Arquitectura de sistemas

Abelardo Pardo

University of Sydney
School of Electrical and Information Engineering
NSW, 2006, Australia
Autor principal del curso de 2009 a 2012

Iria Estévez Ayres

Damaris Fuentes Lorenzo

Pablo Basanta Val

Pedro J. Muñoz Merino

Hugo A. Parada

Derick Leony

Universidad Carlos III de Madrid
Departamento de Ingeniería Telemática
Avenida Universidad 30, E28911 Leganés (Madrid), España

Tabla de contenidos

7.1. Actividades

7.1.1. Implementación de una tabla hash

Recursos

- Tablas Hash
- Implementación de las funciones necesarias para manipular una lista encadenada de elementos (crear, insertar, borrar, buscar, etc.)

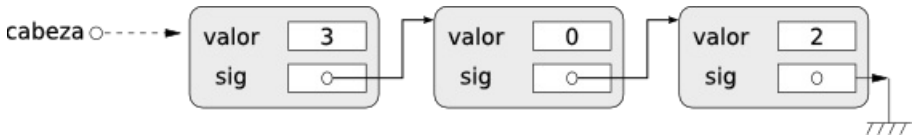
Plan de trabajo

1. Leer el documento " Tablas Hash ".
2. Propón una estructura de datos que contenga una tabla hash. Piensa en que esa estructura es la que se manejará en toda la aplicación, esto es, se creará con una función que escribes tú, y se recibirá como parámetro en las funciones de buscar, añadir, borrar, etc.
3. Escribe la función para crear una tabla hash. Decide qué parámetros necesita recibir.
4. Escribe la función que dada una tabla hash y una clave, busca si tal clave existe. Devuelve el valor asociado a esa clave o `NULL` en caso contrario.
5. Escribe la función que dada una tabla hash, una clave y un valor, añade el par (clave, valor) a la tabla. Se asume que la clave no está presente en la tabla antes de realizarse la operación.
6. Escribe una función que recibe una tabla hash y la destruye. Esto incluye destruir todas las lista de colisiones internas.
7. Modifica la estructura de la tabla para que tenga un parámetro denominado "densidad". Cuando la densidad de la tabla supere ese valor, se debe redimensionar incrementando su tamaño un 25%. Esto debe suceder de forma transparente cuando se ejecuta una operación de inserción de un nuevo par (clave, valor).

7.1.2. Lista encadenada de enteros

Recursos

Un programa en C necesita almacenar un número indeterminado de enteros. El tamaño de los datos no se sabe hasta que el programa está ejecutando. Para hacer un uso más óptimo de memoria se opta por representarlos mediante una lista encadenada. Cada elemento de esa lista es una estructura que contiene el número a almacenar y un puntero al siguiente elemento. Los elementos de la lista están, por tanto, encadenados. La siguiente figura muestra un ejemplo de esta estructura:



La lista se manipula en el programa únicamente mediante un puntero a su primer elemento. Escribe las siguientes porciones de código (necesitarás ayuda con alguna de ellas, con lo que esperamos que hagas uso del foro de la asignatura):

1. Define la estructura de datos necesaria. ¿Cómo representas una lista vacía?
2. Función que no recibe parámetros y devuelve una lista vacía (no debería tener más de una línea de código).
3. Función que, dado un entero, devuelve una lista con un único elemento con ese entero.
4. Función que, dada una lista y un entero, añade el entero a la lista (el lugar donde se añade el entero es irrelevante).
5. Función que devuelve el número los elementos de una lista.
6. Función que, dada una lista y un entero, borra ***todas las apariciones*** de ese entero (si hay alguna) en la lista.
7. Función que, dadas dos listas, devuelve una lista que es la concatenación de ambas.
8. Función que borra totalmente el contenido de una lista.

Sugerencia

Escribe estas estructuras y funciones en un fichero con su propia función `main` e incluye en ella llamadas para verificar que tu estructura de datos funciona correctamente.

Plan de trabajo

1. Decide primero qué representación vas a tener de la lista y cómo vas a almacenar los elementos. Piensa si la estructura que propones es la adecuada para ser manipulada por las funciones que se piden.
2. Para cada función, primero piensa en su prototipo, esto es, el tipo de resultado y los tipos y número de parámetros necesarios. Una vez que tengas esto claro, diseña el cuerpo de la función.
3. Una vez hayas terminado la implementación de las funciones, mueve la función `main` a un fichero aparte. Crea un fichero con extensión "`*.h`" en el que incluyas las definiciones y prototipos necesarios para que, al incluirlo en el fichero del "`main`", se compile sin advertencias.

Evaluación

Comprueba que tu solución es correcta antes de la sesión presencial comparando tu solución con la de algún compañero y posteando en el foro del curso.

Autoevaluación automática

Autoevaluación automática

En esta sección, trabajaremos con la lista de alumnos de un curso (de tamaño variable) donde se guardará, para cada alumno, un nombre (de tamaño variable), sus notas (otra lista de tamaño variable) y su identificador de alumno (un entero).

1. Elija la definición de datos adecuada para guardar las notas de cada alumno.

   - ```
     struct score
     {
         int score;
         struct score next;
     };
     ```

   - ```
     struct score
     {
         int score;
     };
     ```

   - ```
     struct score
     {
         int score;
         struct score *next;
     };
     ```

2. Elija la definición de datos adecuada para la lista de alumnos.

   - ```
     struct lista
     {
         int id;
         char *name;
         struct score scores_student;
         struct lista next;
     };
     ```

   - ```
     struct lista
     {
         int id;
         char *name;
         struct score *scores_student;
         struct lista *next;
     };
     ```

   - ```
     ```

```
#define SIZE 30
struct lista
{
    int id;
    char name[SIZE];
    struct score *scores_student;
    struct lista *next;
};
```

-

```
#define SIZE 30
struct lista
{
    int id;
    char name[SIZE];
    struct score scores_student;
    struct lista next;
};
```

3. Se desea inicializar en el programa principal la lista como vacía. Elija la correcta declaración.

-

```
struct lista *clase= (struct lista *) malloc (sizeof(struct lista));
*clase=NULL;
```

-

```
struct lista *clase= (struct lista *) malloc (sizeof(struct lista));
clase=NULL;
```

-

```
struct lista *clase= (struct lista *) malloc (sizeof(struct lista));
```

-

```
struct lista clase=NULL;
```

-

```
struct lista *clase=NULL;
```

4. Se desea implementar una función que, dado un identificador y un nombre de alumno, devuelva una lista con un único elemento con la lista de notas vacía. La función debe copiar el nombre del alumno. Elija el código correcto para dicha función.

-

```
struct lista *new_student(int id, char *name)
{
    struct lista *aux= (struct lista *) malloc (sizeof(struct lista));
```

```
    if (aux == NULL)
    {
        return NULL;
    }
    aux->id=id;
    aux->name=name;
    aux->scores_student=NULL;
    aux->next=NULL;
    return aux;
}
```

-

```
struct lista *new_student(int id, char *name)
{
    struct lista *aux= (struct lista *) malloc (sizeof(struct lista));
    if (aux == NULL)
    {
        return NULL;
    }
    aux->id=id;
    aux->name=(char *) malloc (sizeof(char)*(strlen(name)+1));
    if (aux->name == NULL)
    {
        free(aux);
        return NULL;
    }
    strcpy(aux->name,name);
    aux->scores_student=NULL;
    return aux;
}
```

-

```
struct lista *new_student(int id, char *name)
{
    struct lista *aux= (struct lista *) malloc (sizeof(struct lista *));
    if (aux == NULL)
    {
        return NULL;
    }
    aux->id=id;
    aux->name=(char *) malloc (sizeof(char *)*(strlen(name)));
    if (aux->name == NULL)
    {
        free(aux);
       return NULL;
    }
    strcpy(aux->name,name);
    aux->scores_student=NULL;
    aux->next=NULL;
    return aux;
}
```

-

```
struct lista *new_student(int id, char *name)
{
    struct lista *aux= (struct lista *) malloc (sizeof(struct lista));
    if (aux == NULL)
    {
        return NULL;
    }
    aux->id=id;
    aux->name=(char *) malloc (sizeof(char)*(strlen(name)+1));
    if (aux->name == NULL)
    {
        free(aux);
        return NULL;
    }
    strcpy(aux->name,name);
    aux->scores_student=NULL;
    aux->next=NULL;
    return aux;
}
```

-

```
struct lista *new_student(int id, char *name)
{
    struct lista *aux= (struct lista *) malloc (sizeof(struct lista));
    if (aux == NULL)
    {
        return NULL;
    }
    aux->id=id;
    aux->name=(char *) malloc (sizeof(char)*(strlen(name)+1));
    if (aux->name == NULL)
    {
        free(aux);
        return NULL;
    }
    strcpy(aux->name,name);
    return aux;
}
```

5. Se desea llamar a la función anterior desde el programa principal. Elija el código correcto.

-

```
#define SIZE 30
struct lista *nuevo;
char nombre[SIZE]="pepe";
int id=5;
nuevo=new_student(id,nombre);
```

-

```
#define SIZE 30
struct lista nuevo;
char nombre[SIZE]="pepe";
int id=5;
nuevo=new_student(id,nombre);
```

-

```
#define SIZE 30
struct lista *nuevo;
char *nombre="pepe";
int id=5;
nuevo=new_student(id,nombre);
```

-

```
#define SIZE 30
struct lista *nuevo;
char nombre[SIZE]="pepe";
int id=5;
nuevo=new_student(id,&nombre);
```

-

```
#define SIZE 30
struct lista *nuevo;
char *nombre="pepe";
int id=5;
nuevo=new_student(id,&nombre);
```

6. Se desea implementar una función que, dada una lista, un identificador y un nombre de alumno, modifique la lista añadiendo el nuevo alumno al final de la lista y devuelva la nueva longitud de la misma. Elija el prototipo correcto de dicha función.

-

```
int add_new_student(struct lista *list,int id, char *name);
```

-

```
int add_new_student(struct lista **list,int id, char *name);
```

-

```
struct lista *add_new_student(int id, char *name);
```

-

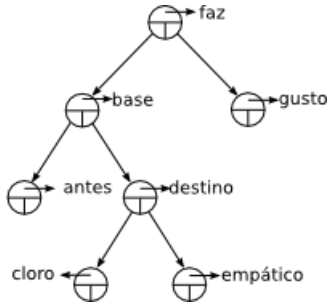
```
void add_new_student(struct lista *list,int id, char *name);
```

- El lenguaje C no permite implementar una función de esas características.

7.1.3. Árbol de cadenas de texto

Plan de trabajo

En un programa en C necesita almacenar un número indeterminado de palabras. La estructura de datos seleccionada es un árbol binario en el que cada nodo almacena una palabra. Además, el árbol tiene la propiedad de que la palabra almacenada en cada nodo sigue lexicograficamente a las del sub-arbol izquierdo, y precede a las del sub-arbol derecho. La siguiente figura muestra un ejemplo de esta estructura de datos.



La aplicación necesita manipular varios conjuntos de palabras independientes.

1. Define las estructuras de datos necesarias para manipular estos árboles.
2. Define los ***prototipos*** de las siguientes funciones:
    - Función que crea un nuevo conjunto vacío.
    - Función que busca una palabra en un conjunto y la devuelve (la palabra devuelta se puede modificar).
    - Función que añade una palabra en un árbol. Si ya existe, no hace nada.
    - Función que destruye un árbol entero.
    - Función que borra una palabra de un árbol.
3. Escribe el método `main` que contenga un bucle y mediante la función `getline` lea de teclado una cadena y la inserta en dos árboles. El programa se detiene cuando `getline` devuelve el valor `NULL`.
4. Dividir el trabajo en equipos y que cada uno haga uno de los métodos de búsqueda, inserción o borrado de todos los elementos. La función `strcmp` recibe como parámetros dos enteros y devuelve un número menor, igual o mayor que cero si la primera palabra precede, es idéntica o antecede a la segunda respectivamente.
5. Implementar la función de borrado de una cadena.
6. Responder a las siguientes preguntas:
    1. Si comparamos esta estructura de datos con una lista encadenada de palabras, ¿qué estructura ofrece una inserción de nuevas palabras más eficiente?
    2. ¿Cómo comparan estas dos estructuras con respecto a la búsqueda?
    3. Dos árboles contienen exactamente las mismas palabras. ¿Quiere decir esto que tienen idéntica estructura?
    4. Si en la aplicación C que utiliza estas funciones, el 99% de las operaciones que se ejecutan son de búsqueda de palabras, ¿qué mejora propondrías a la estructura de datos para ganar eficiencia?

7.1.4. Uso de listas encadenadas

Recursos

- Programa creado en una actividad anterior en la que se implementa un menú con varias opciones.
- Carpeta con nombre `List_aps` en tu espacio compartido en Subversion.

Plan de trabajo

Realiza las siguientes modificaciones en el programa que maneja opciones en un menú:

1. Crea un nuevo subdirectorio en tu espacio de trabajo compartido por Subversion para el programa que has de crear a continuación.
2. Define un tipo de datos `struct node` que, conteniendo la misma información que la estructura `struct ap_scan_info`, sirva como base para una lista enlazada.
3. Implementa la función `struct node *create_node(struct ap_scan_info *cell)` que recibe la dirección de una celda de la tabla y crea el nodo que guarda una copia de dicha información.
4. Implementa la función `void print_node(struct node *node_ptr)` que recibe la dirección de un nodo y muestra su información por pantalla.

Cuando acabes, sube el fichero al repositorio con `svn commit`.

7.1.5. Creación de una lista encadenada

Recursos

- Programa que se implementa un menú con varias opciones y las funciones `struct node *create_node(struct ap_scan_info *cell)` y `void print_node(struct node *node_ptr)` ya implementadas.
- Carpeta con nombre `List_aps` en tu espacio compartido en Subversion.

Plan de trabajo

Partiendo del programa del menú con las funciones mencionadas, realiza las siguientes modificaciones:

1. Implementa la función `struct node *create_list(struct ap_scan_info *array, int size)` que recibe una tabla de estructuras `struct ap_scan_info` y devuelve dicha tabla en una lista encadenada. Para ello usa la definición de tipo de datos y la función `struct node *create_node(struct ap_scan_info *cell)` de las actividades previas.
2. Añade e implementa una nueva opción en el menú que permita al usuario crear una lista encadenada a partir del array. Almacena esta lista en una variable para que pueda ser utilizada en otras opciones.

3. Añade e implementa una nueva opción en el menú que muestre al usuario todos los puntos de acceso de la lista enlazada. Usa la función `void print_node(struct node *node_ptr)` de las actividades previas.
4. Añade código para que la lista, si existe, se destruya al terminar el programa.

Cuando acabes, sube el fichero al repositorio con `svn commit`.

7.1.6. Borrar nodos de una lista encadenada

Recursos

- Programa que se implementa un menú con varias opciones y las funciones `struct node *create_list(struct ap_scan_info *array, int size)` y el código para destruir la lista al terminar el programa.
- Carpeta con nombre `List_aps` en tu espacio compartido en Subversion.

Plan de trabajo

Partiendo del programa del menú de opciones realiza las siguientes modificaciones:

1. Implementa la función `struct node *delete_essid_in_list(struct node *, char *essid)` que recibe una lista enlazada y una cadena con el essid de una red y borra todos los puntos de acceso que tengan dicho essid. La función devuelve la lista modificada.
2. Añade e implementa una nueva opción en el menú que permita al usuario borrar todos los nodos con el mismo essid de la lista. Usa la función `getline` para ello

Cuando acabes, sube el fichero al repositorio con `svn commit`.