Estructuras de datos abstractos (ADTs) Listas

mat-151

problema de Josephus

problema de Josephus

- Hay ***n*** personas paradas en circulo esperando para ser ejecutadas. Después de ser ejecutado el primer hombre (el 0), saltan a ***k-1*** personas, y la ***k***-ésima persona es ejecutada.

problema de Josephus

- Hay ***n*** personas paradas en circulo esperando para ser ejecutadas. Después de ser ejecutado el primer hombre (el 0), saltan a ***k-1*** personas, y la ***k***-ésima persona es ejecutada.

- El proceso de ejecución continua alrededor del círculo hasta que la última persona es liberada.

problema de Josephus

- Hay ***n*** personas paradas en circulo esperando para ser ejecutadas. Después de ser ejecutado el primer hombre (el 0), saltan a ***k-1*** personas, y la ***k***-ésima persona es ejecutada.

- El proceso de ejecución continua alrededor del círculo hasta que la última persona es liberada.

- La identidad de la persona liberada es una función de ***n*** y ***k*** y se conoce como *función de Josephus.*

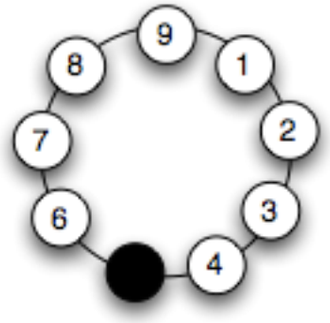
problema de Josephus

- Hay ***n*** personas paradas en circulo esperando para ser ejecutadas. Después de ser ejecutado el primer hombre (el 0), saltan a ***k-1*** personas, y la ***k***-ésima persona es ejecutada.

- El proceso de ejecución continua alrededor del círculo hasta que la última persona es liberada.

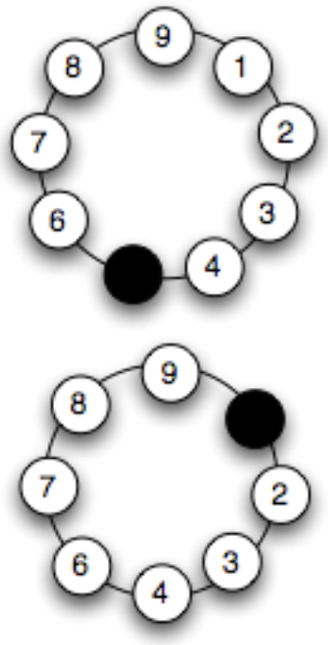
- La identidad de la persona liberada es una función de ***n*** y ***k*** y se conoce como *función de Josephus.*

- ¿Cómo podemos simular (saber) computacionalmente el orden en que las personas serán ejecutadas?



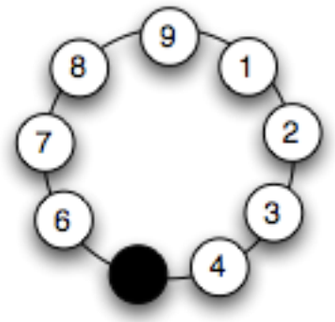
$$n = 9$$
$$k = 5$$

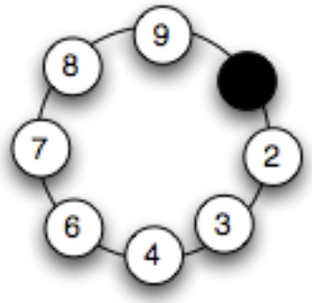
Problema de Josephus

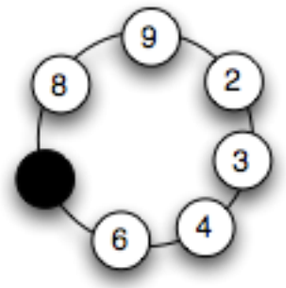


$$n = 9$$
$$k = 5$$

Problema de Josephus

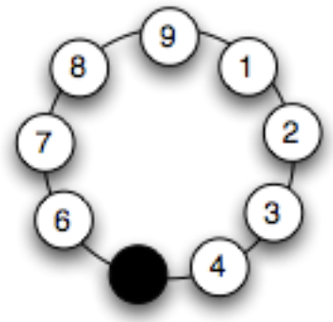


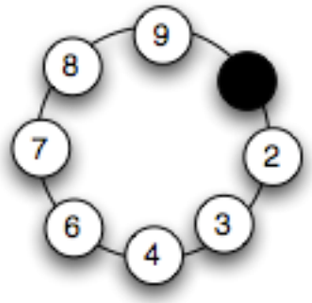


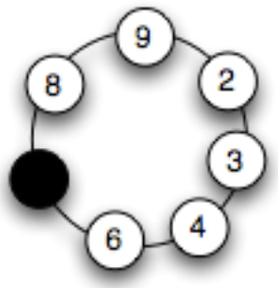


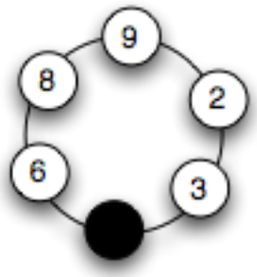
$$n = 9$$
$$k = 5$$

Problema de Josephus



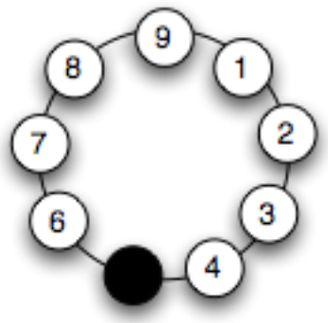


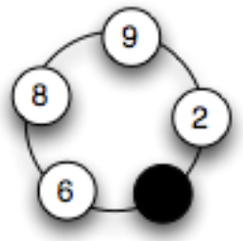




$$n = 9$$
$$k = 5$$

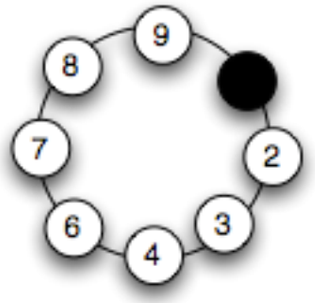
Problema de Josephus

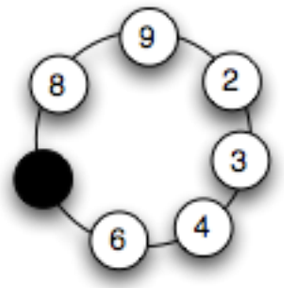


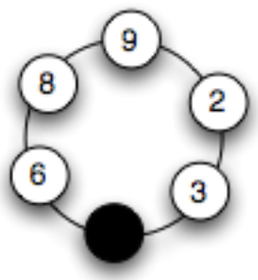


$n = 9$

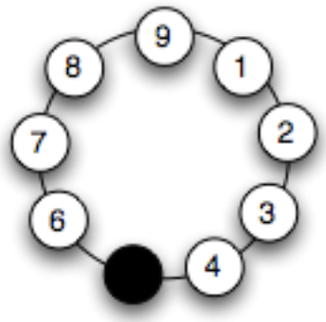
$k = 5$

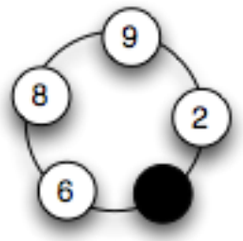






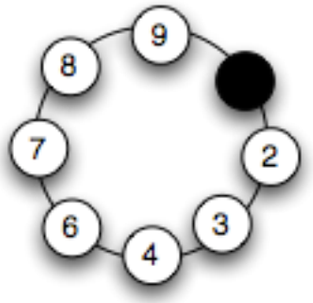
Problema de Josephus

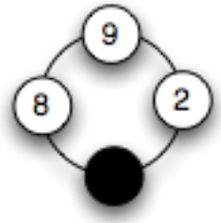


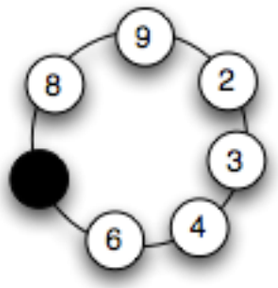


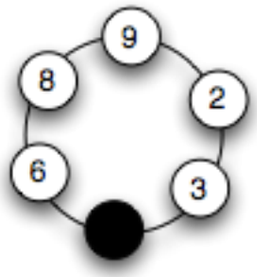
$n = 9$

$k = 5$

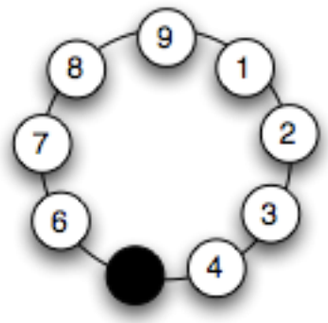


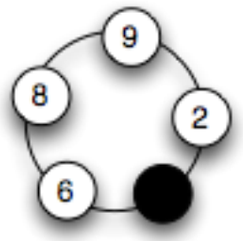






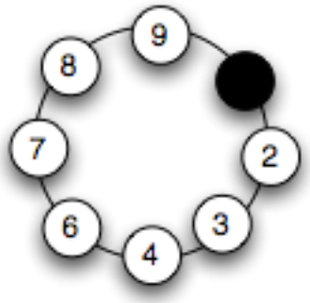
Problema de Josephus

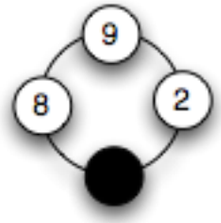


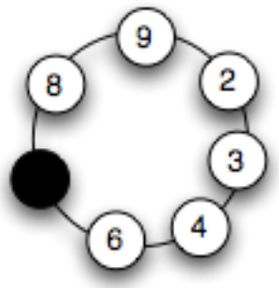


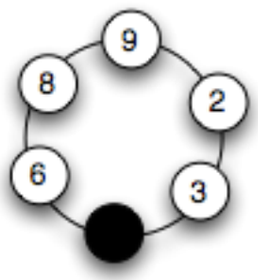
$n = 9$

$k = 5$

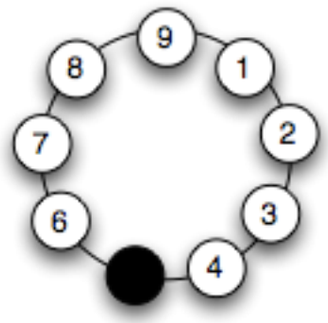


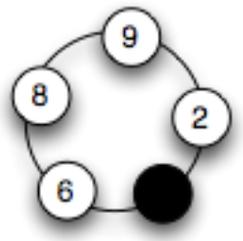






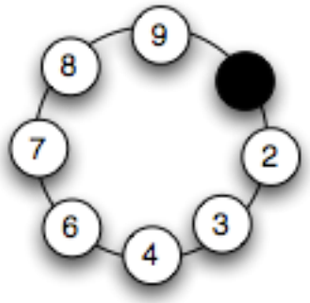
Problema de Josephus

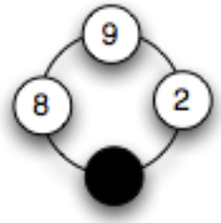


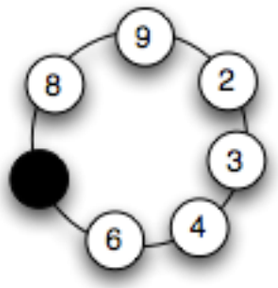


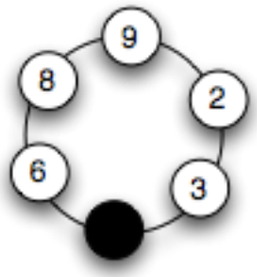
$n = 9$

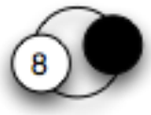
$k = 5$



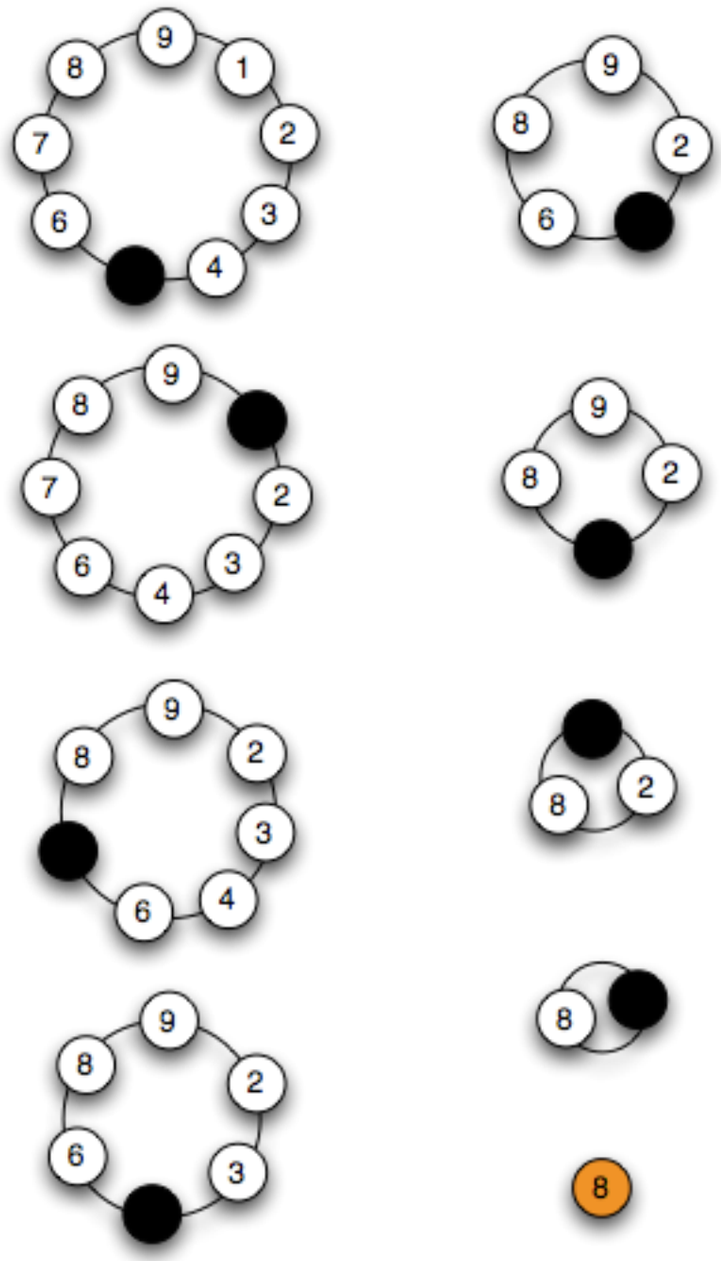








Problema de Josephus



$n = 9$

$k = 5$

Problema de Josephus

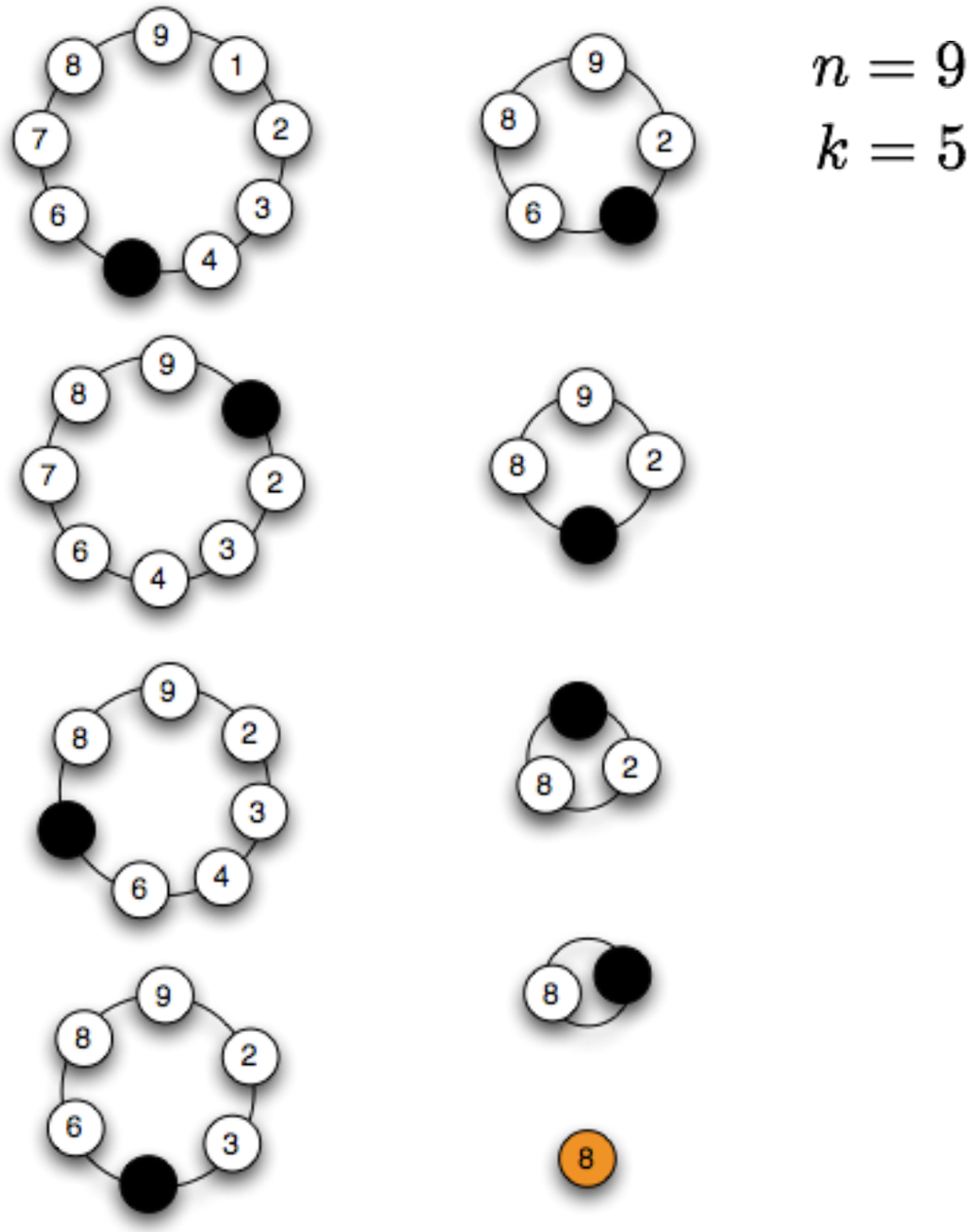
```cpp
#include <iostream>

using namespace std;

struct node {
    int item;
    node* next;
    node(int x, node* t) { item = x; next=t; }
};
typedef node *liga;

int main( int argc, char *argv[])
{
    int i, n=atoi(argv[1]), k=atoi(argv[2]);

    liga t = new node(1,NULL);
    t->next=t;
    liga x = t;
    for (i=2; i<=n; i++){
        x = (x->next = new node(i,t));
    }
    cout << "mueren ";
    while (x != x->next){
        for (i=1; i<k; i++){
            x = x->next;
        }
        t = x->next;
        x->next = t->next;
        cout << t->item << " ";
        delete t;
    }
    cout << "sobrevive: " << x->item << endl;
}
```



Problema de Josephus

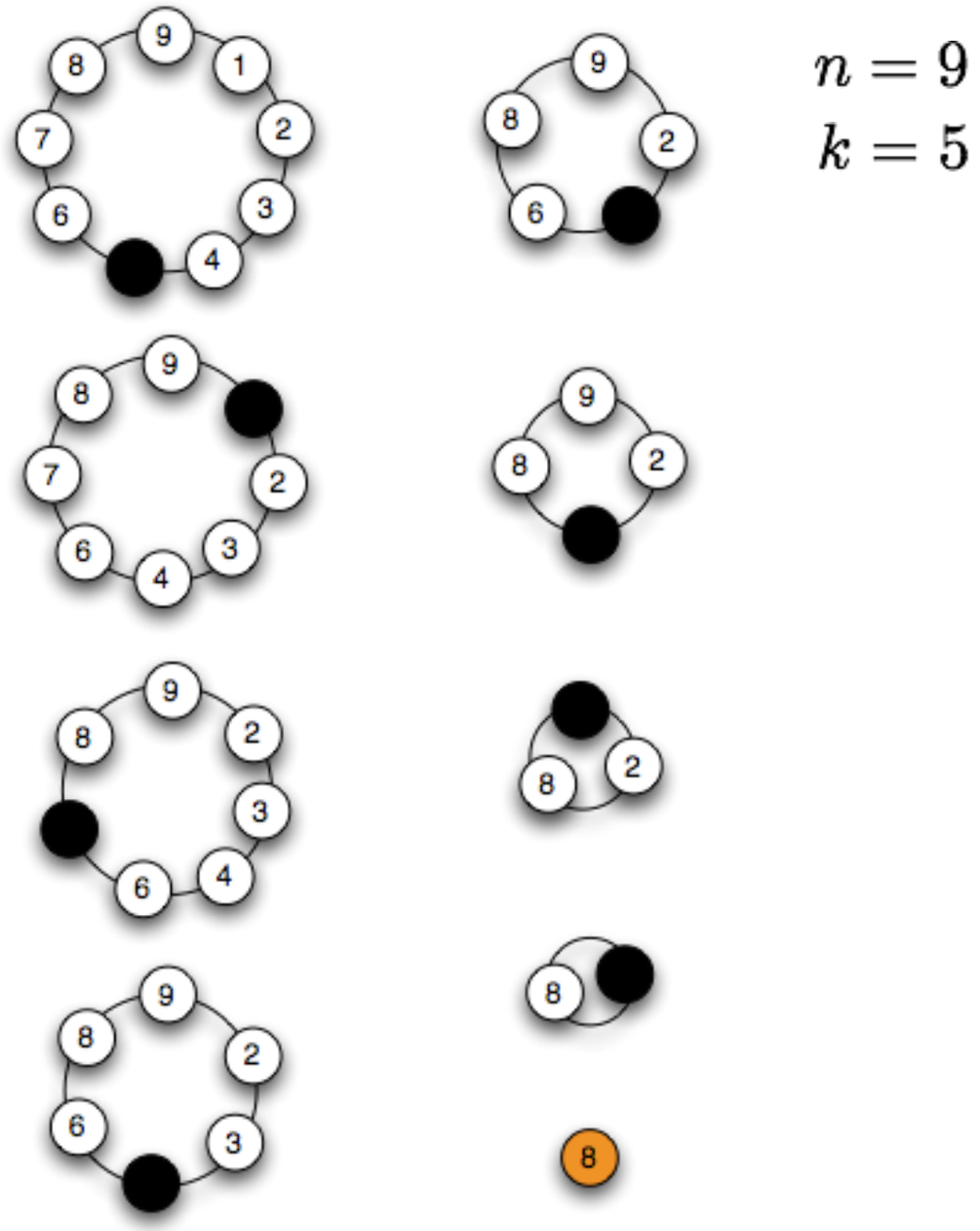
```cpp
#include <iostream>

using namespace std;

struct node {
    int item;
    node* next;
    node(int x, node* t) { item = x; next=t; }
};
typedef node *liga;

int main( int argc, char *argv[])
{
    int i, n=atoi(argv[1]), k=atoi(argv[2]);

    liga t = new node(1,NULL);
    t->next=t;
    liga x = t;
    for (i=2; i<=n; i++){
        x = (x->next = new node(i,t));
    }
    cout << "mueren ";
    while (x != x->next){
        for (i=1; i<k; i++){
            x = x->next;
        }
        t = x->next;
        x->next = t->next;
        cout << t->item << " ";
        delete t;
    }
    cout << "sobrevive: " << x->item << endl;
}
```



Problema de Josephus

```
% ./josephus 9 5
mueren 5 1 7 4 3 6 9 2 sobrevive: 8
```

arreglos vs. listas ligadas

arreglos vs. listas ligadas

- ¿ Cuál estructura es la más adaptada para la criba de Eratóstenes (encontrar números primos menores a N) y para el problema de Josephus ?

arreglos vs. listas ligadas

- ¿ Cuál estructura es la más adaptada para la criba de Eratóstenes (encontrar números primos menores a N) y para el problema de Josephus ?

  ➡ en la criba de Eratóstenes necesitamos acceder rápidamente a los elementos del arreglo para que el algoritmo sea eficiente.

arreglos vs. listas ligadas

- ¿ Cuál estructura es la más adaptada para la criba de Eratóstenes (encontrar números primos menores a N) y para el problema de Josephus ?

  ➡ en la criba de Eratóstenes necesitamos acceder rápidamente a los elementos del arreglo para que el algoritmo sea eficiente.

  ➡ en el problema de Josephus necesitamos eliminar elementos rápidamente para que el algoritmo sea eficiente.

arreglos vs. listas ligadas

- ¿ Cuál estructura es la más adaptada para la criba de Eratóstenes (encontrar números primos menores a N) y para el problema de Josephus ?

  ➡ en la criba de Eratóstenes necesitamos acceder rápidamente a los elementos del arreglo para que el algoritmo sea eficiente.

  ➡ en el problema de Josephus necesitamos eliminar elementos rápidamente para que el algoritmo sea eficiente.

- al elegir una estructura debemos estar conscientes de los efectos de esta elección en la eficiencia de los algoritmos que procesan estos datos.

arreglos vs. listas ligadas

- ¿ Cuál estructura es la más adaptada para la criba de Eratóstenes (encontrar números primos menores a N) y para el problema de Josephus ?

  ➡ en la criba de Eratóstenes necesitamos acceder rápidamente a los elementos del arreglo para que el algoritmo sea eficiente.

  ➡ en el problema de Josephus necesitamos eliminar elementos rápidamente para que el algoritmo sea eficiente.

- al elegir una estructura debemos estar conscientes de los efectos de esta elección en la eficiencia de los algoritmos que procesan estos datos.

- esta combinación datos/algoritmos es el corazón del diseño de algoritmos.

operaciones con listas

- recorrer una lista:

```
for ( link t = x; t != NULL; t = t->next ){
    do_something(t->item);
}
```

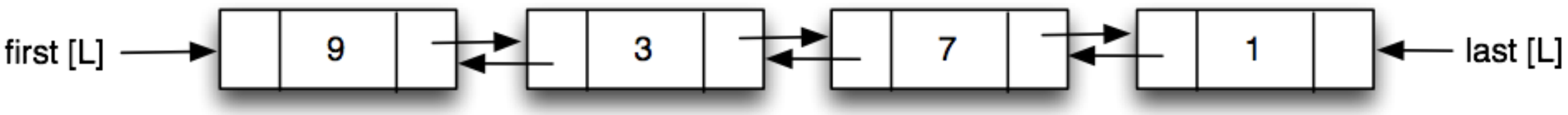
- recorrer un arreglo:

```
for ( int i=0; i<N; i++ ){
    do_something(a[i]);
}
```

¿Limitaciones de las listas ligadas?

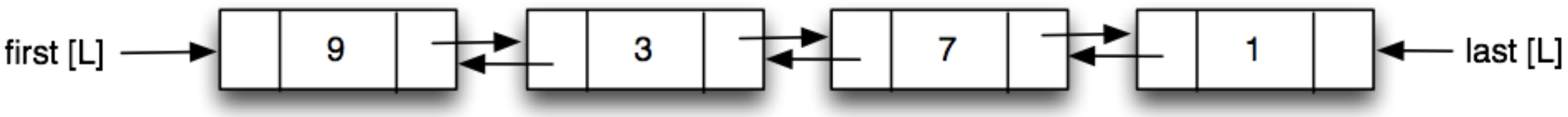
- ¿cuál nodo podemos borrar?

listas doblemente ligadas



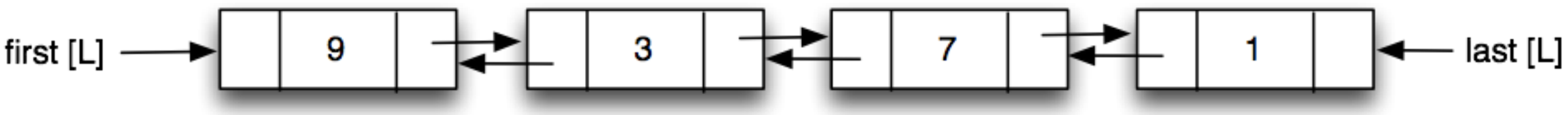
listas doblemente ligadas

- se pueden recorrer en ambas direcciones incluyendo además un **e.Prev** que es un apuntador al elemento previo de la lista.



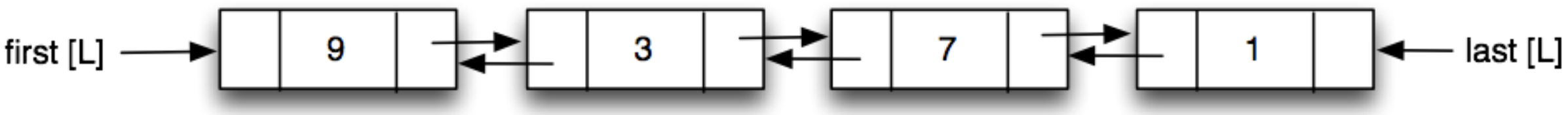
listas doblemente ligadas

- se pueden recorrer en ambas direcciones incluyendo además un **e.Prev** que es un apuntador al elemento previo de la lista.
- **e.Prev =** *null* si **e** es el primer elemento de la lista.



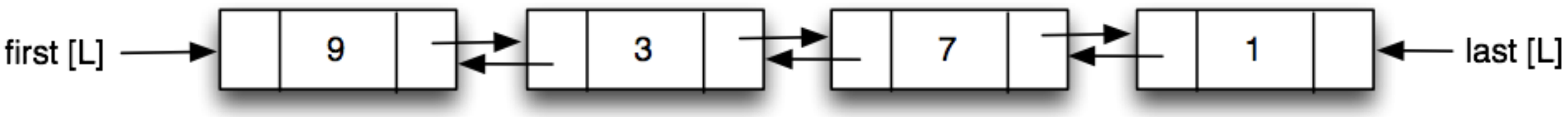
listas doblemente ligadas

- se pueden recorrer en ambas direcciones incluyendo además un **e.Prev** que es un apuntador al elemento previo de la lista.
- **e.Prev** = *null* si **e** es el primer elemento de la lista.
- análogamente a **first** se incluye un apuntador **last** al último elemento de la lista.



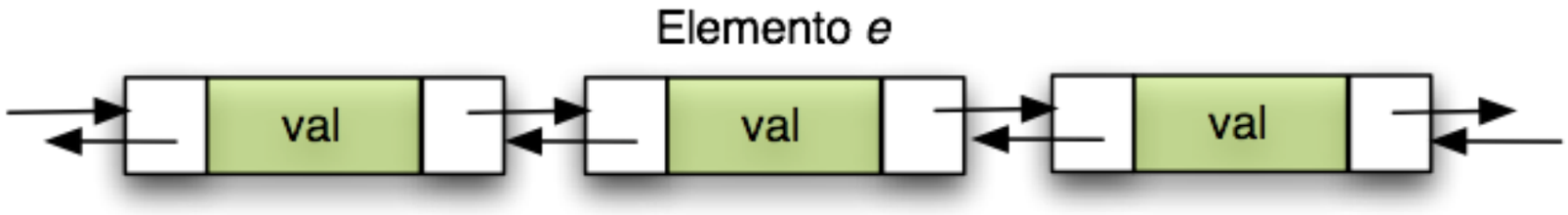
listas doblemente ligadas

- se pueden recorrer en ambas direcciones incluyendo además un **e.Prev** que es un apuntador al elemento previo de la lista.
- **e.Prev** = *null* si **e** es el primer elemento de la lista.
- análogamente a **first** se incluye un apuntador **last** al último elemento de la lista.
- el espacio requerido para los **links,** al igual que el **número de manipulaciones básicas** para una operación simple es el doble que en una lista simplemente ligada.



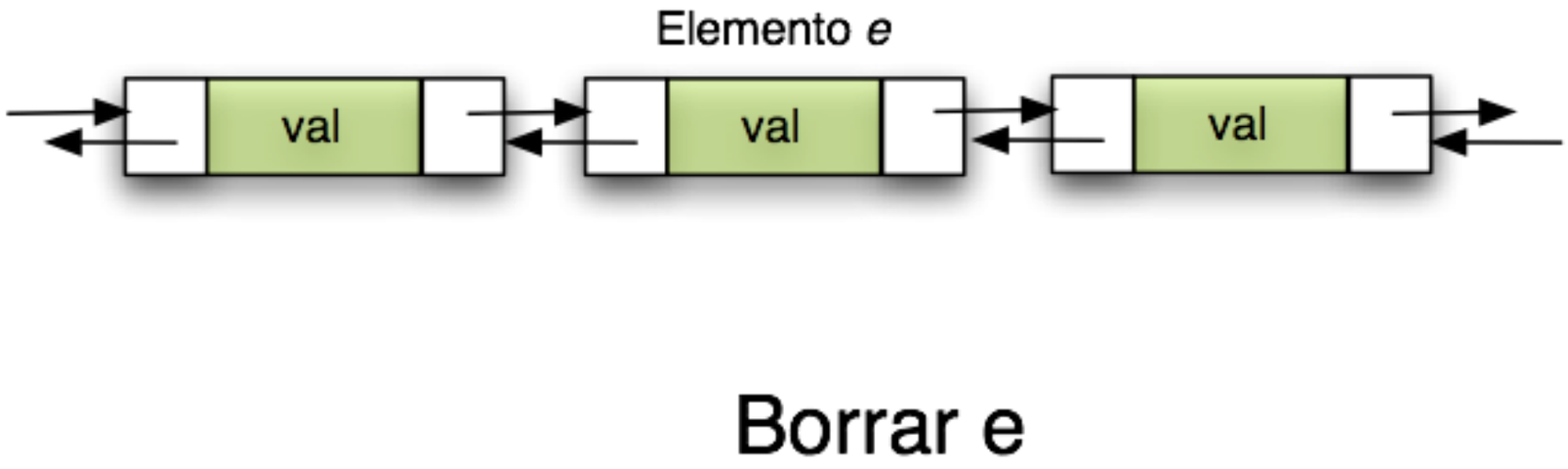
listas doblemente ligadas: remover nodos

- podemos retirar un nodo cuando la única información que tenemos es un **link** a este, por ejemplo ***e***.



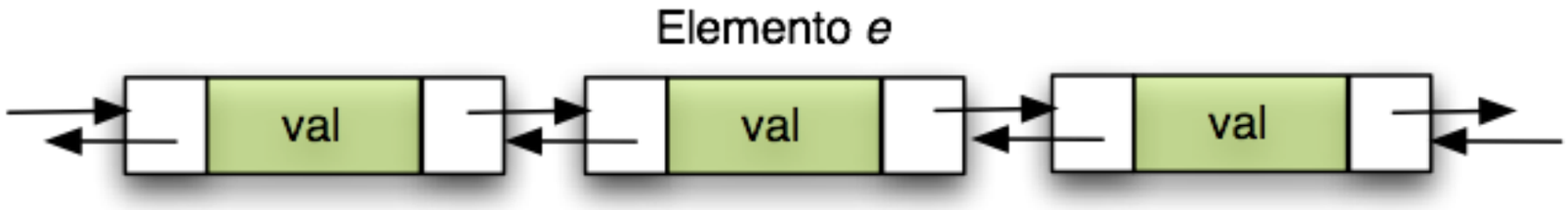
listas doblemente ligadas: remover nodos

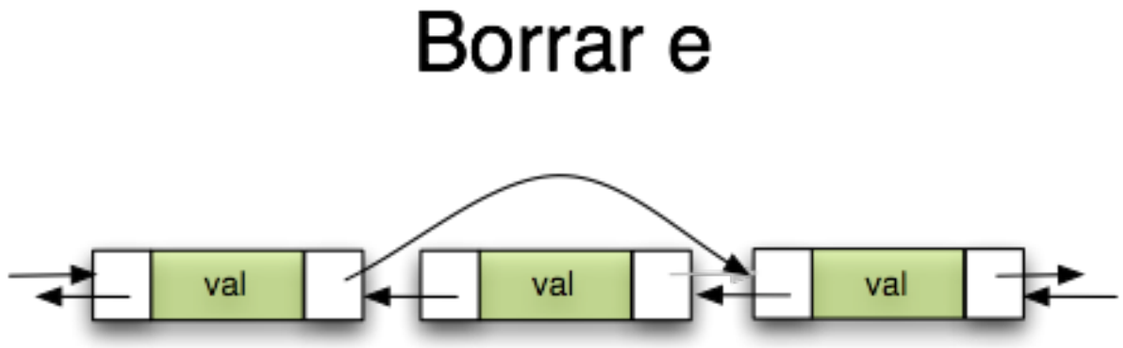
- podemos retirar un nodo cuando la única información que tenemos es un **link** a este, por ejemplo ***e***.



listas doblemente ligadas: remover nodos

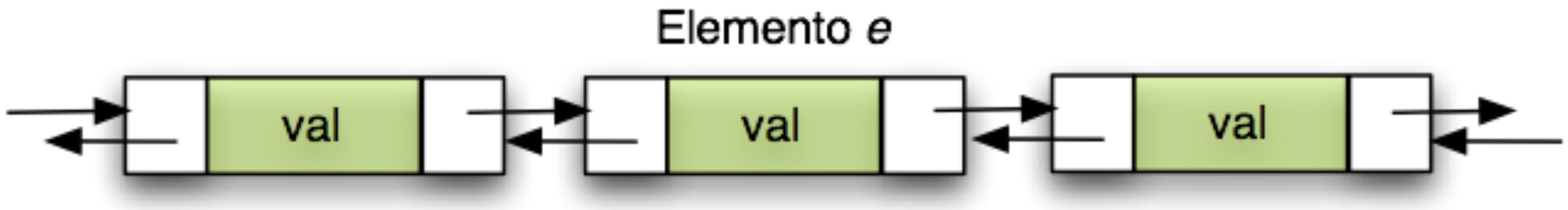
- podemos retirar un nodo cuando la única información que tenemos es un **link** a este, por ejemplo ***e***.

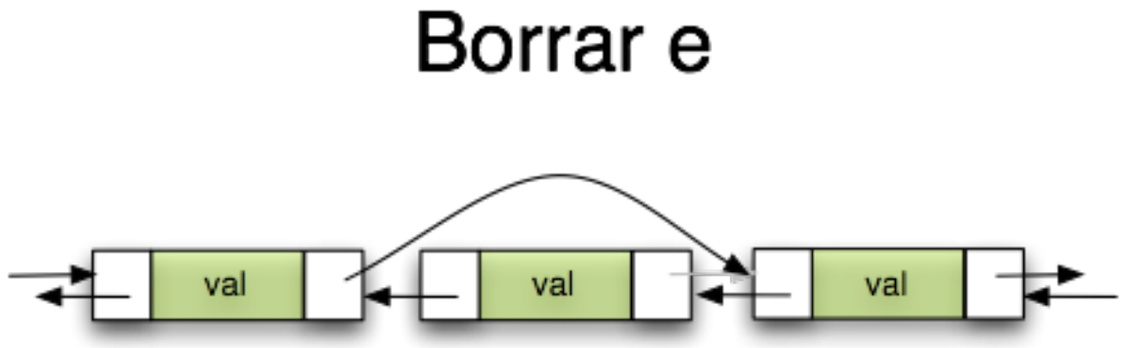




listas doblemente ligadas: remover nodos

- podemos retirar un nodo cuando la única información que tenemos es un **link** a este, por ejemplo ***e***.

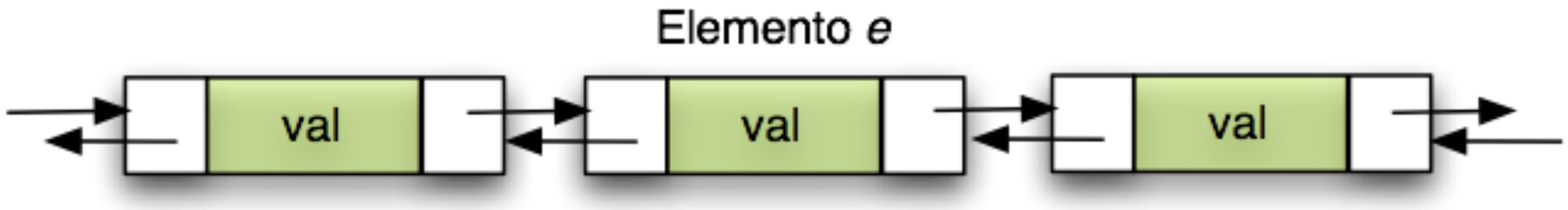




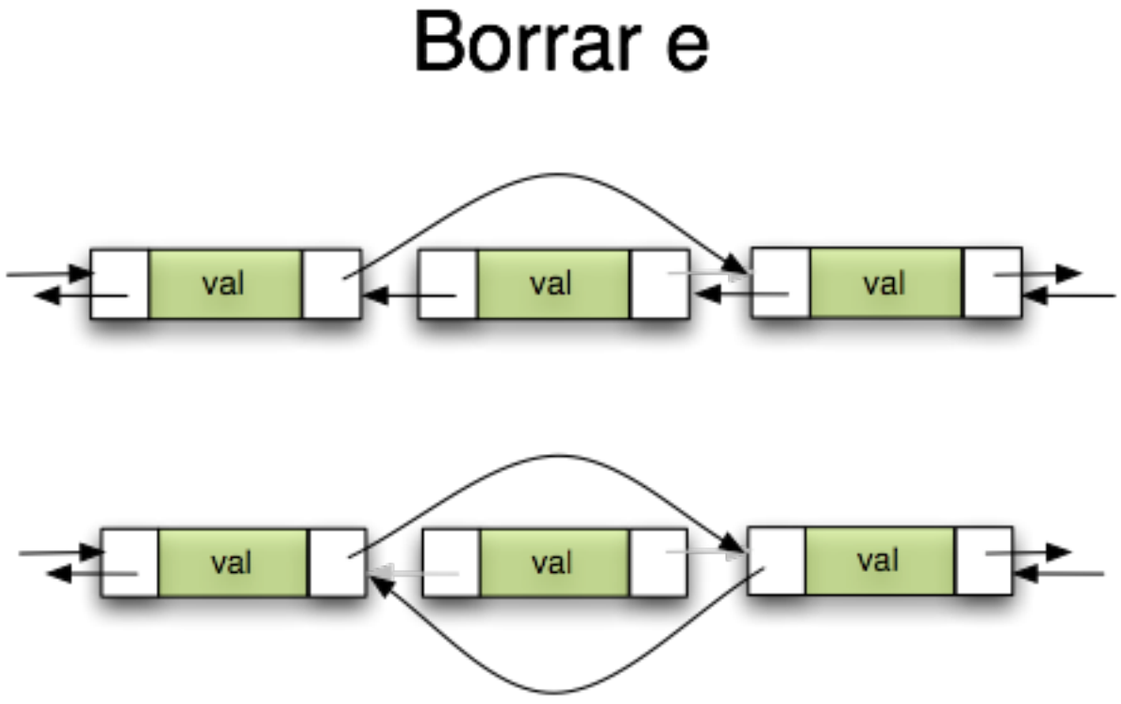

```
e->prev->next = e->next;
```

listas doblemente ligadas: remover nodos

- podemos retirar un nodo cuando la única información que tenemos es un **link** a este, por ejemplo ***e***.

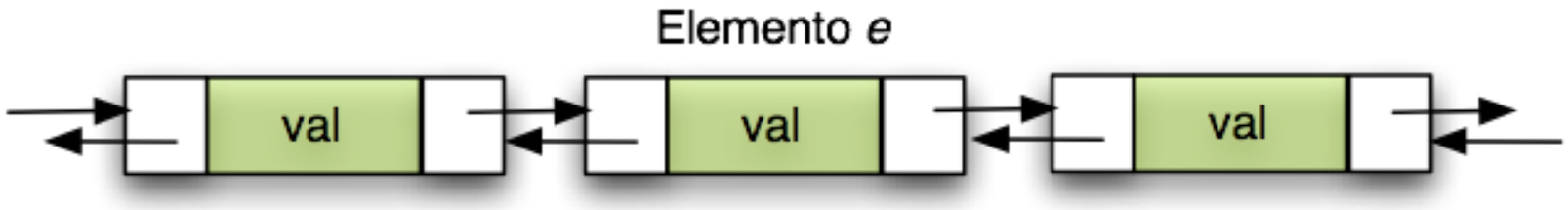



```
e->prev->next = e->next;
```

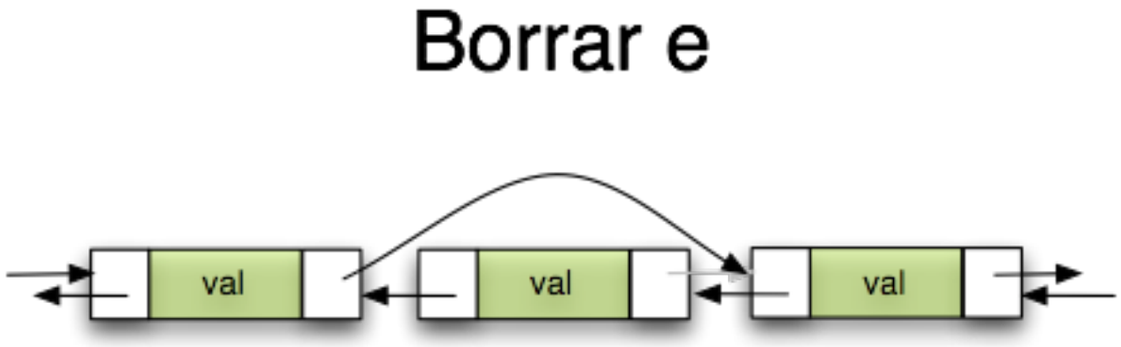


listas doblemente ligadas: remover nodos

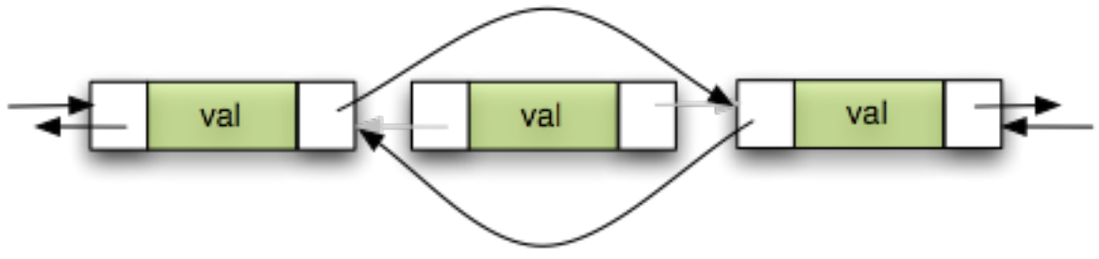
- podemos retirar un nodo cuando la única información que tenemos es un **link** a este, por ejemplo ***e***.




```
e->prev->next = e->next;
```



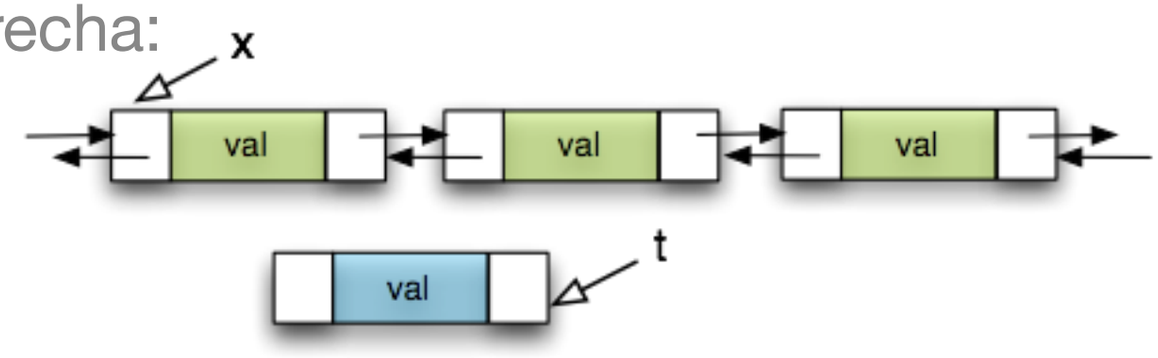

```
e->next->prev = e->prev;
```



listas doblemente ligadas: insertar nodos

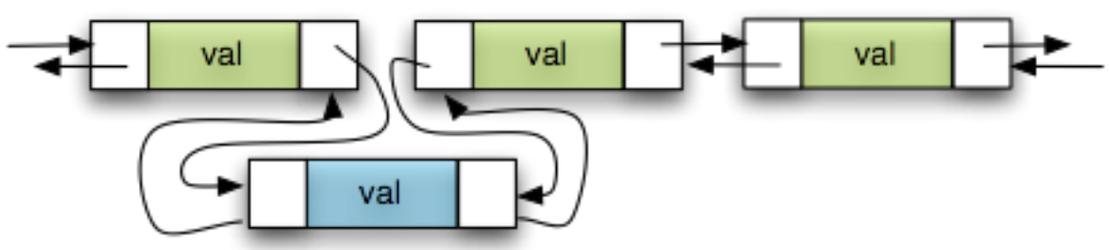
- insertar el nodo t después de x, parte derecha:

```
t->next = x->next;
x->next->prev = t;
```



- parte izquierda

```
x->next = t;
t->prev = x;
```



Listas ligadas

- Las listas se pueden implementar en la mayoría de los lenguajes de programación. Algunos lenguajes, como **LISP, Scheme** o **Java** tienen esta estructura pre-definida, así como las operaciones para acceder a ella. Otros lenguajes procedurales u orientados a objetos como **C, C++** cuentan con referencias o punteros para crearlas.

- la memoria de los nodos no es continua (necesariamente)

listas ligadas vs. listas doblemente ligadas

listas ligadas vs. listas doblemente ligadas

- un elemento de una lista doblemente ligada se puede borrar sin necesidad de decir “borra el nodo que está después de x”.

listas ligadas vs. listas doblemente ligadas

- un elemento de una lista doblemente ligada se puede borrar sin necesidad de decir “borra el nodo que está después de x”.

- una lista doblemente ligada se puede recorrer en los dos sentidos.

listas ligadas vs. listas doblemente ligadas

- un elemento de una lista doblemente ligada se puede borrar sin necesidad de decir “borra el nodo que está después de x”.
- una lista doblemente ligada se puede recorrer en los dos sentidos.
- las listas doblemente ligadas toman el doble de espacio para los links y

listas ligadas vs. listas doblemente ligadas

- un elemento de una lista doblemente ligada se puede borrar sin necesidad de decir “borra el nodo que está después de x”.

- una lista doblemente ligada se puede recorrer en los dos sentidos.

- las listas doblemente ligadas toman el doble de espacio para los links y

- las listas doblemente ligadas manipulan el doble de links por operación básica.

Tipos de datos abstractos (ADTs)

Tipos de datos abstractos (ADTs)

- Todos los sistemas computacionales están basados en niveles de abstracción:

Tipos de datos abstractos (ADTs)

- Todos los sistemas computacionales están basados en niveles de abstracción:
    - modelo de bit como valores 0-1 y no como propiedades físicas del silicio y otros materiales.

Tipos de datos abstractos (ADTs)

- Todos los sistemas computacionales están basados en niveles de abstracción:
    - modelo de bit como valores 0-1 y no como propiedades físicas del silicio y otros materiales.
    - modelo abstracto de máquina y no las propiedades dinámicas de un conjunto de bits.

Tipos de datos abstractos (ADTs)

- Todos los sistemas computacionales están basados en niveles de abstracción:
    - modelo de bit como valores 0-1 y no como propiedades físicas del silicio y otros materiales.
    - modelo abstracto de máquina y no las propiedades dinámicas de un conjunto de bits.
    - lenguaje de máquina y no control directo sobre la máquina.

Tipos de datos abstractos (ADTs)

- Todos los sistemas computacionales están basados en niveles de abstracción:
    - modelo de bit como valores 0-1 y no como propiedades físicas del silicio y otros materiales.
    - modelo abstracto de máquina y no las propiedades dinámicas de un conjunto de bits.
    - lenguaje de máquina y no control directo sobre la máquina.
    - lenguaje de alto nivel y no lenguaje de máquina.

Tipos de datos abstractos (ADTs)

- Todos los sistemas computacionales están basados en niveles de abstracción:
    - modelo de bit como valores 0-1 y no como propiedades físicas del silicio y otros materiales.
    - modelo abstracto de máquina y no las propiedades dinámicas de un conjunto de bits.
    - lenguaje de máquina y no control directo sobre la máquina.
    - lenguaje de alto nivel y no lenguaje de máquina.
    - tipos de datos abstractos.

Tipos de datos abstractos (ADTs)

- Todos los sistemas computacionales están basados en niveles de abstracción:
    - modelo de bit como valores 0-1 y no como propiedades físicas del silicio y otros materiales.
    - modelo abstracto de máquina y no las propiedades dinámicas de un conjunto de bits.
    - lenguaje de máquina y no control directo sobre la máquina.
    - lenguaje de alto nivel y no lenguaje de máquina.
    - tipos de datos abstractos.
    - Y arriba de todo, abstracción de un problema de selección de personas.