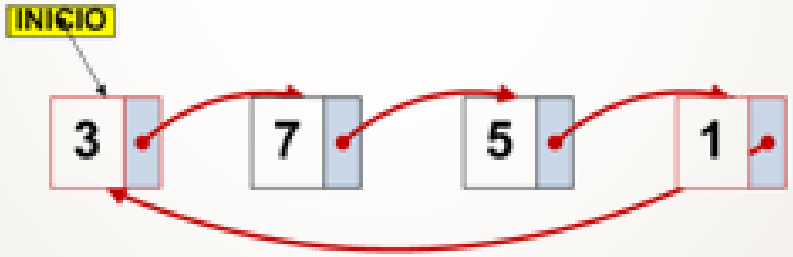
Analista Programador Universitario

Estructura de Datos

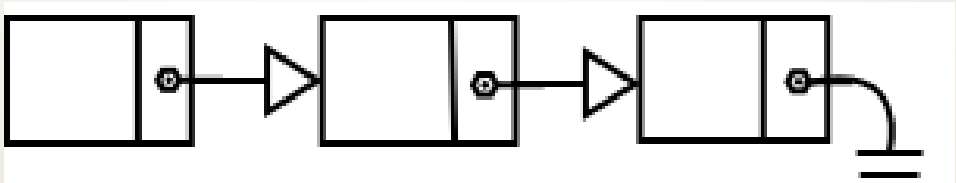
Unidad I: Listas Simples



Facultad de Ingeniería
Universidad Nacional de Jujuy

Listas Simples (1)

- Una lista simple es una colección de elementos (nodos) ordenada según su posición, cuyo acceso/recorrido se realiza mediante punteros que enlazan los nodos.
- Una lista es una estructura lineal en la que los elementos (nodos) se disponen de tal forma que cada uno tiene un predecesor y un sucesor, salvo el primero y el último.



Lista enlazada

Listas Simples (2)

- Un nodo es un registro con 2 campos esenciales:
  - Campo de datos (tipos de datos simples o compuestos)
  - Campo puntero (un puntero hacia otro nodo del mismo tipo.)

Listas Simples (3)

nodo=REGISTRO

datos: tipo_dato (simple, compuesto)

siguiente: puntero a nodo

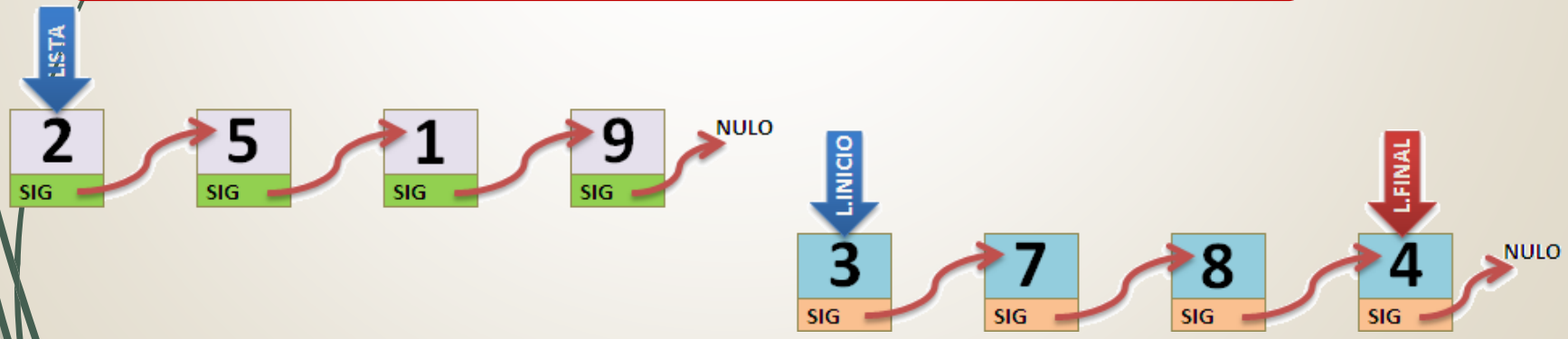
FIN_REGISTRO

Operaciones Fundamentales

- Sobre una lista simple definen las siguientes operaciones:
  - Iniciar lista
  - Crear nodo
  - Agregar nodo
    - ✓ agregar_inicio
    - ✓ agregar_final
    - ✓ agregar en orden
  - Quitar nodo
    - ✓ quitar_inicio
    - ✓ quitar_final
    - ✓ quitar_nodo_especifico
  - Mostrar (recorrido de la lista)
  - Buscar un valor en la lista

Alternativas de Implementación

- Básicamente, la implementación del TDA lista requiere de la definición de un registro (datos y puntero a próximo elemento) y punteros que permitan acceder a la lista. La implementación puede presentar las siguientes variantes:
  - Un puntero al inicio de la lista
  - Un puntero al inicio y otro al final de la lista



Implementación (1)

- TDA lista: Implementación del tipo nodo y del puntero a éste.

```
typedef struct tnodo *pnodo;
typedef struct tnodo{
                        int dato;
                        pnodo sig;
                       };
typedef struct tlista{
                        pnodo inicio;
                        pnodo final;
                       };
```

  - *inicio*: es el puntero que indica el primer nodo de la lista.
  - *final*: es el puntero que indica el último nodo de la lista.

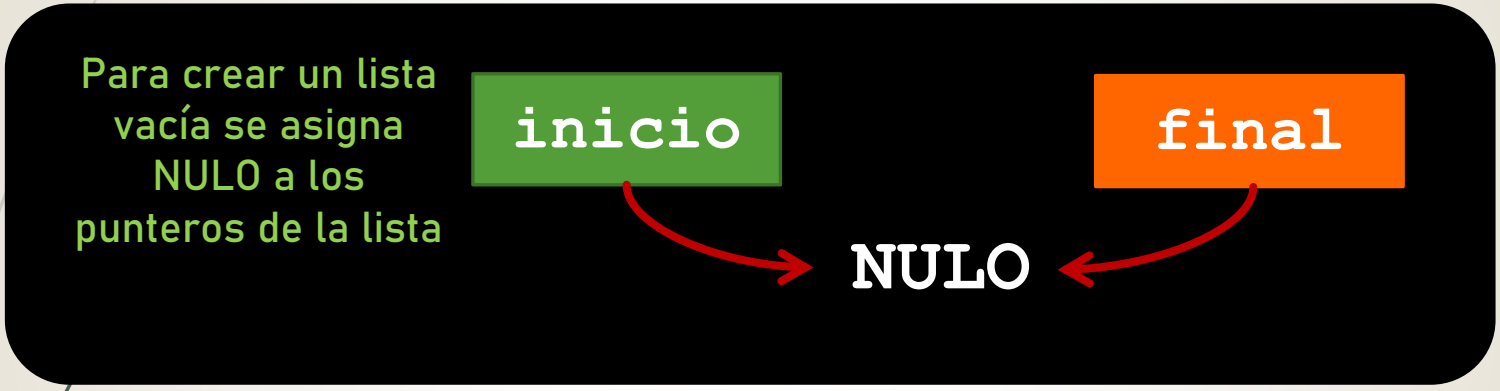
Implementación (2)

- ¿Cómo se modifican las operaciones fundamentales al utilizar 2 punteros?
  - Se debe inicializar 2 punteros.
  - Al agregar el primer nodo deben actualizar ambos punteros.
  - Se simplifica la inserción de elementos al final de la lista.
  - Si la lista está ordenada, se simplifica la búsqueda de datos.
  - Al eliminar un nodo único se deben actualizar ambos punteros.
  - Al eliminar un nodo del final de la lista debe actualizarse el puntero correspondiente.

Implementación (3)

- Operación *iniciar lista*

- Iniciar lista
  - Propósito: inicializar la lista (esto genera una lista vacía).
  - Entrada: una lista (puntero de inicio de la lista).
  - Salida: una lista vacía (puntero de inicio de la lista en valor NULO).
  - Restricciones: ninguna.



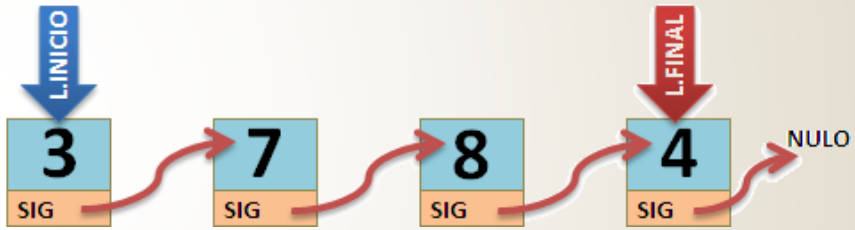

```
void inicia_lista(tlista &lista)
{
      lista.inicio=NULL;
      lista.final=NULL;
}
```

Implementación (4)

- Crear nodo
  - Propósito: crear un nuevo nodo (se reserva memoria para un nuevo elemento).
  - Entrada: un puntero a nodo.
  - Salida: un puntero con la dirección del nodo creado. Si el nodo no puede crearse, retorna NULO.
  - Restricciones: ninguna.

- Operación *crear nodo*

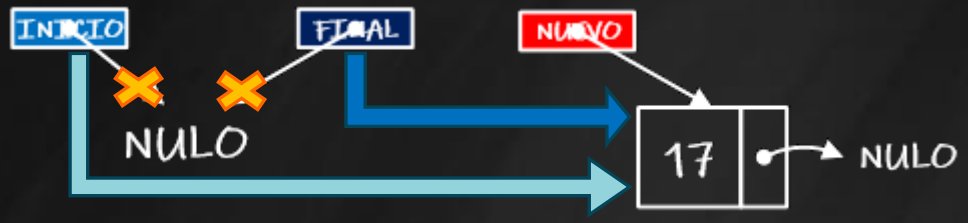
```
void crear(pnodo &nuevo, int valor)
{
   nuevo=new tnodo;
   if (nuevo!=NULL)
       { nuevo->dato=valor;
         nuevo->sig=NULL;
       }
   else
       cout << "MEMORIA INSUFICIENTE" << endl;
}
```



Operación agregar al inicio

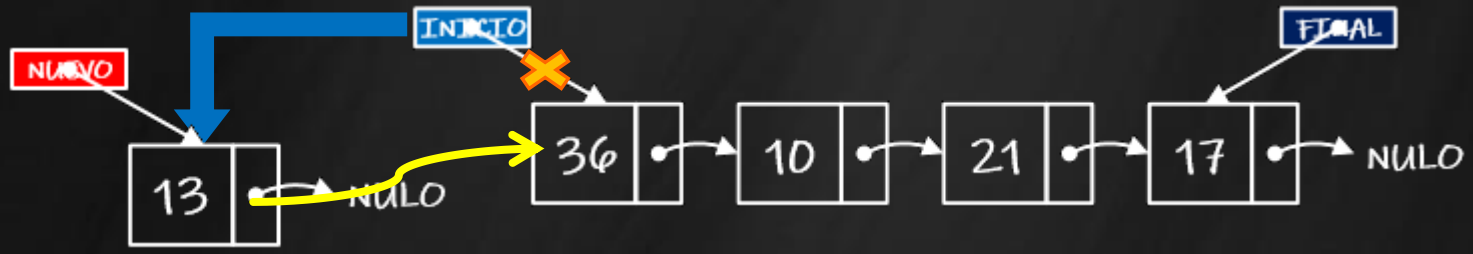
Permite agregar un nodo al comienzo de la lista

Caso 1: Lista Vacía




```
lista.inicio=nuevo;
lista.final=nuevo;
```

Caso 2: Lista con elementos




```
nuevo->sig=lista.inicio;
lista.inicio=nuevo;
```

Implementación (5)

- Operación *agregar al inicio*

```
void agregar_inicio(tlista &lista,pnodo nuevo)
{ if (lista.inicio==NULL)
     { lista.inicio=nuevo;
       lista.final=nuevo;
  else
    { nuevo->sig=lista.inicio;
      lista.inicio=nuevo;
    }
}
```

Lista Vacía

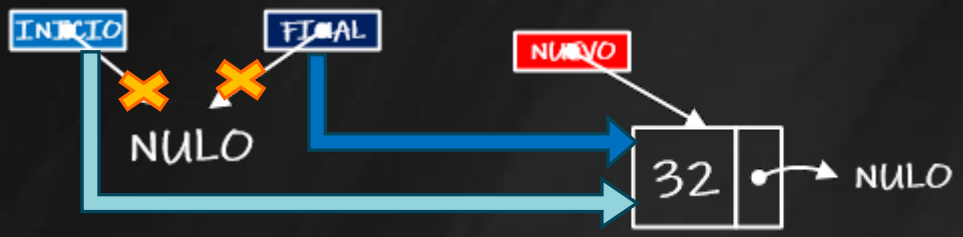
Lista con elementos

- Agregar un nodo al inicio
  - Propósito: agregar un nodo al inicio de la lista.
  - Entrada: una lista y un nuevo dato.
  - Salida: una lista con un nuevo nodo al principio.
  - Restricciones: una lista inicializada y espacio en memoria para creación del nuevo nodo.

Operación agregar al final

Permite agregar un nodo al final de la lista

Caso 1: Lista Vacía




```
lista.inicio=nuevo;
lista.final=nuevo;
```

Caso 2: Lista con elementos

¿RECORRIDO?

```
lista.final->sig=nuevo;
lista.final=nuevo;
```

Implementación (6)

- Agregar un nodo al final
  - Propósito: agregar un nodo al final de la lista.
  - Entrada: una lista y un nuevo dato.
  - Salida: una lista con un nuevo nodo al principio.
  - Restricciones: una lista inicializada y espacio en memoria para creación del nuevo nodo.

- Operación *agregar al final*

```
void agregar_final(tlista &lista,pnodo nuevo)
{
  if (lista.inicio==NULL)
     {lista.inicio=nuevo;
      lista.final=nuevo;
     }
  else
     {lista.final->sig=nuevo;
      lista.final=nuevo;
     }
}
```

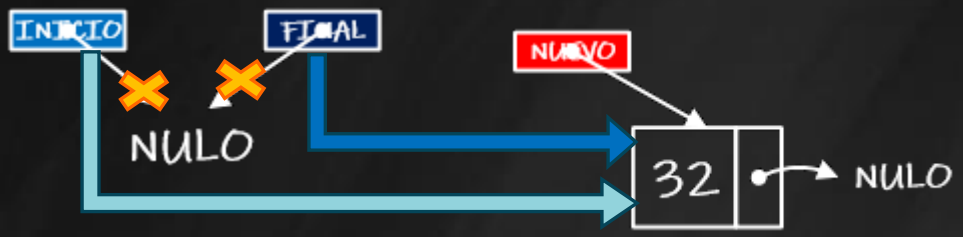
Agregar en una lista vacía

Agregar al final de la lista

Operación agregar en orden

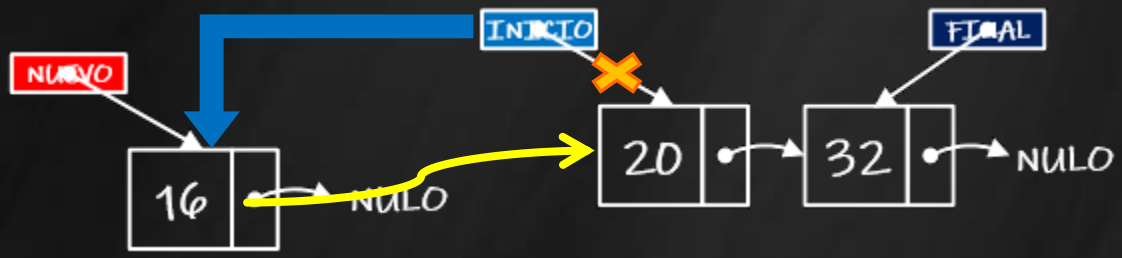
Permite agregar un nodo a la lista con un criterio de orden

Caso 1: Lista Vacía




```
lista.inicio=nuevo;
lista.final=nuevo;
```

Caso 2: el nuevo valor es menor que el primer elemento de la lista



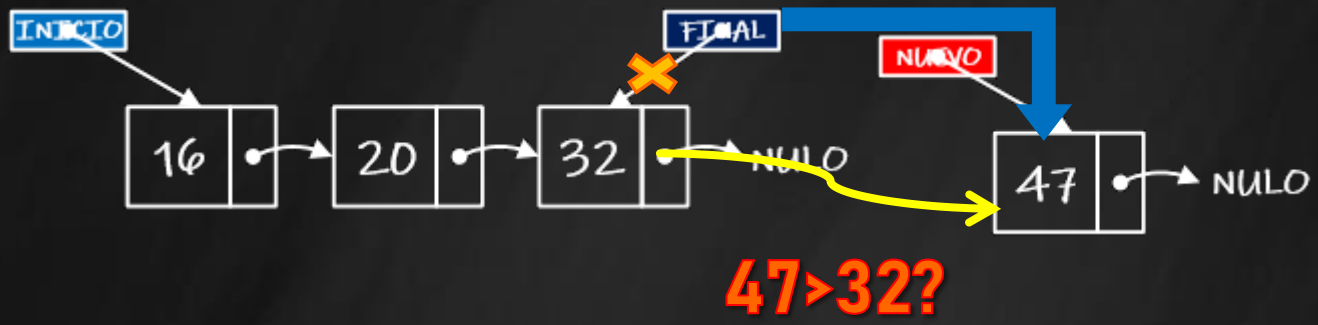
16<20?

```
nuevo->sig=lista.inicio;
lista.inicio=nuevo;
```

Operación agregar en orden

Permite agregar un nodo a la lista con un criterio de orden

Caso 3: el nuevo valor es mayor que el último elemento de la lista

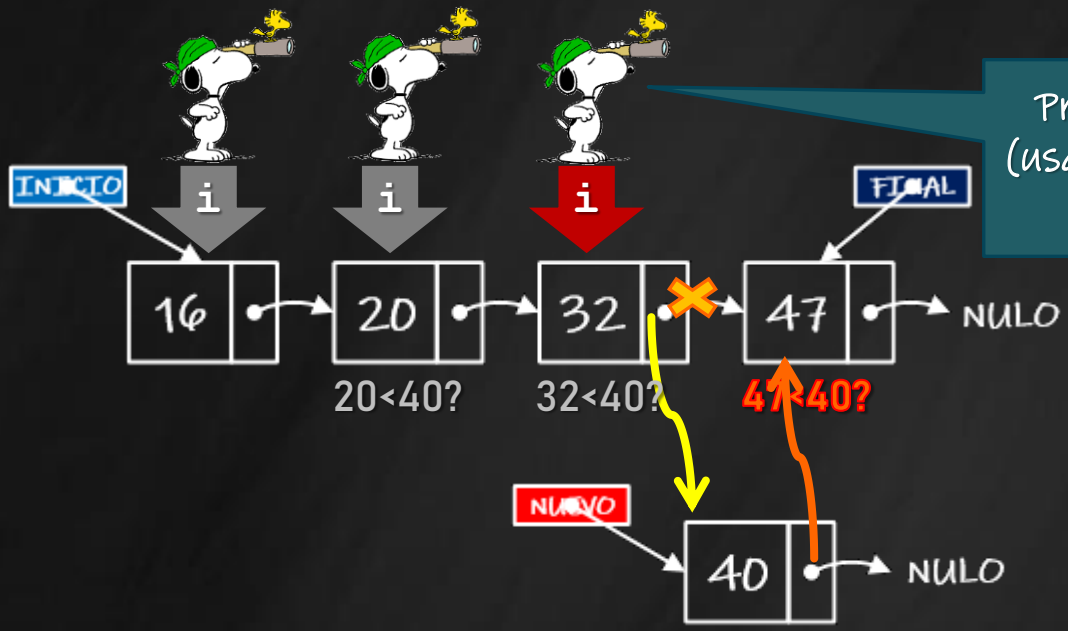



```
lista.final->sig=nuevo;
lista.final=nuevo;
```

Operación agregar en orden

Permite agregar un nodo a la lista con un criterio de orden

Caso 4: el nuevo valor debe ubicarse en el medio de la lista



Preguntamos de forma anticipada (usando el puntero sig) para conectar correctamente el nuevo nodo

```
i!=lista.final
```

```
for ( i=lista.inicio ; i->sig!=NULL && (i->sig)->dato < nuevo->dato  ;  i=i->sig );
 nuevo->sig=i->sig;
 i->sig=nuevo;
```

Implementación (7)

- Agregar un nodo en orden
  - Propósito: agregar, en orden, un nodo a la lista.
  - Entrada: una lista y un nuevo dato.
  - Salida: una lista, ordenada, con un nuevo nodo.
  - Restricciones: una lista inicializada y espacio en memoria para la creación del nuevo nodo.

- Operación *agregar en orden*

```
void agregar_orden(tlista &lista,pnodo nuevo)
{ pnodo i;
  if (lista.inicio==NULL)
   { lista.inicio=nuevo;
     lista.final=nuevo;
  else
  { if (nuevo->dato < lista.inicio->dato)
     { nuevo->sig=lista.inicio;
       lista.inicio=nuevo;}
     else
      if (nuevo->dato > lista.final->dato)
       { lista.final->sig=nuevo;
         lista.final=nuevo; }
      else
      { for(i=lista.inicio;i->sig!=NULL && nuevo->dato > (i->sig)->dato
           ;i=i->sig);
        nuevo->sig=i->sig;
        i->sig=nuevo; }
   }
}
```

Lista vacía

Agregar al inicio
(el nuevo valor es menor que el primero)

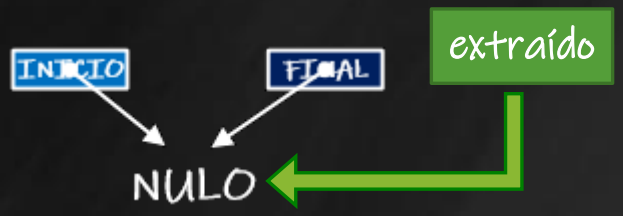
Agregar al final
(el nuevo valor es mayor que el último)

Agregar al medio o final

Operación quitar inicio

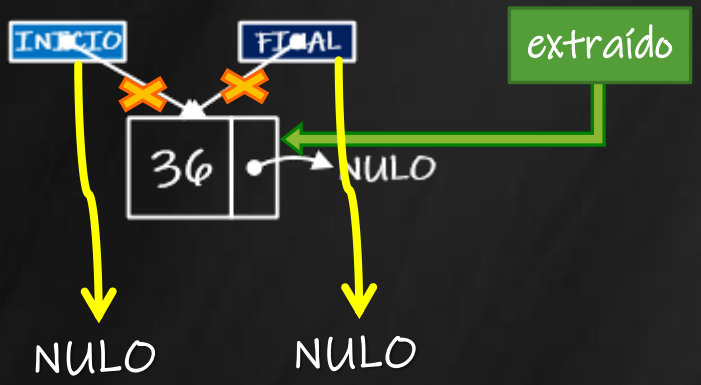
Permite extraer el primer nodo de la lista.

Caso 1: Lista Vacía




```
extraido=NULL;
```

Caso 2: Lista con un único elemento

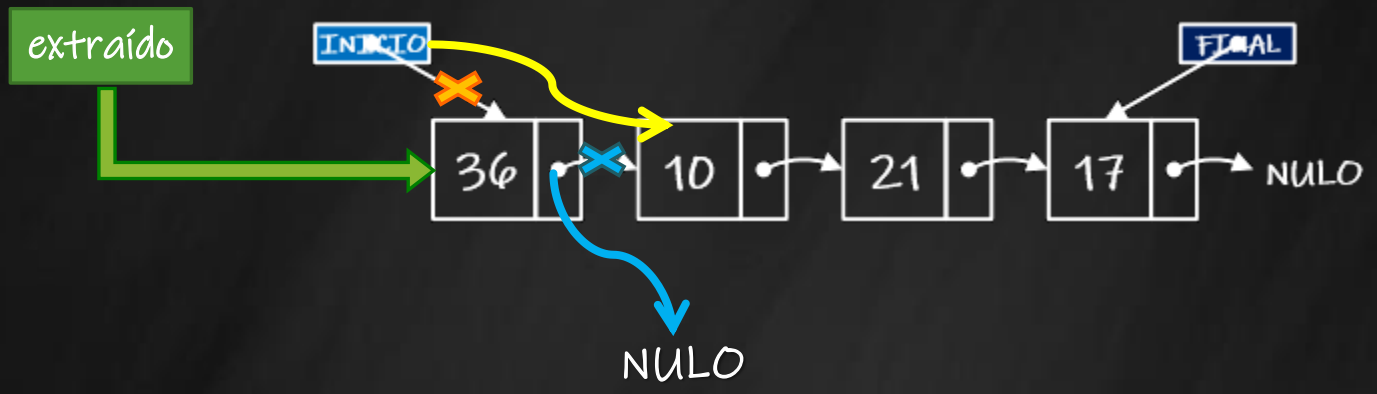



```
extraido=lista.inicio;
lista.inicio=NULL;
lista.final=NULL;
```

Operación quitar inicio

Caso 3: Lista con 2 o más elementos

Permite extraer el primer nodo de la lista.




```
extraido=lista.inicio;
lista.inicio=lista.inicio->sig;
extraido->sig=NULL;
```

Implementación (8)

- Quitar un nodo del inicio
  - Propósito: quitar el primer nodo de la lista.
  - Entrada: una lista.
  - Salida: una lista con un nodo menos (extraído del inicio) y la dirección del elemento extraído.
  - Restricciones: una lista inicializada y no vacía.

- Operación *quitar del inicio*

```
pnodo quitar_inicio(pnodo &lista)
{ pnodo extraido;
  if (lista.inicio==NULL)
     extraido=NULL;
  else
    if (lista.inicio==lista.final)
    { extraido=lista.inicio;
      lista.inicio=NULL;
      lista.final=NULL;  }
    else
    { extraido=lista.inicio;
      lista.inicio=lista.inicio->sig;
      extraido->sig=NULL;
   }
  return extraido;
}
```

Extracción de lista vacía

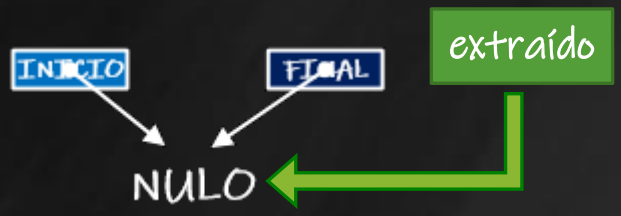
Extracción del único nodo

Extracción del primer nodo

Operación quitar final

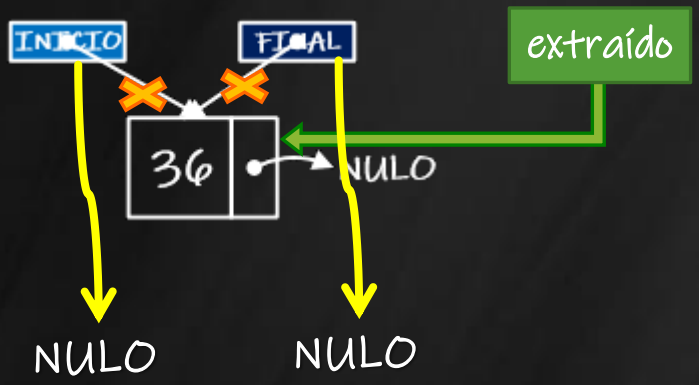
Permite extraer el último nodo de la lista.

Caso 1: Lista Vacía




```
extraido=NULL;
```

Caso 2: Lista con un único elemento

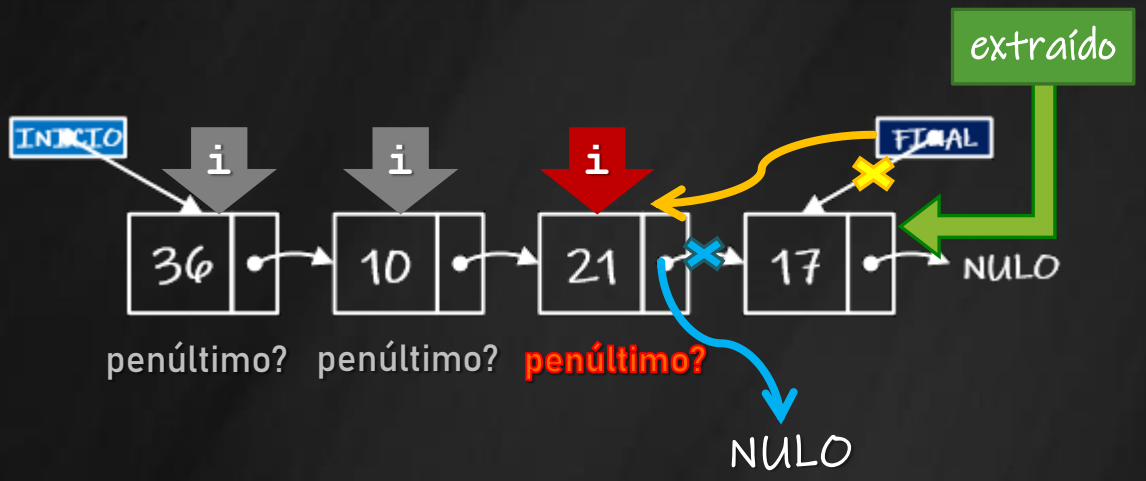



```
extraido=lista.inicio;
lista.inicio=NULL;
lista.final=NULL;
```

Operación quitar final

Caso 3: Lista con 2 o más elementos

Permite extraer el último nodo de la lista.



i->sig!=lista.final

```
for(i=lista.inicio ;(i->sig)->sig!=NULL; i=i->sig );
extraido=i->sig;
i->sig=NULL;
lista.final=i;
```

Implementación (9)

- Quitar un nodo del final
  - Propósito: quitar el último nodo de la lista.
  - Entrada: una lista.
  - Salida: una lista con un nodo menos (extraído del final) y la dirección del elemento extraído.
  - Restricciones: una lista inicializada y no vacía.

- Operación *quitar del final*

```
pnodo quitar_final(tlista &lista)
{ pnodo extraido,i;
  if (lista.inicio==NULL)
     extraido=NULL;
  else
   {if (lista.inicio==lista.final)
      { extraido=lista.inicio;
        lista.inicio=NULL;
        lista.final=NULL; }
    else
      { for(i=lista.inicio;(i->sig)->sig!=NULL;i=i->sig);
        extraido=lista.final;
        lista.final=i;
        lista.final->sig=NULL; }
   }
  return extraido;
}
```

Extracción de lista vacía

Extracción del único nodo

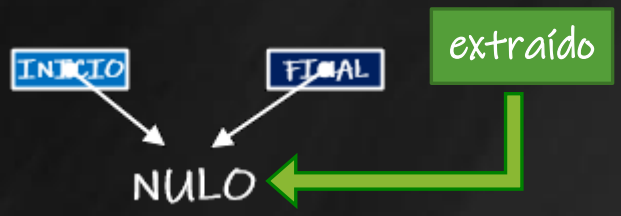
Extracción del último nodo

Operación quitar nodo específico

Permite extraer un nodo de la lista que contiene un valor específico.

Caso 1: Lista Vacía

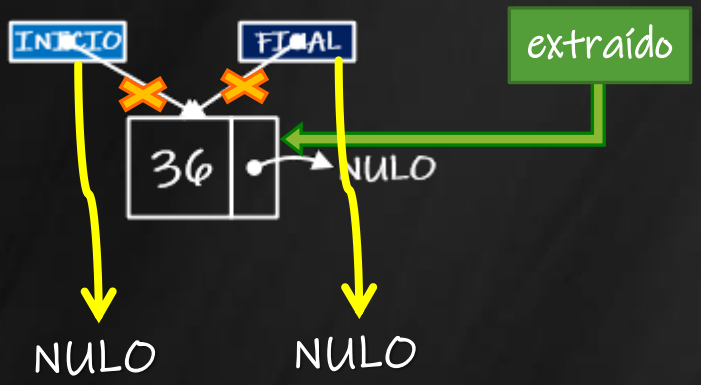
Extraer 36




```
extraido=NULL;
```

Caso 2: Extracción del primer elemento (único elemento)

Extraer 36



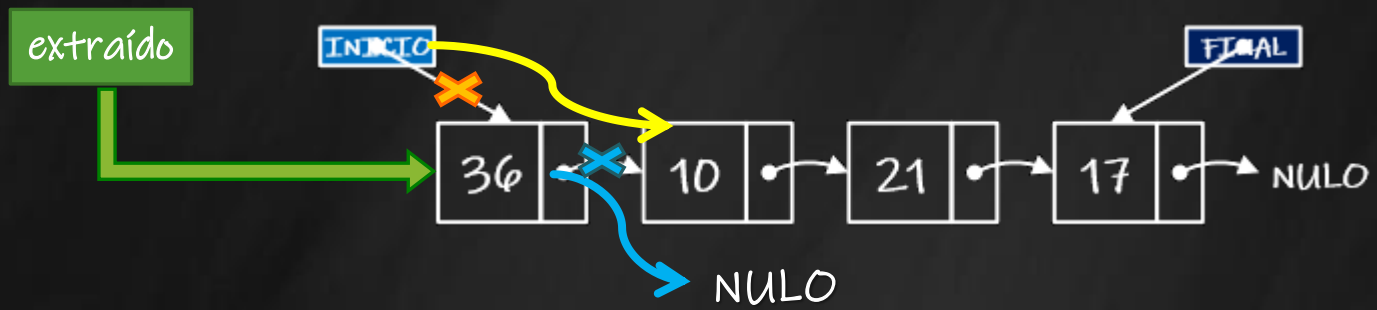

```
extraido=lista.inicio;
lista.inicio=NULL;
lista.final=NULL;
```

Operación quitar nodo específico

Permite extraer un nodo de la lista que contiene un valor específico.

Caso 3: Extracción del primer elemento de un lista con 2 más elementos.

Extraer 36

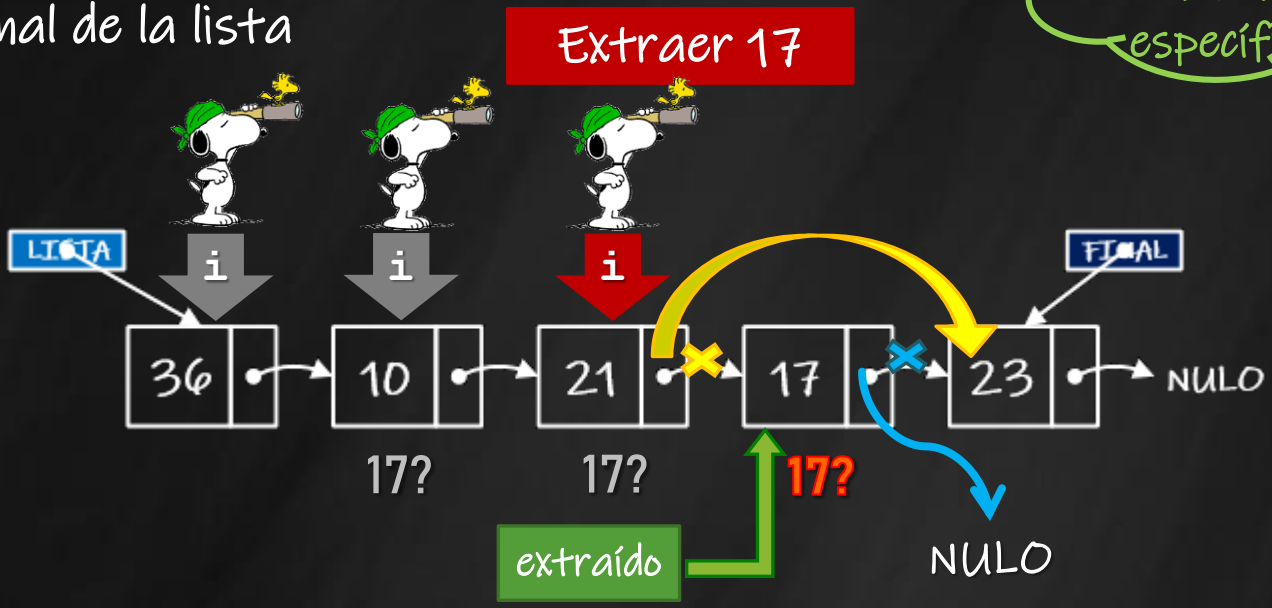



```
extraido=lista.inicio;
lista.inicio=lista.inicio->sig;
extraido->sig=NULL;
```

Operación quitar nodo específico

Permite extraer un nodo de la lista que contiene un valor específico.

Caso 4: Extracción de un valor del medio o final de la lista



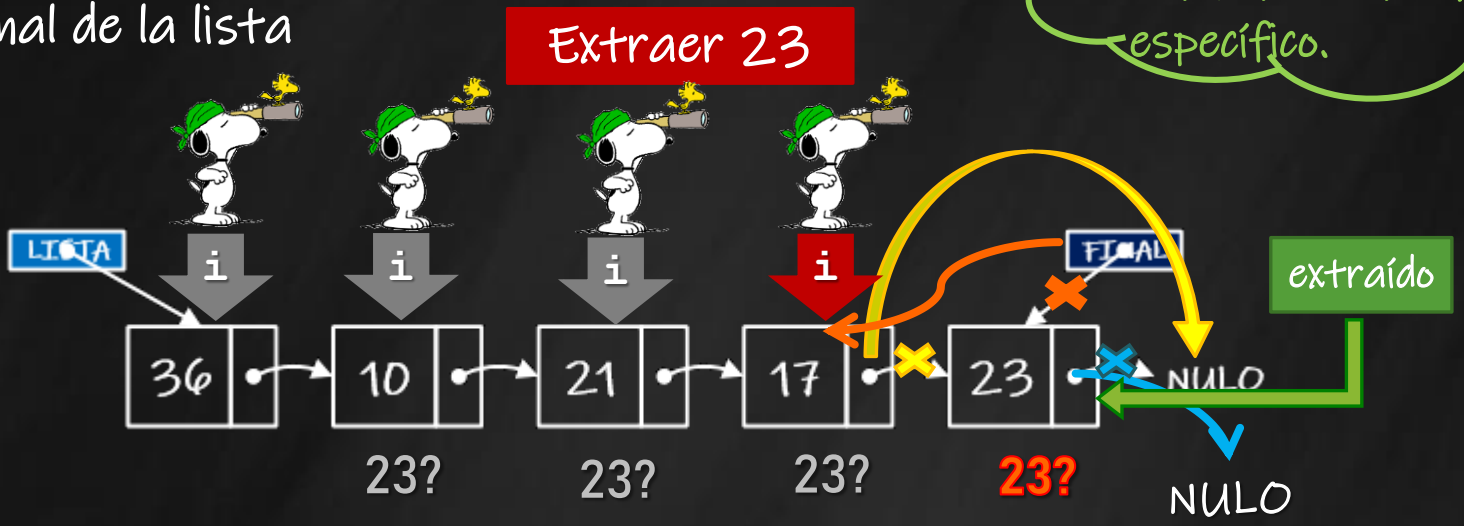

```
for (i=lista.inicio ; i->sig!=NULL && (i->sig)->dato != buscado ; i=i->sig );
 extraido=i->sig;
 i->sig=extraido->sig;
 extraido->sig=NULL;
```

¿Qué ocurre si el valor a extraer es el último de la lista?

Operación quitar nodo específico

Permite extraer un nodo de la lista que contiene un valor específico.

Caso 4: Extracción de un valor del medio o final de la lista




```
for (i=lista.inicio; i->sig!=NULL && (i->sig)->dato != buscado; i=i->sig);
extraido=i->sig;
i->sig=extraido->sig;
extraido->sig=NULL;
if (extraido==lista.final)
    lista.final=i;
```

Sólo si i->sig!=NULL

¿Qué ocurre si el valor a extraer no se encuentra en la lista?

Implementación (10)

- Quitar un nodo según un valor especificado
  - Propósito: quitar un nodo con valor específico.
  - Entrada: una lista y el valor a extraer.
  - Salida: una lista con un nodo menos (extraído el valor solicitado) y la dirección del nodo extraído.
  - Restricciones: una lista inicializada y no vacía.

- Operación *quitar un nodo según un valor solicitado*

```
pnodo quitar_nodo(tlista &lt,int valor)
{ pnodo extraido,i;
  if (lt.inicio==NULL)
    extraido=NULL;
  else
    if (lt.inicio->dato==valor)
     { if (lt.inicio==lt.final)
         { extraido=lt.inicio;
           lt.inicio=NULL;
           lt.final=NULL; }
       else
         { extraido=lt.inicio;
           lt.inicio=lt.inicio->sig;
           extraido->sig=NULL;
     }                ...
```

Extracción de lista vacía

Extracción del único nodo

Extracción del primer nodo

Implementación (10)

- Quitar un nodo según un valor especificado
  - Propósito: quitar un nodo con valor específico.
  - Entrada: una lista y el valor a extraer.
  - Salida: una lista con un nodo menos (extraído el valor solicitado) y la dirección del nodo extraído.
  - Restricciones: una lista inicializada y no vacía.

- Operación *quitar un nodo según un valor solicitado*

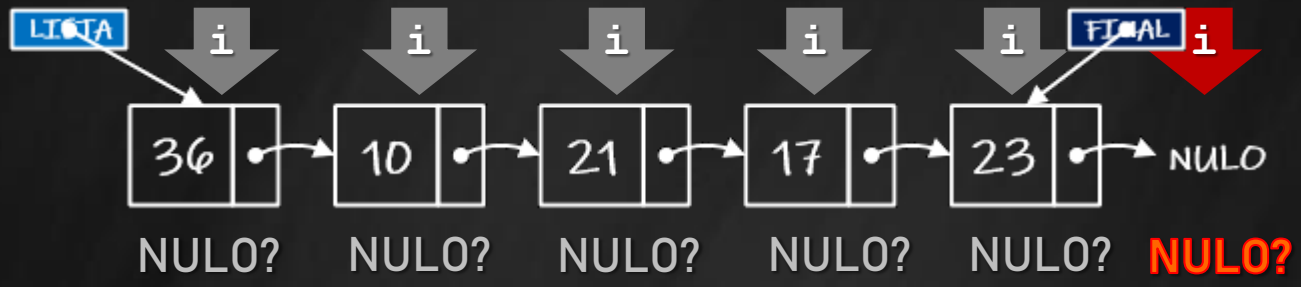
```
pnodo quitar_nodo(tlista &lt,int valor)
{ pnodo extraido,i;
                    ...
    else
    { for(i=lt.inicio;i->sig!=NULL && valor!=(i->sig)->dato;i=i->sig);
      if (i->sig!=NULL)
       { extraido=i->sig;
         i->sig=extraido->sig;
         extraido->sig=NULL;
         if (extraido==lt.final)
            lt.final=i;
       }
       else
         extraido=NULL;
    }
   return extraido;
}
```

Extracción del medio o final

Valor no encontrado

Operación mostrar lista

Permite mostrar nodo a nodo el contenido de una lista.




```
36   10   21   17   23
```

Implementación (12)

- Mostrar lista
  - Propósito: mostrar el contenido de la lista.
  - Entrada: una lista (puntero de inicio de la lista).
  - Salida: se muestran por pantalla los datos almacenados en los nodos.
  - Restricciones: una lista inicializada y no vacía.

- Operación *mostrar datos de la lista*

```
void mostrar(tlista lista)
{ pnodo i;
  if (lista.inicio!=NULL)
     for(i=lista.inicio;i!=NULL;i=i->sig)
       cout << "Nodo: " << i->dato << endl;
  else
     cout << "LISTA VACIA";
}
```

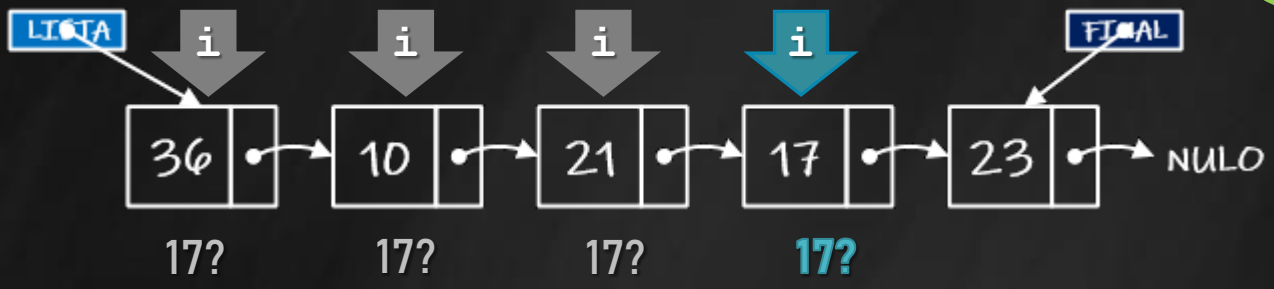
Se recorre la lista, nodo a nodo, hasta que el puntero *i* sea NULO

Operación buscar dato

Determina si un valor se encuentra o no almacenado en una lista.

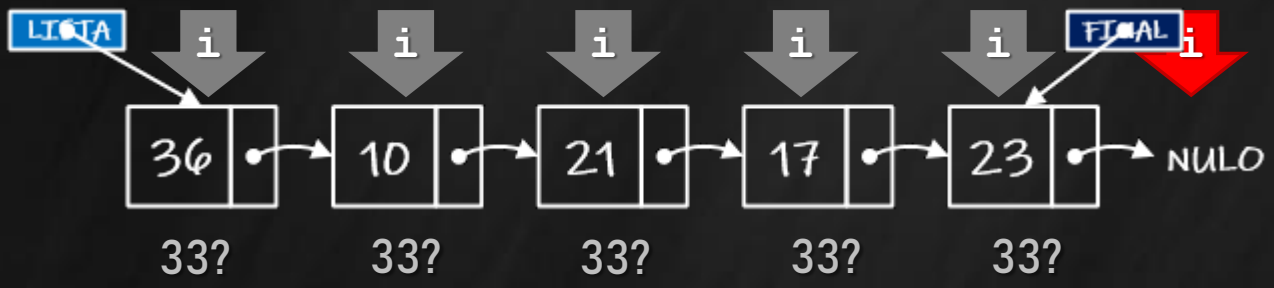
Caso Positivo: buscando un valor existente

Buscar 17



Caso Negativo: buscando un valor inexistente

Buscar 33



Implementación (13)

- Buscar un dato en la lista
  - Propósito: buscar un valor específico en la lista
  - Entrada: una lista y el valor a buscar.
  - Salida: valor true si el dato buscado se encuentra en la lista, caso contrario, false.
  - Restricciones: una lista inicializada y no vacía.

- Operación *buscar un dato en la lista*

```
bool buscar_nodo(tlista lista, int valor)
{ pnodo i;
  bool encontrado=false;
  if (lista.inicio!=NULL)
    for(i=lista.inicio;i!=NULL && encontrado==false;i=i->sig)
      if (i->dato==valor)
        encontrado=true;
  return encontrado;
}
```

Se recorre la lista, nodo a nodo, hasta que el puntero *i* sea NULO o se detecte el valor buscado.

```
pnodo buscar_nodo(tlista lista, int valor)
{ pnodo i;
  pnodo encontrado=NULL;
  if (lista.inicio!=NULL)
   for(i=lista.inicio;i!=NULL && encontrado==NULL;i=i->sig)
     if (i->dato==valor)
       encontrado=i;
  return encontrado;
}
```

¿Cómo se modifica la operación si debe obtenerse la dirección del nodo?

Bibliografía

- Joyanes Aguilar *et al*. Estructuras de Datos en C++. Mc Graw Hill. 2007.
- De Giusti, Armando *et al.* Algoritmos, datos y programas, conceptos básicos. Editorial Exacta. 1998.
- Joyanes Aguilar, Luis. Fundamentos de Programación. Mc Graw Hill. 1996.
- Hernández, Roberto *et al*. Estructuras de Datos y Algoritmos. Prentice Hall. 2001.