el símbolo de que la inserción fue exitosa (✔). La última clave a dispersar, que es la clave que originalmente produjo la saturación, puede volver a

provocar una nueva saturación, en ese caso se muestra el símbolo de inserción con saturación (✖) y el proceso de resolución de la misma se inicia nuevamente.

5.3 Descripción de diseño e interface

E-HASH presenta un diseño formal e intuitivo, con un estilo para abordar y considerar el tema en cuestión con la importancia que el mismo requiere. No contiene pantallas extremadamente cargadas de información, se cuenta con la información justa y precisa para que el alumno pueda adquirir los conceptos necesarios y entender claramente cada tema relacionado con un ambiente de dispersión.

Debido a la importancia de cada uno de los temas que involucra dispersar un archivo, se presenta una interface evolutiva dividida por solapas, donde cada solapa se concentra en tratar un tema en particular. El diseño de la interface de esta manera, permite que el alumno pueda comprender cuán importante es la elección de cada parámetro correspondiente para un ambiente de dispersión.

Cuando se configura un ambiente de dispersión, son varios parámetros que se deben tener en cuenta para evaluar su desempeño. Dividir cada conjunto de parámetros bajo su categoría, hace que el alumno no tenga que visualizar, al mismo tiempo, todo el conjunto de información y pueda concentrarse solamente en el conjunto de interés.

Validaciones durante la simulación

Las validaciones durante el proceso de simulación evitan que el alumno pueda realizar acciones no pertinentes para el ambiente de dispersión configurado.

E-HASH requiere que se cumplan ciertas condiciones mínimas para que pueda llevar a cabo cualquiera de las operaciones posibles sobre un archivo.

Como primera condición una clave debe estar compuesta por al menos un elemento. Esto significa que la clave debe contener al menos un carácter (numérico o alfabético). La Figura 5.24 presenta el mensaje descripto cuando se intenta operar sin valor de clave alguno. Como máximo se permite una clave de ocho caracteres.

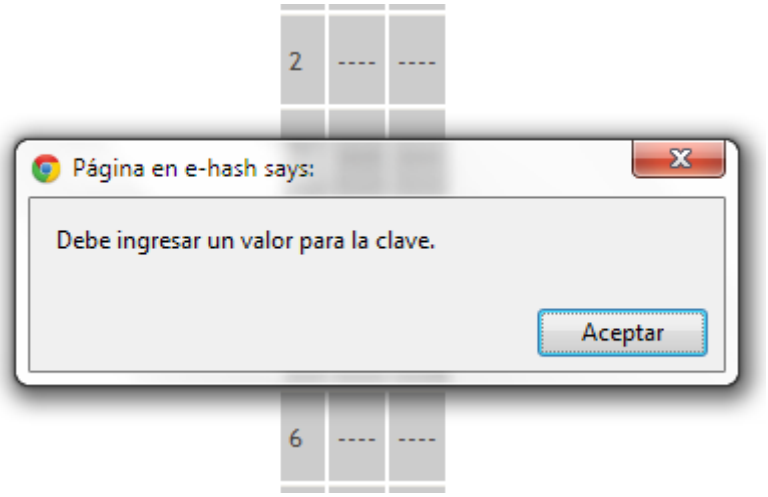


Figura 5.24. Mensaje de que no se ha ingresado un valor de clave para realizar la operación deseada.

La organización de archivos mediante dispersión no permite manejar claves duplicadas en el archivo, algo que es muy importante respetar y que *E-HASH* tiene incorporado. Al intentar dispersar una clave que ya existe en el archivo, se notará un breve parpadeo de toda la dirección en donde reside la clave que coincide con la ingresada. Si bien en este caso no se expresa mensaje alguno, se indica de manera animada que la clave ya existe en el archivo y que no es posible volver a insertarla.

Cuando se configuró el ambiente de dispersión, uno de los parámetros definidos fue la cantidad de claves a dispersar. Cuando se intenta ingresar una clave más de las, se presenta un mensaje para indicar que se llego al máximo de claves posibles a dispersar. La Figura 5.25 presenta el mensaje correspondiente a la situación descripta.

Si se desea verificar el comportamiento al dispersar una clave y se ha llegado al límite máximo permitido, para no volver a configurar nuevamente el ambiente de dispersión, se puede eliminar algunas de las claves existentes, reduciendo así el número de claves dispersadas para darle lugar a la nueva clave.

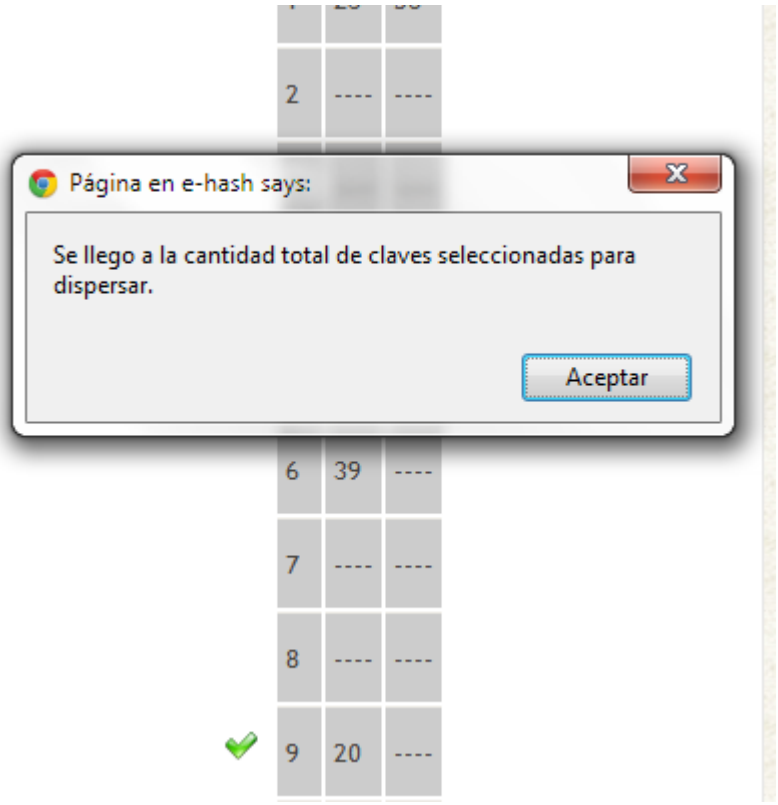


Figura 5.25. Mensaje de que no es posible ingresar más claves en el archivo, aunque el mismo aún posea lugares sin ocupar.

Una restricción importante es que cada operación para ser iniciada, debe esperar a que finalice la operación inmediata anterior, es decir, que no es posible realizar más de una operación al mismo tiempo. Esto se ha diseñado así para evitar confusiones durante la simulación de una operación determinada.

Respetar los tiempos de simulación de cada operación, permite que el alumno pueda concentrar su atención en cada paso de la operatoria en curso, lo que posibilita mayor atención en la operación.

En la Figura 5.26 presenta al menú de operaciones cuando no hay una operatoria en curso.

Figura 5.26. Menú de operaciones.

Al iniciar una operación, las opciones son reemplazadas por una barra de progreso. Una vez que la simulación se finaliza, se vuelven a presentar las opciones correspondientes.

La Figura 5.27 presenta como se ve el menú de operaciones cuando existe una operación en curso.

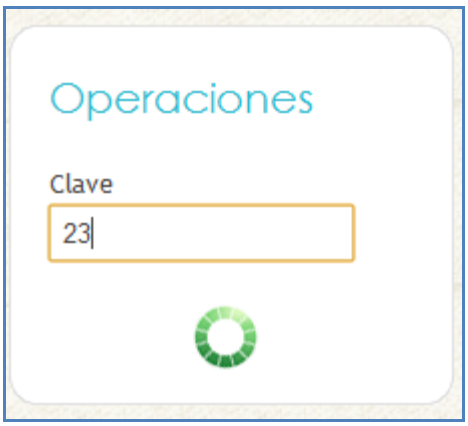


Figura 5.27. Operación en curso.

Objetivo del diseño

La interface y animación que presenta *E-HASH* está pensada para que el alumno pueda, de una manera amigable y entretenida, comprender los conceptos más importantes que brinda la organización de archivos mediante dispersión.

Sin realizar pasos extremadamente largos de animación se trata de mostrar los aspectos más importantes cuando se realiza una operación, con el fin de que la misma sea exhaustivamente explicada y pueda ser comprendida por el estudiante.

Capítulo 6

Tecnología utilizada.

6.1 Introducción

El objetivo de este capítulo es describir la tecnología utilizada en la creación de *E-HASH*, como así también cuales fueron los motivos para su elección.

Seleccionar una tecnología adecuada para este trabajo, es parte de lo que se denomina *arquitectura del software*, y la misma cumple un papel importante a la hora de determinar el éxito o el fracaso de un desarrollo desde el principio a fin.

La arquitectura del software se refiere al diseño de más alto nivel de la estructura de un sistema, programa o aplicación y tiene la responsabilidad de diversas tareas, definir los módulos principales, definir las responsabilidades que tendrá cada uno de estos módulos, definir la interacción que existirá entre dichos módulos, control y flujo de datos, entre otras.

6.2 Elección de la tecnología

En el momento de analizar qué tecnología utilizar para la creación de *E-HASH*, existieron dos puntos importantes:

- la facilidad de acceso a la herramienta, y
- las animaciones, a través de las cuales, se muestra como queda, paso a paso, un archivo dispersado.

La tecnología web avanzó a lo largo de tiempo, esto hace que la información y las herramientas estén cada vez más al alcance de la mano de los usuarios. Ante esta situación, uno de los objetivos planteados fue obtener una herramienta que pueda ser accedida vía web. En función de este objetivo, surgió la posibilidad que aquellos usuarios que tengan acceso a internet puedan acceder utilizando la red, mientras los que no, se puedan bajar la herramienta e instalarla sin inconvenientes. Después de analizar lo

anteriormente mencionado, se planteo que *E-HASH* sea creado con tecnología web que se ejecuta en el medio informático del cliente, algo conocido como "*tecnología que se ejecuta del lado del cliente*".

Es necesario aclarar la diferencia entre el software que se ejecuta del lado del cliente y el software que se ejecuta del lado del servidor. Cuando el software es ejecutado del lado del cliente, todo el procesamiento lo realiza la máquina donde se está ejecutando la aplicación. Una definición más formal sería que la ejecución de los programas o scripts[7] se realizan en el navegador del usuario, es decir, en la computadora del usuario. A diferencia del software que se ejecuta del lado del servidor, un navegador del lado del cliente genera solicitudes a un servidor, en el cual se realiza el trabajo, y posteriormente se devuelve una respuesta como resultado de tal solicitud.

Desarrollar la herramienta con tecnología web que se ejecute del lado del cliente, permite también que *E-HASH* sea alojado en un servidor y que pueda ser accedido a través de internet, aunque el procesamiento se realice en el cliente; o que el usuario se lo descargue en su computadora y pueda utilizarlo sin más requerimientos que tener un navegador web instalado en su computadora.

Otro de los puntos a considerar para la elección de la tecnología utilizada fue contar con la posibilidad de realizar animaciones que facilitaran el aprendizaje en técnicas de dispersión.

Presentar animaciones hace que la herramienta sea amigable y atrape, aún más, el interés del alumno por querer trabajar con ella. Es importante que la utilización de *E-HASH* no provoque una carga extra para el alumno, sino que por el contrario, que se focalice en el aprendizaje de dispersión de archivos sin tener que poseer conocimientos previos acerca de la utilización de la herramienta y logre interactuar con ella de forma sencilla y rápida.

E-HASH utiliza la librería *JQuery*, que será explicada en detalle más adelante, la cual permite realizar animaciones de manera sencilla y cumple con todos los objetivos propuestos. *JQuery* está implementada en su totalidad utilizando el lenguaje *JavaScript,* el

[7] En informática, un script es un programa usualmente simple y por lo regular se almacena en un archivo de texto plano. El uso habitual de los scripts es realizar diversas tareas, como combinar componentes, interactuar con el sistema operativo o con el usuario [26].

cual se ejecuta del lado del cliente y satisface también así otro de los puntos a cumplir por la herramienta presentada.

En resumen, se puede decir que *E-HASH,* en su implementación*,* es un software web, escrito en código *HTML (HyperText Markup Language)*, con el lenguaje *JavaScript* embebido, utilizando como complemento la librería *JQuery* para realizar las animaciones necesarias.

6.3 Descripción de la tecnología

La finalidad de esta sección es describir muy brevemente la tecnología utilizada en la construcción de *E-HASH*. Es importante aclarar que la intención es dar una idea de las herramientas utilizadas, sin profundizar en exceso cada una de ellas dado que excede el marco de este trabajo.

HTML (HyperText Markup Language - Lenguaje de Marcado de HiperTexto)

HTML es el lenguaje con el que se definen las páginas web. Se trata de un conjunto de etiquetas que sirven para definir el texto y otros elementos que conforman una página web.

HTML se creó en un principio con el objetivo de mostrar información con formato de texto y algunas imágenes. Originalmente, no se pensó que llegara a ser utilizado para crear áreas de ocio y consulta con carácter multimedia, de modo que, *HTML* se creó sin dar respuesta a todos los posibles usos que se le iba a dar.

HTML es un lenguaje de marcación de elementos para la creación de documentos *hipertexto*[8], es fácil de aprender y entender, lo que permite que cualquier persona, aunque sea un principiante en informática, pueda enfrentarse a la tarea de crear una página web con dicho lenguaje.

Para escribir el código de una página *HTML* se utiliza directamente cualquier procesador de texto tradicional. El único detalle a tener en cuenta es que el archivo que se debe

[8] En informática, es el nombre que recibe el texto que en la pantalla de un dispositivo electrónico, permite conducir a otros textos relacionados, pulsando con el ratón o el teclado en ciertas zonas sensibles y destacadas. La forma más habitual de hipertexto en informática es la de hipervínculos o referencias cruzadas automáticas que van a otros documentos. Si el usuario selecciona un hipervínculo el programa muestra el documento enlazado [31].

generar es un archivo con extensión "*html*" o "*htm*". Esto demuestra lo sencillo y práctico que resulta utilizar *HTML*, se pueden lograr interesantes resultados rápidamente.

HTML consta de etiquetas de apertura, que tienen la forma de "<*table*>" o "<label>" y etiquetas de cierre "</table>" o "</label>". Cada etiqueta posee un significado y una interpretación, la Tabla 6.1 muestra algunas de las etiquetas más comunes que componen el *HTML*. Existen decenas de etiquetas más y cada una posee su propio significado. Se debe recordar que *HTML* es propiamente un lenguaje muy completo [14].

Etiqueta	Significado
<html> </html>	Define el inicio del documento HTML.
<title> </title>	Define el título de la página.
<style> </style>	Coloca el estilo interno de la página.
<body> </body>	Contenido principal o cuerpo del documento.
<table> </table>	Define una tabla.
<a> 	Hipervínculo o enlace
o	Ambas representan un texto en negrita
	Representa un texto en cursiva
<u> </u>	Subrayado
	Salto de párrafo
 	Salto de línea
<img />	Imagen

Tabla 6.1. Algunas etiquetas de *HTML*.

JavaScript

JavaScript es un lenguaje de programación interpretado. Se define como orientado objetos, basados en prototipos, débilmente tipado y dinámico.

Se utiliza principalmente en su forma del lado del cliente (*client-side*), implementado como parte de un navegador web, permite mejoras en la interfaz de usuario y páginas web dinámicas, aunque existe una forma de *JavaScript* del lado del servidor (*Server-side JavaScript o SSJS*).

JavaScript se diseñó con una sintaxis similar a la del leguaje *C*, aunque adopta nombres y convenciones del lenguaje de programación *Java*. Sin embargo *Java* y *JavaScript* no están relacionados, y tienen semánticas y propósitos diferentes.

Todos los navegadores modernos interpretan código *JavaScript,* independientemente de la plataforma en la que se estén ejecutando. Para interactuar con una página web se provee al lenguaje *JavaScript* de una implementación del *Modelo en Objetos para la Representación de Documentos* (*DOM - Document Object Model*).

DOM es esencialmente una interfaz de programación de aplicaciones (*API*) que proporciona un conjunto estándar de objetos para representar documentos *HTML* y *XML*[9], un modelo estándar sobre cómo pueden combinarse dichos objetos, y una interfaz estándar para acceder a ellos y manipularlos. A través del *DOM*, los programas pueden acceder y modificar el contenido, estructura y estilo de los documentos *HTML* y *XML*, que es para lo que principalmente se diseñó [28].

Tradicionalmente, *JavaScript* se ha utilizado en páginas web *HTML* para realizar operaciones y únicamente en el marco de la aplicación cliente, sin acceso a funciones del servidor.

JavaScript se interpreta en el agente de usuario (aplicación informática que funciona como cliente en un protocolo de red), al mismo tiempo que las sentencias van descargándose junto con el código *HTML*.

Esta particularidad conlleva una notable serie de ventajas y desventajas según el uso que se le deba dar y teniendo en cuenta la relación que se establece entre el mecanismo *cliente-servidor*. Para explicar con pocas palabras dicha relación, se puede decir que el servidor envía los datos al cliente y estos datos pueden llegar en dos formatos:

- en formato de texto (o *ASCII*) o
- en formato binario o código máquina.

[9] XML, siglas en inglés de eXtensible Markup Language (lenguaje de marcas extensible), es un metalenguaje extensible de etiquetas. Permite definir la gramática de lenguajes específicos. Por lo tanto XML no es realmente un lenguaje en particular, sino una manera de definir lenguajes para diferentes necesidades, de ahí que se le denomine metalenguaje [29].

El cliente sabe como comprender solo el formato binario (es decir, 0 y 1), por lo que si los datos llegan en este formato son inmediatamente ejecutables (no dejan abierta la posibilidad de efectuar controles), mientras que si el formato es otro, dichos datos tendrán que ser interpretados y traducidos al formato binario. En estos casos el cliente necesita un filtro, o mejor dicho, un intérprete que sepa leer estos datos y los pueda traducir al formato binario.

Los datos en formato texto son visibles al usuario como simples combinaciones de caracteres y de palabras y son, por tanto, fáciles de manipular. Se requiere más tiempo para su interpretación a causa de los distintos pasos y de las transformaciones a las que deben someterse para que el cliente pueda comprenderlos. Los datos en formato binario, sin embargo, son muy difíciles de comprender por el usuario, pero inmediatamente ejecutables por el cliente ya que no requieren fases intermedias.

Algunas de las ventajas y desventajas respectivas de los lenguajes de scripting (programa usualmente simple, que por lo regular se almacena en un archivo de texto plano, como lo es *JavaScript*) son:

1. El lenguaje de scripting es seguro y fiable.

2. Los script generados tienen capacidades limitadas, por razones de seguridad, por lo cual no es posible hacer todo con *JavaScript*, sino que es necesario usarlo conjuntamente con otros lenguajes, posiblemente más seguros, como Java.

3. Un problema importante es que el código es visible y puede ser leído por cualquiera, incluso si está protegido con las leyes del *copyright*[10] *(derecho de autor)*.

4. El código *JavaScript* se ejecuta en el cliente por lo que el servidor no es solicitado más de lo debido; un script ejecutado en el servidor, sin embargo, sometería a éste a dura prueba y los servidores de capacidades más limitadas podrían resentir de una continua solicitud por un mayor número de usuarios.

[10] El derecho de autor es un conjunto de normas jurídicas y principios que regulan los derechos morales y patrimoniales que la ley concede a los autores, por el solo hecho de la creación de una obra literaria, artística, musical, científica o didáctica, esté publicada o inédita. En el derecho anglosajón se utiliza la noción de copyright (traducido literalmente como "derecho de copia") que, por lo general, comprende la parte patrimonial de los derechos de autor [27].

5. El código del script debe descargarse completamente antes de poder ser ejecutado. Si los datos que un script utiliza son muchos, el tiempo que tardará en descargarse será extenso.

JQuery

Las aplicaciones web son cada vez más complejas ya que incorporan varios efectos visuales. El desarrollo de algunos efectos desde cero puede resultar complicado y requerir bastante tiempo. En este tipo de situaciones el empleo de librerías como *jQuery* facilita el desarrollo de la aplicación.

JQuery es una librería de *JavaScript* que permite simplificar la manera de interactuar con los documentos *HTML*, manipular el árbol *DOM*, manejar eventos, desarrollar animaciones y agregar interacción con la técnica *AJAX*[11] a páginas web. Fue presentada el 14 de enero de 2006 en el BarCamp NYC [16].

JQuery es software libre y de código abierto, bajo la Licencia *MIT* (*Massachusetts Institute of Technology*) y la *Licencia Publica General* de *GNU v2*, permitiendo su uso en proyectos libres y privativos [16].

JQuery, al igual que otras bibliotecas, ofrece una serie de funcionalidades basadas en *JavaScript* que de otra manera requerirían de mucho más código, es decir, con las funciones propias de esta biblioteca se logran grandes resultados en menos tiempo y espacio. JQuery consiste en un único fichero *JavaScript* que contiene las funcionalidades comunes de *DOM*, eventos, efectos y *AJAX*.

La característica principal de la biblioteca es que permite cambiar el contenido de una página web sin necesidad de recargarla, mediante la manipulación del árbol *DOM* y peticiones *AJAX*. Para ello utiliza las funciones *$()* o *jQuery()*.

[11] Ajax, acrónimo de Asynchronous JavaScript And XML (JavaScript asíncrono y XML), es una técnica de desarrollo web para crear aplicaciones interactivas. Estas aplicaciones se ejecutan en el cliente, es decir, en el navegador de los usuarios mientras se mantiene la comunicación asíncrona con el servidor en segundo plano. De esta forma es posible realizar cambios sobre las páginas sin necesidad de recargarlas, lo que significa aumentar la interactividad, velocidad y usabilidad en las aplicaciones [30].

La forma de interactuar con la página es mediante la función $(), un alias de JQuery(), que recibe como parámetro una expresión *CSS* o el nombre de una etiqueta *HTML* y devuelve todos los nodos (elementos) que concuerden con la expresión.

$("#tabla"); // Devolverá el elemento con *id = "tabla".*

$(".activo"); // Devolverá una matriz de elementos con propiedad *class = "activo".*

Una vez obtenidos los nodos, se les puede aplicar cualquiera de las funciones que facilita la biblioteca.

Se elimina el estilo (con *removeClass()*) y se aplica uno nuevo (con *addClass()*) a todos los nodos con propiedad *class = "activo"*

$(".activo").removeClass("activo").addClass("inactivo");

Ejemplo de efectos gráficos:

$(".activo").slideToggle("slow"); // Anima todos los componentes con class="activo"

Generalmente, antes de realizar cualquier acción en el documento con *jQuery()*, el documento debe estar listo. Para eso usamos *$(document).ready()*, por ejemplo:

```
$(document).ready(function() {
    // Código del documento.
});
```

Cuando se utiliza una librería, es importante adaptarse a su mecanismo de uso. Tratar de imponer conocimientos sobre un tema y aplicarlos a la fuerza, puede ser frustrante. Lo más adecuado es ir viendo cual es la mecánica de trabajar con dicha librería, evaluando problemas muy sencillos e ir complicándolo a medida que se comprende su funcionamiento.

Capítulo 7

Conclusiones y Trabajos Futuros

7.1 Conclusiones

La dispersión constituye una de las estrategias más importantes para organizar archivos de datos. Mediante esta estrategia se logra una organización de archivos con “acceso directo”. Esto se debe a que para la mayoría de las operaciones (alta, eliminación, modificación o consulta) se necesita en promedio menos de dos accesos a memoria secundaria.

Uno de los objetivos previstos en la asignatura *Introducción a las Bases de Datos* es que el alumno comprenda los beneficios de este tipo de organización de archivos, junto a como se desarrollan los algoritmos para las operaciones ante mencionadas.

E-HASH constituye una herramienta interactiva que le permite al alumno resolver los ejercicios definidos en las guías prácticas, a fin de agilizar el proceso de aprendizaje. El alumno puede plantear sus propias configuraciones y realizar la operatoria sobre archivos, en un ambiente de simulación adecuado.

No es propósito de *E-HASH* reemplazar la práctica tradicional en papel y lápiz, sino brindar un complemento para el alumno al momento de evacuar una duda o comprender mejor una situación específica. El alumno además, puede sentirse seguro dado que la herramienta generada sigue el marco conceptual de la asignatura mencionada.

En la experiencia como docentes de la asignatura, concluimos que desarrollar herramientas con un propósito educativo (*CASER, HEA, E-HASH*), refuerza e incentiva al alumno de informática a experimentar nuevas alternativas para la ejercitación propuesta. Poder asistir al alumno en cualquier momento agiliza su proceso de aprendizaje, debido a que no siempre deberá esperar a un docente para realizar consultas.

La herramienta aprovecha los avances tecnológicos en pos de fortalecer un proceso de enseñanza y aprendizaje, dado que la mayoría de los alumnos de hoy en día cuentan con algún medio informático.

Se espera que E-HASH resulte en la práctica en una herramienta robusta, eficaz y amigable, de total utilidad, tanto para alumnos como para los docentes respecto a la extensa temática que involucra Dispersión de Archivos como técnica de organización de Archivos.

7.2 Trabajos futuros

E-HASH es una herramienta que cubre todos los objetivos para la cual fue construida, pero no obstante, se debe ser cauto y esperar observaciones en cuanto a su utilización por parte de los alumnos.

En una primera etapa, y luego de una prueba de campo, se espera poder optimizar la interface y animación, de acuerdo a los requerimientos de usabilidad que se planteen.

Posteriormente se prevé agregar nuevas funciones de dispersión y nuevas técnicas de resolución de colisiones, con el fin de ampliar el marco de comparación entre las diversas configuraciones posibles.

Finalmente se tiene previsto agregar la posibilidad de guardar y/o exportar el resultado del ambiente de dispersión en algún formato que sea de utilidad común por la mayoría de los alumnos. Contar con la posibilidad de guardar el resultado de haber dispersado un archivo, da la posibilidad de posteriormente poder comparar con algún otro resultado, evaluando los ambientes respectivos y la configuración de parámetros elegida para cada uno de ellos.

Obviamente que el uso masivo de la herramienta generará requerimientos de cambios y/o mejoras que exceden las posibles previsiones realizadas al momento, y que como es de esperar, serán consideradas por los autores de este trabajo a fin de poder mejorar aún más a *E-HASH*.

Referencias

1. Michael Folk, Bill Zoellick, Greg Ricciardi. *Estructuras de Archivos*. Addision Wesley 1992. ISBN: 0-201-62923-2.

2. Rodolfo Bertone, Pablo Thomas. *Introducción a las Bases de Datos*. *Fundamentos y Diseño*. Pearson Latinoamérica 2011. ISBN: 978-987-615-136-8.

3. Ramez Elmasri, Shamkant B. Navathe. *Fundamentos de Sistemas de Bases de Datos*. Addision Wesley 2002. ISBN 10: 84-7829-051-6. ISBN 13: 978-84-7829-051-2.

4. Peter Smith, Michael Barnes. *Files & Databases: An Introduction*. Addison Wesley 1987. ISBN: 0-201-10746-5.

5. C. J. Date. *Introducción a los Sistemas de Bases de Datos*. Prentice Hall 2001. ISBN: 968444-419-2.

6. Gary W. Hansen, James V. Hansen. *Diseño y Administración de Bases de Datos*. Prentice Hall 1997. ISBN: 84-8322-002-4.

7. Rob Peter, Coronel Carlos. *Sistemas de Bases de Datos. Diseño, Implementación y Administración*. Thomson 2002. ISBN: 970-686-286-2.

8. James L. Johnson. *Bases de Datos. Modelos, Lenguajes, Diseño*. Oxford University Press 1997. ISBN: 970-613-461-1.

9. Abraham Silberschatz, Henry F. Korth, S. Sudarshan. *Fundamentos de Bases de Datos*. Mc Graw Hill 2002. ISBN: 84-481-3654-3.

10. Rodolfo Bertone, Emanuel Nucilli, Pablo Thomas. *HEA: Herramienta de Software para enseñanza de árboles*. XVI Congreso Argentino de Ciencias de la Computación. Universidad de Morón. Año 2010

11. Rodolfo Bertone, Pablo Thomas, S. Antonetti, A. Miglio. *CasER: Herramienta para la enseñanza de Modelado Conceptual de Bases de Datos*. XV Congreso

Argentino de Ciencias de la Computación. Universidad Nacional de Jujuy. Año 2009.

12. Alejandra Durán, María F. Rius, Rodolfo Bertone, Pablo Thomas. *CasER 2.0: Herramienta para la enseñanza de Modelado de Bases de Datos*. XVII Congreso Argentino de Ciencias de la Computación. Universidad Nacional de La Plata. Año 2011.

13. Ariel Sobrado, Luciano Marrero, Rodolfo Bertone, Pablo Thomas. *E-HASH: Herramienta de Software para Dispersión de Archivos*. XVII Congreso Argentino de Ciencias de la Computación. Universidad Nacional de La Plata. Año 2011.

14. Eric Jeltsch F. Organización Física de los Sistemas de Bases de Datos. [http://dns.uls.cl/~ej/fit_2004/Lecturas/LogProp%28c%29.pdf]. Accedido en Noviembre del 2011.

15. Bases de Datos. [http://es.wikipedia.org/wiki/Base_de_datos]. Accedido en Noviembre del 2011.

16. HTML. [http://es.wikipedia.org/wiki/HTML]. Accedido en Diciembre del 2011.

17. JavaScript. [http://es.wikipedia.org/wiki/JavaScript]. Accedido en Diciembre del 2011.

18. JQuery. [http://jquery.com/] [http://jqueryui.com/] [http://es.wikipedia.org/wiki/JQuery]. Accedido en Diciembre del 2011.

19. ASCII (acrónimo inglés de American Standard Code for Information Interchange — Código Estándar Estadounidense *para el Intercambio de Información*) [http://es.wikipedia.org/wiki/ASCII]. Accedido en Diciembre 2011.

20. ANSI-SPARC Architecture [http://en.wikipedia.org/wiki/ANSI-SPARC_Architecture]. Accedido en Diciembre del 2011.

21. Buffer [http://es.wikipedia.org/wiki/Buffer]. Accedido en Diciembre del 2011.

22. Sistema de Entrada/Salida. [http://es.wikipedia.org/wiki/Entrada/salida]. Accedido en Diciembre del 2011.

23. Pseudocódigo. [http://es.wikipedia.org/wiki/Pseudocódigo]. Accedido en Diciembre del 2011.

24. Nanosegundo [http://es.wikipedia.org/wiki/Nanosegundo]. Accedido en Diciembre del 2011.

25. Milisegundo [http://es.wikipedia.org/wiki/Milisegundo]. Accedido en Diciembre del 2011.

26. Script [http://es.wikipedia.org/wiki/Script]. Accedido en Diciembre del 2011.

27. Derecho de autor o *copyright* [http://es.wikipedia.org/wiki/Derecho_de_autor]. Accedido en Febrero del 2012.

28. Document Object Model [http://es.wikipedia.org/wiki/Document_Object_Model]. Accedido en Febrero del 2012.

29. XML [http://es.wikipedia.org/wiki/Extensible_Markup_Language]. Accedido en Febrero del 2012.

30. AJAX [http://es.wikipedia.org/wiki/AJAX]. Accedido en Febrero del 2012.

31. Hipertexto [http://es.wikipedia.org/wiki/Hipertexto]. Accedido en Febrero del 2012.