Estructuras dinámicas en C

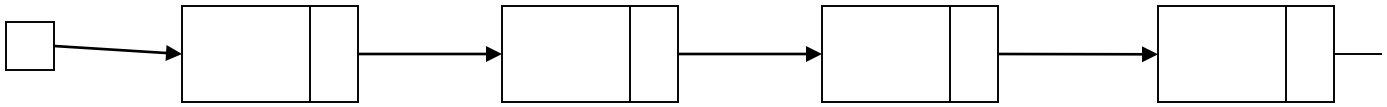
Antonio Espín Herranz

Estructuras dinámicas de almacenamiento de datos

- Listas Enlazadas.
- Pilas.
- Colas.
- Árboles Binarios

Listas enlazadas

- Cadena elemento o nodo tiene dos partes.
- En la primera almacenamos datos, ya sean tipos básicos o una estructura mas compleja.
- Y en la 2ª tendremos un puntero al siguiente elemento.



Características sobre la lista

- Es una colección secuencial de datos del mismo tipo.
- Se empiezan a recorrer siempre por el nodo raíz.
- Avanzamos de uno en uno y no podemos retroceder.
- Cualquier lista se puede dividir en sublistas → uso de funciones recursivas.

Definición de la Estructura

- Podemos utilizar anidamiento de estructuras, para separar la parte de datos de los enlaces.

```
typedef struct l {
    char nombre[20];
    int edad;
    struct l *sig;  // Estructura auto referenciada.
} Tnodo;
```

Definición de la raíz:

```
Tnodo *raiz = NULL;
```

Operaciones sobre la lista

- Añadir un nuevo elemento:
  - Al final de la lista.
  - De forma ordenada.
  - Al inicio de la lista.
- Eliminar un elemento de la lista:
  - Por posición.
  - Buscando el elemento en concreto.
- Recorrer todos los elementos de la lista.
  - Para imprimir por pantalla.
  - Grabar en un fichero.
- Resolver preguntas como:
  - Número de elementos de la lista.
  - Está vacía.
  - Existe un elemento.
- Liberar TODOS los elementos de la lista

Añadir un elemento

- Al final:
  - Nos posicionamos al final de la lista, es decir, cuando alcancemos NULL, ya sea de forma recursiva o iterativa. La forma de avanzar por una lista es:

    TLista *actual = lista;

    actual = actual→siguiente;
  - Crear el nuevo nodo. Con malloc.
  - Damos los valores (datos) y siguiente = NULL.
  - Y hacemos que la lista apunte al nuevo nodo.

Añadir un elemento

- De forma ordenada:
  - Si está vacía elegir la opción anterior.
  - En caso contrario tenemos que llevar un puntero con el nodo actual, y un puntero al anterior elemento.
  - Crear el nodo, con sus datos.
  - nodoNuevo→siguiente = actual;
  - anterior→siguiente = nodoNuevo;

Añadir un elemento

- Al inicio:
  - Crear el nuevo nodo, con sus datos.
  - nuevoNodo→siguiente = inicio_lista;
  - inicio_lista = nuevoNodo;

Liberar todos los elementos de la lista

- Recursivamente, vamos avanzando hasta el último nodo.
- Hacemos una llamada recursiva con el siguiente.
- Y a la vuelta de la recursividad vamos liberando.

Eliminar un nodo de la lista

- Búsqueda del nodo que contiene el dato.
- Se debe almacenar la dirección actual y la dirección anterior.
- El puntero siguiente del nodo anterior debe apuntar al siguiente al nodo a eliminar.
- En caso de que el nodo a eliminar sea el primero de la lista, se modifica el inicio de la lista para que tenga la dirección del nodo siguiente.
- Se libera la memoria del nodo.

Pilas

- Sería lo mas parecido a una pila de papeles.
- Crecen hacia arriba, el primer elemento que sale es el que está en la cima de la pila.
- Para recuperar un elemento tenemos que extraer lo que están por encima de él.
- **LIFO**: (last-in, first-out): último entrar –primero en salir.

cima
3
4
5
0

Definición de la estructura

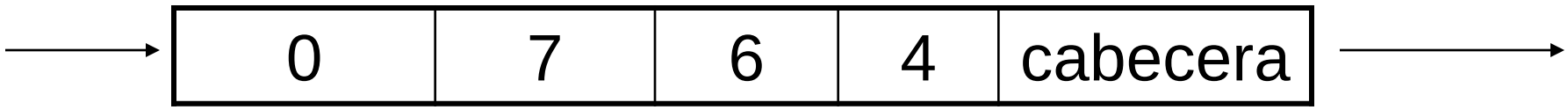
- Podemos apoyarnos en una lista enlazada o utilizar un array dinámico.
- Si la implementamos con una lista enlazada podemos añadir siempre al principio de la lista.
- Y siempre obtendremos el primer el elemento de la lista.

Operaciones sobre la pila

- Está vacía.
- Número de elementos.
- Apilar: Introducir un nuevo elemento en la pila, siempre crece por la cima.
- Desapilar: Consiste en sacar el elemento que se encuentra en la cima de la pila. Se elimina el elemento de la pila.
- Recorrer la pila: Ya sea para imprimir por pantalla los elementos o grabar en un fichero.

Colas

- Es como una fila de personas.
- Los elementos se van añadiendo por el final.
- Para poder sacar un elemento antes hay que procesar los que están por delante de él.
- **FIFO**: First-in, First-out, primero en entrar, primero en salir.



Operaciones sobre la cola

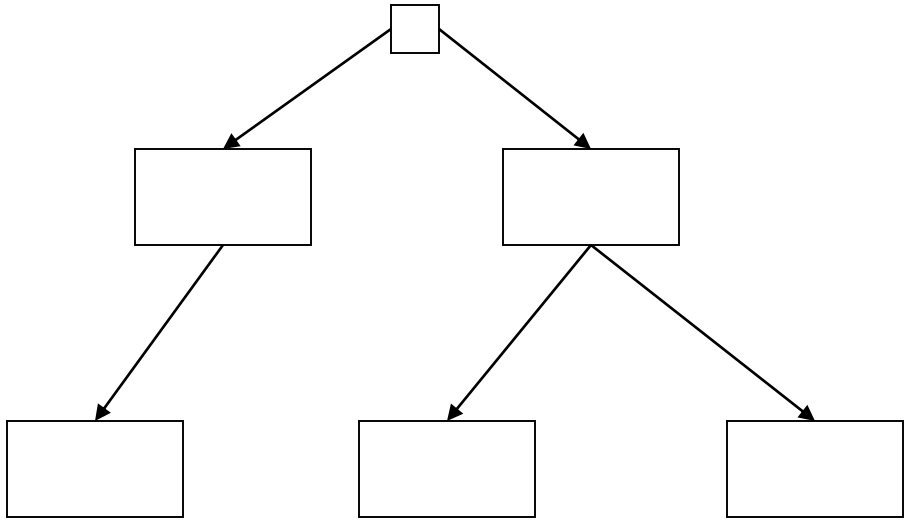
- Está vacía.
- Número de elementos.
- Encolar: Introducir un nuevo elemento en la cola, siempre crece por el final.
- Desencolar: Consiste en sacar el elemento que se encuentra en la cabecera de la cola. Se elimina el elemento de la cola.
- Recorrer la cola: Ya sea para imprimir por pantalla los elementos o grabar en un fichero.

Definición de la estructura

- Al igual que en las pilas, nos podemos apoyar en un array o en una lista enlazada.
- Podemos trabajar con dos punteros, para trabajar mas rápido, uno apunta a la cabecera de la cola, es decir, por donde salen los elementos.
- Y otro apunta al final de la cola, por donde entran los elementos.

Árboles Binarios

- Es un árbol que ningún nodo puede tener mas de dos subárboles.
- En un árbol binario cada nodo puede tener, cero, uno o dos hijos (subárboles).
- Nodo Izquierda → Hijo Izq. Nodo Derecha → Hijo Der.



Características de los árboles

- Es una estructura recursiva.
- Cada nodo es el raíz de su propio subárbol y tiene hijos, que son raíces de árboles llamados los subárboles derecho e izquierdo del nodo.

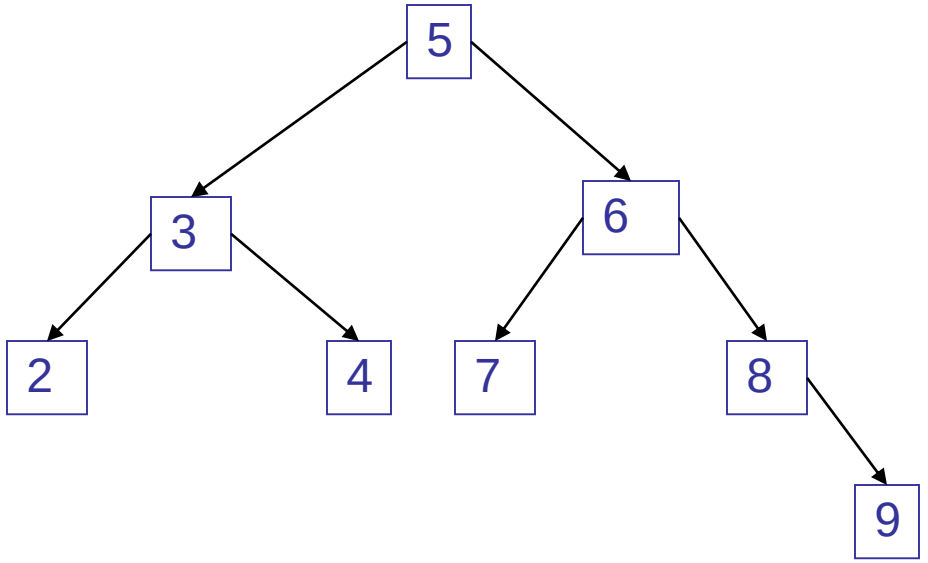
Características del árbol

- Cada nodo está partido en 3 partes:
  - Tendremos los datos a almacenar. Puede ser cualquier tipo o estructura mas compleja.
  - Un puntero a la izquierda.
  - Un puntero a la derecha.

- Árboles binarios de búsqueda: se crean de forma ordenada.
- A partir de un nodo raíz, en el subárbol izq. Estarán todos los nodos menores que la raíz.
- Y en el caso de los mayores se colocarán en el subárbol derecho.

Construcción de árboles binarios y los 3 tipos de recorridos

- Inserción de los números:
  - 5, 3, 2, 4, 6, 8, 7, 9
- Recorridos:
  - PreOrden: R – I – D
    - 5, 3, 2, 4, 6, 8, 7, 9
  - EnOrden: I – R – D
    - 2, 3, 4, 5, 6, 7, 8, 9
  - PostOrden: I – D – R
    - 2, 4, 3, 7, 9, 8, 6, 5



Definición de la estructura

- Podemos utilizar anidamiento de estructuras, para separar la parte de datos de los enlaces.

```
typedef struct l {
    char nombre[20];
    int edad;
    struct l *izq;
    struct l *der;
} Tnodo;
```

Definición de la raíz:

```
Tnodo *arbol = NULL;
```

Operaciones sobre el árbol

- Crear nuevos nodos.
  - Añadir la información e inicializar a NULL los dos subárboles.
- Determinar la altura del árbol. Será 1 mas que la mayor de las alturas de sus subárboles izquierdo y derecho.
- Liberar toda la memoria del árbol → utilizaremos un recorrido postorden.
- Búsqueda de un elemento.
- Visualizar el árbol. A partir de un recorrido vamos imprimiendo el valor de cada nodo.
- Recorridos por el árbol.
  - PreOrden: Raíz (ver datos), Preorden(Izq) y Preorden(Der).
  - EnOrden: EnOrden(Izq), Raíz(Ver datos), EnOrden(Der).
  - PostOrden: PostOrden(Izq), PostOrden(Der), Raíz(ver datos)