Universidad Andrés Bello
Conectar · Innovar · Liderar

Listas Enlazadas

Contenidos

Introducción

- Los arreglos son típicamente las estructuras lineales conocidas hasta el momento.
- Los arreglos presentan dos grandes ventajas para el almacenamiento de colecciones lineales de datos:
  - Acceso aleatorio: se puede acceder a cualquier posición del arreglo en tiempo constante.
  - Uso eficiente de memoria cuando todas las posiciones están ocupadas: por guardarse en posiciones consecutivas de memoria.

Introducción

En cuanto a las desventajas:

- Problemas con el tamaño. Debido a que se debe asignar un tamaño al crear el arreglo, y no se puede cambiar en caso de usar arreglos estáticos.
- En caso de usar arreglos dinámicos se debe redimensionar. Da lugar a problemas:
  - Uso no eficiente de memoria por tener que reservar espacio para el caso peor.
  - Posibilidad de sobrepasar el tamaño reservado en tiempo de ejecución.
- Necesidad de memoria contigua:
  - Puede ocurrir que, pese a haber suficiente memoria libre, no haya un bloque contiguo suficientemente grande.

Introducción

Más desventajas:

- Ciertas operaciones tienen un coste no óptimo:
  - Inserciones y eliminaciones de datos en la primera posición o posiciones intermedias: necesidad de desplazar datos entre posiciones consecutivas.
  - Concatenación de dos o más arreglos: necesidad de copiar los datos a un nuevo arreglos.
  - Partición de un arreglo en varios fragmentos: necesidad de copiar datos a nuevos arreglos.

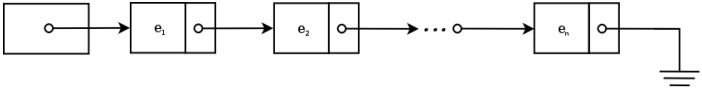
Introducción

- Las listas enlazadas son estructuras de datos dinámicas.
- Al contrario de las estructuras estáticas, en las que su tamaño en memoria se establece durante una compilación y permanece inalterable durante la ejecución del programa.
- Las estructuras de datos dinámicas crecen y se contraen a medida que se ejecuta el programa.
- Son una colección de elementos, denominados nodos, dispuestos uno a continuación de otro.
- Cada uno de ellos conectado al siguiente elemento por un *enlace* o *puntero*.
- Las listas enlazadas son estructuras muy flexibles y con numerosas aplicaciones en el mundo de la programación.

Fundamentos teóricos

- La idea básica consiste en construir una lista cuyos elementos llamados nodos se componen de dos partes o campos:
  - La primera parte o campo contiene la información.
  - La segunda parte o campo es un puntero que apunta al siguiente nodo de la lista.

- La representación más extendida es aquella que utiliza un rectángulo con dos secciones al interior, valor del dato y el enlace. El último nodo apunta `NULL`.



Fundamentos teóricos

Clasificación de las listas enlazadas

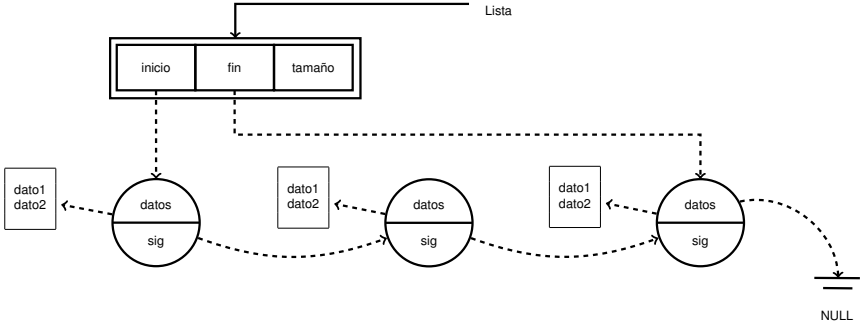
- **Listas simplemente enlazadas:** Cada nodo contienen un único enlace con el nodo siguiente.
- **Listas doblemente enlazadas:** Cada nodo contienen dos enlaces, uno con su predecesor y otro con su sucesor.
- **Listas circulares simplemente enlazadas**: Es una lista simplemente enlazada en la que el último elemento se enlaza al primer elemento.
- **Listas circulares doblemente enlazadas**: Es una lista doblemente enlazada en la que el último elemento se enlaza al primer elemento.

Fundamentos teóricos

Operaciones en las listas enlazadas

- Inicialización o creación, con declaración de los nodos.
- Insertar elementos en una lista.
- Eliminar elementos de una lista.
- Buscar elementos de una lista (comprobar la existencia de elementos en una lista).
- Recorrer una lista enlazada.
- Comprobar si la lista enlazada está vacía.

Declaración de una Lista

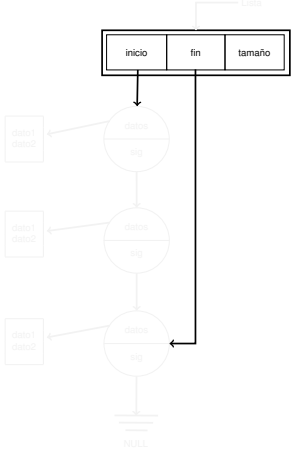


Declaración de una Lista

La lista

- Una lista vacía no contiene nodos.
- Se debe conocer el comienzo y el final.
- Se debe conocer el tamaño actual de la lista.

```
1 typedef struct lista {
2     Nodo *inicio;
3     Nodo *fin;
4     int tamano;
5 } Lista;
```

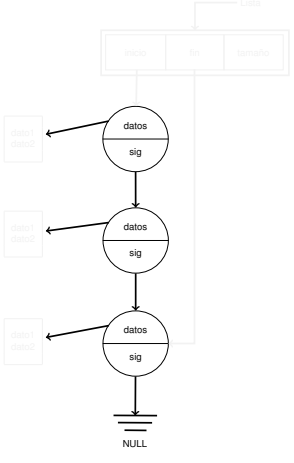


Declaración de una Lista

El nodo

- Cada nodo debe almacenar información.
- Cada nodo debe almacenar la dirección del nodo siguiente.

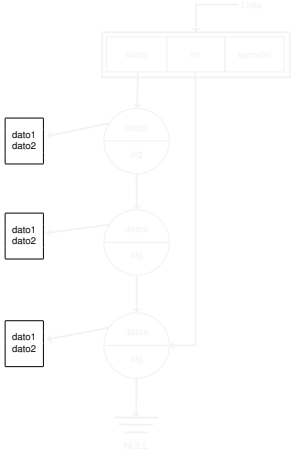
```
1 typedef struct nodo {
2     Info *datos;
3     struct nodo *siguiente;
4 } Nodo;
```



Declaración de una Lista

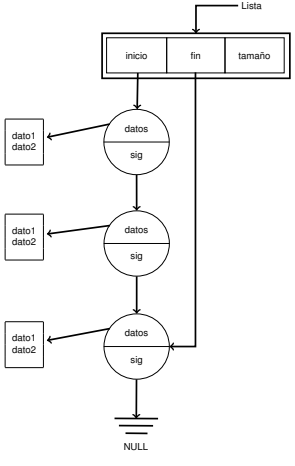
La información

```
1 typedef struct info {
2     int dato1;
3     /* int dato2; */
4     /* ... */
5 } Info;
```



Declaración de una Lista

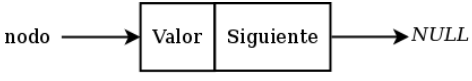
```c
typedef struct info {
    int dato1;
    /* int dato2; */
    /* ... */
} Info;
/**
 * Estructura que guarda el nodo, un puntero que apunta a la Info,
 * y otro que apunta al nodo siguiente
 */
typedef struct nodo {
    Info *datos;
    struct nodo *siguiente;
} Nodo;
/**
 * Estructura que guarda la Lista, con un puntero que apunta al inicio
 *  y al fin de ésta. Además, almacena el tamaño de la lista.
 */
typedef struct lista {
    Nodo *inicio;
    Nodo *fin;
    int tamano;
} Lista;
```



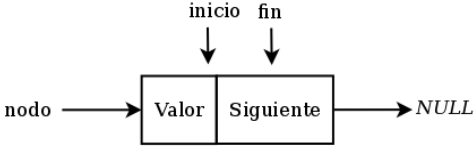
Operaciones sobre listas

Insertar en una Lista vacía

1. Crear un nodo y hacer que su siguiente apunte a `NULL`.



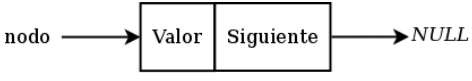
2. Hacer que *inicio* y *fin* apunten a nodo.



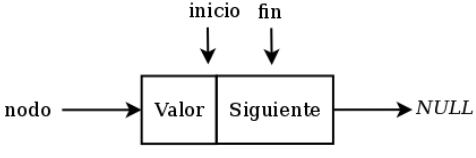
Operaciones sobre listas

Insertar en una Lista vacía

1. Crear un nodo y hacer que su siguiente apunte a `NULL`.



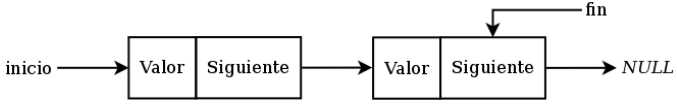
2. Hacer que *inicio* y *fin* apunten a nodo.



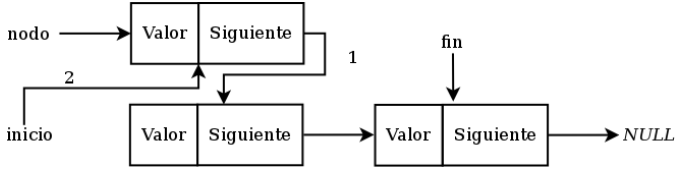
Insertar en una lista no vacía

1. Crear un nodo y hacer que su siguiente apunte al *inicio*.
2. Hacer que *inicio* apunte al nodo.

- Antes:



- Después:



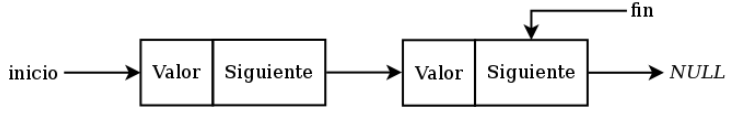
Eliminar un elemento

Suponiendo que se parte de una lista con uno o más nodos, considere un apuntador auxiliar nodo:

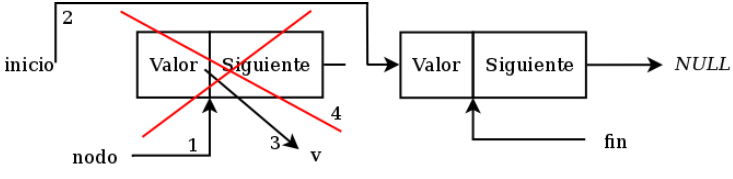
1. Hacer que nodo *aux* apunte al primer elemento de la lista, es decir al *inicio*.
2. Asignar a *inicio* la dirección del segundo nodo de la lista; es decir, el de su nodo siguiente.
3. Guardar el contenido del nodo para devolverlo como retorno.
4. Liberar la memoria asignada al nodo, que es el que se desea eliminar.

Eliminar un elemento

- Antes:



- Después:



Ventajas de las listas enlazadas

- Inserción y extracción de nodos con coste independiente del tamaño de la lista.
- Concatenación y partición listas con coste independiente del tamaño de las listas.
- No hay necesidad de grandes cantidades de memoria contigua.
- El uso de memoria se adapta dinámicamente al número de datos almacenados en la lista en cada momento.

Desventajas de las listas enlazadas

- Acceso a posiciones intermedias con coste dependiente del tamaño de la lista.
- Necesidad de memoria adicional para almacenar los nodos con sus atributos.

Eficiencia

Operaciones	Arreglo Estático	Arreglo Dinámico	Lista Enlazada
Acceder	$O(1)$	$O(1)$	$O(n)$
Buscar	$O(n)$	$O(n)$	$O(n)$
Insertar	–	$O(n)$	$O(1)$
Eliminar	–	$O(n)$	$O(1)$

Ejercicios

- Diseñe una función que muestre la lista desde el primer elemento hasta el último.
- Diseñe una función que muestre la lista desde el último elemento hasta el primero. Pista: utilice recursividad.

Universidad Andrés Bello
Conectar · Innovar · Liderar

Listas Enlazadas