Lección 20: Ficheros indexados

- Motivación
- Índices simples
- Operaciones con índices simples
- Índices secundarios
- Operaciones con índices secundarios

Motivación

Una empresa de venta de libros por Internet necesita mantener un extenso catálogo de 100000 títulos que posibilite la búsqueda rápida por isbn y autor. Los campos de descripción y opiniones son textos de un tamaño arbitrario.

Mantener el catálogo completo permanentemente en memoria es inasumible.

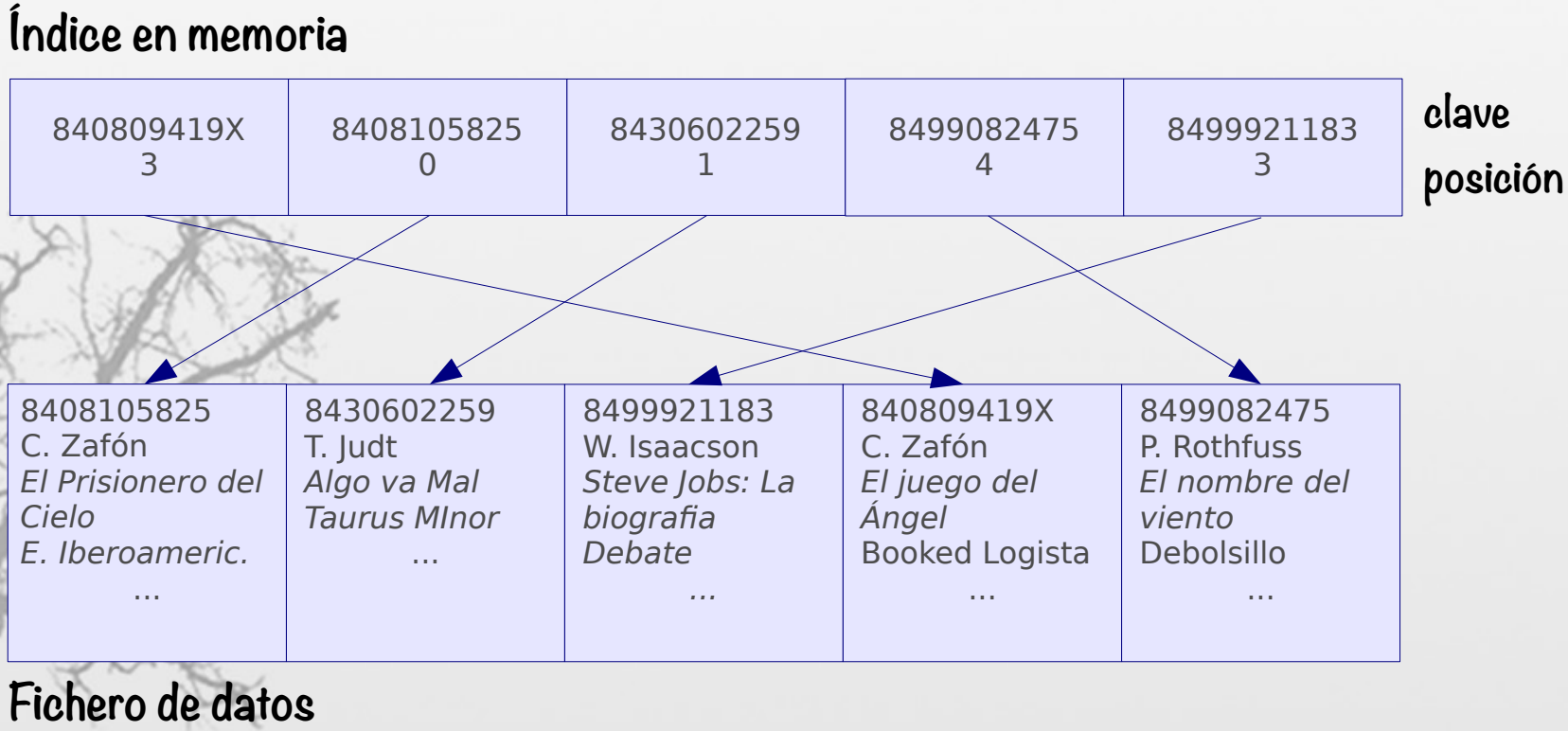
Libro
string isbn string autor string titulo string editorial int numPaginas string descripcion string opiniones

Indexación de ficheros

- La indexación de ficheros es una técnica que permite de manera eficiente:
  - Localizar cualquier registro del fichero de datos a partir de su clave
  - Determinar si un registro existe o no
  - Listar de manera ordenada los registros del fichero de datos sin que esté ordenado físicamente
- Para ello utiliza una estructura de datos en memoria: el índice del fichero
- El consumo de memoria del índice es pequeño comparado con el del fichero completo

Índices de ficheros

- Un índice es una lista en memoria ordenada por clave que mantiene la posición de cada registro en el fichero de datos



Índices de ficheros

- Para implementar el índice podemos usar varias de las estructuras de datos que hemos estudiado:
  - Vectores dinámicos ordenados
  - Listas enlazadas ordenadas
  - Mapas implementados mediante ABBs
- Si la clave es la llave o clave primaria (una clave que identifica unívocamente cada registro), entonces el índice es un **índice simple o primario**

Operaciones

- Para buscar:
  - Se busca eficientemente por clave en el índice
  - Con la posición obtenida se lee el registro del fichero de datos y se pasa a memoria
- Para insertar:
  - Se inserta el registro en el fichero de datos
  - Se inserta su clave y posición en el índice
- Para borrar:
  - Se localiza el registro mediante el índice en memoria
  - Con la posición obtenida se borra el registro (borrado lógico) y se elimina igualmente la entrada del índice

Construcción del índice

- Para construir el índice:
  - Se recorre el fichero de datos registro a registro
  - Se obtiene la posición del registro en el fichero (operación tellg()) y se extrae la clave
  - Se inserta la tupla (clave, posición) en el índice en memoria
- Es un proceso lento, por tanto el índice una vez construido se guarda en un fichero propio: **el fichero de índices**
- Al arrancar se busca este fichero y se carga el índice directamente de él. Si no existe o está desactualizado, se reconstruye.

Desactualización

- Si el índice es modificado en memoria y no es guardado al terminar, el fichero de índices queda desactualizado y **no debe utilizarse para cargar el índice la próxima vez**
- Para detectar este problema se utiliza una marca de *actualizado* en el fichero de índice:
  - Nunca se utilizará un fichero índice con la marca puesta a *falso*
  - Al cargar el índice se cambia a *falso*
  - Al guardar al terminar se cambia a *verdadero*

Implementación índice

```
#include <iostream>
#include <fstream>
#include <map>
using namespace std;

class ErrorFicheroIndice {};

class IndiceLibros {
    map<string, long> ind;
public:
    Indice() : ind() {}
    void cargar(string &fich);
    void salvar(string &fich);

    void insertar(string &isbn, long pos) {
        ind.insert(pair<string, long>(isbn, pos));
    }

    void eliminar(string &isbn) {
        ind.erase(isbn);
    }

    long buscar(string &isbn) {
        map<string, long>::iterator i = ind.find(isbn);
        return (i != ind.end() ? i->second : -1);
    }
};
```

Implementación índice

```
struct EntradaIndice {
    char isbn[11];
    long pos;
};

void IndiceLibros::salvar(string &fich) {
    ofstream f(fich.c_str());

    if (!f)
        throw  ErrorFicheroIndice();

    char actualizado = 'V';
    f.write(&actualizado, sizeof(actualizado));

    EntradaIndice entrada;
    map<Clave, long>::iterator i = ind.begin();
    while (i != ind.end()) {
        strcpy(entrada.isbn, (i->first).c_str());
        entrada.pos = i->second;
        f.write((const char *)&entrada, sizeof(pair<Clave, long>));
        ++i;
    }
}
```

Implementación índice

```
void IndiceLibros::cargar(string &fich) {
    fstream f(fich.c_str(), ios::in | ios::out);

    if (!f)
        throw  ErrorFicheroIndice();

    char actualizado;
    f.read(&actualizado, sizeof(actualizado));

    if (actualizado == 'F')
        throw ErrorFicheroIndice();

    EntradaIndice entrada;
    while (f.read((char *) &entrada, sizeof(entrada))) {
        ind.insert(pair<string, long>(entrada.isbn, entrada.pos));
    }

    // Actualizar marca
    actualizado = 'F';
    f.seekp(0, ios::beg);
    f.write(&actualizado, sizeof(actualizado));
}
```

Implementación fichero indexado

```
#include “indice.h”
#include “fichero.h”
using namespace std;

class FicheroIndexadoLibros {
    FicheroLibros fichLibros; //lección 19
    IndiceLibros indPrim;

public:
    FicheroIndexadoLibros(string &f);
    ~FicheroIndexadoLibros();
    bool buscar(string &isbn, Libro &libro);
    void insertar(Libro &libro);
    bool borrar(string &isbn);
    bool modificar(string &isbn, Libro &nuevoLibro);
};
```

Implementación fichero indexado

```
bool FicheroIndexadoLibros::buscar(string &isbn, Libro &libro) {
    long pos = indPrim.buscar(isbn);
    if (pos == -1) return false;

    fichLibros.leer(pos, libro);
    return true;
}

void FicheroIndexadoLibros::insertar(Libro &libro) {
    long pos = fichLibros.insertar(libro);
    indPrim.insertar(libro.verIsbn(), pos);
}

bool FicheroIndexadoLibros::borrar(string &isbn) {
    long pos = indPrim.buscar(isbn);
    if (pos == -1) return false;

    inPrim.borrar(isbn);
    fichLibros.borrar(pos);
    return true;
}

bool FicheroIndexadoLibros::modificar(string &isbn, Libro &nuevoLibro)
{
    long pos = indPrim.buscar(isbn);
    if (pos == -1) return false;

    fichLibros.modificar(pos, nuevoLibro);
    return true;
}
```

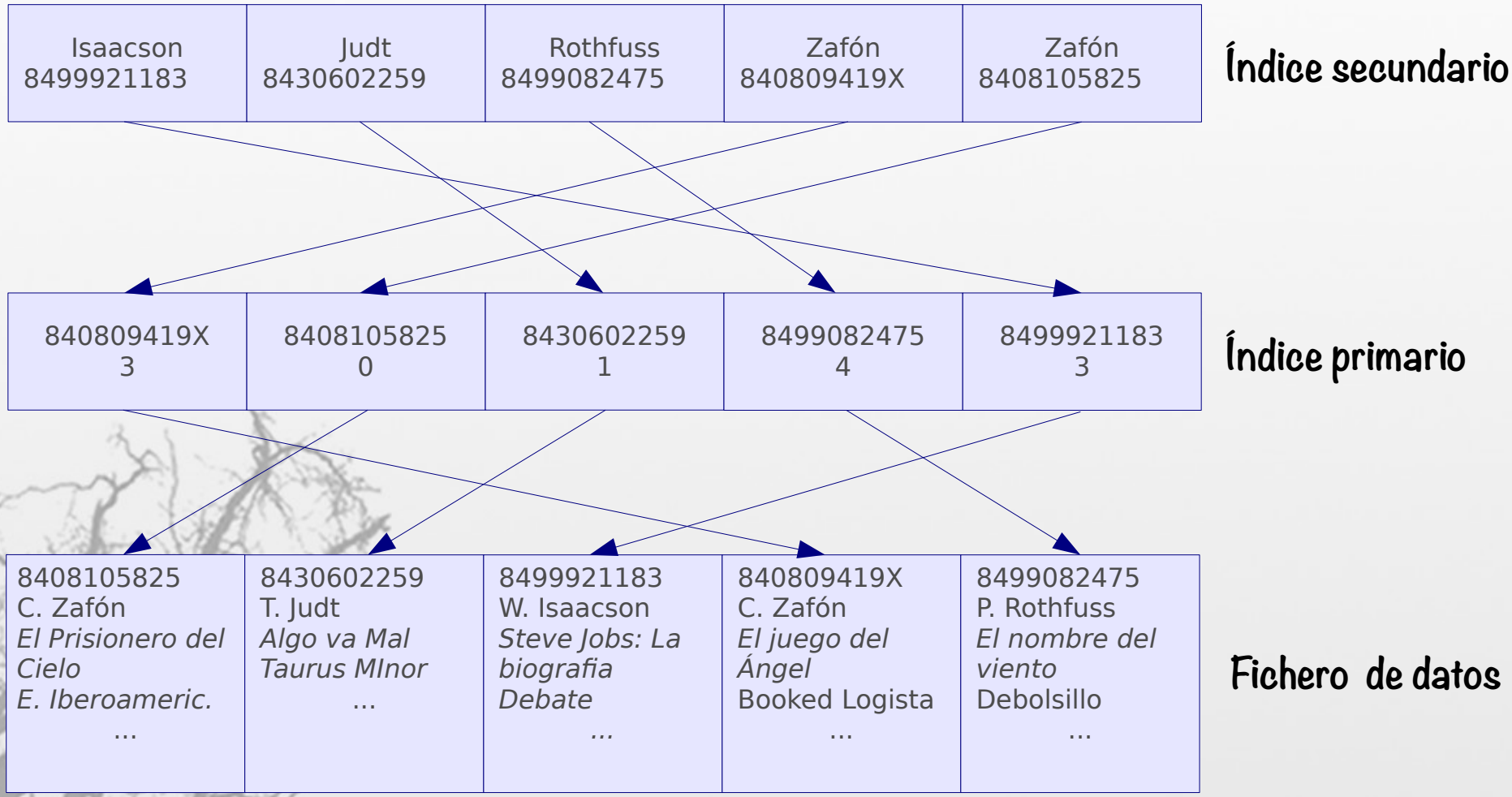
Implementación fichero indexado

```
FicheroIndexadoLibros::FicheroIndexadoLibros(string &fich) :
    ficheroLibros(fich),
    indPrim()
{
    try {

        indPrim.cargar(fich + ".idx");

    } catch (ErrorFicheroIndice e) {
        IteradorLibros i = ficheroLibros.inicio();
        while (i.haySiguiente()) {
            indPrim.insertar(
                pair<string, pos>(i.libro().verIsbn(), i.pos())
            );
            i.siguiente();
        }
    }
}
```

Índices secundarios

- Un índice primario sólo permite búsquedas por la clave primaria
- Podemos montar índices adicionales (**índices secundarios**) para buscar por otros campos
- Su estructura es similar a los índices primarios con dos diferencias:
  - La clave puede repetirse (al no ser una llave primaria)
  - En lugar de la posición se guarda la llave primaria del registro

Índices secundarios



Estructuras de datos para índices secundarios

- Similares a las de un índice primario (vectores o listas ordenadas, mapas)
- Al existir claves repetidas hay dos opciones:
  - Usar entradas permitiendo claves repetidas

```
multimap<ClaveSec, ClavePrim> indSec;
```

  - Usar una entrada por clave y una lista de ocurrencias (listas invertidas)

```
map<ClaveSec, list<ClavePrim> > indSec;
```

- Debe guardarse en un fichero al igual que un índice primario

Operaciones con índices sec.

- Para buscar por clave secundaria:
  - Se busca en el índice secundario y se obtienen las claves primarias de los registros buscados
  - Usando el índice primario obtenemos las posiciones de los registros en el fichero de datos
- Inserciones y borrados de registros
  - Se insertan o borran las entradas correspondientes en el índice secundario
- Modificaciones de registros
  - Si afectan a la clave secundaria, se borran y vuelven a insertar las entradas correspondientes

Implementación índice sec.

```
#include <iostream>
#include <fstream>
#include <map>
using namespace std;

class ErrorFicheroIndice {};

class IndiceSecAutor {
    map<string, list<string> > ind;
public:
    Indice() : ind() {}
    void cargar(string &fich);
    void salvar(string &fich);

    void insertar(string &autor, string &isbn) {
        ind[autor].push_back(isbn);
    }

    list<string> buscar(string &autor) {
        map<string, list<string> >::iterator i = ind.find(autor);
        return (i != ind.end() ? i->second : list<string>());
    }

    void eliminar(string &autor, string &isbn) {
        // Localizar entrada del autor y borrar isbn de la lista
    }
};
```

Implementación fichero indexado con índice secundario

```
#include “indice.h”
#include “fichero.h”
using namespace std;

class FicheroIndexadoLibros {
    FicheroLibros fichLibros;
    IndiceLibros indPrim;
    IndiceSecAutor indAutor;

public:
    FicheroIndexadoLibros(string &f);
    ~FicheroIndexadoLibros();
    void insertar(Libro &libro);
    void eliminar(string &isbn);
    bool buscar(string &isbn, Libro &libro);
    list<Libro> buscarPorAutor(string &autor);
    bool modificar(string &isbn, Libro &nuevoLibro);
};
```

Implementación fichero indexado con índice secundario

```
bool FicheroIndexadoLibros::insertar(Libro &libro) {
    long pos = fichLibros.insertar(libro);
    indPrim.insertar(libro.verIsbn(),  pos);
    indAutor.insertar(libro.verAutor(),  libro.verIsbn());
}

list<Libro> FicheroIndexadoLibros::buscarPorAutor(string &autor) {
    list<string> listaIsbn = indAutor.buscar(autor);

    Libro libro;
    list<Libro> libros;
    list<string>::iterator i = listaIsbn.begin();
    while (i != listaIsbn.end()) {
        if (buscar(*i, libro))
            libros.push_back(libro);
        ++i;
    }
    return libros;
}

// Resto de operaciones
```

Conclusiones

- La indexación de ficheros es una técnica sencilla para manejar un fichero de gran tamaño con información que debe ser recuperada eficientemente
- Permite localizar registros por su clave primaria o claves secundarias
- Problema: consume memoria y puede llegar a ser un problema con ficheros extremadamente grandes