Estructuras de datos lineales

Listas

M.S.C. Jacob Green • 09-11-2020

Lista

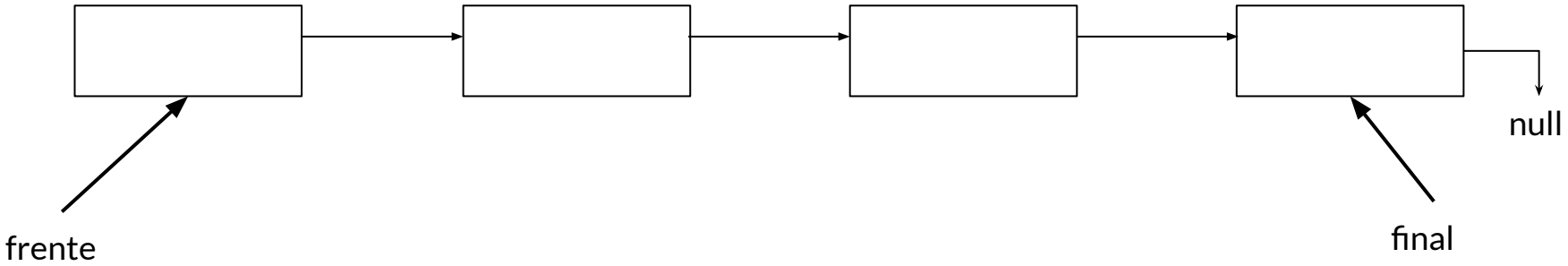
- Es una estructura de datos dinámica que almacena una colección de elementos del mismo tipo con un orden establecido entre ellos.
- Existen diferentes tipos de listas:
  - Listas simplemente enlazadas.
  - Listas doblemente enlazadas.
  - Listas circulares.

Definiendo el TDA Lista

Datos	Operaciones	Función	Operaciones	Función
Nodo *frente Nodo *final	void push_front(T d)	Inserta el elemento **d** al frente de la lista	void push(T d, int i)	Inserta el elemento **d** en la posición i-ésima
	void push_back(T d)	Inserta el elemento **d** al final de la lista		
	void pop_front( )	Elimina el frente de la lista	void pop(int i)	Elimina el nodo en la posición i-ésima
	void pop_back( )	Elimina el final de la lista		
	T front( )	Devuelve el frente de la lista	T get(int i)	Devuelve el dato en la posición i-ésima
	T back( )	Devuelve el final de la lista		
	void clear( )	Elimina todos los nodos de la lista	void set(T d, int i);	Actualizar el valor en la posición i-ésima, utilizando **d**.
int tam	int size( )	Devuelve el tamaño de la lista		
	bool empty( )	Verifica si la lista está vacía		

Lista simplemente enlazada

- Cada nodo apunta al siguiente nodo.
- Guardamos el primer nodo de la lista, llamado **frente**.
- Adicionalmente podemos mantener un apuntador al último nodo, llamado **final**.



Lista simplemente enlazada

- void push(T d);
- Inserción en lista vacía.

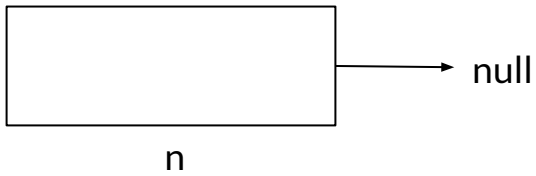
frente=null final=null

Lista simplemente enlazada

- void push(T d);
- Inserción en lista vacía.

Nodo *n = new nodo(d);

frente=null final=null



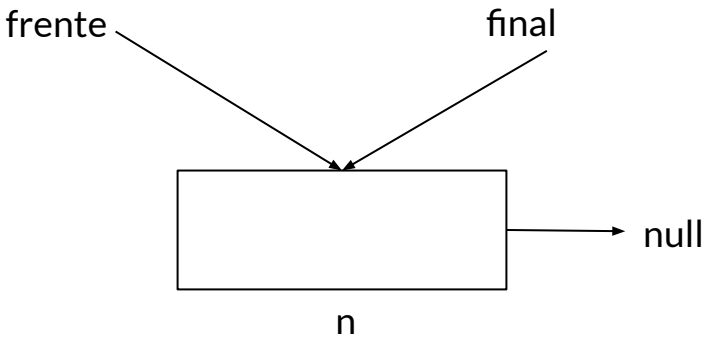
Lista simplemente enlazada

- void push(T d);
- Inserción en lista vacía.

Nodo *n = new nodo(d);

frente = n;

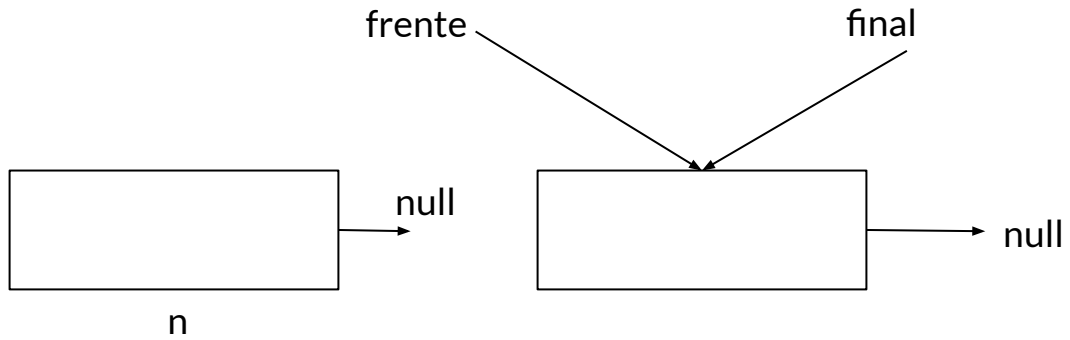
final = n;



Lista simplemente enlazada

- void push_front(T d);
- Inserción al frente.

Nodo *n = new nodo(d);

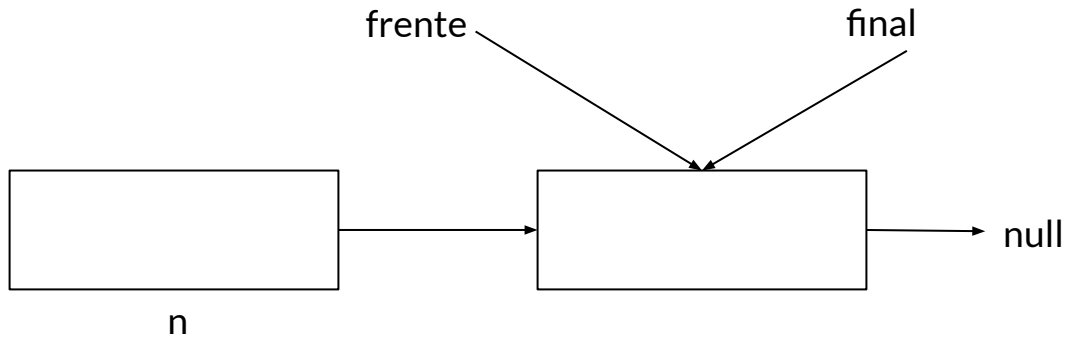


Lista simplemente enlazada

- void push_front(T d);
- Inserción al frente.

Nodo *n = new nodo(d);

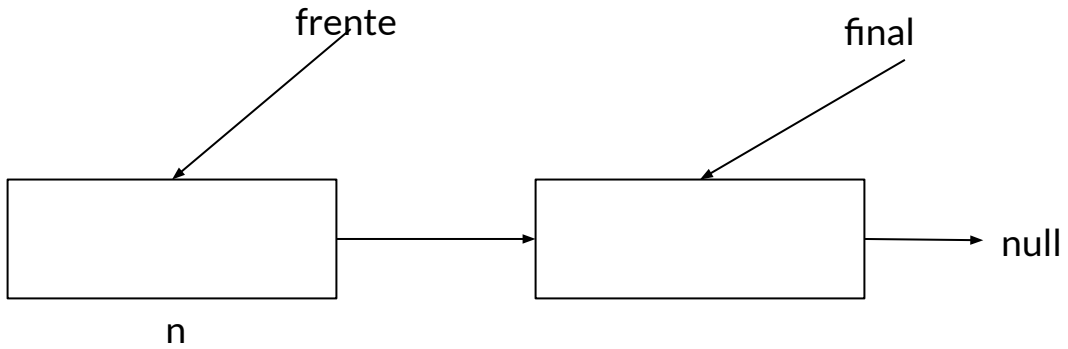
n->siguiente=frente;



Lista simplemente enlazada

- void push_front(T d);
- Inserción al frente.

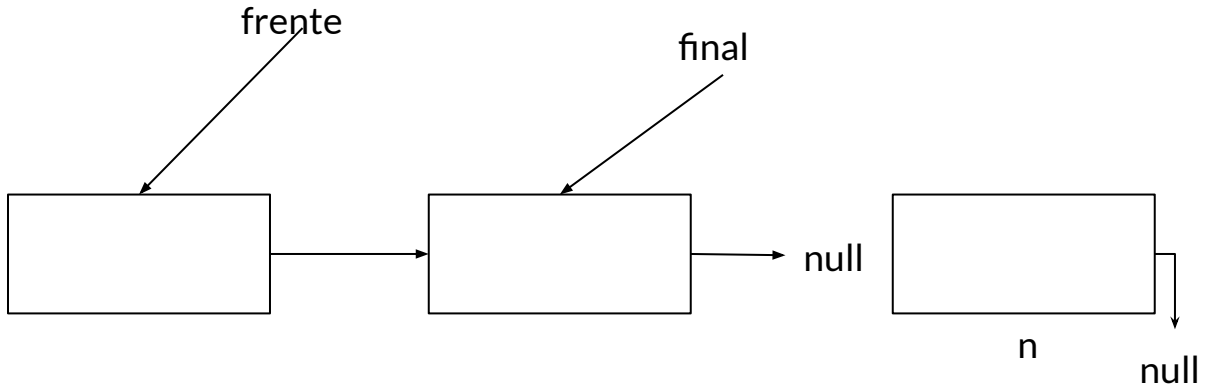
```
Nodo *n = new nodo(d);

n->siguiente=frente;

frente = n;
```



Lista simplemente enlazada

- void push_back(T d);
- Inserción al final.

Nodo *n = new nodo(d);

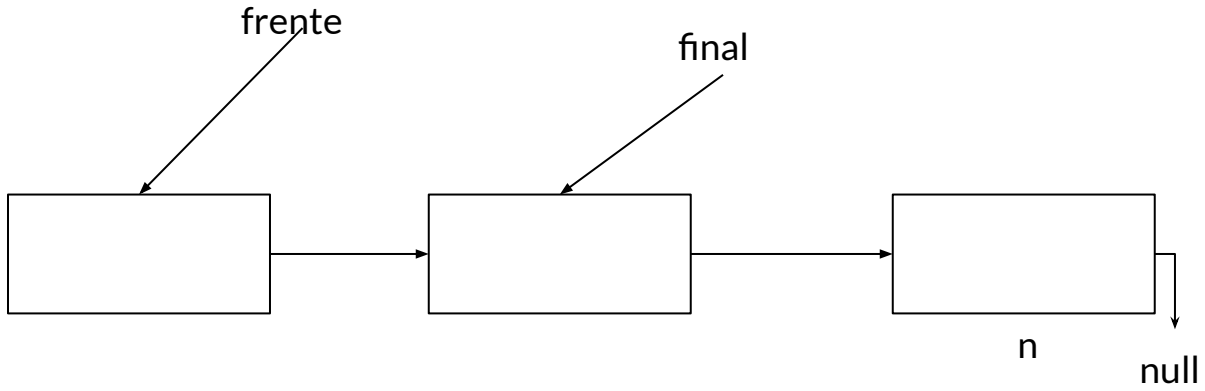


Lista simplemente enlazada

- void push_back(T d);
- Inserción al final.

Nodo *n = new nodo(d);

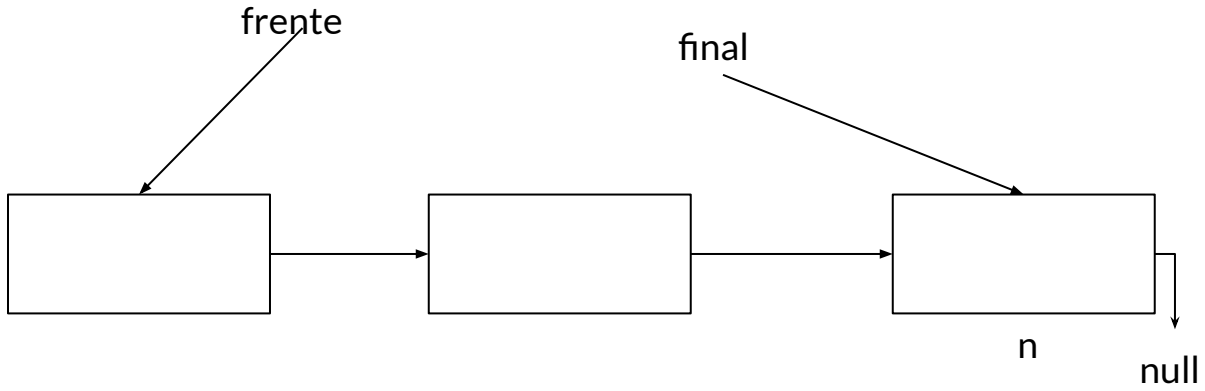
final->siguiente=n;



Lista simplemente enlazada

- void push_back(T d);
- Inserción al final.

```
Nodo *n = new nodo(d);

final->siguiente=n;

final = n;
```

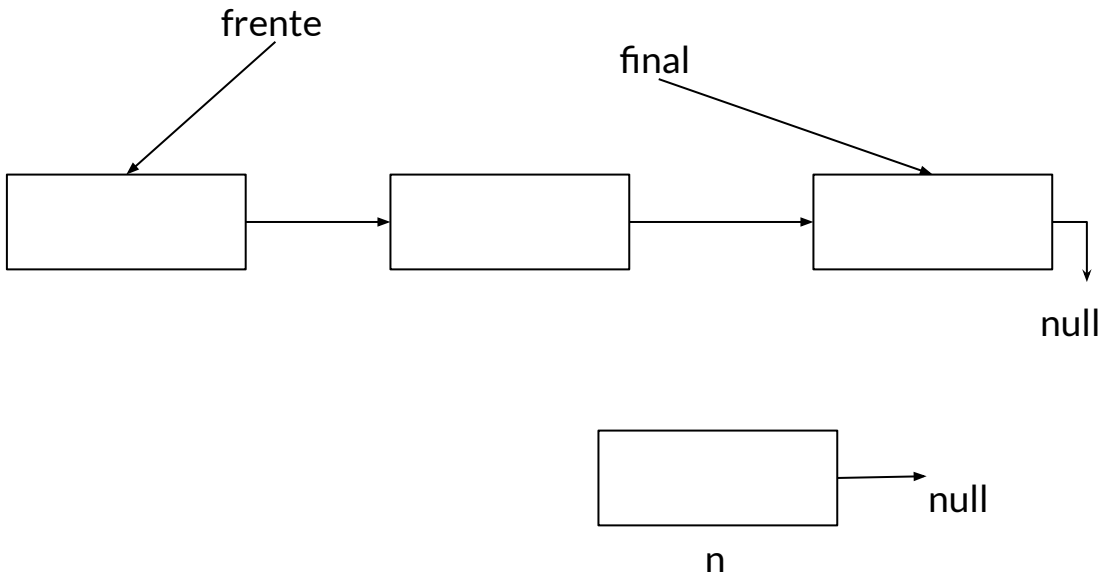


Lista simplemente enlazada

- void push(T d, int i);
- Inserción en la posición i-ésima.

int i = 2;

Nodo *n = new nodo(d);

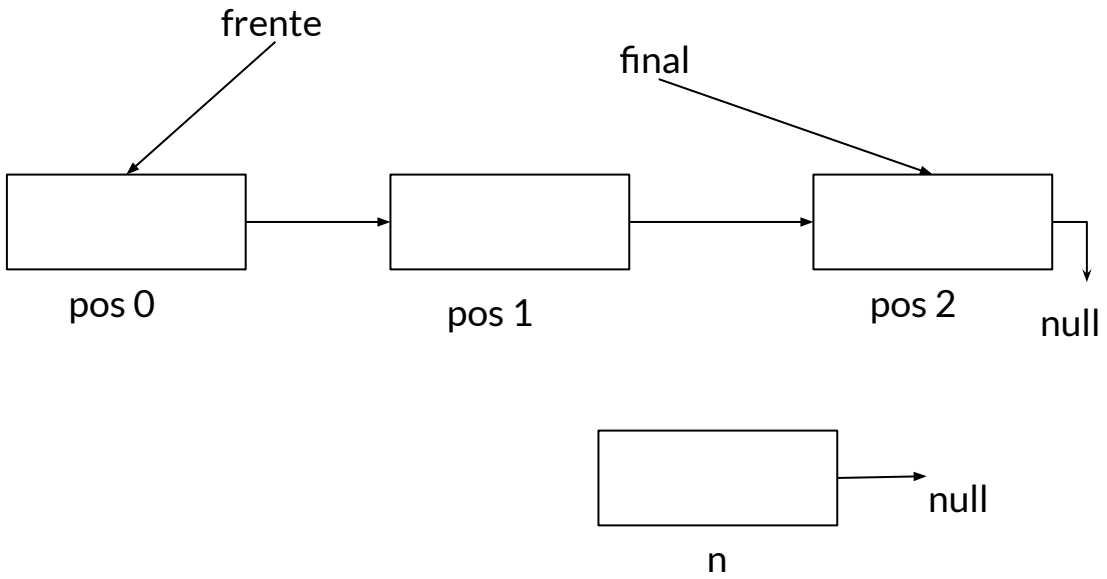


Lista simplemente enlazada

- void push(T d, int i);
- Inserción en la posición i-ésima.

int i = 2;

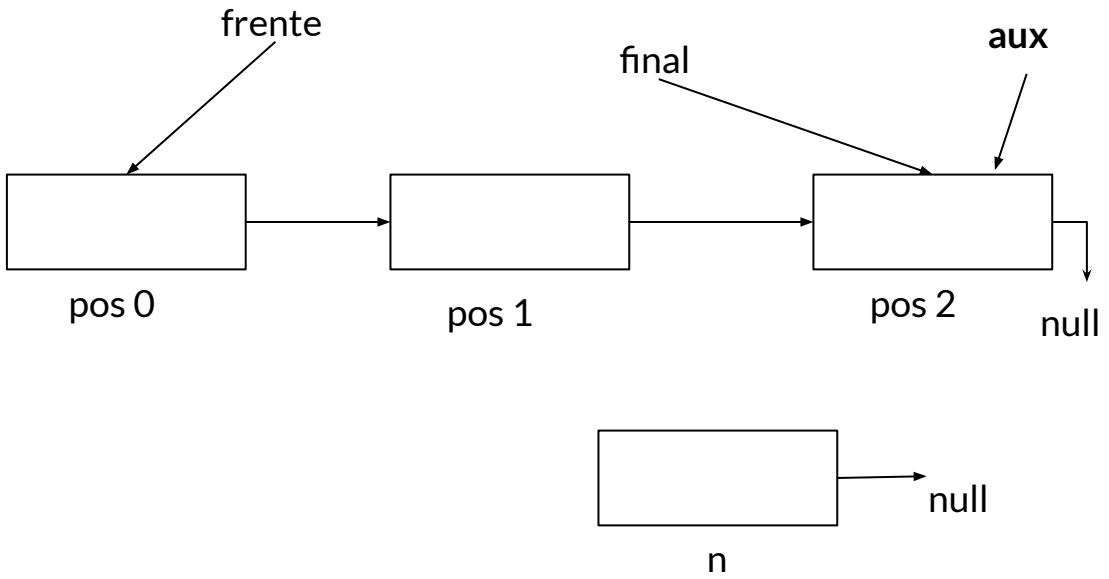
Nodo *n = new nodo(d);



Lista simplemente enlazada

- void push(T d, int i);
- Inserción en la posición i-ésima.

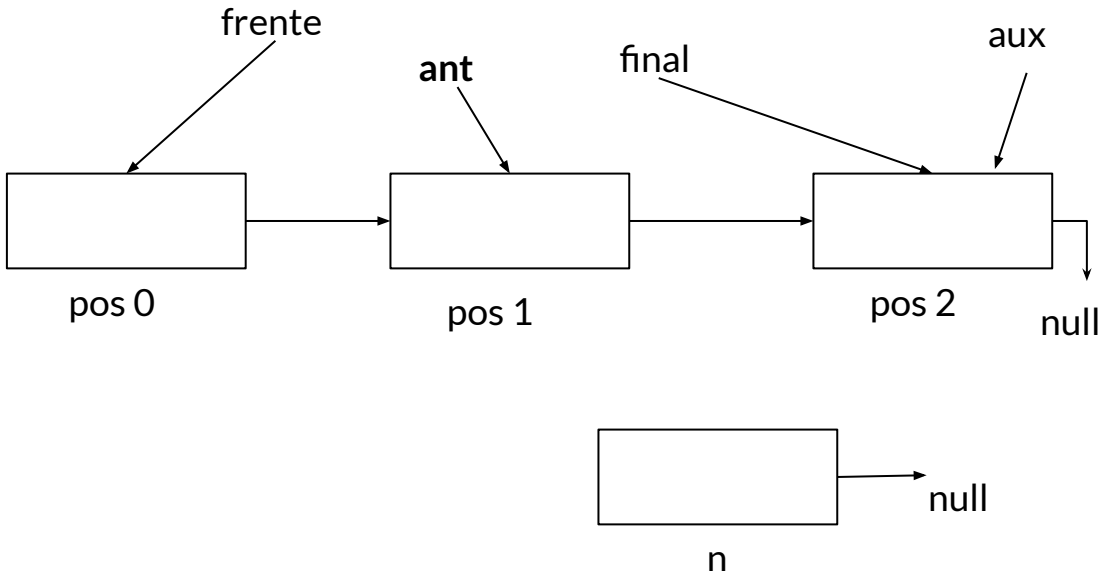
int i = 2;
Nodo *n = new nodo(d);
Nodo ***aux*** = nodo en la posición i;



Lista simplemente enlazada

- void push(T d, int i);
- Inserción en la posición i-ésima.

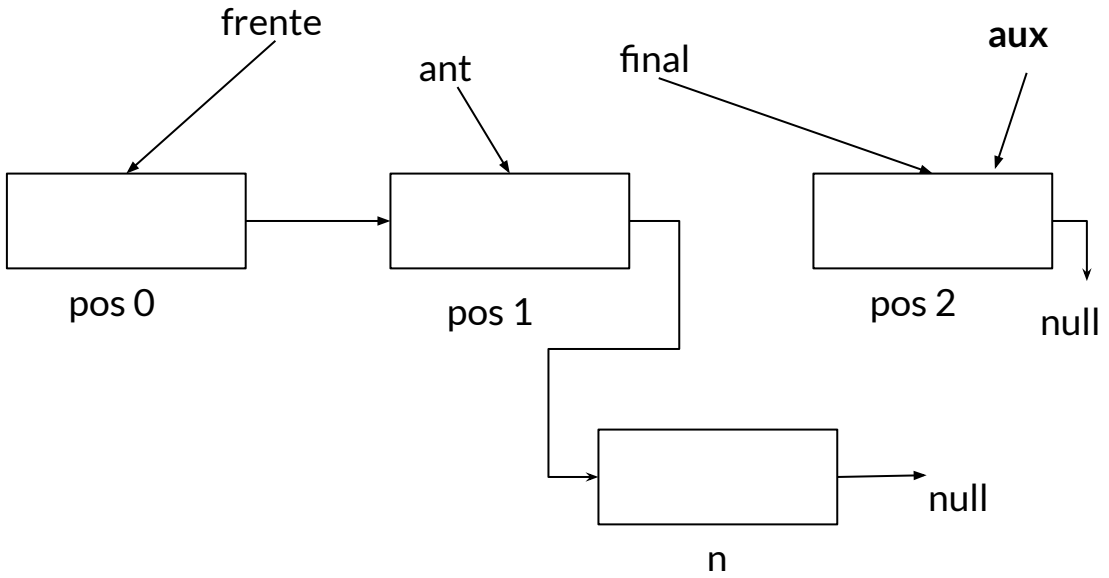
int i = 2;
Nodo *n = new nodo(d);
Nodo *aux = nodo en la posición i;
Nodo ***ant*** = nodo en la posición anterior a aux;



Lista simplemente enlazada

- void push(T d, int i);
- Inserción en la posición i-ésima.

```
int i = 2;
Nodo *n = new nodo(d);
Nodo *aux = nodo en la posición i;
Nodo *ant = nodo en la posición
anterior a aux;

ant->siguiente = n;
```

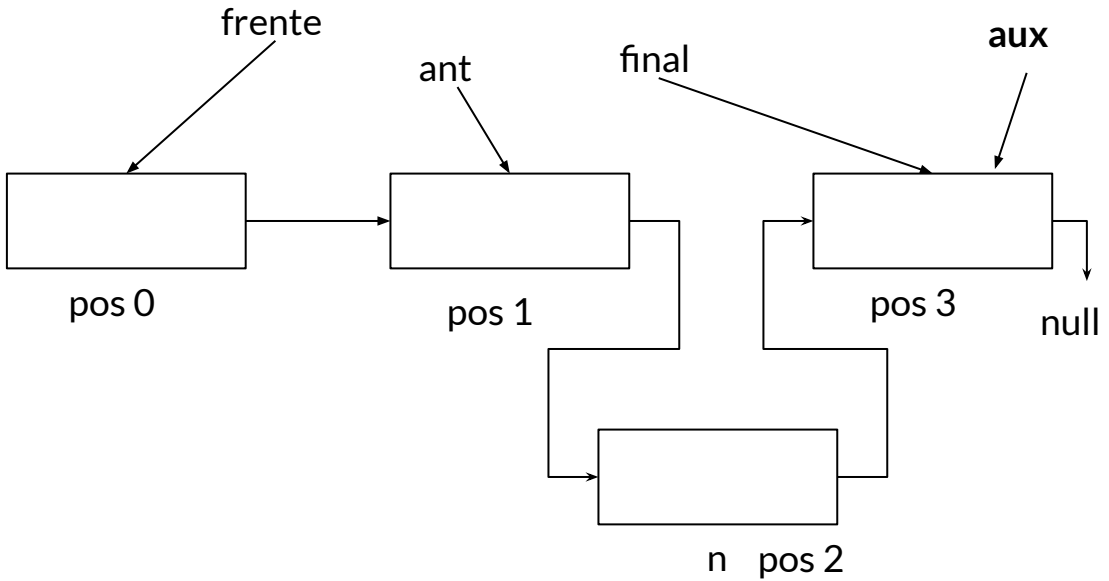


Lista simplemente enlazada

- void push(T d, int i);
- Inserción en la posición i-ésima.

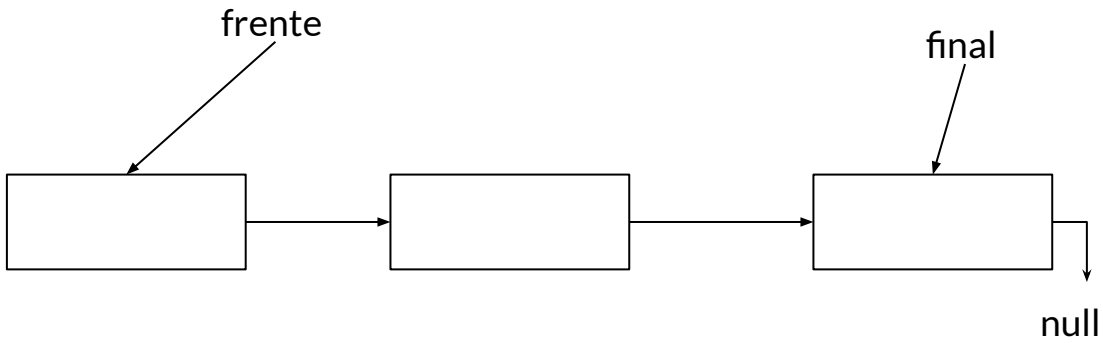
int i = 2;
Nodo *n = new nodo(d);
Nodo *aux = nodo en la posición i;
Nodo *ant = nodo en la posición anterior a aux;

ant->siguiente = n;
n->siguiente = aux;



Lista simplemente enlazada

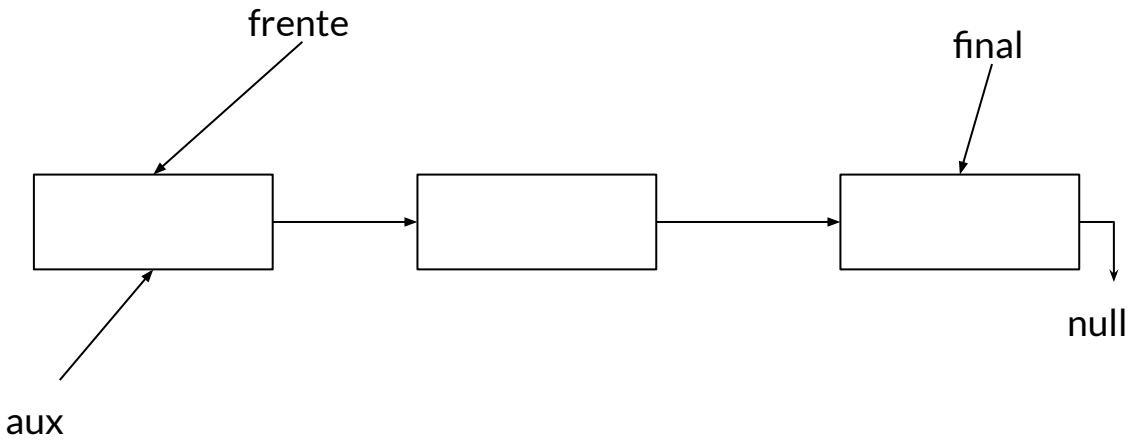
- void pop_front( );
- Eliminar el frente.



Lista simplemente enlazada

- void pop_front( );
- Eliminar el frente.

Nodo *aux = frente;

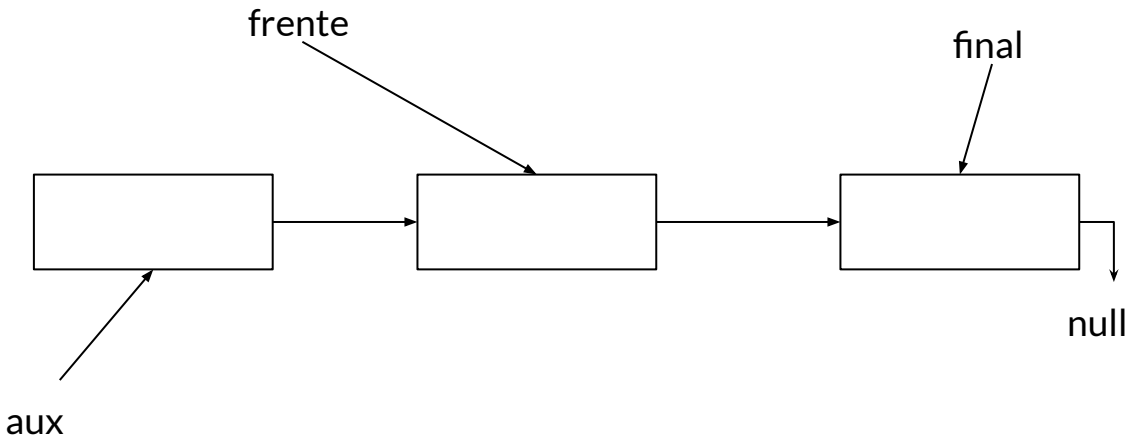


Lista simplemente enlazada

- void pop_front( );
- Eliminar el frente.

Nodo *aux = frente;

frente = frente->siguiente;



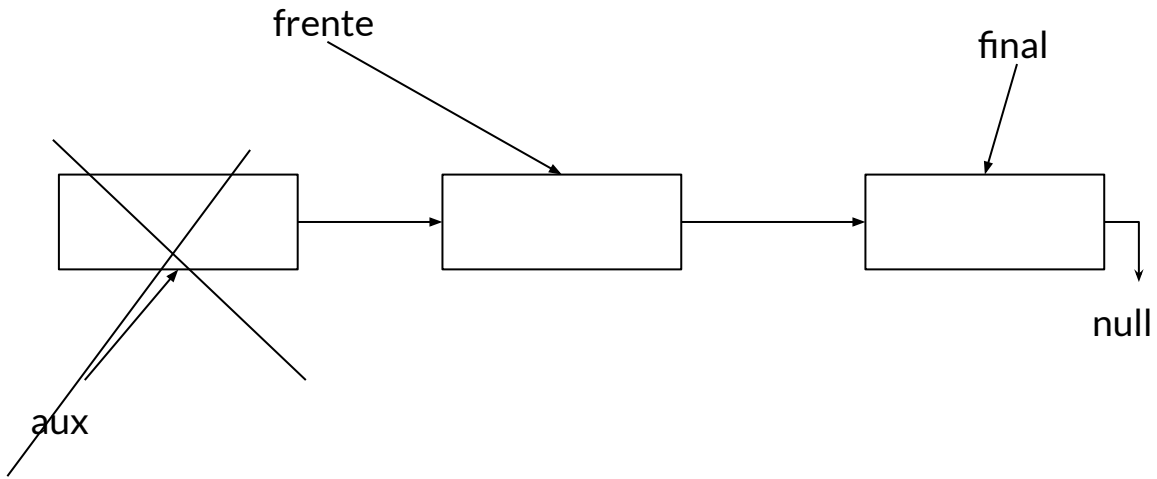
Lista simplemente enlazada

- void pop_front( );
- Eliminar el frente.

Nodo *aux = frente;

frente = frente->siguiente;

delete aux;



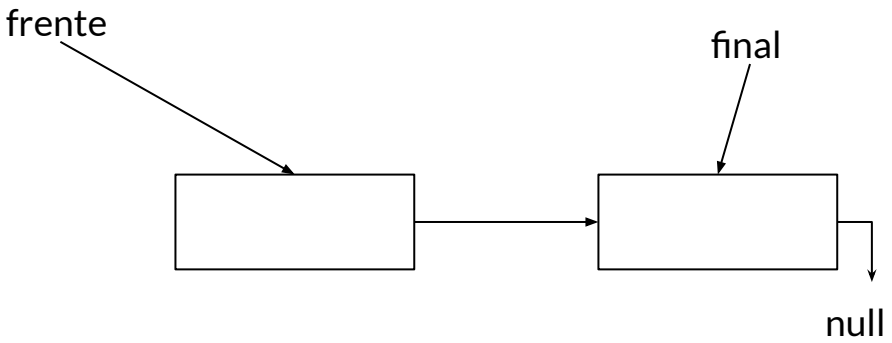
Lista simplemente enlazada

- void pop_front( );
- Eliminar el frente.

Nodo *aux = frente;

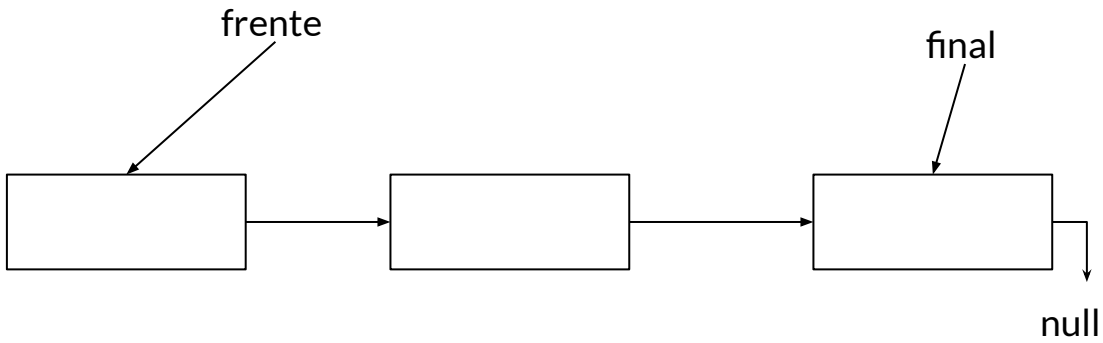
frente = frente->siguiente;

delete aux;



Lista simplemente enlazada

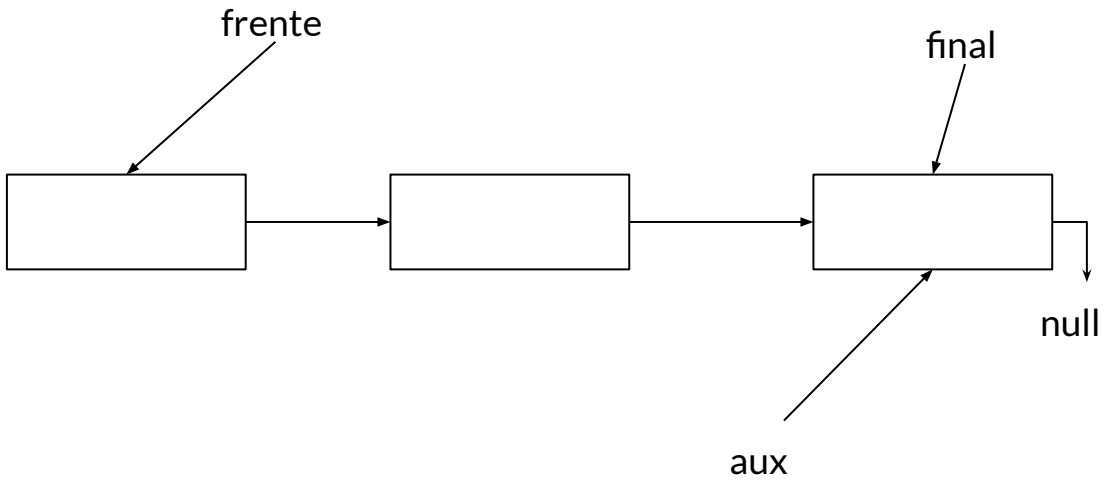
- void pop_back( );
- Eliminar el final.



Lista simplemente enlazada

- void pop_back( );
- Eliminar el final.

Nodo *aux = final;

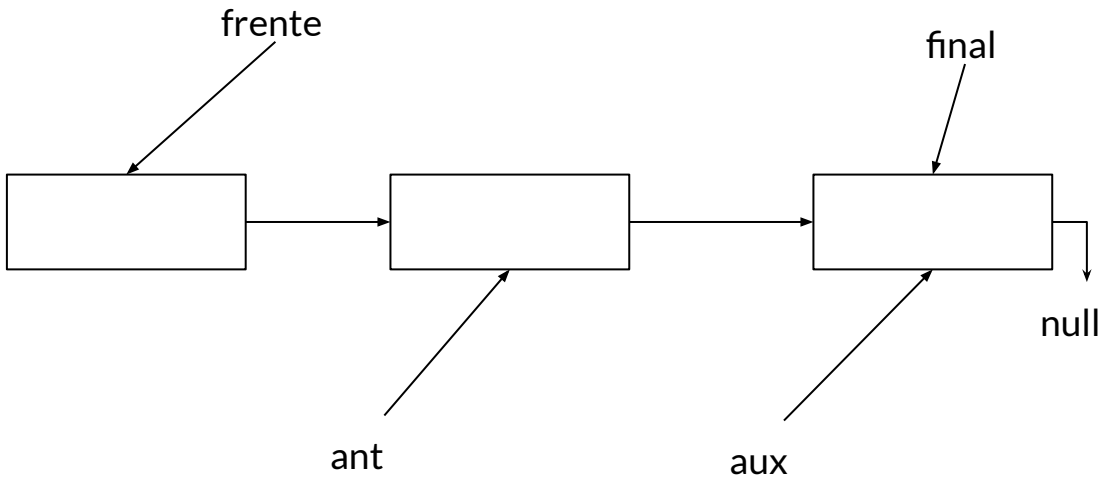


Lista simplemente enlazada

- void pop_back( );
- Eliminar el final.

Nodo *aux = final;

Nodo *ant = nodo anterior a aux;



Lista simplemente enlazada

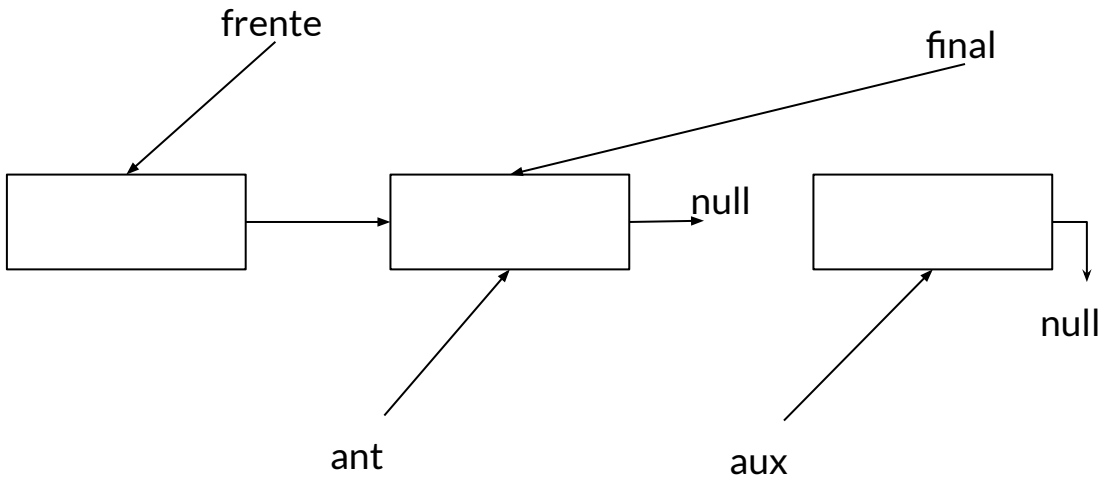
- void pop_back( );
- Eliminar el final.

Nodo *aux = final;

Nodo ant = nodo anterior a aux;

final = ant;

final->siguiente = null;



Lista simplemente enlazada

- void pop_back( );
- Eliminar el final.

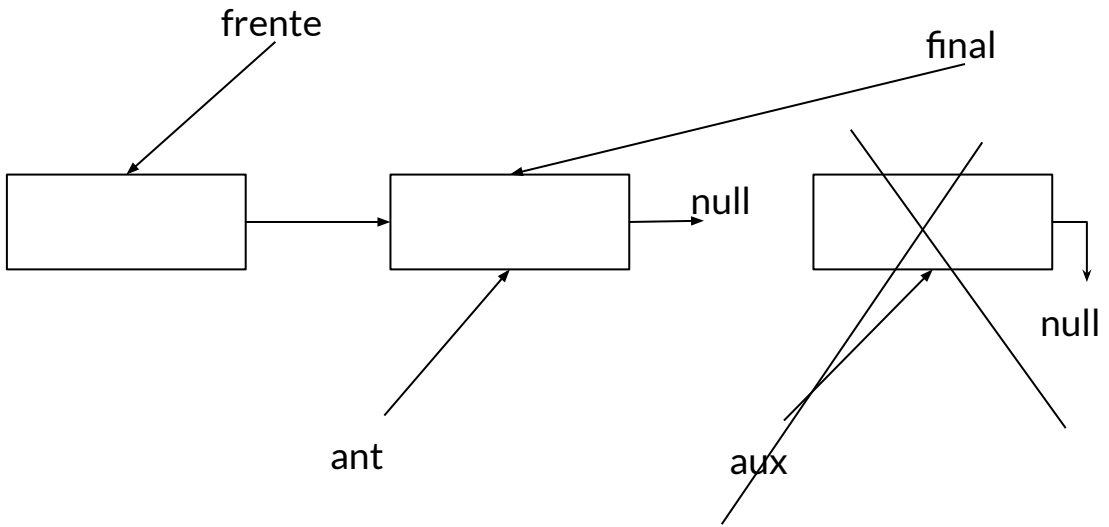
Nodo *aux = final;

Nodo ant = nodo anterior a aux;

final = ant;

final->siguiente = null;

delete aux;



Lista simplemente enlazada

- void pop_back( );
- Eliminar el final.

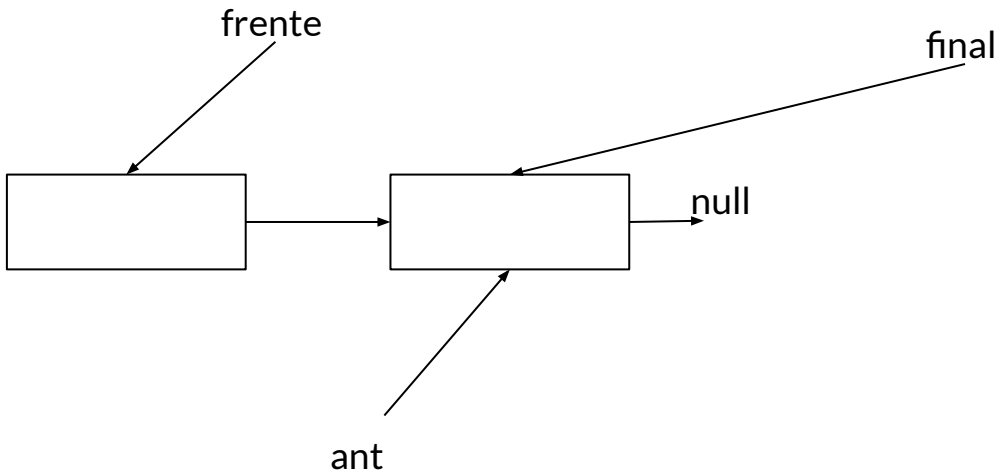
Nodo *aux = final;

Nodo *ant = nodo anterior a aux;

final = ant;

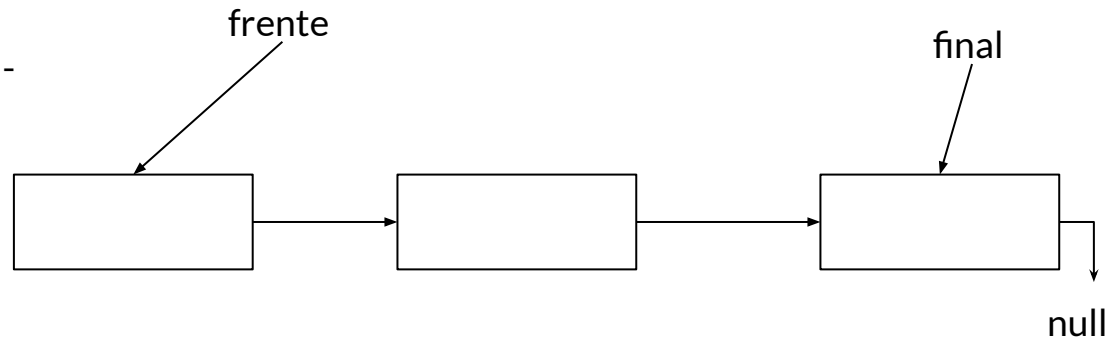
final->siguiente = null;

delete aux;



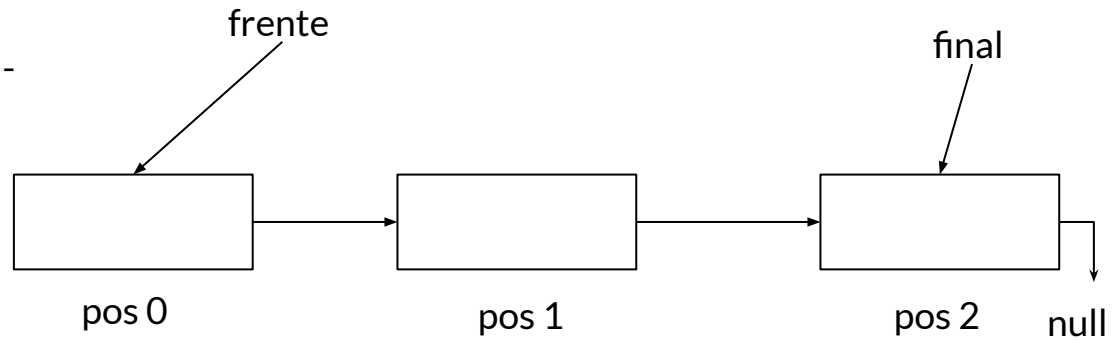
Lista simplemente enlazada

- void pop(int i);
- Eliminar el nodo en la posición i-ésima.



Lista simplemente enlazada

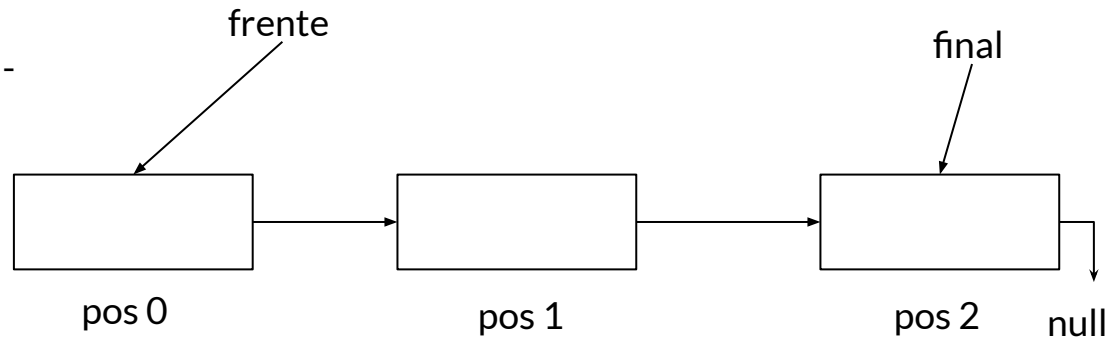
- void pop(int i);
- Eliminar el nodo en la posición i-ésima.



Lista simplemente enlazada

- void pop(int i);
- Eliminar el nodo en la posición i-ésima.

int i = 1;

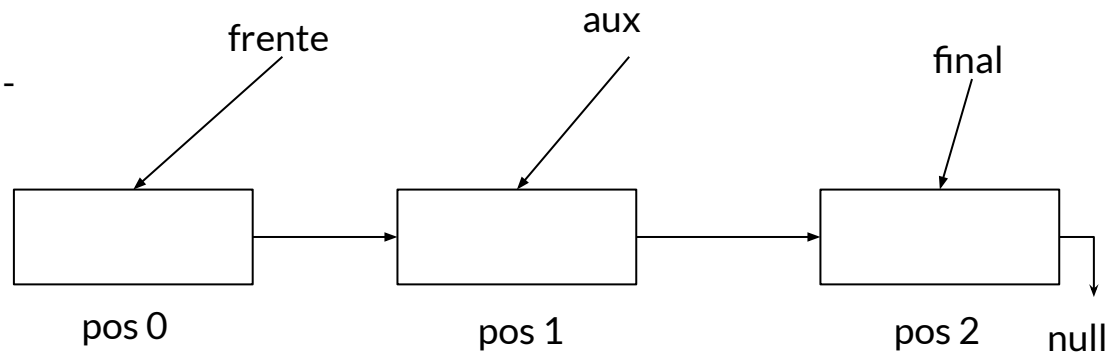


Lista simplemente enlazada

- void pop(int i);
- Eliminar el nodo en la posición i-ésima.

int i = 1;

Nodo *aux = nodo en la posición i;



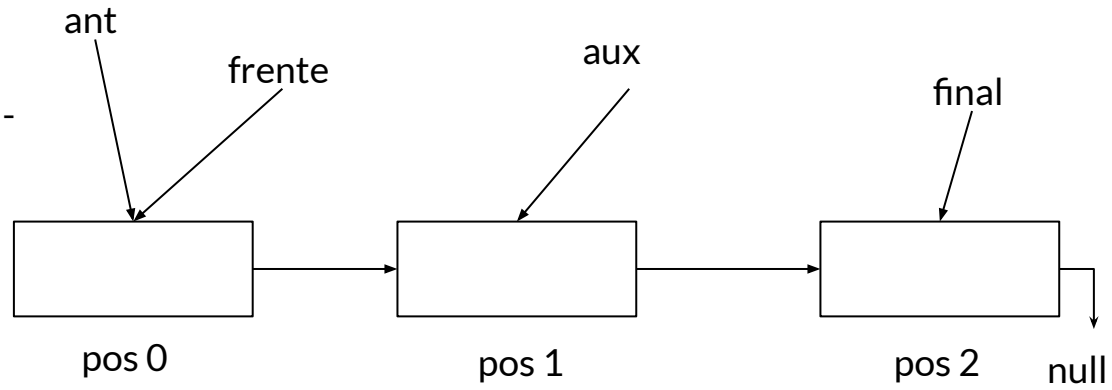
Lista simplemente enlazada

- void pop(int i);
- Eliminar el nodo en la posición i-ésima..

int i = 1;

Nodo *aux = nodo en la posición i;

Nodo *ant = nodo en la posición anterior a aux;



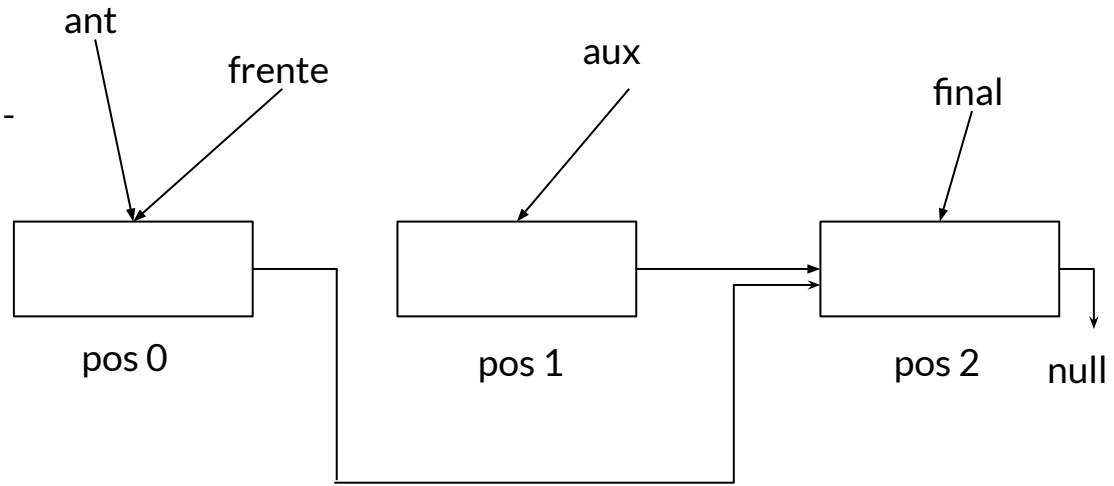
Lista simplemente enlazada

- void pop(int i);
- Eliminar el nodo en la posición i-ésima..

int i = 1;

Nodo *aux = nodo en la posición i;
Nodo *ant = nodo en la posición anterior a aux;

ant->siguiente = aux->siguiente;



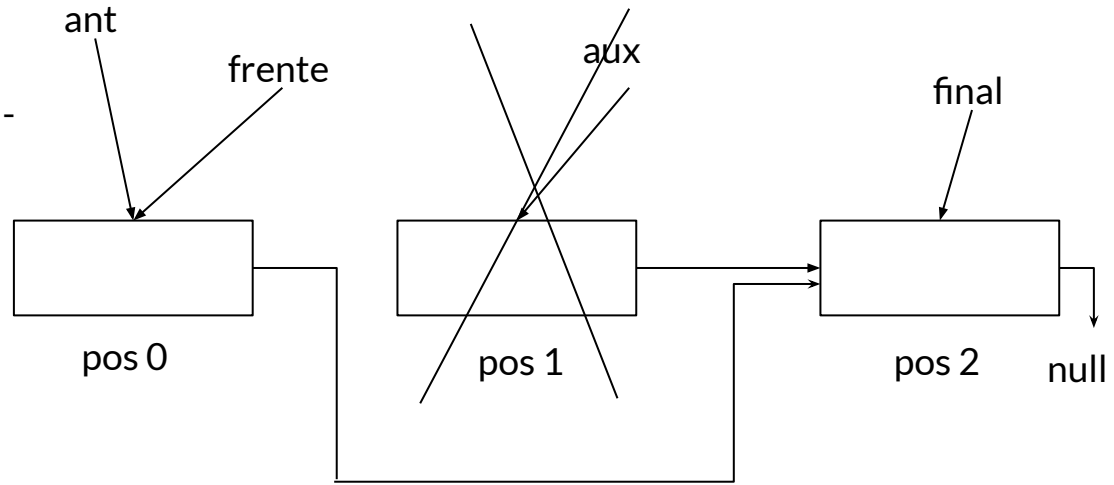
Lista simplemente enlazada

- void pop(int i);
- Eliminar el nodo en la posición i-ésima..

int i = 1;

Nodo *aux = nodo en la posición i;
Nodo *ant = nodo en la posición anterior a aux;
ant->siguiente = aux->siguiente;

delete aux;



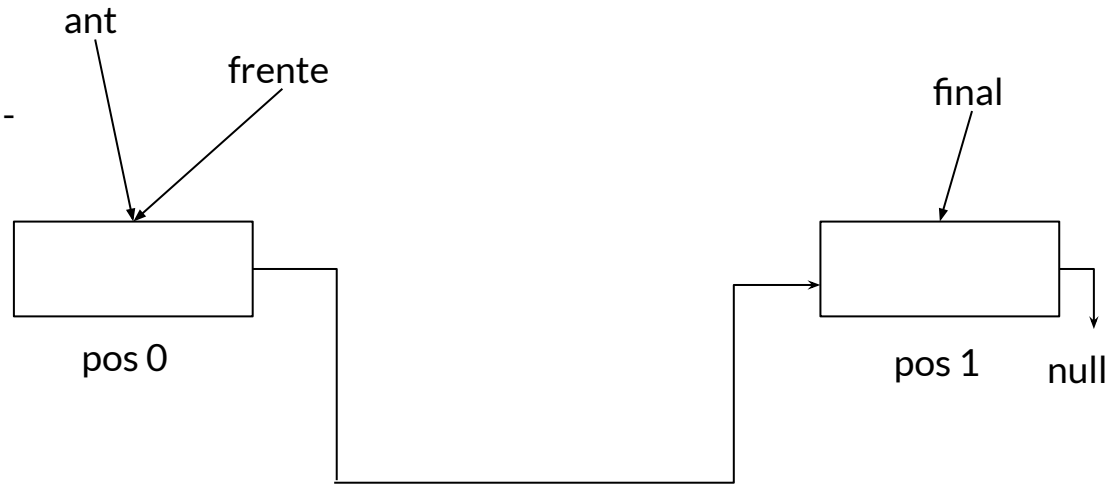
Lista simplemente enlazada

- void pop(int i);
- Eliminar el nodo en la posición i-ésima..

int i = 1;

Nodo *aux = nodo en la posición i;
Nodo *ant = nodo en la posición anterior a aux;
ant->siguiente = aux->siguiente;

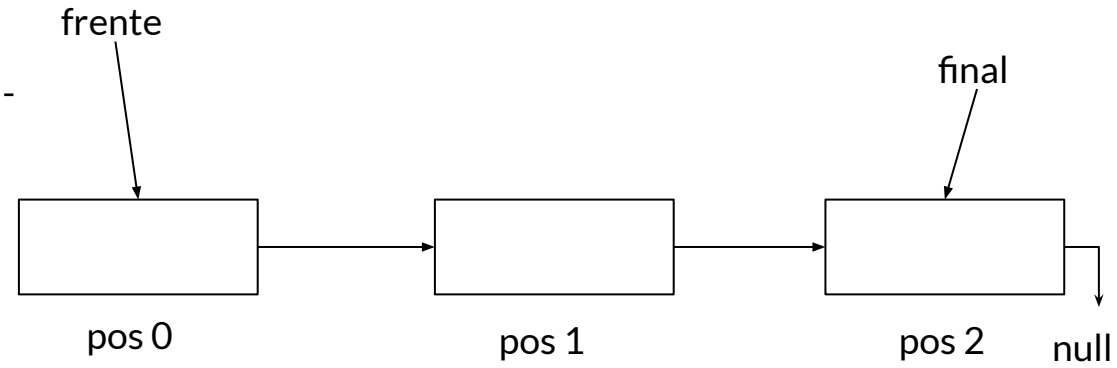
delete aux;



Lista simplemente enlazada

- T get(int i);
- Obtener el dato en la posición i-ésima.

int i = 1;

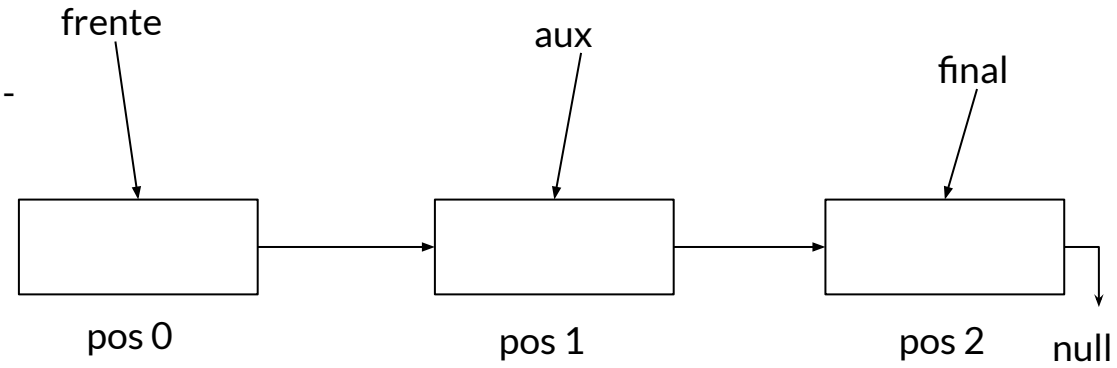


Lista simplemente enlazada

- T get(int i);
- Obtener el dato en la posición i-ésima.

int i = 1;

Nodo *aux = nodo en la posición i;



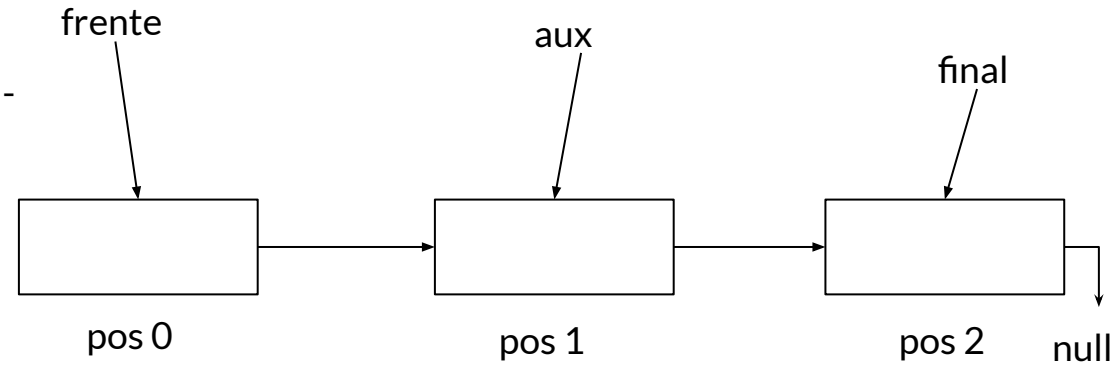
Lista simplemente enlazada

- T get(int i);
- Obtener el dato en la posición i-ésima.

int i = 1;

Nodo *aux = nodo en la posición i;

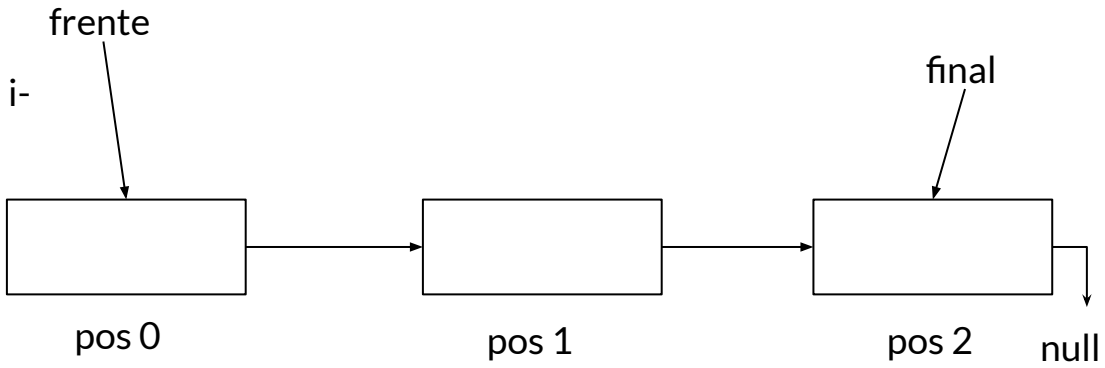
return aux->dato;



Lista simplemente enlazada

- void set(T d, int i);
- Actualizar el dato en la posición i-ésima.

int i = 1;

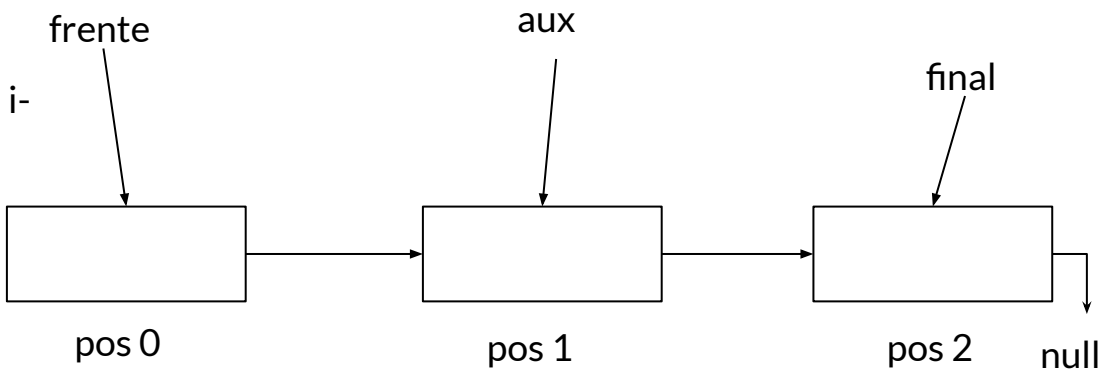


Lista simplemente enlazada

- void set(T d, int i);
- Actualizar el dato en la posición i-ésima.

int i = 1;

Nodo *aux = nodo en la posición i-ésima;



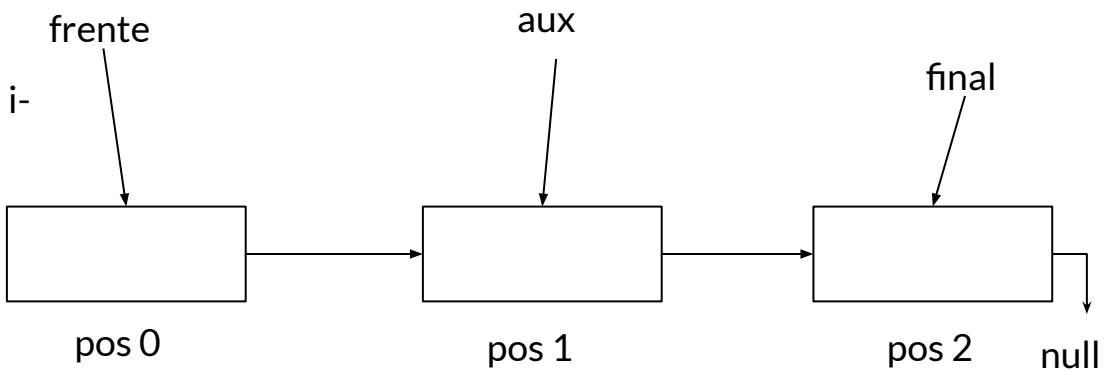
Lista simplemente enlazada

- void set(T d, int i);
- Actualizar el dato en la posición i-ésima.

int i = 1;

Nodo *aux = nodo en la posición i-ésima;

aux->dato = d;

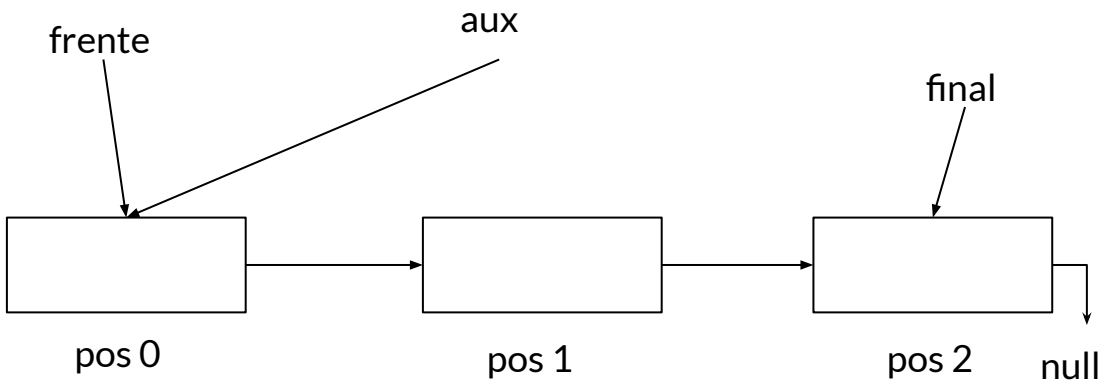


Lista simplemente enlazada

- Buscar el nodo en la posición i-ésima.

int i = 1;

Nodo *aux = frente;



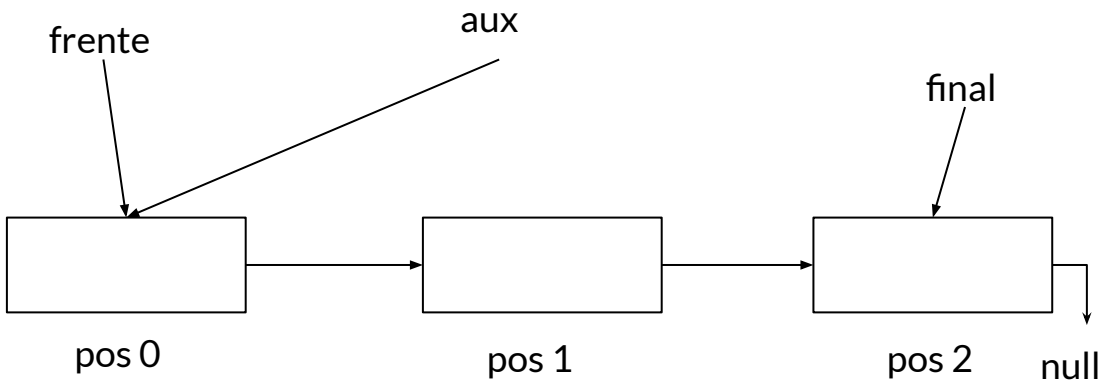
Lista simplemente enlazada

- Buscar el nodo en la posición i-ésima.

int i = 1;

Nodo *aux = frente;

Hacemos aux = aux->siguiente hasta que lleguemos a la posición buscada.



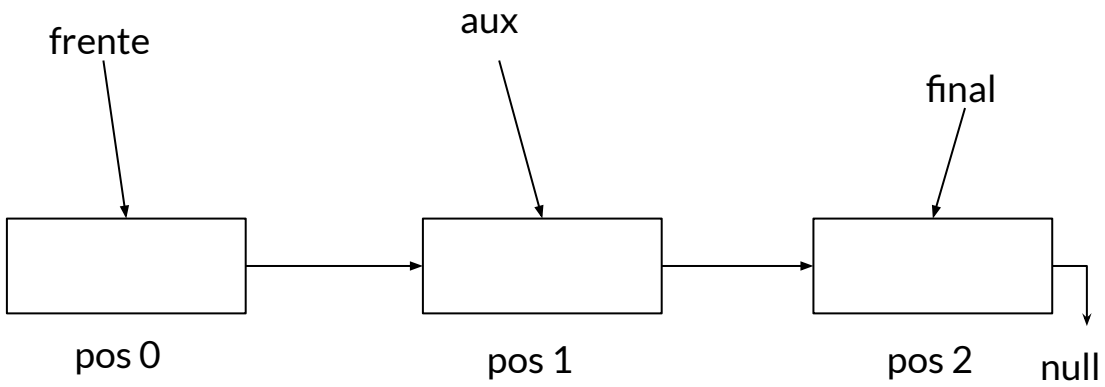
Lista simplemente enlazada

- Buscar el nodo en la posición i-ésima.

int i = 1;

Nodo *aux = frente;

Hacemos aux = aux->siguiente hasta que lleguemos a la posición buscada.



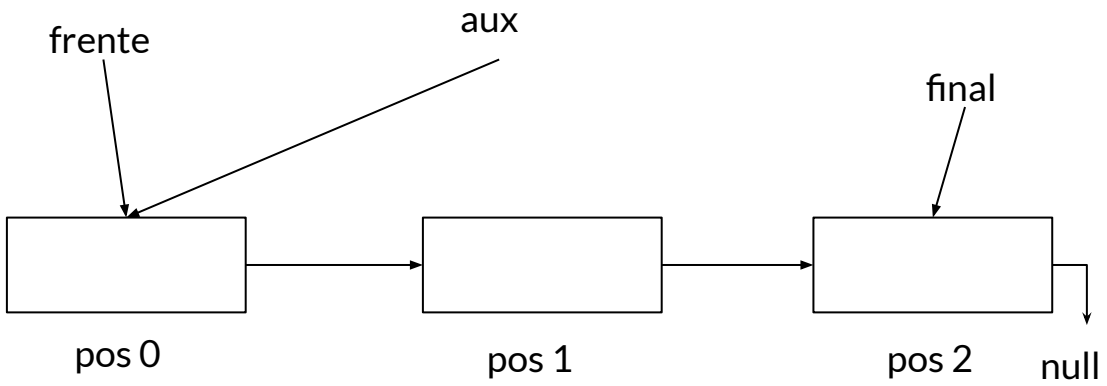
Lista simplemente enlazada

- Buscar el nodo en la posición i-ésima y su anterior.

int i = 1;

Nodo *aux = frente;

Nodo *ant = null;



ant = null

Lista simplemente enlazada

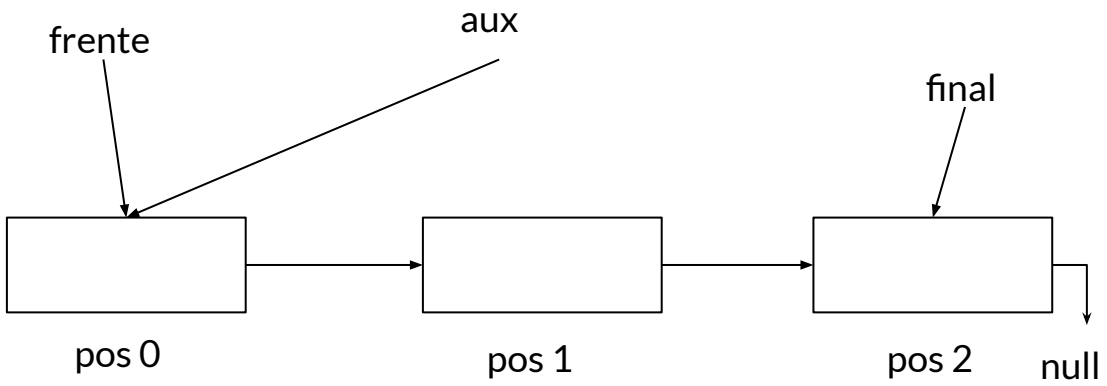
- Buscar el nodo en la posición i-ésima y su anterior.

int i = 1;

Nodo *aux = frente;

Nodo *ant = null;

Hacemos ant = aux y aux = aux->siguiente hasta que lleguemos a la posición buscada.



ant = null

Lista simplemente enlazada

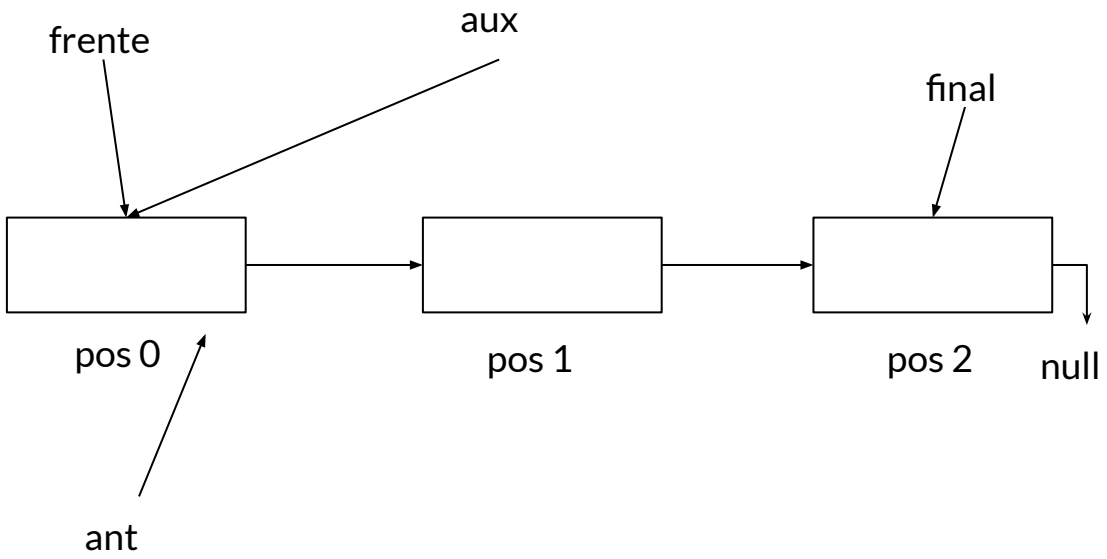
- Buscar el nodo en la posición i-ésima y su anterior.

int i = 1;

Nodo *aux = frente;

Nodo *ant = null;

Hacemos ant = aux y aux = aux->siguiente hasta que lleguemos a la posición buscada.



Lista simplemente enlazada

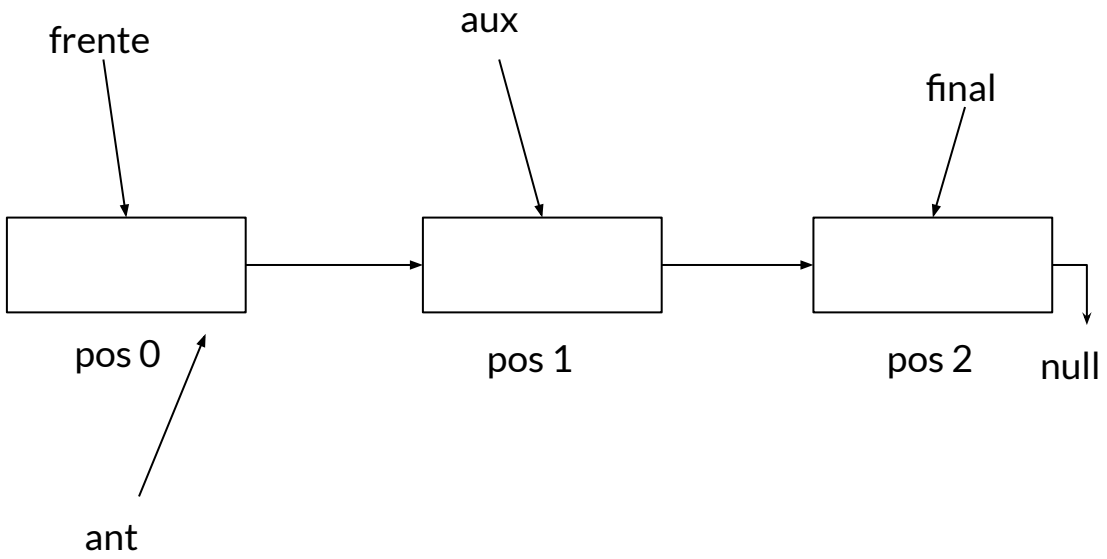
- Buscar el nodo en la posición i-ésima y su anterior.

int i = 1;

Nodo *aux = frente;

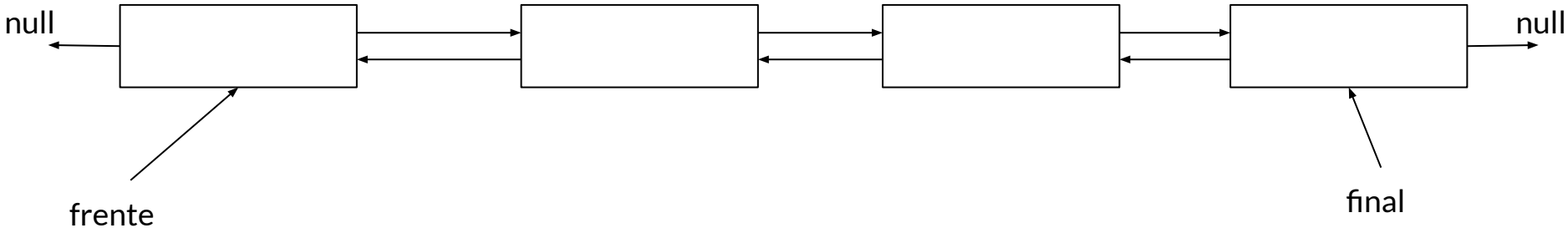
Nodo *ant = null;

Hacemos ant = aux y aux = aux->siguiente hasta que lleguemos a la posición buscada.



Lista doblemente enlazada

- Cada nodo apunta al nodo anterior y al nodo siguiente.
- Puede verse como una cola doble pero sin restricciones de acceso, eliminación e inserción.
- Mantenemos un apuntador al frente y al final.



Lista doblemente enlazada

- void push(T d);
- Inserción en una lista vacía.

frente = null final = null

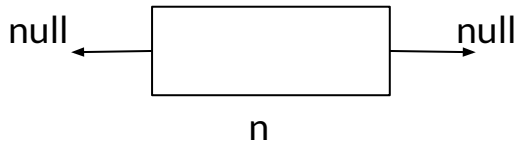
Lista doblemente enlazada

- void push(T d);
- Inserción en una lista vacía.

Nodo *n = new Nodo(d);

frente = null

final = null



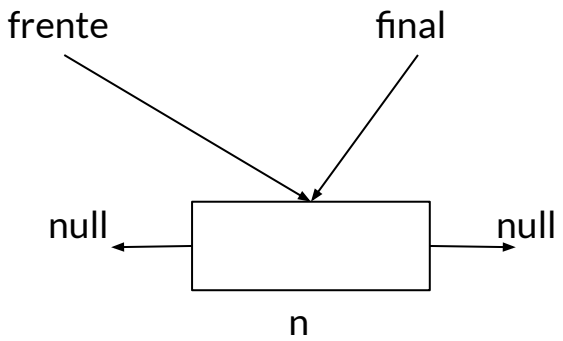
Lista doblemente enlazada

- void push(T d);
- Inserción en una lista vacía.

Nodo *n = new Nodo(d);

frente = n;

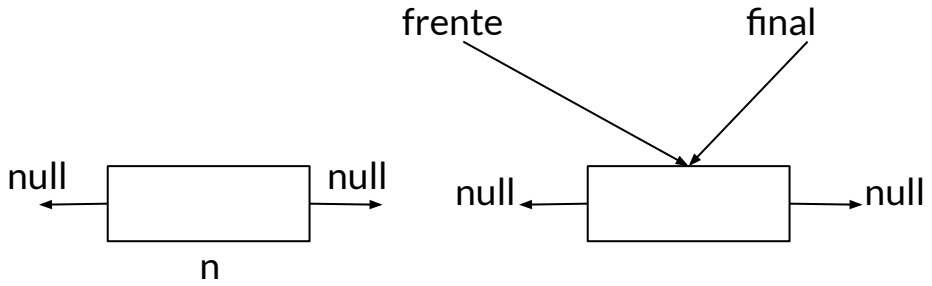
final = n;



Lista doblemente enlazada

- void push_front(T d);
- Inserción al frente.

Nodo *n = new Nodo(d);

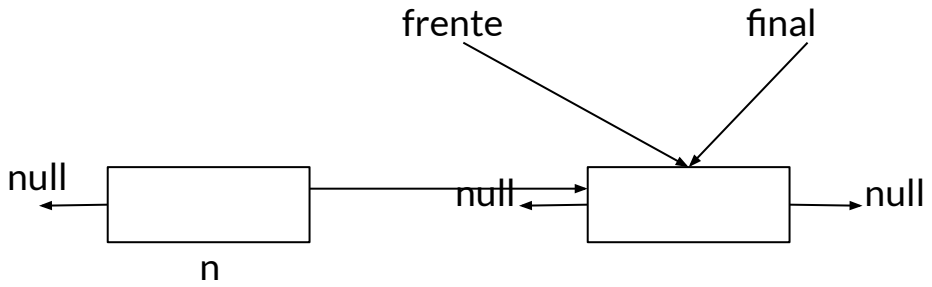


Lista doblemente enlazada

- void push_front(T d);
- Inserción al frente.

Nodo *n = new Nodo(d);

n->siguiente = frente;



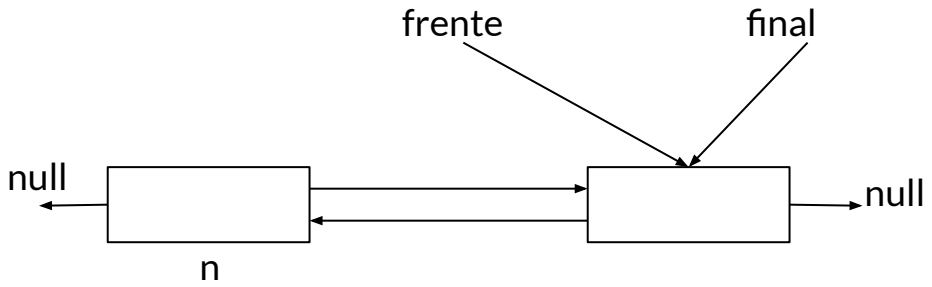
Lista doblemente enlazada

- void push_front(T d);
- Inserción al frente.

Nodo *n = new Nodo(d);

n->siguiente = frente;

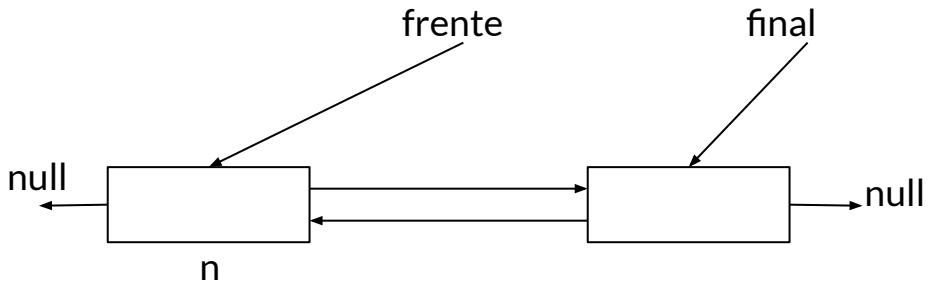
frente->anterior = n;



Lista doblemente enlazada

- void push_front(T d);
- Inserción al frente.

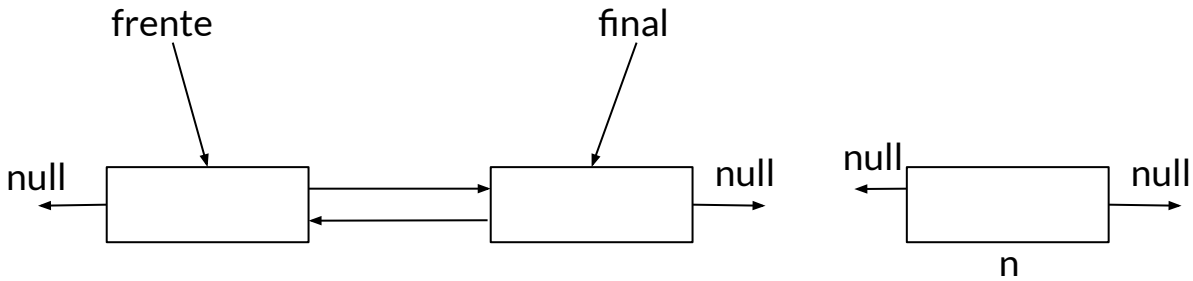
```
Nodo *n = new Nodo(d);

n->siguiente = frente;

frente->anterior = n;

frente = n;
```



Lista doblemente enlazada

- void push_back(T d);
- Inserción al final.

Nodo *n = new Nodo(d);

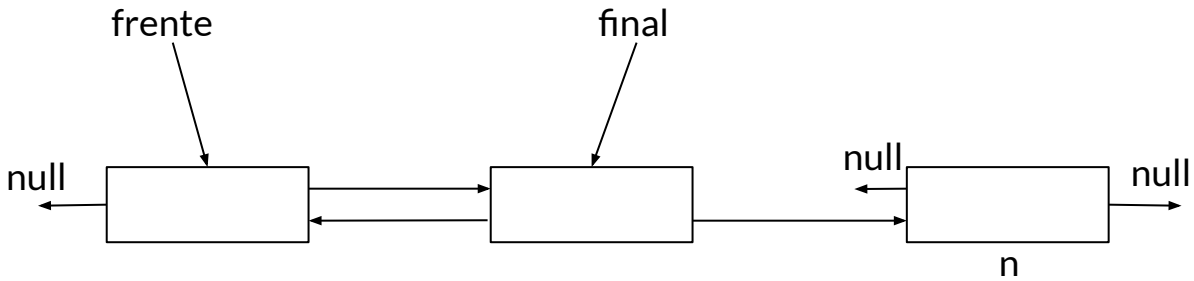


Lista doblemente enlazada

- void push_back(T d);
- Inserción al final.

Nodo *n = new Nodo(d);

final->siguiente = n;



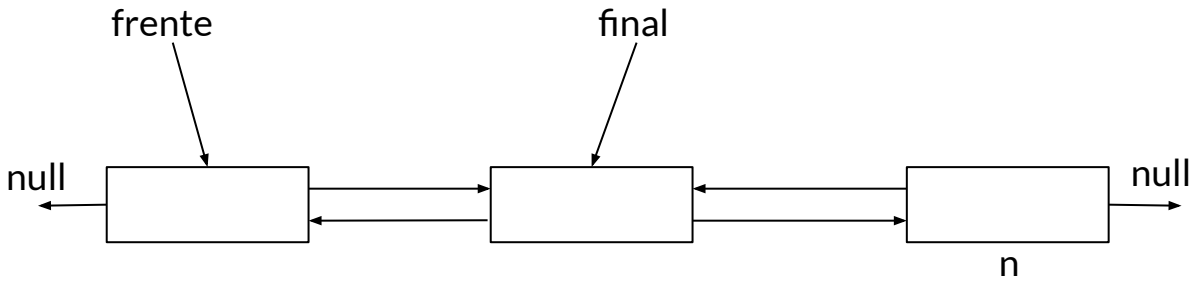
Lista doblemente enlazada

- void push_back(T d);
- Inserción al final.

Nodo *n = new Nodo(d);

final->siguiente = n;

n->anterior = final;



Lista doblemente enlazada

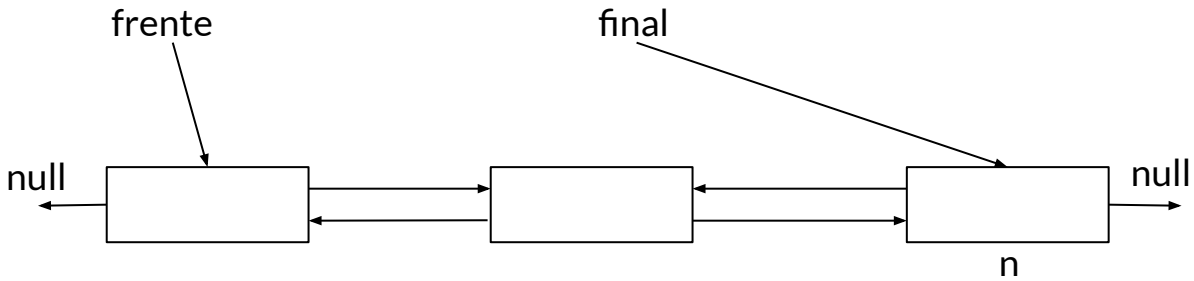
- void push_back(T d);
- Inserción al final.

Nodo *n = new Nodo(d);

final->siguiente = n;

n->anterior = final;

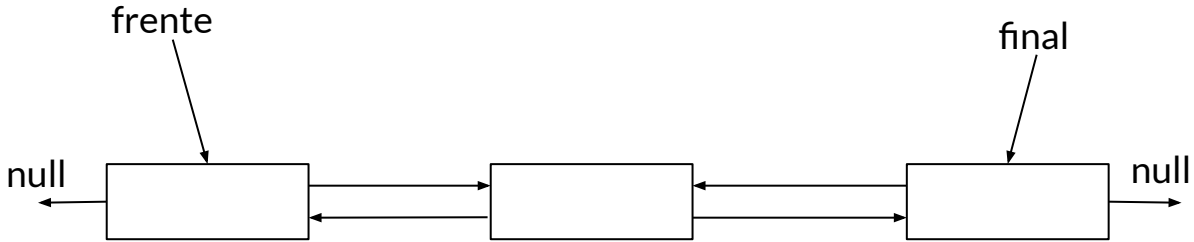
final = n;

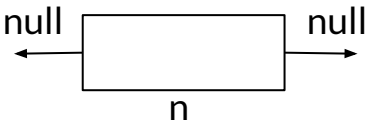


Lista doblemente enlazada

- void push(T d, int i);
- Inserción en la i-ésima posición.

Nodo *n = new Nodo(d);



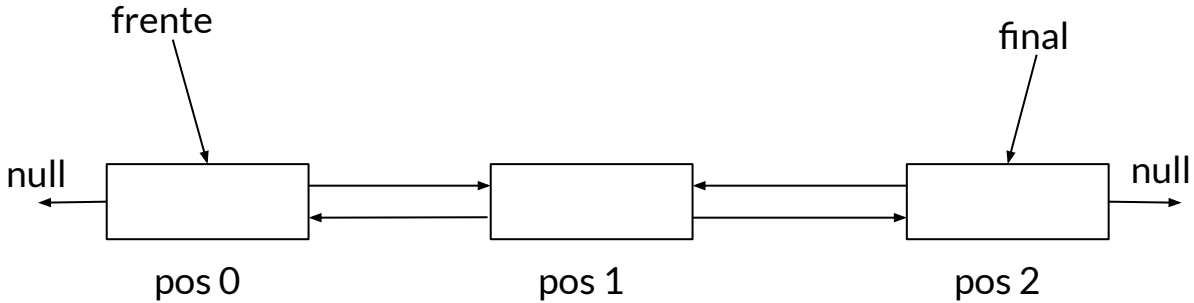


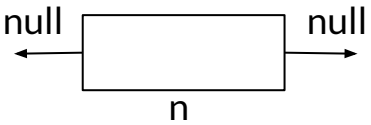
Lista doblemente enlazada

- void push(T d, int i);
- Inserción en la i-ésima posición.

int i = 2;

Nodo *n = new Nodo(d);





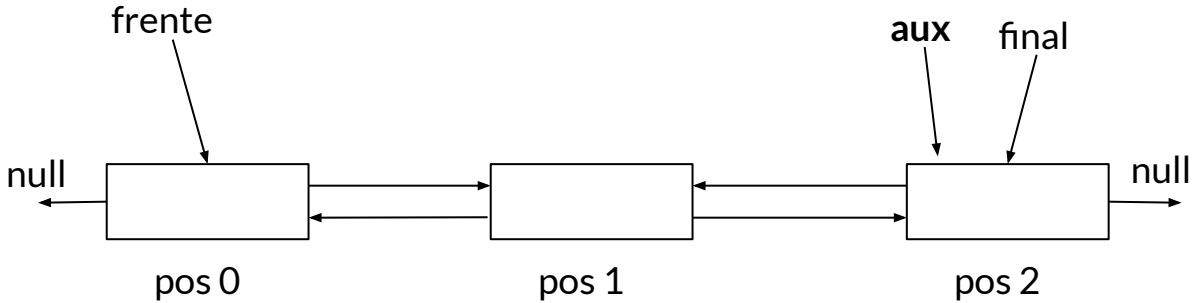
Lista doblemente enlazada

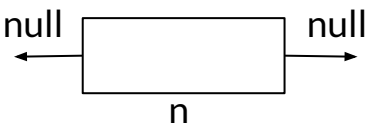
- void push(T d, int i);
- Inserción en la i-ésima posición.

int i = 2;

Nodo *n = new Nodo(d);

Nodo ***aux*** = nodo en la posición i;





Lista doblemente enlazada

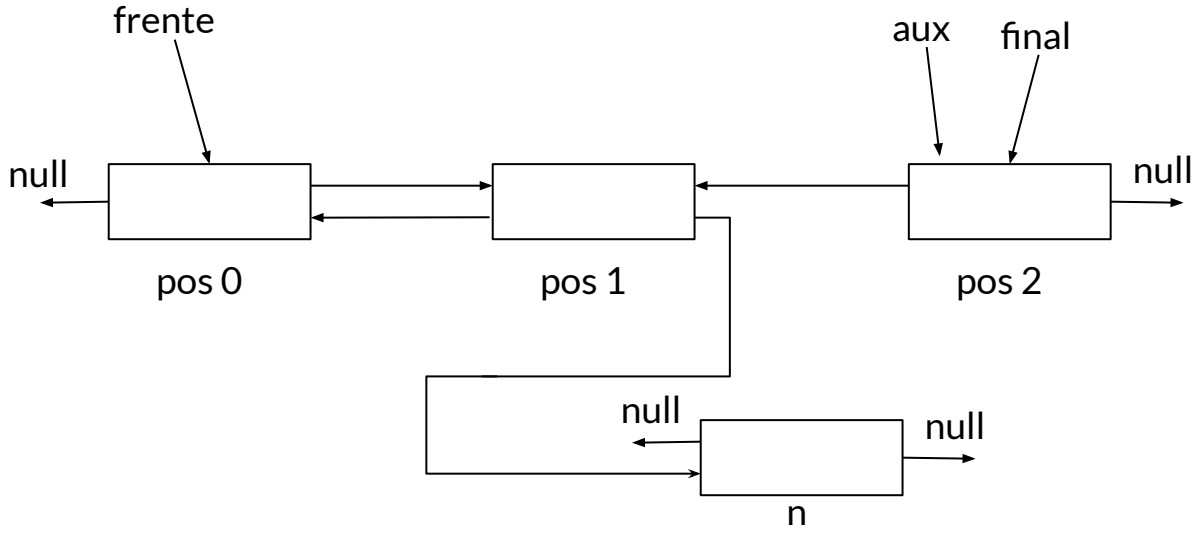
- void push(T d, int i);
- Inserción en la i-ésima posición.

int i = 2;

Nodo *n = new Nodo(d);

Nodo *aux = nodo en la posición i;

aux->anterior->siguiente = n;



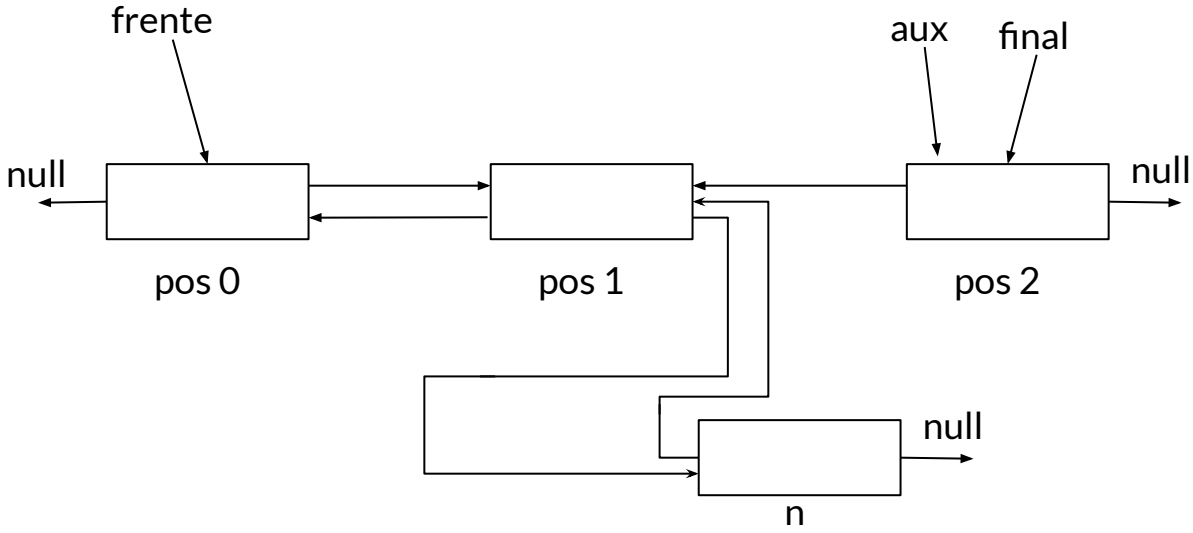
Lista doblemente enlazada

- void push(T d, int i);
- Inserción en la i-ésima posición.

int i = 2;

Nodo *n = new Nodo(d);

Nodo *aux = nodo en la posición i;
aux->anterior->siguiente = n;
n->anterior = aux->anterior;



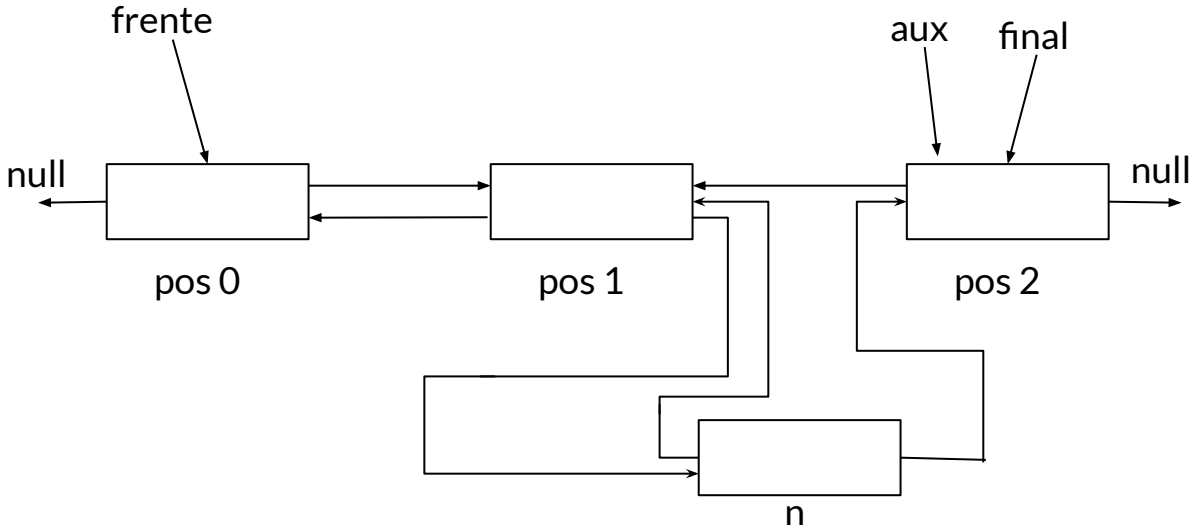
Lista doblemente enlazada

- void push(T d, int i);
- Inserción en la i-ésima posición.

int i = 2;

Nodo *n = new Nodo(d);

Nodo *aux = nodo en la posición i;
aux->anterior->siguiente = n;
n->anterior = aux->anterior;
n->siguiente = aux;



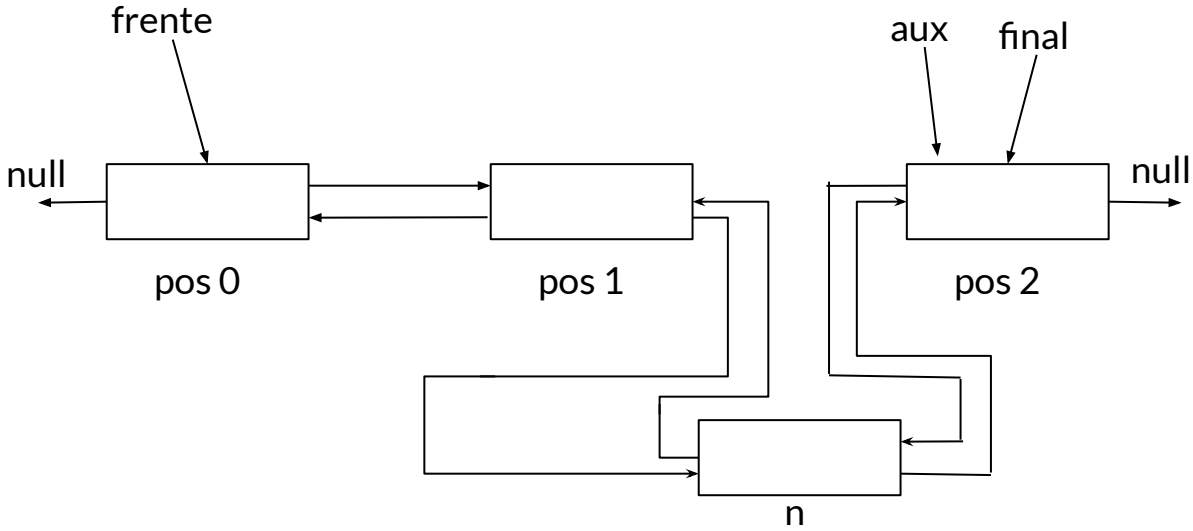
Lista doblemente enlazada

- void push(T d, int i);
- Inserción en la i-ésima posición.

int i = 2;

Nodo *n = new Nodo(d);

Nodo *aux = nodo en la posición i;
aux->anterior->siguiente = n;
n->anterior = aux->anterior;
n->siguiente = aux;
aux->anterior = n;



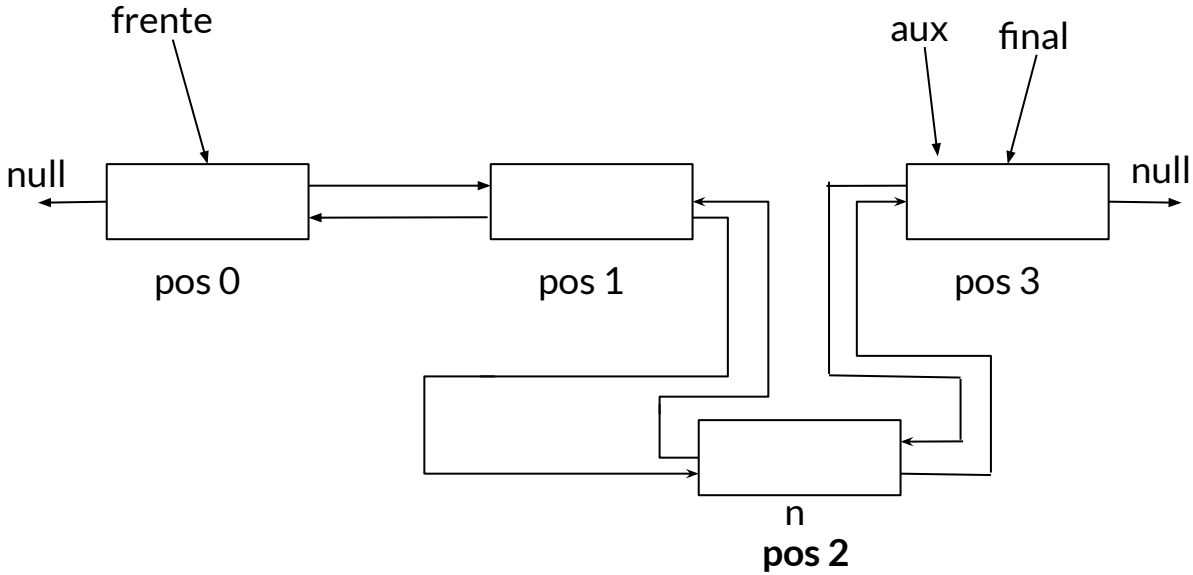
Lista doblemente enlazada

- void push(T d, int i);
- Inserción en la i-ésima posición.

int i = 2;

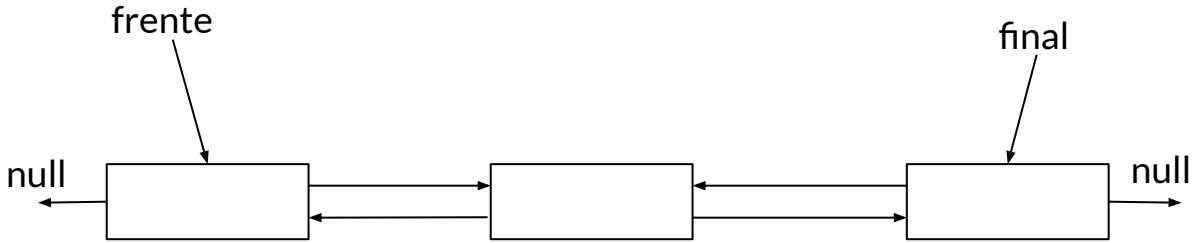
Nodo *n = new Nodo(d);

Nodo *aux = nodo en la posición i;
aux->anterior->siguiente = n;
n->anterior = aux->anterior;
n->siguiente = aux;
aux->anterior = n;



Lista doblemente enlazada

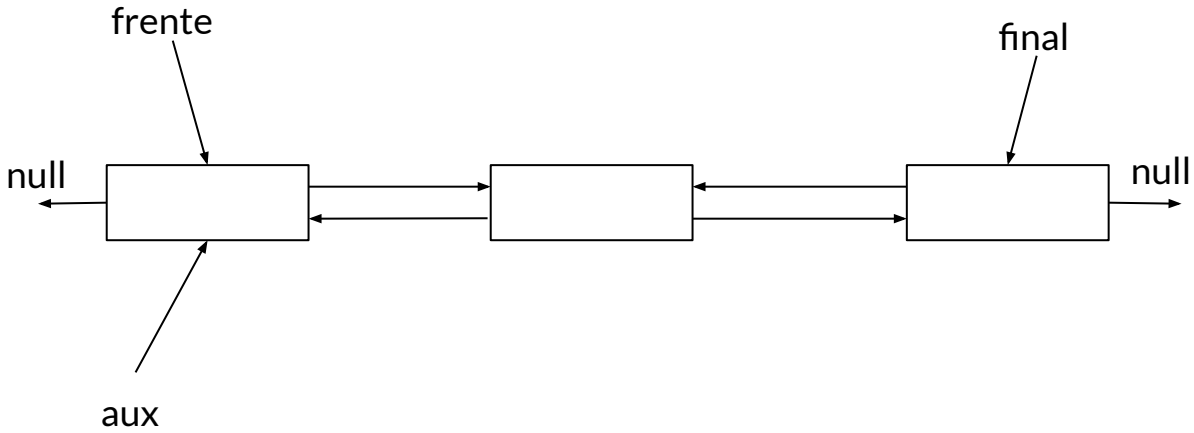
- void pop_front( );
- Eliminar al frente.



Lista doblemente enlazada

- void pop_front( );
- Eliminar al frente.

Nodo *aux = frente;

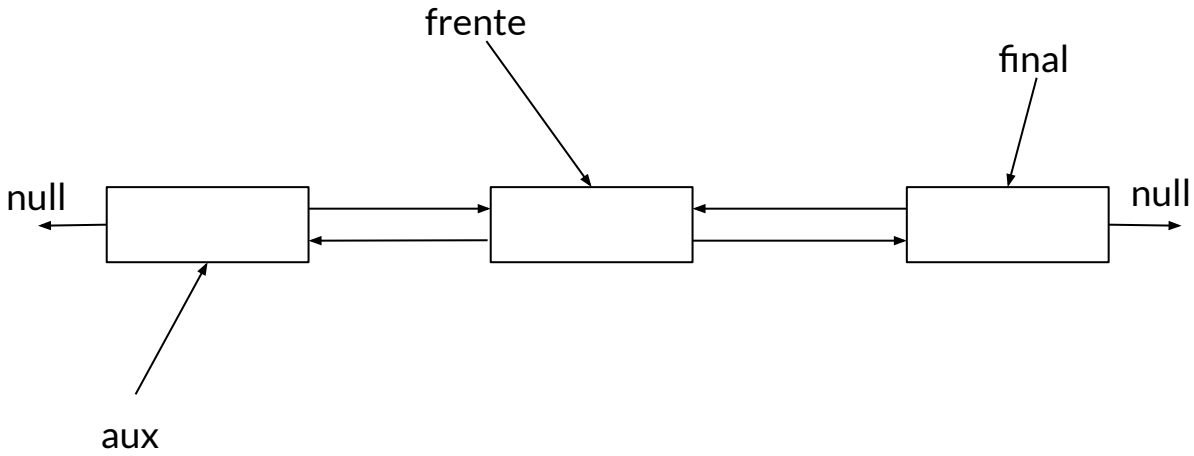


Lista doblemente enlazada

- void pop_front( );
- Eliminar al frente.

Nodo *aux = frente;

frente = frente->siguiente;



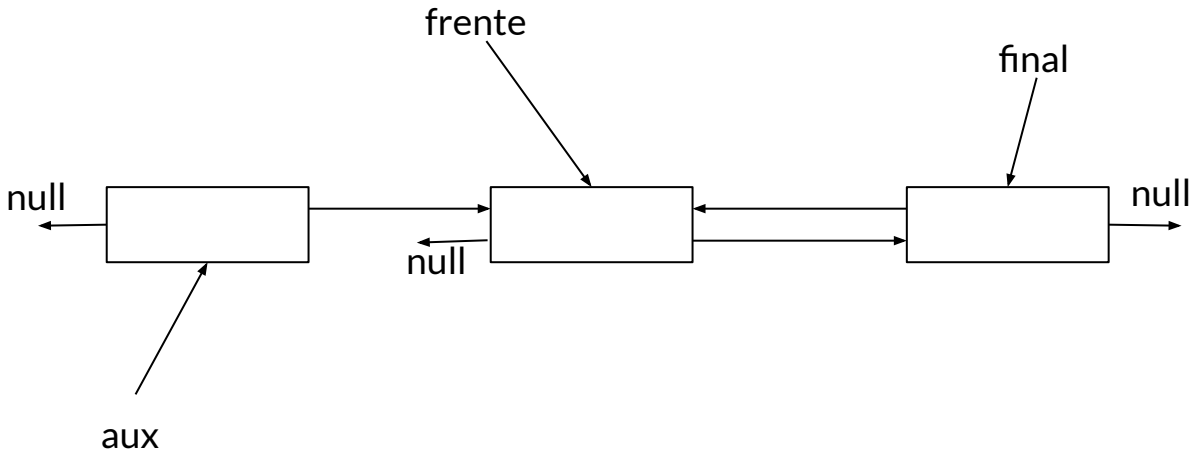
Lista doblemente enlazada

- void pop_front( );
- Eliminar al frente.

Nodo *aux = frente;

frente = frente->siguiente;

frente->anterior = null;



Lista doblemente enlazada

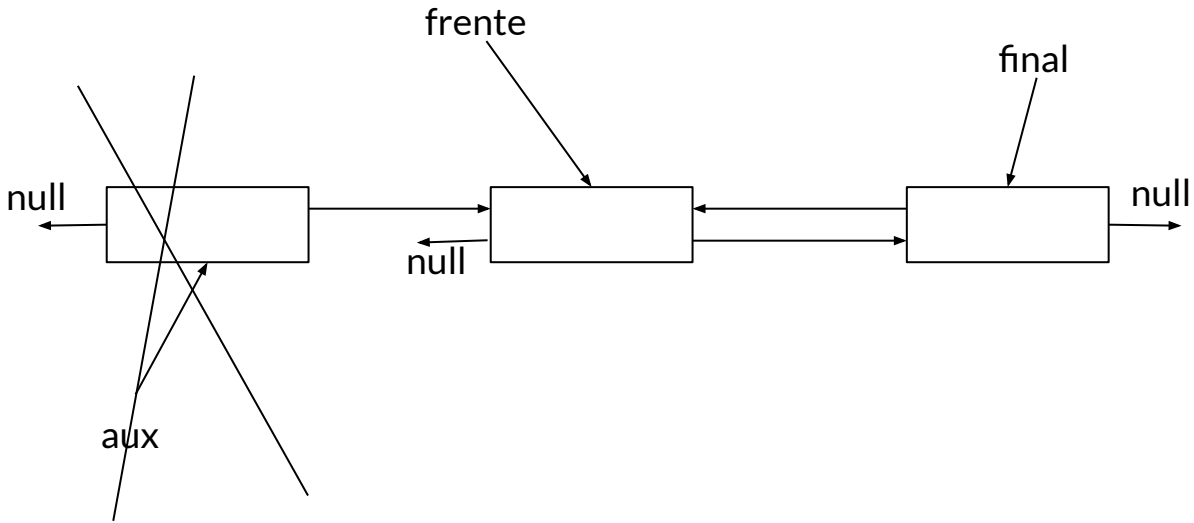
- void pop_front( );
- Eliminar al frente.

Nodo *aux = frente;

frente = frente->siguiente;

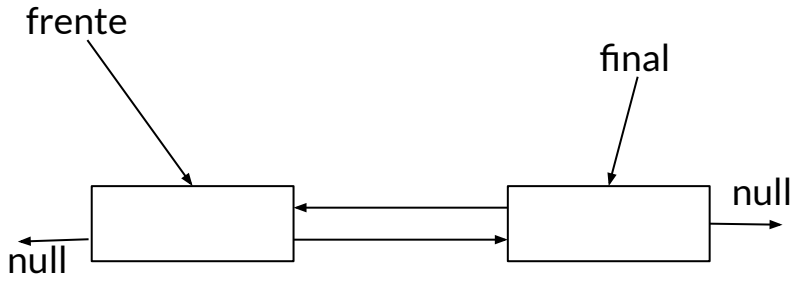
frente->anterior = null;

delete aux;



Lista doblemente enlazada

- void pop_front( );
- Eliminar al frente.



Nodo *aux = frente;

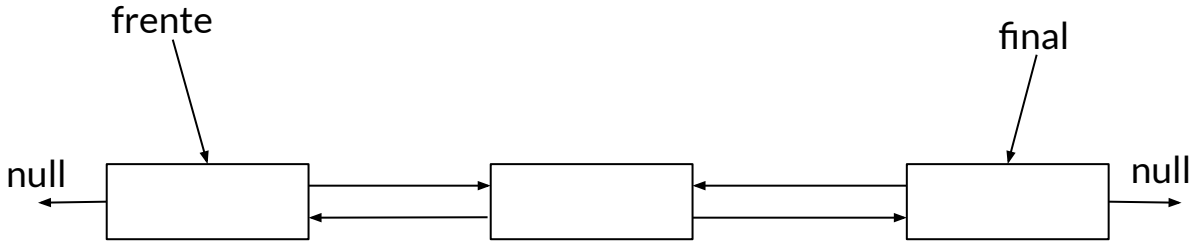
frente = frente->siguiente;

frente->anterior = null;

delete aux;

Lista doblemente enlazada

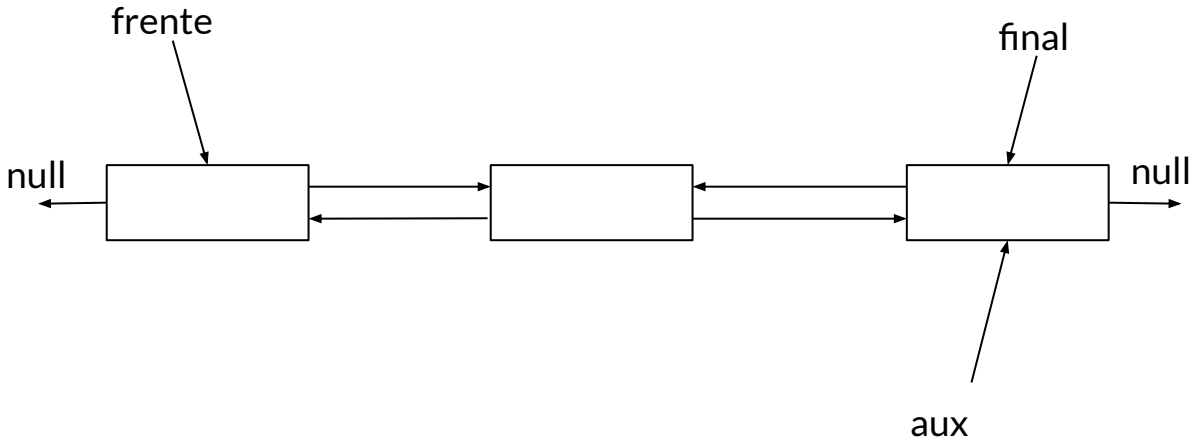
- void pop_back( );
- Eliminar al final.



Lista doblemente enlazada

- void pop_back( );
- Eliminar al final.

Nodo *aux = final;

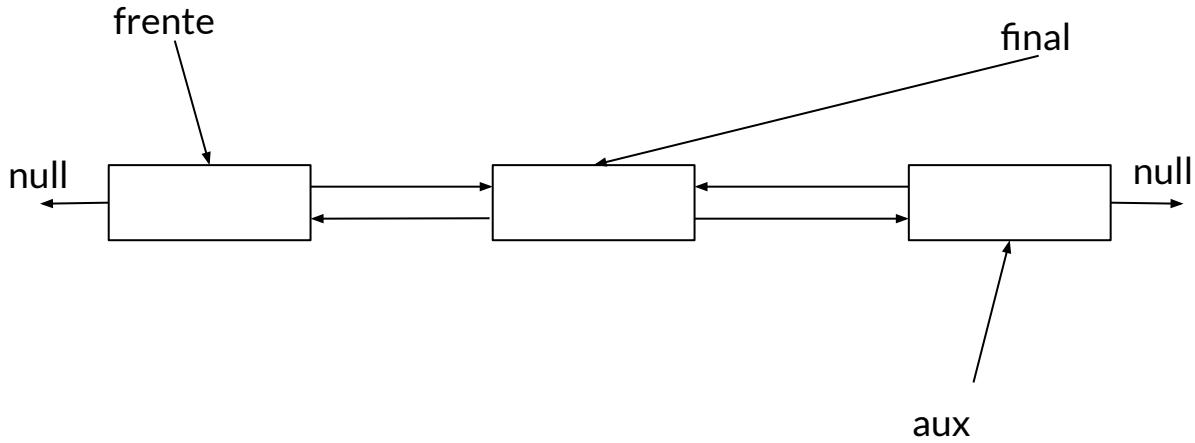


Lista doblemente enlazada

- void pop_back( );
- Eliminar al final.

Nodo *aux = final;

final = final->anterior;



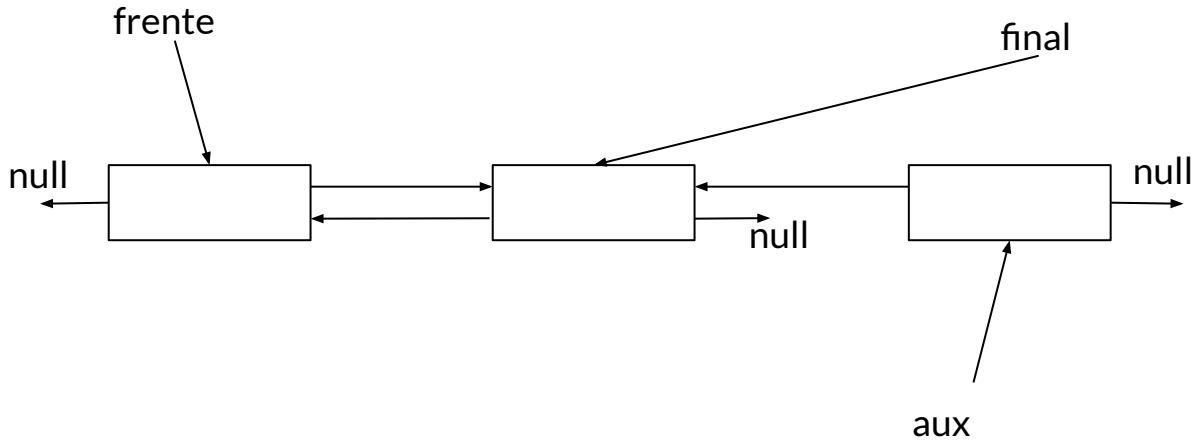
Lista doblemente enlazada

- void pop_back( );
- Eliminar al final.

Nodo *aux = final;

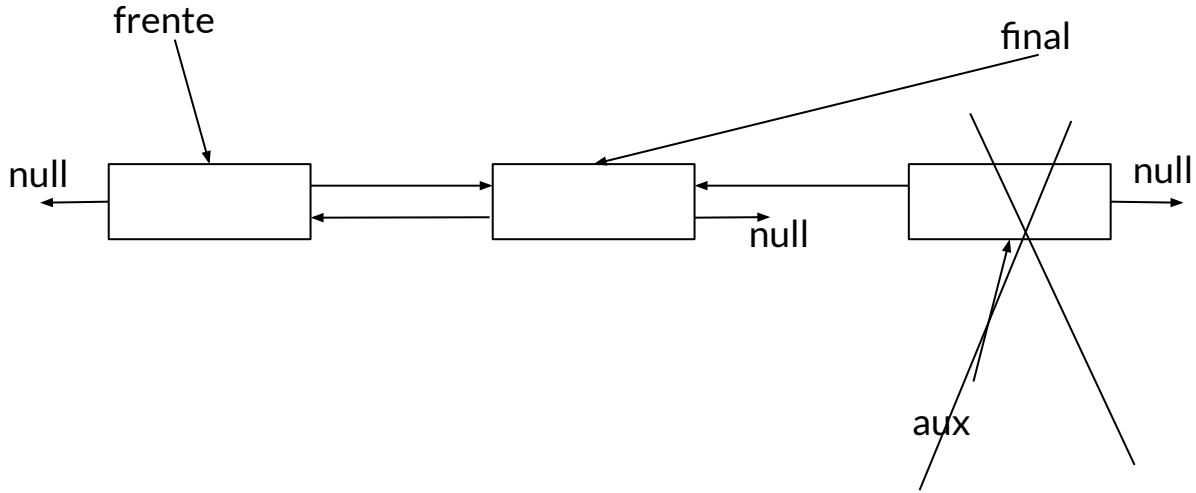
final = final->anterior;

final->siguiente = null;



Lista doblemente enlazada

- void pop_back( );
- Eliminar al final.



Nodo *aux = final;

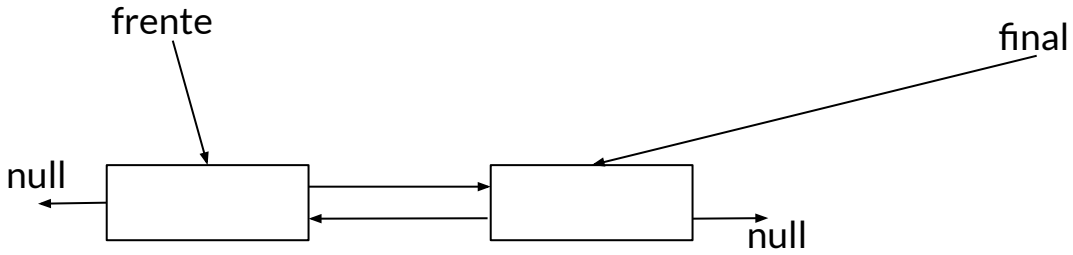
final = final->anterior;

final->siguiente = null;

delete aux;

Lista doblemente enlazada

- void pop_back( );
- Eliminar al final.



Nodo *aux = final;

final = final->anterior;

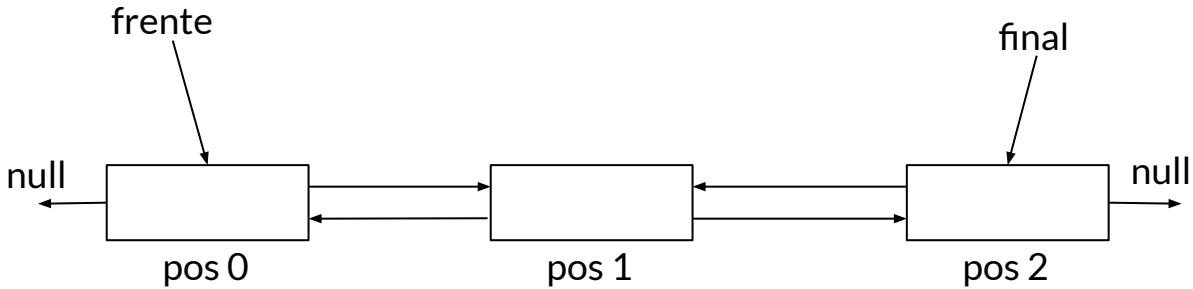
final->siguiente = null;

delete aux;

Lista doblemente enlazada

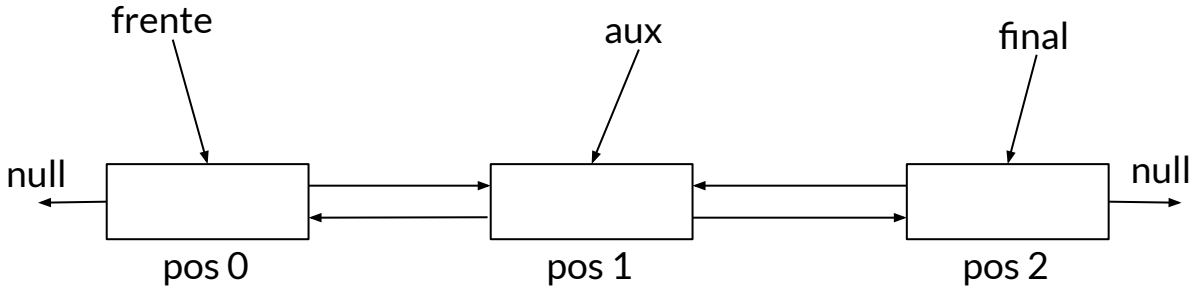
- void pop(int i);
- Eliminar el nodo en la posición i-ésima.

int i = 1;



Lista doblemente enlazada

- void pop(int i);
- Eliminar el nodo en la posición i-ésima.

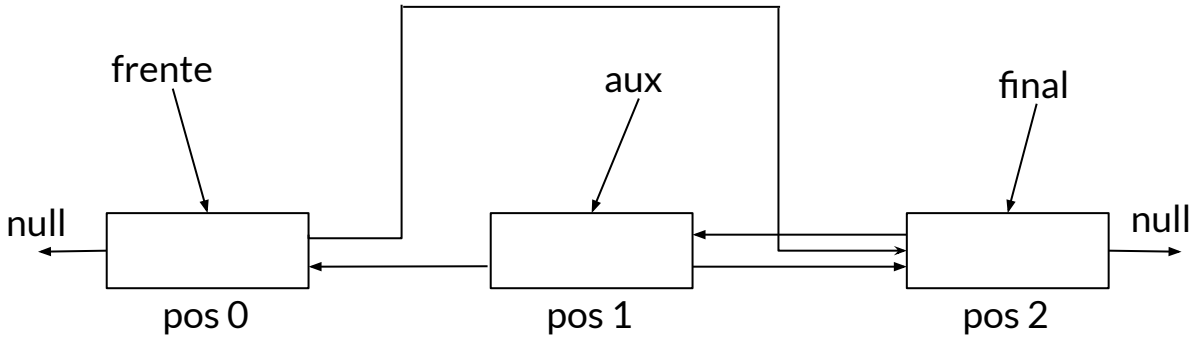


int i = 1;

Nodo *aux = nodo en la posición i;

Lista doblemente enlazada

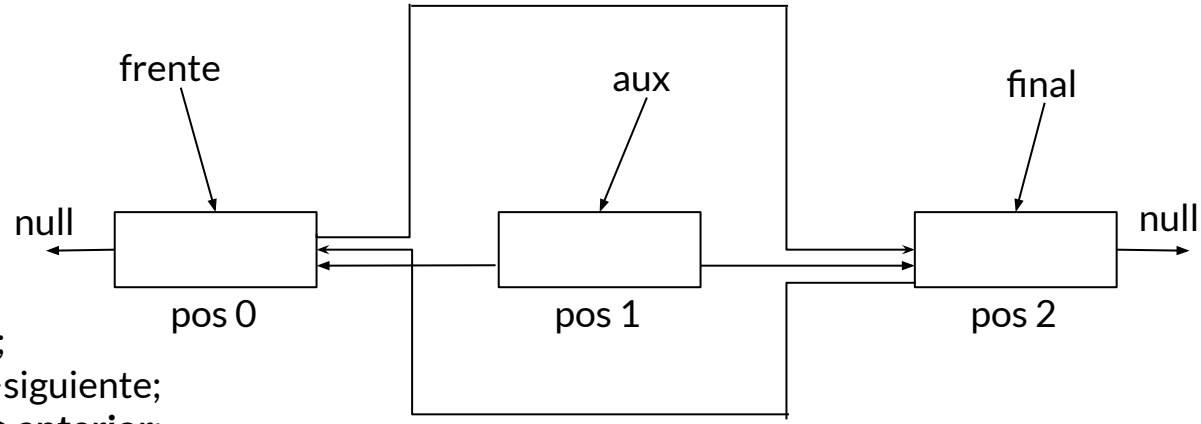
- void pop(int i);
- Eliminar el nodo en la posición i-ésima.



int i = 1;
Nodo *aux = nodo en la posición i;
aux->anterior->siguiente = aux->siguiente;

Lista doblemente enlazada

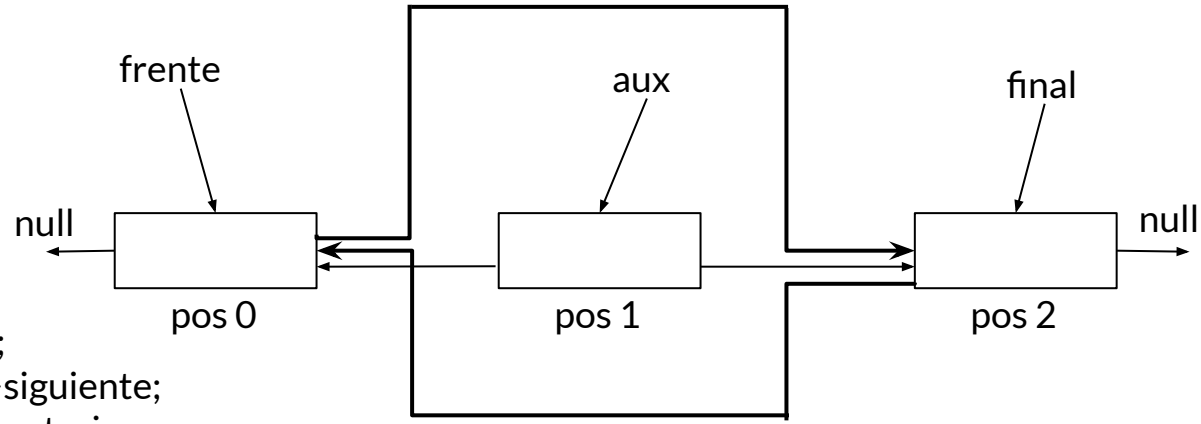
- void pop(int i);
- Eliminar el nodo en la posición i-ésima.



int i = 1;
Nodo *aux = nodo en la posición i;
aux->anterior->siguiente = aux->siguiente;
aux->siguiente->anterior = aux->anterior;

Lista doblemente enlazada

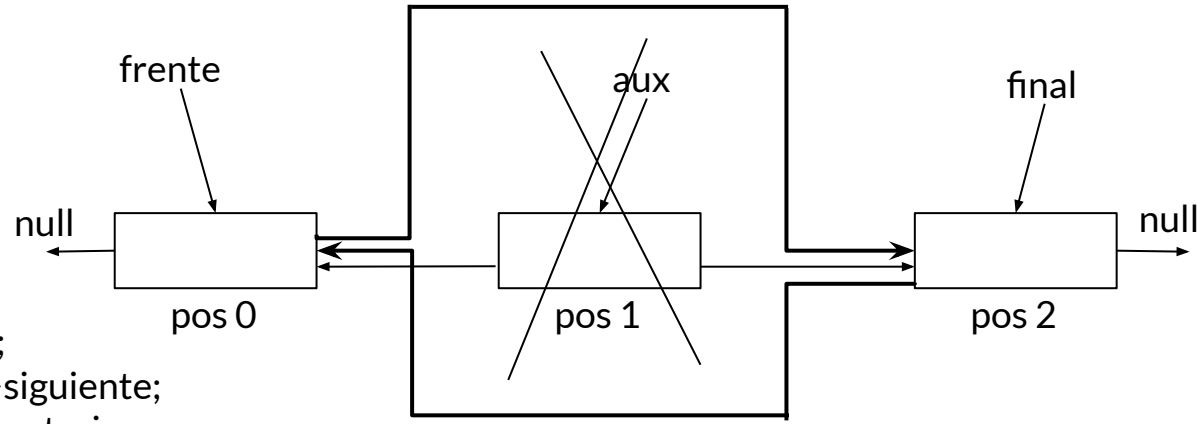
- void pop(int i);
- Eliminar el nodo en la posición i-ésima.



int i = 1;
Nodo *aux = nodo en la posición i;
aux->anterior->siguiente = aux->siguiente;
aux->siguiente->anterior = aux->anterior;

Lista doblemente enlazada

- void pop(int i);
- Eliminar el nodo en la posición i-ésima.

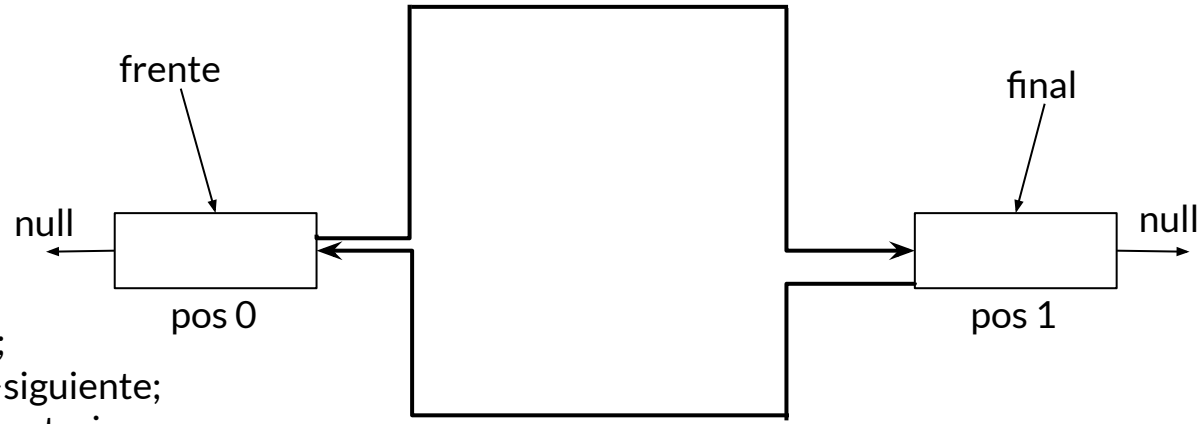


int i = 1;
Nodo *aux = nodo en la posición i;
aux->anterior->siguiente = aux->siguiente;
aux->siguiente->anterior = aux->anterior;

delete aux;

Lista doblemente enlazada

- void pop(int i);
- Eliminar el nodo en la posición i-ésima.

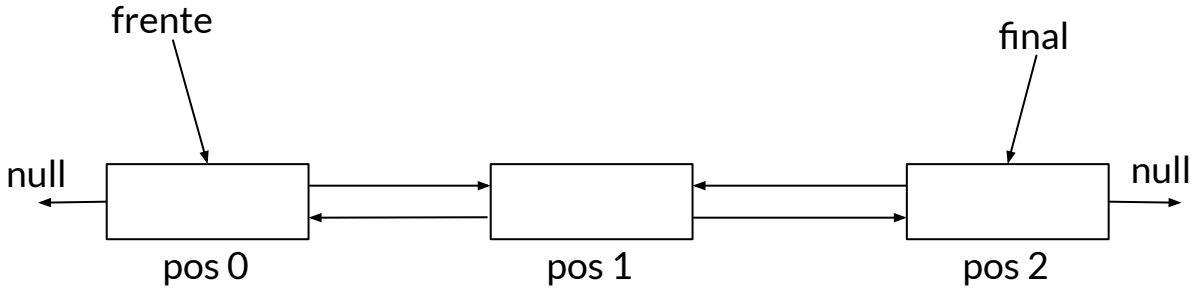



```
int i = 1;
Nodo *aux = nodo en la posición i;
aux->anterior->siguiente = aux->siguiente;
aux->siguiente->anterior = aux->anterior;

delete aux;
```

Lista doblemente enlazada

- T get(int i);
- Devuelve el dato en la posición i.

int i = 1;

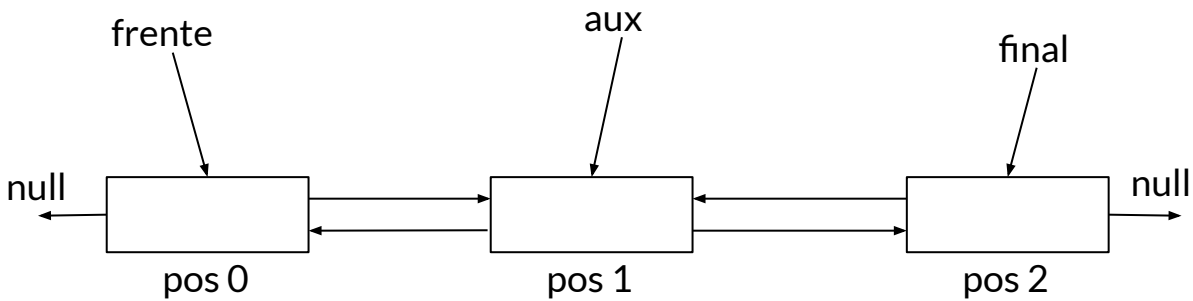


Lista doblemente enlazada

- T get(int i);
- Devuelve el dato en la posición i.

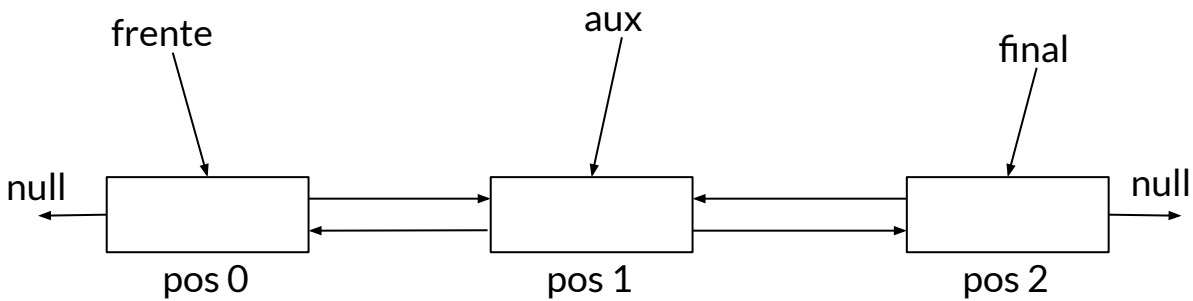
int i = 1;

Nodo *aux = nodo en la posición i.



Lista doblemente enlazada

- T get(int i);
- Devuelve el dato en la posición i.



int i = 1;

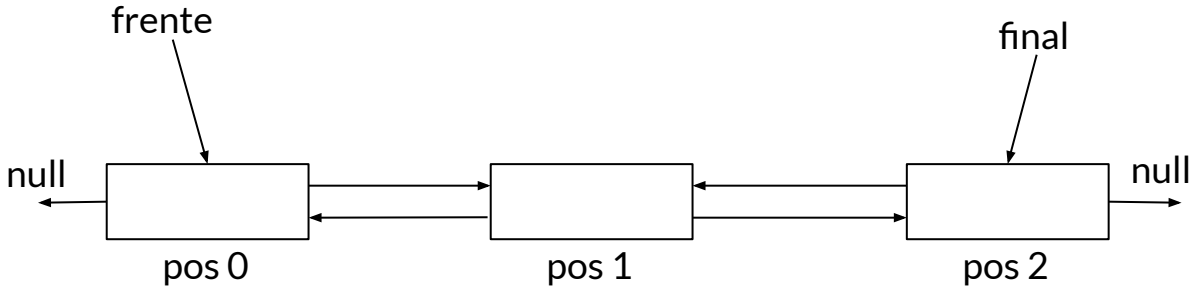
Nodo *aux = nodo en la posición i.

return aux->dato;

Lista doblemente enlazada

- void set(T d, int i);
- Actualiza el dato en la posición i.

int i = 1;

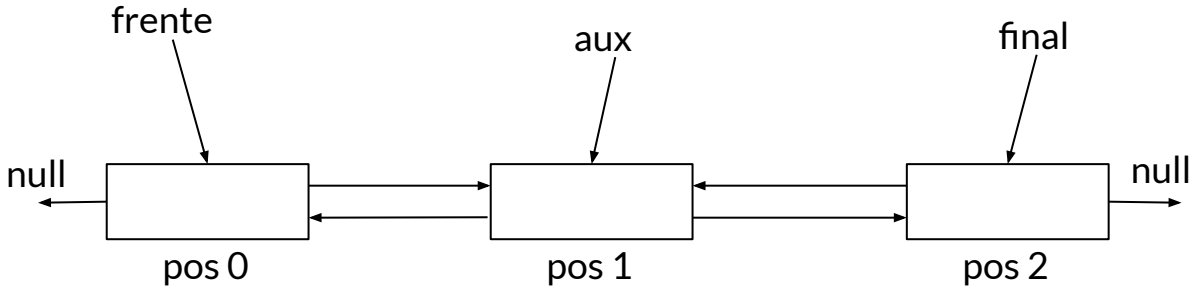


Lista doblemente enlazada

- void set(T d, int i);
- Actualiza el dato en la posición i.

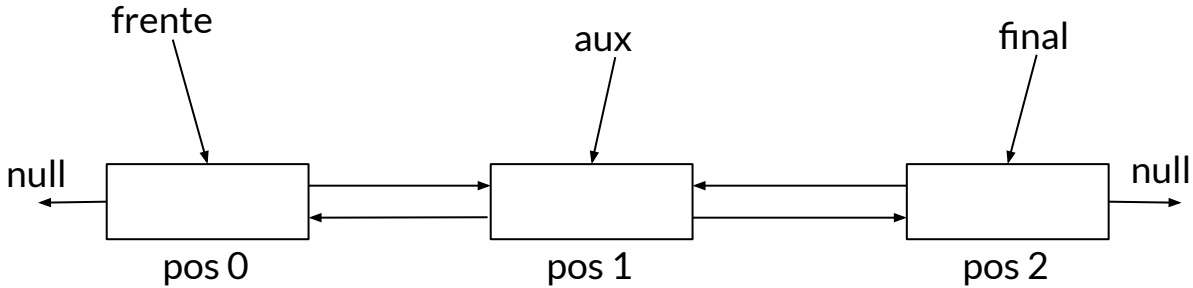
int i = 1;

Nodo *aux = nodo en la posición i.



Lista doblemente enlazada

- void set(T d, int i);
- Actualiza el dato en la posición i.



int i = 1;

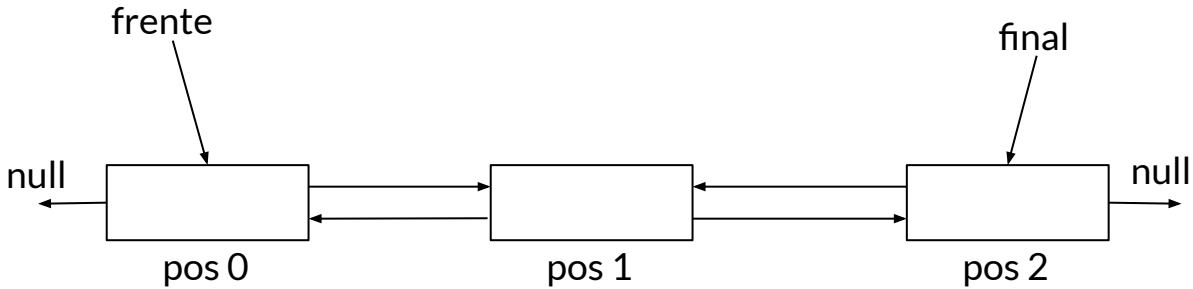
Nodo *aux = nodo en la posición i.

aux->dato = d;

Lista doblemente enlazada

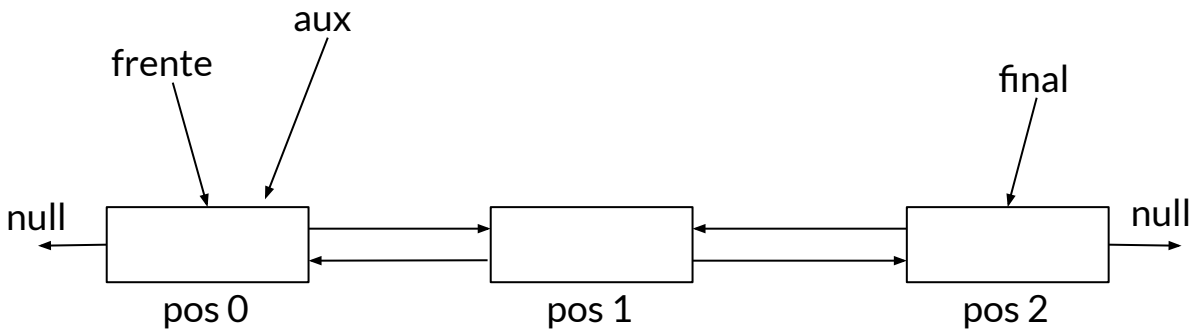
- Buscar el nodo en la posición i-ésima.

int i = 1;



Lista doblemente enlazada

- Buscar el nodo en la posición i-ésima.

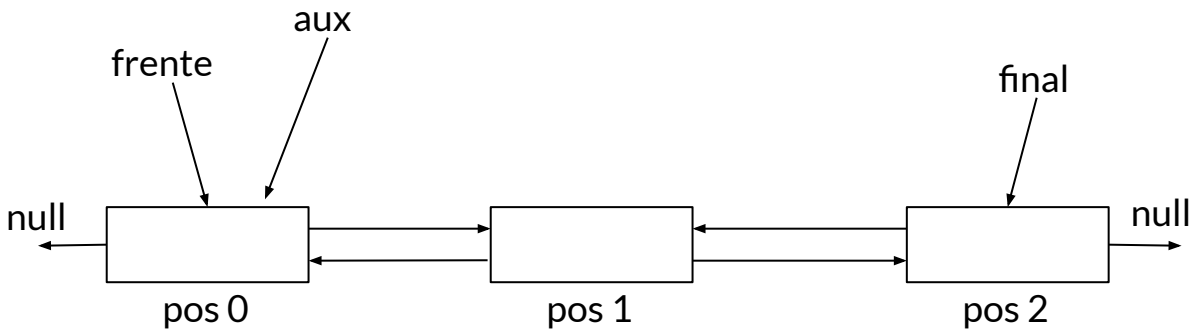


int i = 1;

Nodo *aux = frente;

Lista doblemente enlazada

- Buscar el nodo en la posición i-ésima.



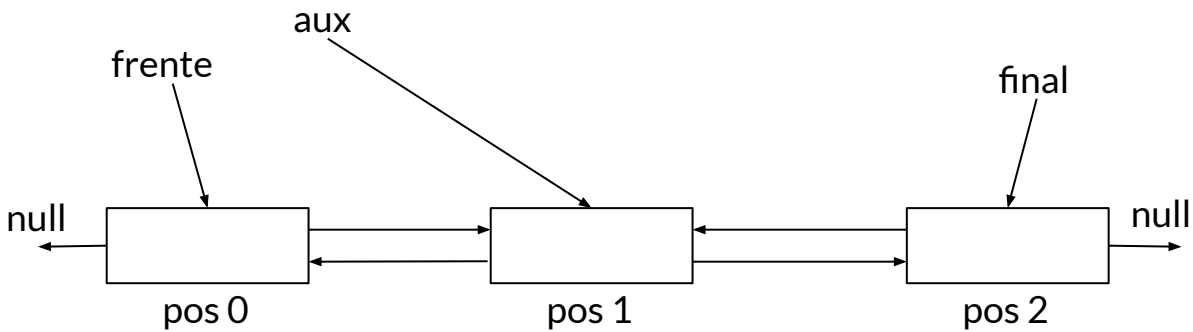
int i = 1;

Nodo *aux = frente;

Hacemos aux = aux->siguiente
hasta llegar a la posición buscada.

Lista doblemente enlazada

- Buscar el nodo en la posición i-ésima.



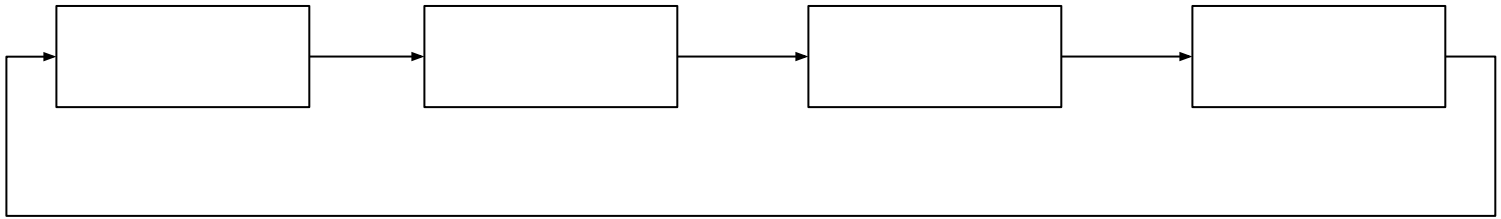
int i = 1;

Nodo *aux = frente;

Hacemos aux = aux->siguiente
hasta llegar a la posición buscada.

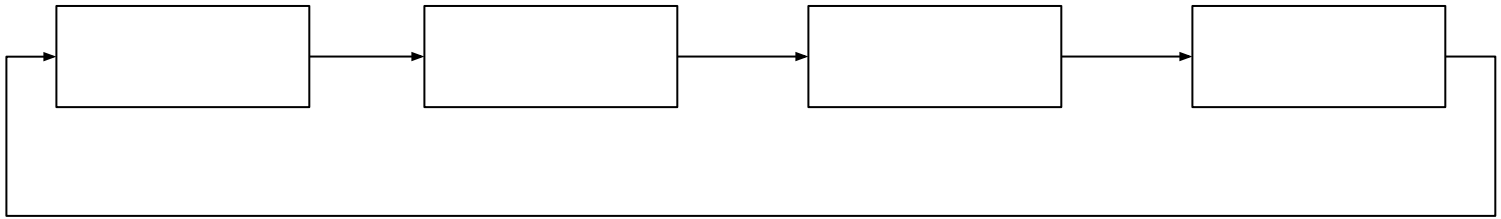
Lista circular

- Es una lista en la que el último nodo apunta al primero.
- Permite realizar operaciones sin considerar casos especiales.
- Se puede implementar tanto de manera enlazada (simple y doble) como utilizando arreglos.



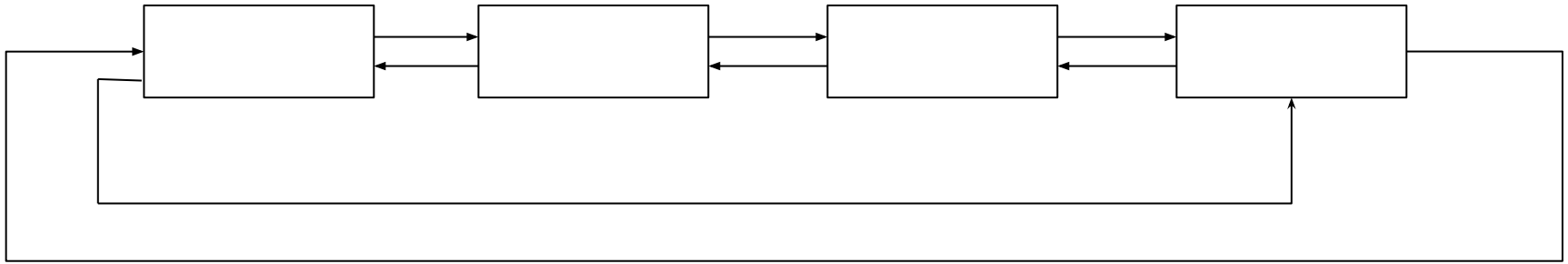
Lista circular

- Es una lista en la que el último nodo apunta al primero.
- Permite realizar operaciones sin considerar casos especiales.
- Se puede implementar tanto de manera enlazada (**simple** y doble) como utilizando arreglos.



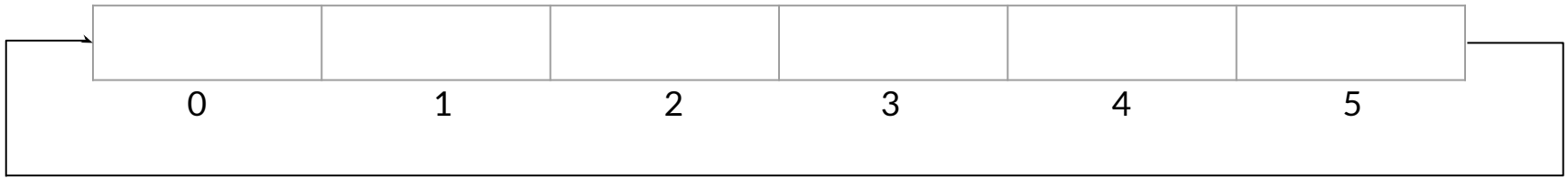
Lista circular

- Es una lista en la que el último nodo apunta al primero.
- Permite realizar operaciones sin considerar casos especiales.
- Se puede implementar tanto de manera enlazada (simple y **doble**) como utilizando arreglos.



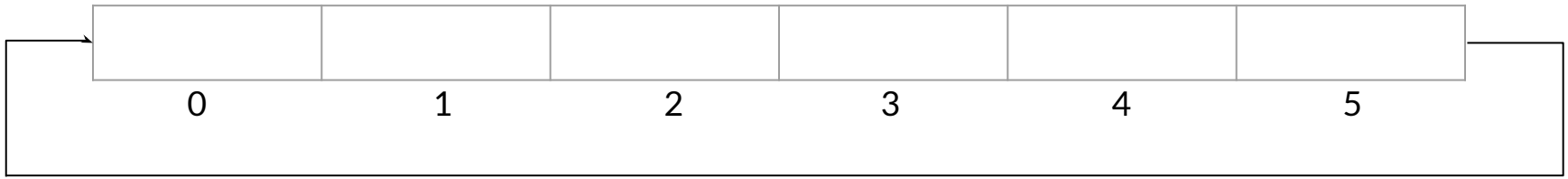
Lista circular

- Es una lista en la que el último nodo apunta al primero.
- Permite realizar operaciones sin considerar casos especiales.
- Se puede implementar tanto de manera enlazada (simple y doble) como utilizando **arreglos**.



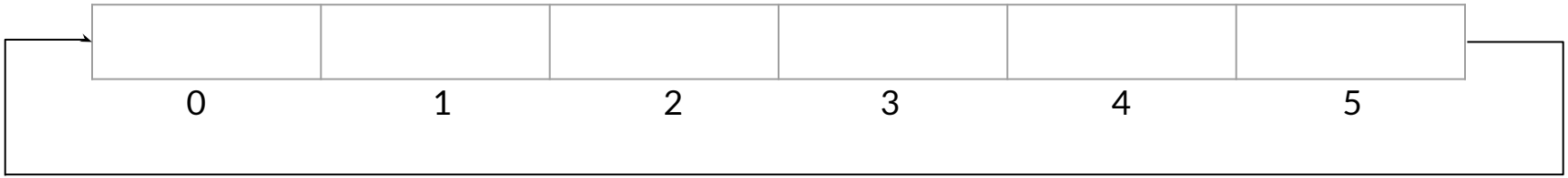
Lista circular basada en arreglo

- Vamos a tratar el caso cuando la lista circular es basada en arreglos, dado que para listas enlazadas solo hay que enlazar el nodo final y frente.



Lista circular basada en arreglo

- La longitud del arreglo determina la **capacidad** de nuestra lista.
- El **tamaño** de la lista hace referencia a la cantidad de elementos que almacena.
- Una lista circular se considera **llena** cuando su tamaño es igual a su capacidad.
- Si la lista está llena y se requiere insertar un nuevo elemento, entonces reemplazamos el arreglo por uno nuevo de mayor longitud que conserva los elementos almacenados.



Lista circular basada en arreglo

- Cuando recién se crea una instancia de la lista, el arreglo puede tener **longitud cero**.
- En este momento la **capacidad** de la lista es también cero.
- Utilizamos dos variables de tipo entero para almacenar las posiciones **frente** y **final** de la lista.
- De manera inicial hacemos **frente = 0** y **final = capacidad - 1**.
- Una vez que se busca insertar el primer elemento, se llama al método resize (redimensionar) para crear un arreglo con la longitud adecuada.

Lista circular basada en arreglo

- Para “**redimensionar**” el arreglo, habremos de considerar un **incremento** en la capacidad actual del arreglo.
- Este incremento se puede calcular como **capacidad / 2**, siempre y cuando la capacidad actual no sea cero. Entonces la nueva capacidad sería **capacidad + (capacidad / 2)**.
- En caso de que la capacidad sea cero, el incremento es un valor preestablecido. Por ejemplo, 12.

Lista circular basada en arreglo

- void resize( );
- Redimensionar el arreglo.

Pseudocódigo

1. Hacer nodos = al arreglo actual
2. Sí capacidad == 0 entonces incremento = 12, de lo contrario incremento = capacidad / 2;
3. Hacer n = capacidad;
4. capacidad = capacidad + incremento;
5. Crear arreglo temp con longitud = capacidad;
6. Copiar todos los elementos de nodos a temp desde 0 a n-1
7. Eliminar nodos
8. Hacer nodos = temp

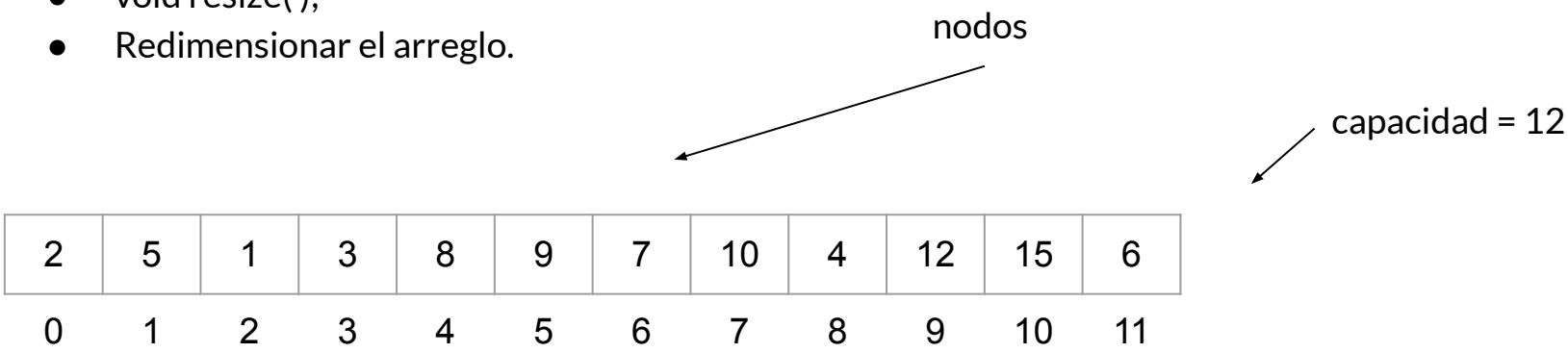
Lista circular basada en arreglo

- void resize( );
- Redimensionar el arreglo.

2	5	1	3	8	9	7	10	4	12	15	6
0	1	2	3	4	5	6	7	8	9	10	11

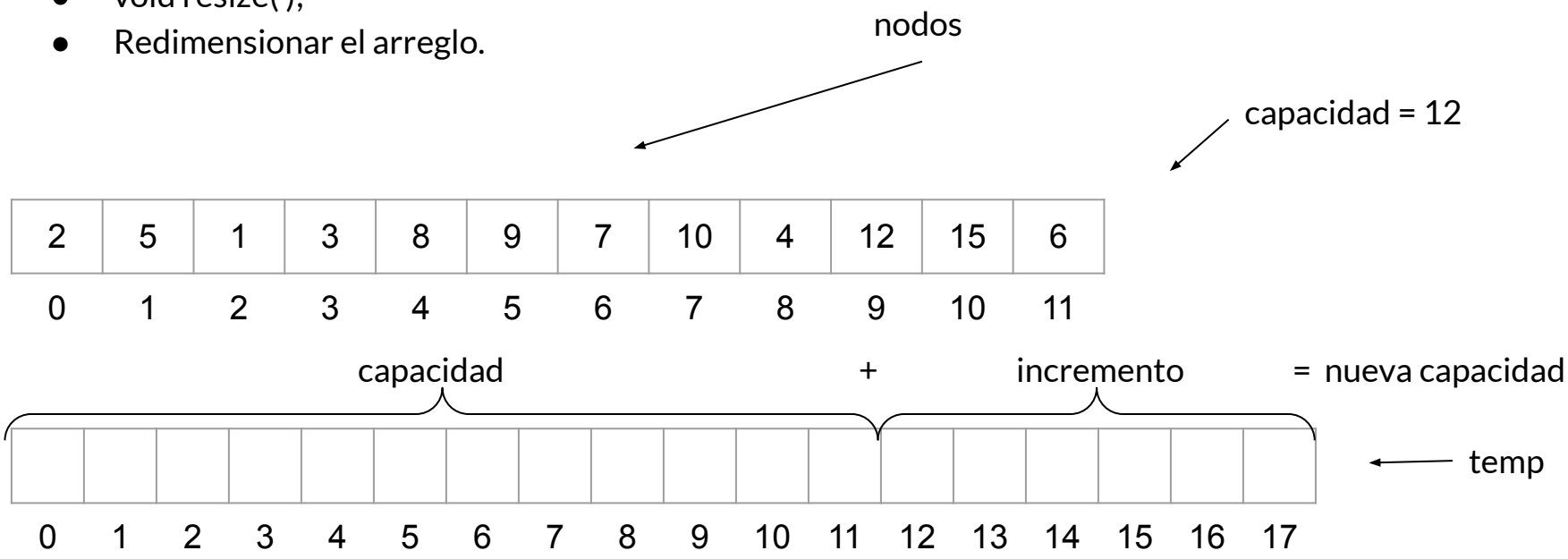
Lista circular basada en arreglo

- void resize( );
- Redimensionar el arreglo.



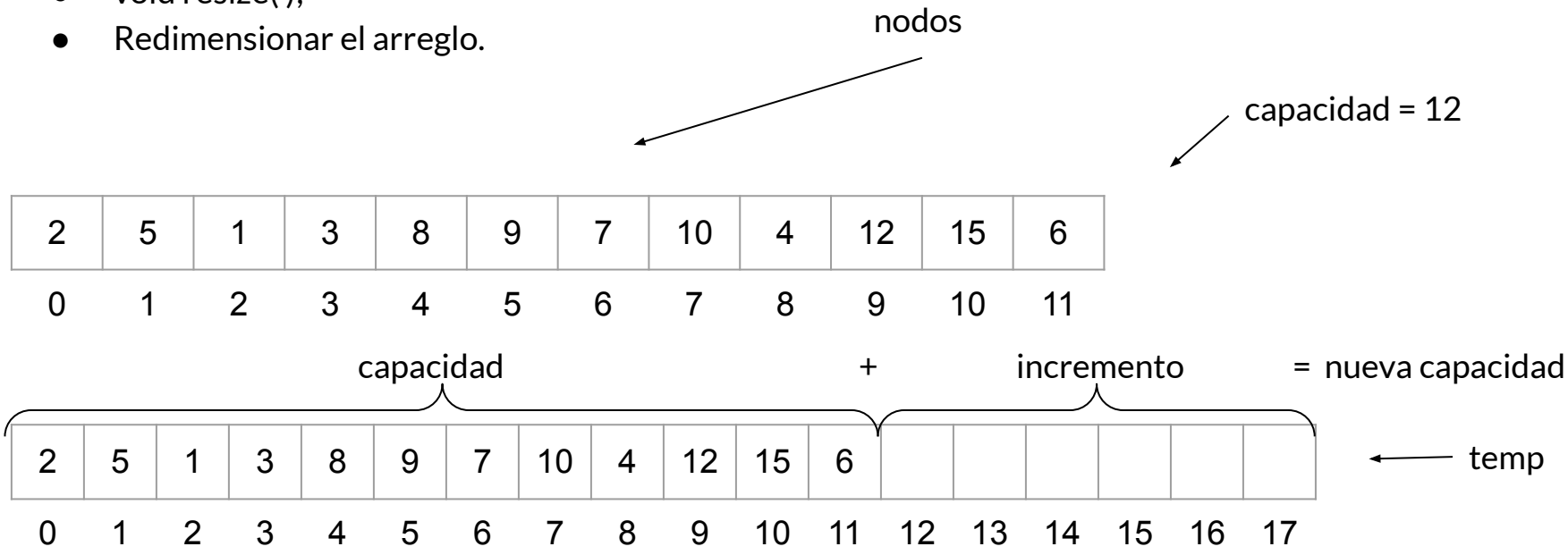
Lista circular basada en arreglo

- void resize( );
- Redimensionar el arreglo.



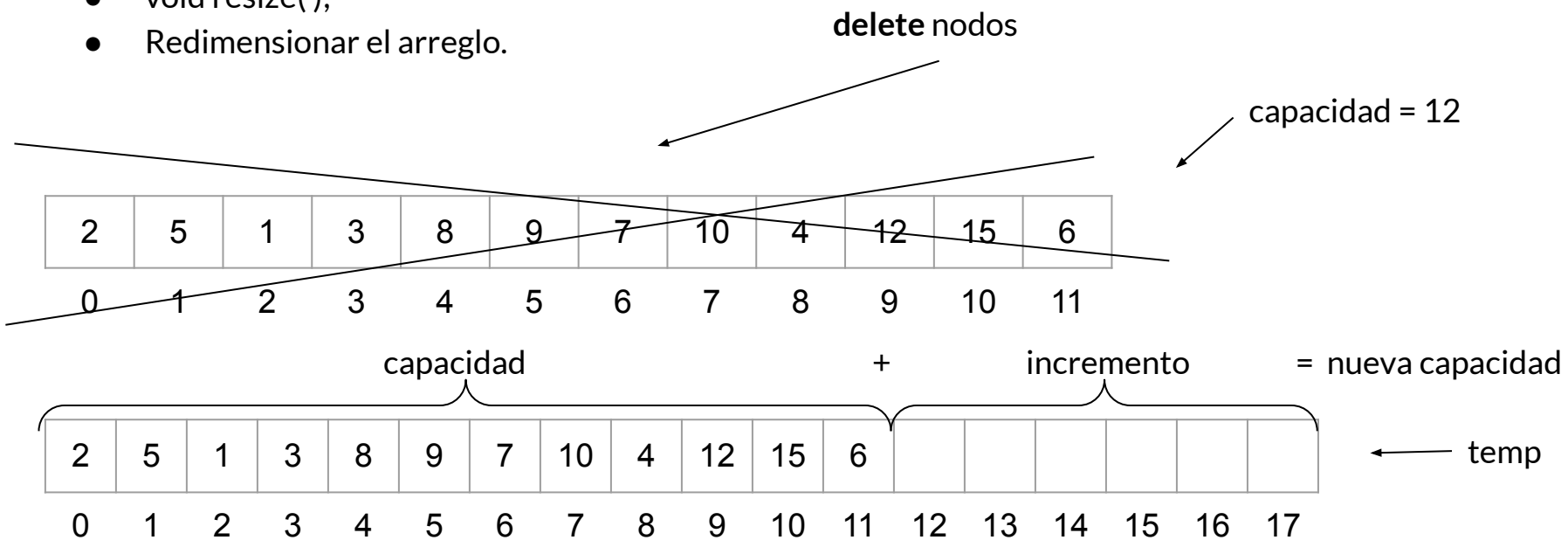
Lista circular basada en arreglo

- void resize( );
- Redimensionar el arreglo.



Lista circular basada en arreglo

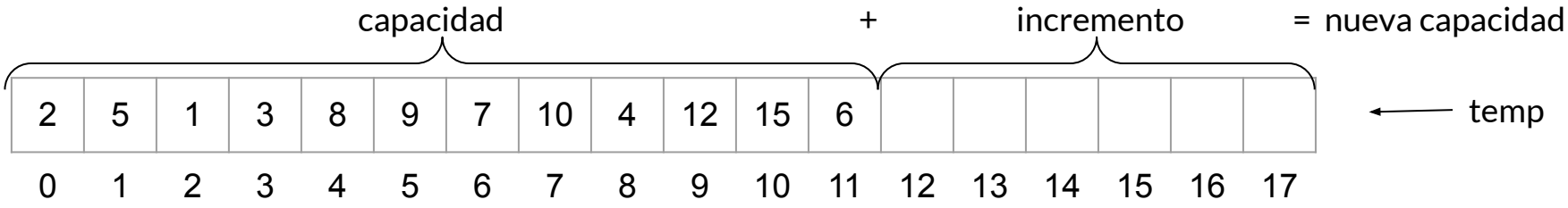
- void resize( );
- Redimensionar el arreglo.



Lista circular basada en arreglo

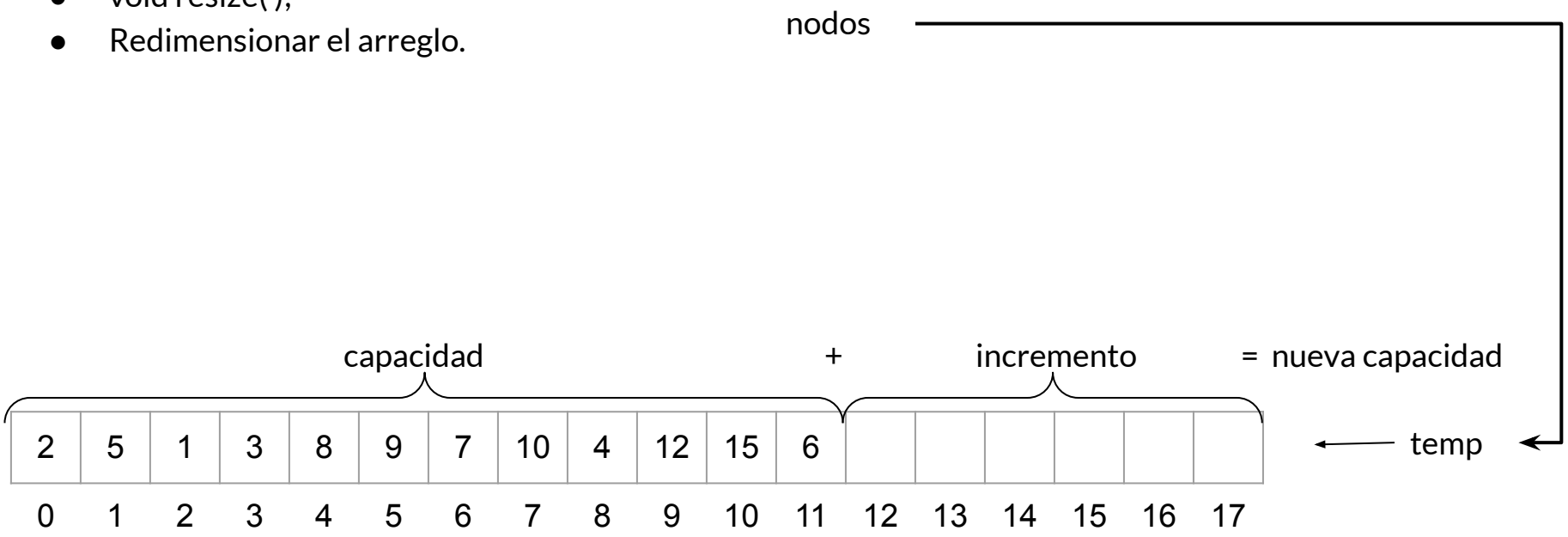
- void resize( );
- Redimensionar el arreglo.

nodos = null



Lista circular basada en arreglo

- void resize( );
- Redimensionar el arreglo.

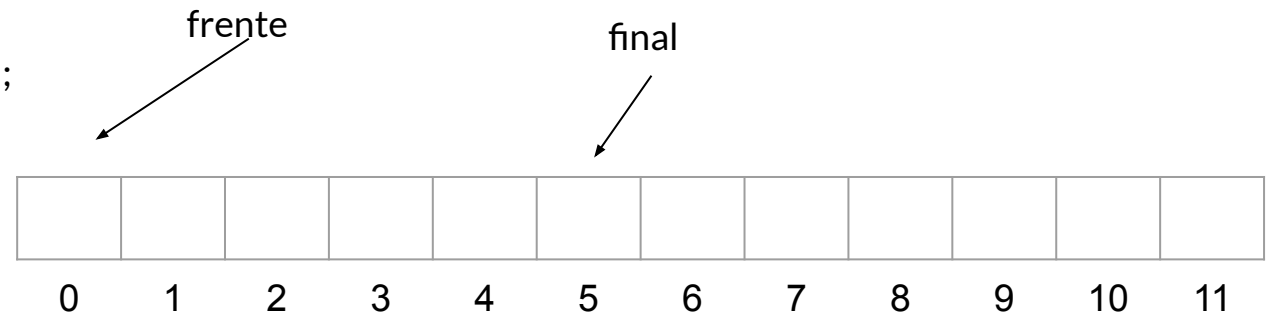


Lista circular basada en arreglo

- int siguiente(int r);
- Calcula la posición que sigue a *r*.

capacidad = 12

```
int siguiente(int r) {
        return (r+1) % capacidad;
}
```

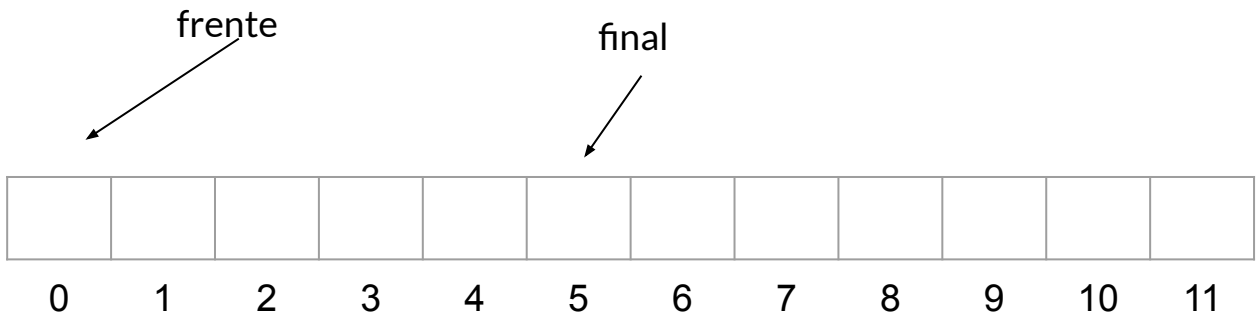


Lista circular basada en arreglo

- int siguiente(int r);
- Calcula la posición que sigue a *r*.

capacidad = 12

```
int siguiente(int r) {
        return (r+1) % capacidad;
}
```



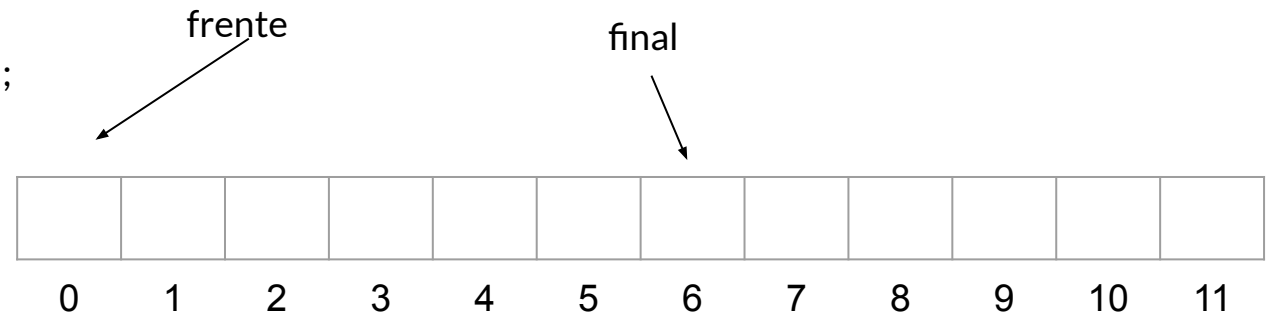
final = siguiente(final);
final = siguiente(5);
final = 6;

Lista circular basada en arreglo

- int siguiente(int r);
- Calcula la posición que sigue a *r*.

capacidad = 12

```
int siguiente(int r) {
        return (r+1) % capacidad;
}
```



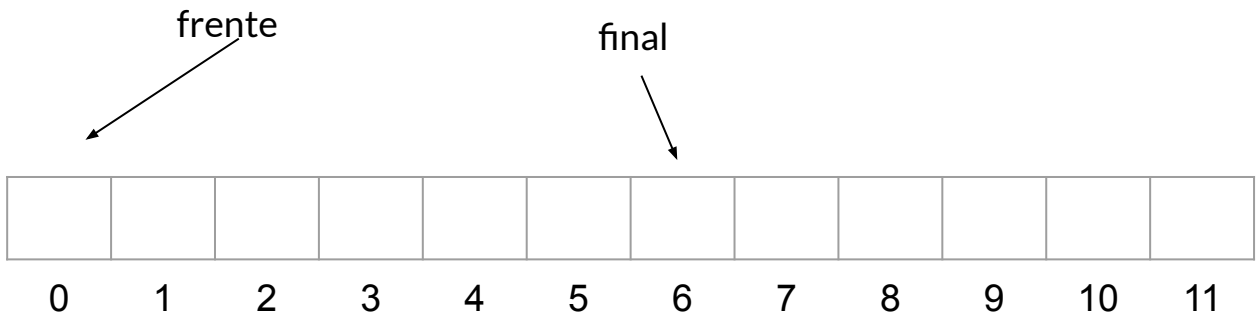
final = siguiente(final);
final = siguiente(5);
final = 6;

Lista circular basada en arreglo

- int siguiente(int r);
- Calcula la posición que sigue a *r*.

capacidad = 12

```
int siguiente(int r) {
        return (r+1) % capacidad;
}
```



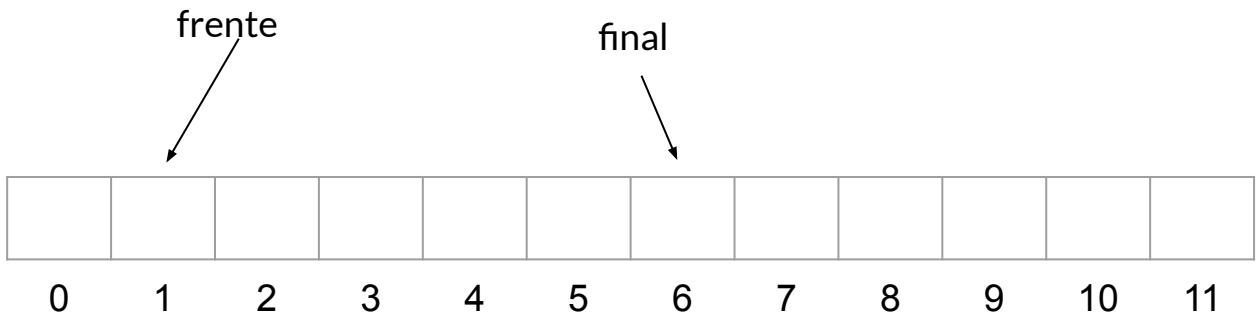
frente = siguiente(frente);
frente = siguiente(0);
frente = 1;

Lista circular basada en arreglo

- int siguiente(int r);
- Calcula la posición que sigue a *r*.

capacidad = 12

```
int siguiente(int r) {
        return (r+1) % capacidad;
}
```



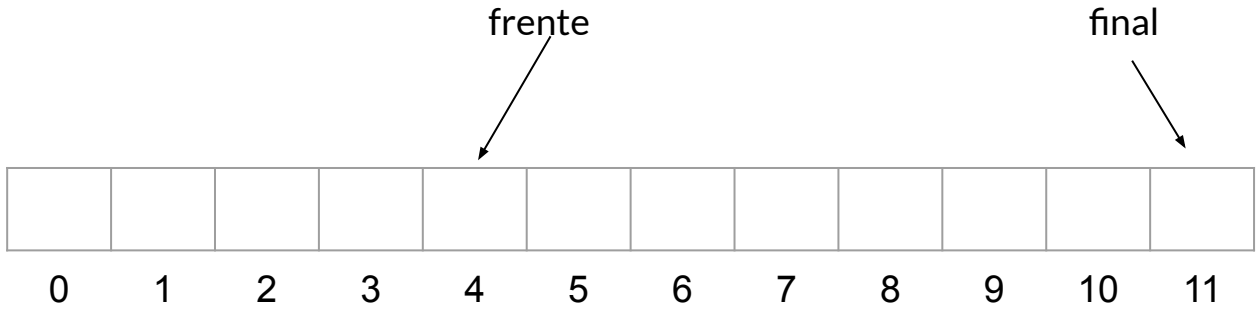
frente = siguiente(frente);
frente = siguiente(0);
frente = 1;

Lista circular basada en arreglo

- int siguiente(int r);
- Calcula la posición que sigue a *r*.

capacidad = 12

```
int siguiente(int r) {
        return (r+1) % capacidad;
}
```

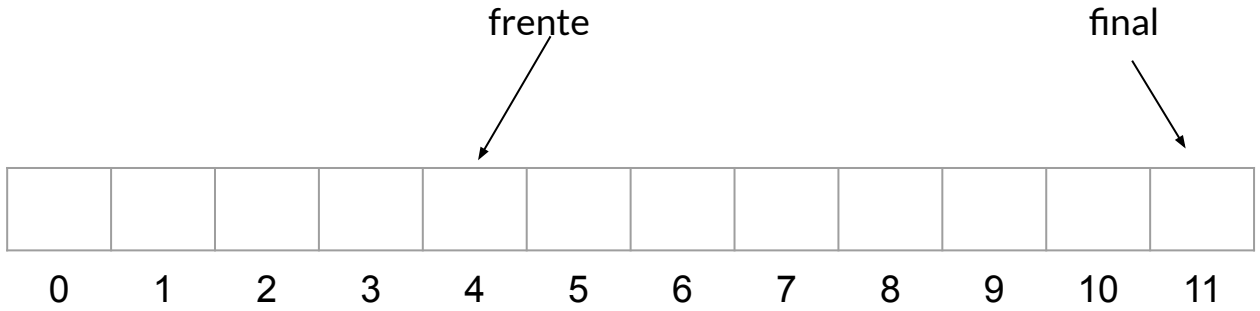


Lista circular basada en arreglo

- int siguiente(int r);
- Calcula la posición que sigue a *r*.

capacidad = 12

```
int siguiente(int r) {
        return (r+1) % capacidad;
}
```



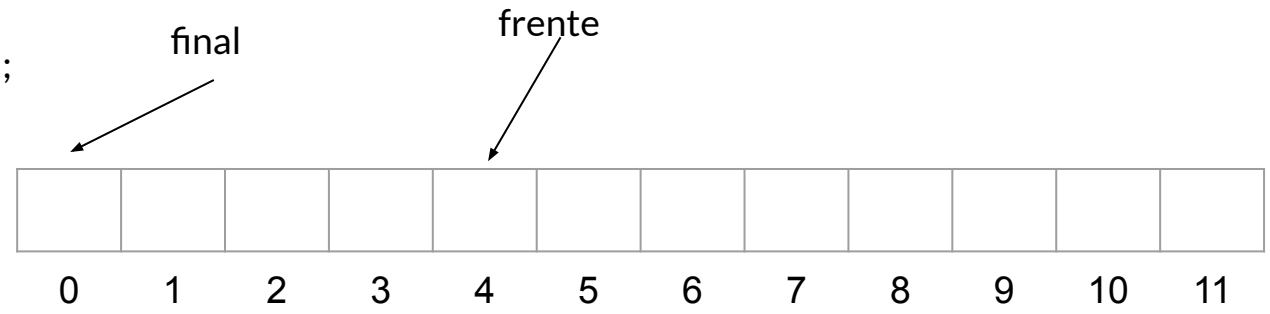
final = siguiente(final);
final = siguiente(11);
final = 0;

Lista circular basada en arreglo

- int siguiente(int r);
- Calcula la posición que sigue a *r*.

capacidad = 12

```
int siguiente(int r) {
        return (r+1) % capacidad;
}
```



final = siguiente(final);
final = siguiente(11);
final = 0;

Lista circular basada en arreglo

- void push_front(T d);
- Insertar al frente.

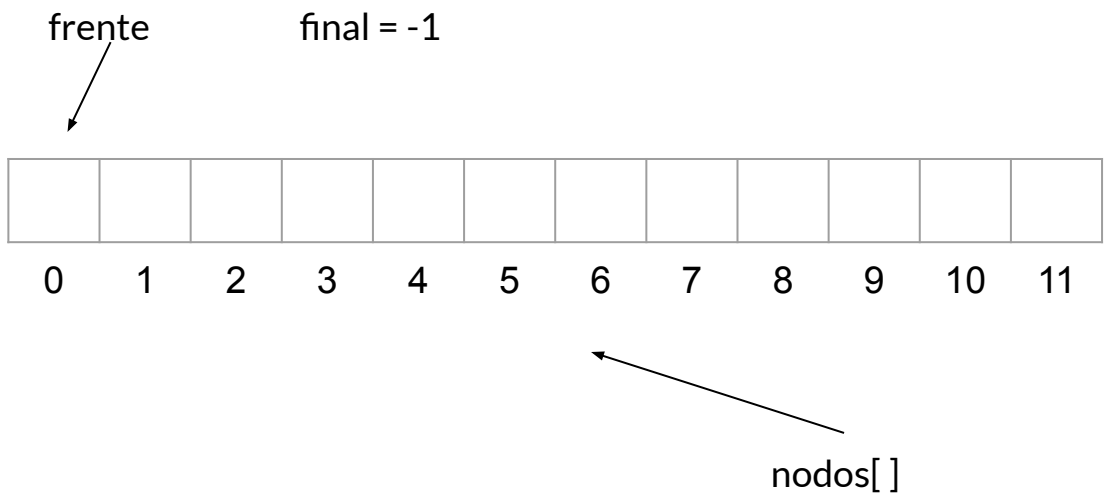
capacidad = 0

frente = 0 final = -1

Lista circular basada en arreglo

- void push_front(T d);
- Insertar al frente.

capacidad = 12

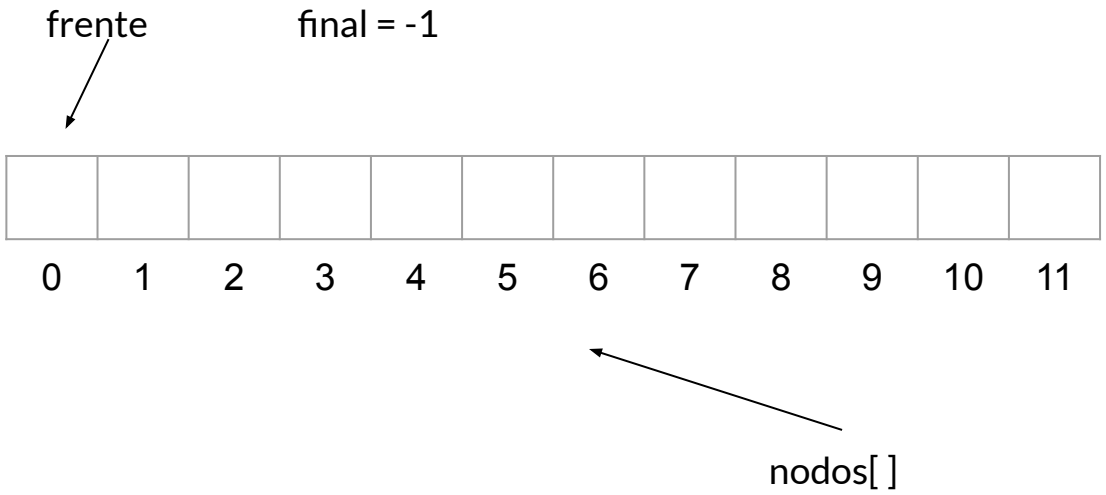


Lista circular basada en arreglo

- void push_front(T d);
- Insertar al frente.

capacidad = 12

Si la lista está vacía (tam == 0)
entonces corresponde a push_back( )



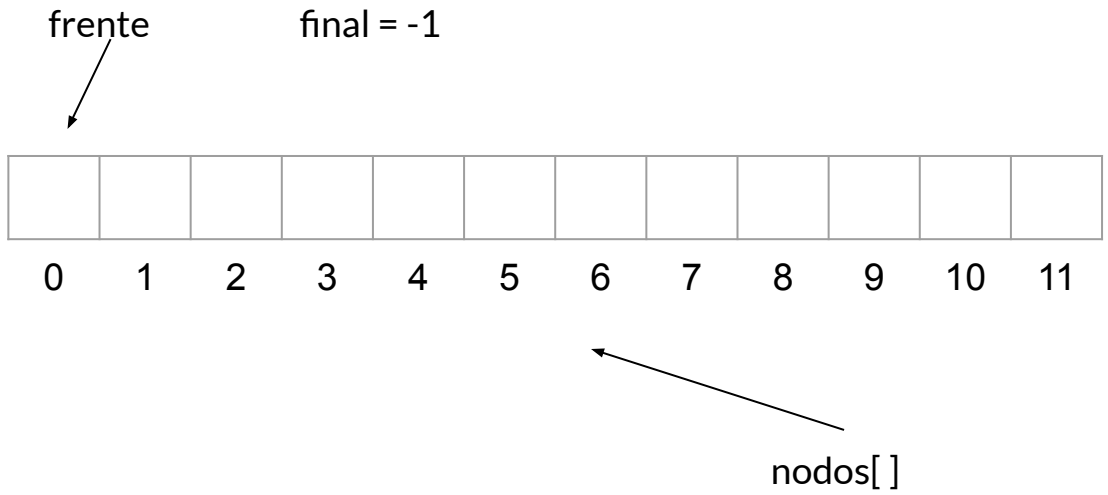
Lista circular basada en arreglo

- void push_front(T d);
- Insertar al frente.

capacidad = 12

Si la lista está vacía (tam == 0)
entonces corresponde a push_back( )

Calculamos final = siguiente(final) = 0



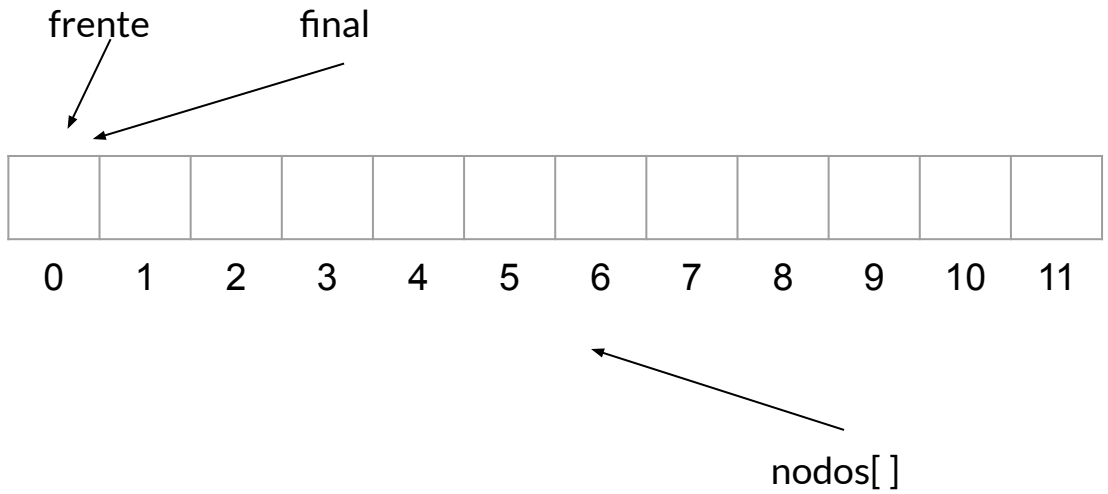
Lista circular basada en arreglo

- void push_front(T d);
- Insertar al frente.

capacidad = 12

Si la lista está vacía (tam == 0)
entonces corresponde a push_back( )

Calculamos final = siguiente(final) = 0



Lista circular basada en arreglo

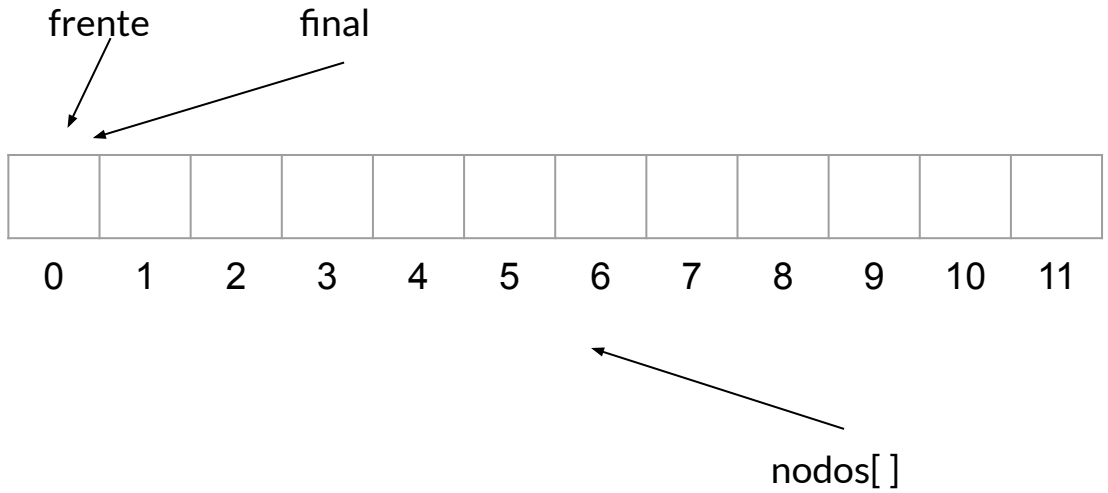
- void push_front(T d);
- Insertar al frente.

capacidad = 12

Si la lista está vacía (tam == 0)
entonces corresponde a push_back( )

Calculamos final = siguiente(final) = 0

Hacemos nodos[final] = d;

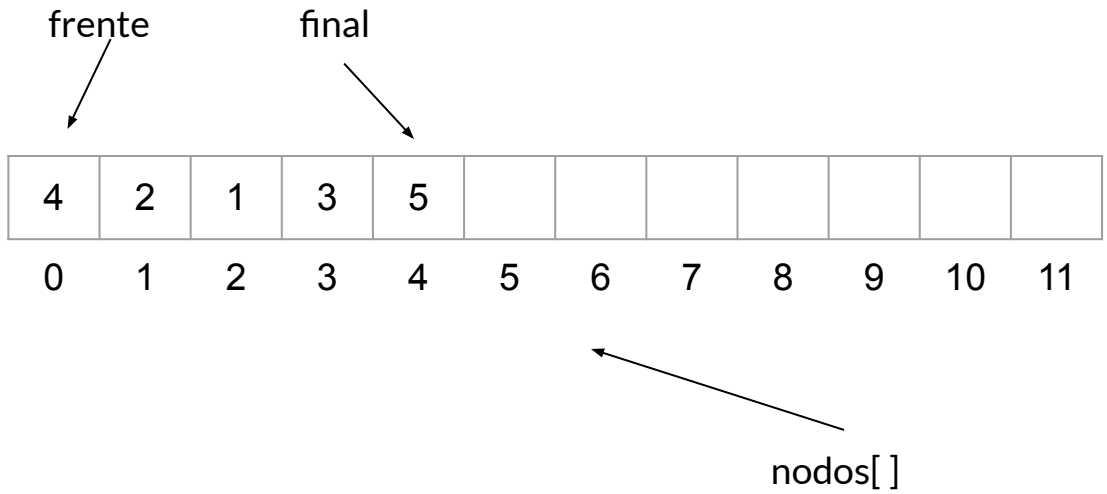


Lista circular basada en arreglo

- void push_front(T d);
- Insertar al frente.

capacidad = 12

Si la lista **no** está vacía (tam != 0) entonces recorremos todos los elementos una posición a la derecha

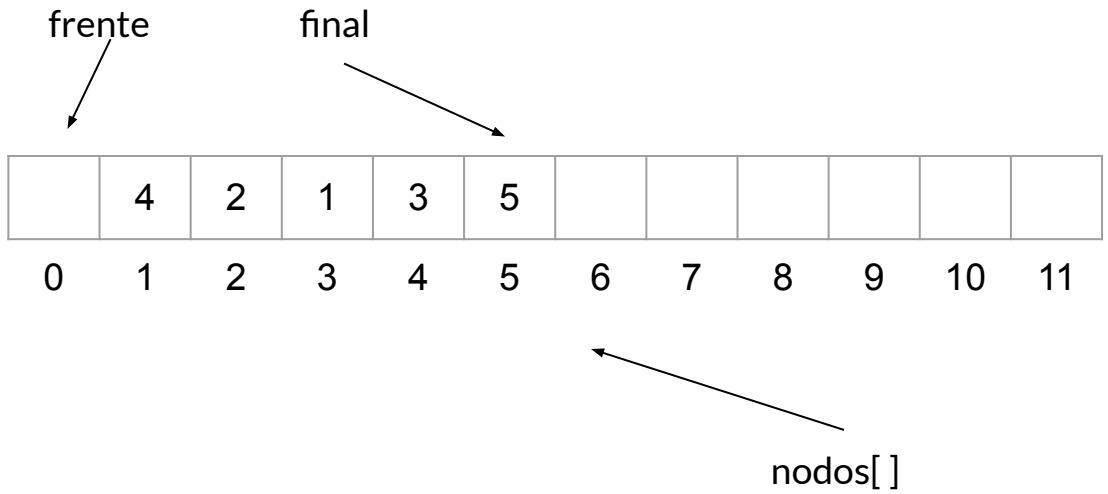


Lista circular basada en arreglo

- void push_front(T d);
- Insertar al frente.

capacidad = 12

Si la lista **no** está vacía (tam != 0) entonces recorremos todos los elementos una posición a la derecha



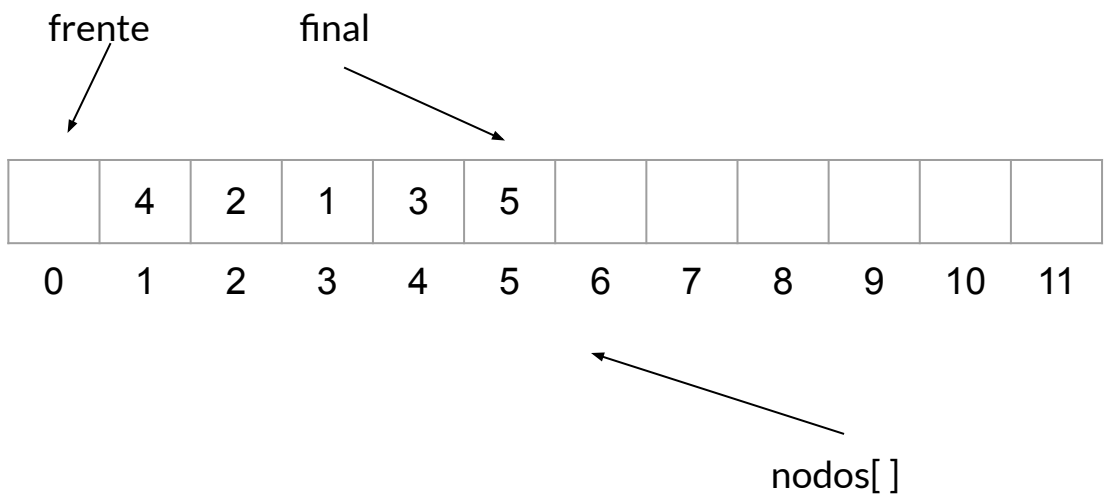
Lista circular basada en arreglo

- void push_front(T d);
- Insertar al frente.

capacidad = 12

Si la lista **no** está vacía (tam != 0) entonces recorremos todos los elementos una posición a la derecha

Hacemos nodos[frente] = d;



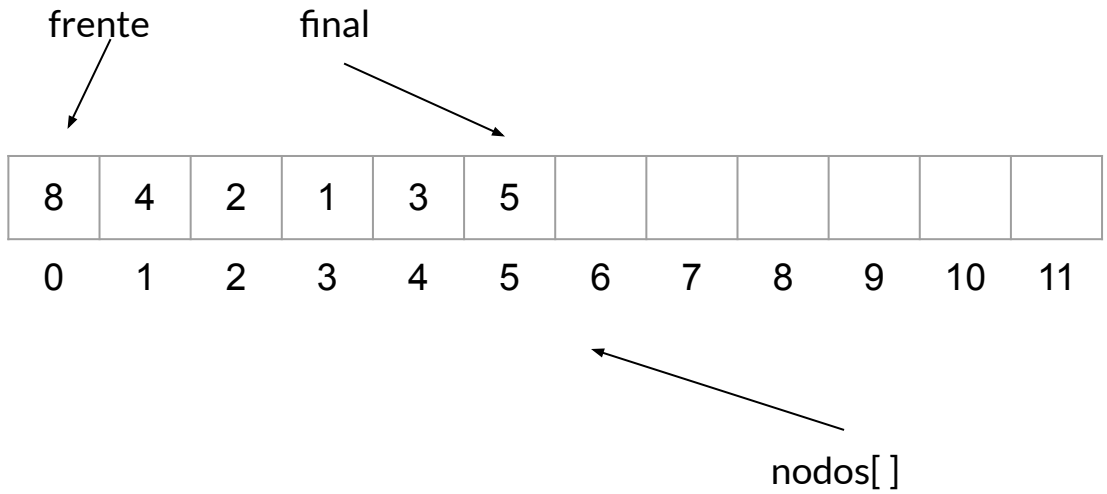
Lista circular basada en arreglo

- void push_front(T d);
- Insertar al frente.

capacidad = 12

Si la lista **no** está vacía (tam != 0) entonces recorremos todos los elementos una posición a la derecha

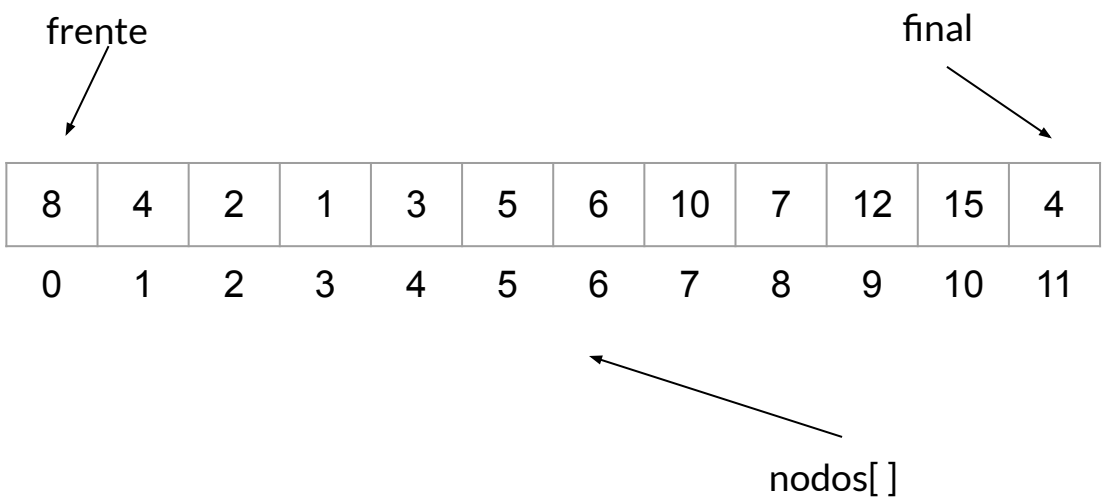
Hacemos nodos[frente] = d;



Lista circular basada en arreglo

- void push_front(T d);
- Insertar al frente.

Si la lista **no** está vacía (tam != 0) y está **llena**, entonces primero llamamos al método **resize**

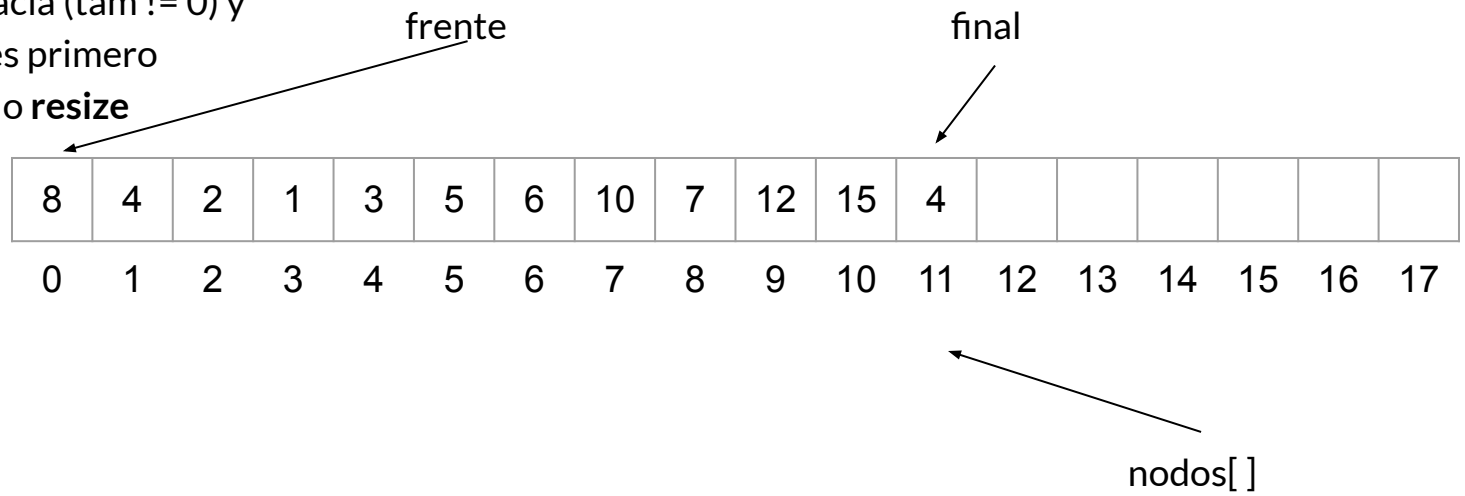


Lista circular basada en arreglo

- void push_front(T d);
- Insertar al frente.

capacidad = 18

Si la lista **no** está vacía (tam != 0) y está **llena**, entonces primero llamamos al método **resize**

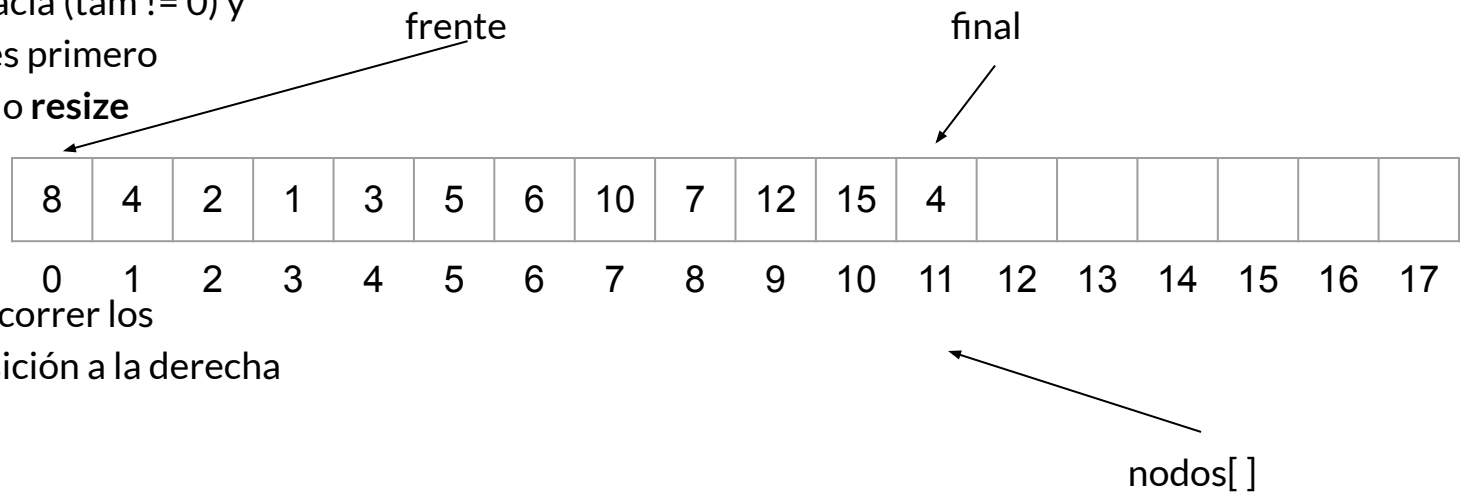


Lista circular basada en arreglo

- void push_front(T d);
- Insertar al frente.

capacidad = 18

Si la lista **no** está vacía (tam != 0) y está **llena**, entonces primero llamamos al método **resize**



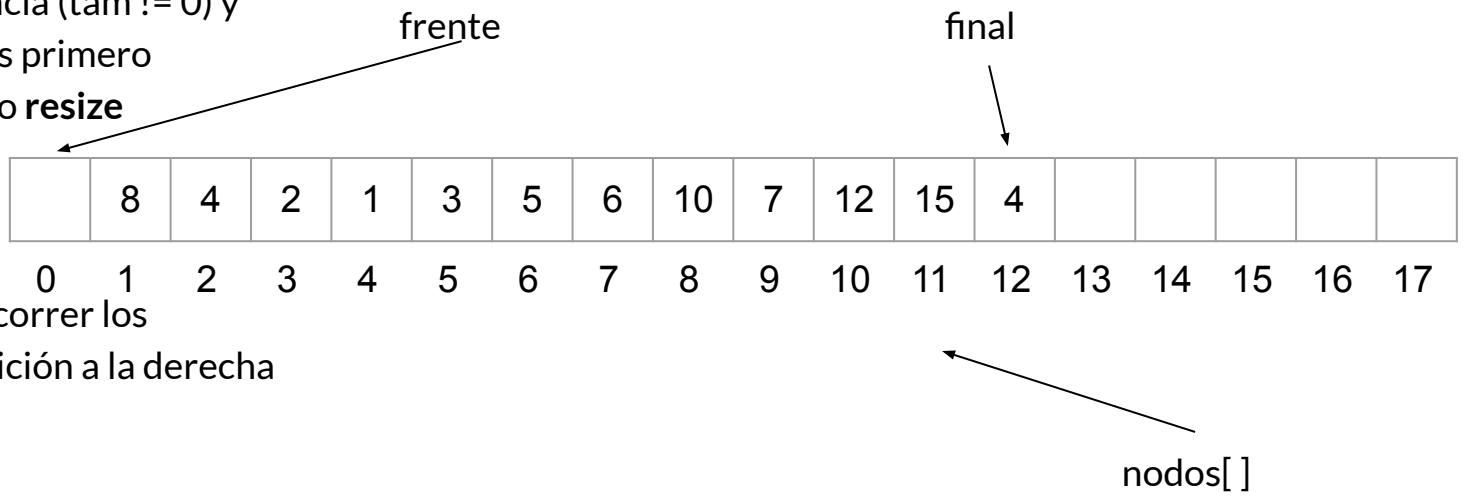
Ahora podemos recorrer los elementos una posición a la derecha

Lista circular basada en arreglo

- void push_front(T d);
- Insertar al frente.

capacidad = 18

Si la lista **no** está vacía (tam != 0) y está **llena**, entonces primero llamamos al método **resize**



Ahora podemos recorrer los elementos una posición a la derecha

Lista circular basada en arreglo

- void push_front(T d);
- Insertar al frente.

capacidad = 18

Si la lista **no** está vacía (tam != 0) y está **llena**, entonces primero llamamos al método **resize**

Ahora podemos recorrer los elementos una posición a la derecha

Hacemos nodos[frente] = d;

Lista circular basada en arreglo

- void push_front(T d);
- Insertar al frente.

capacidad = 18

Si la lista **no** está vacía (tam != 0) y está **llena**, entonces primero llamamos al método **resize**

Ahora podemos recorrer los elementos una posición a la derecha

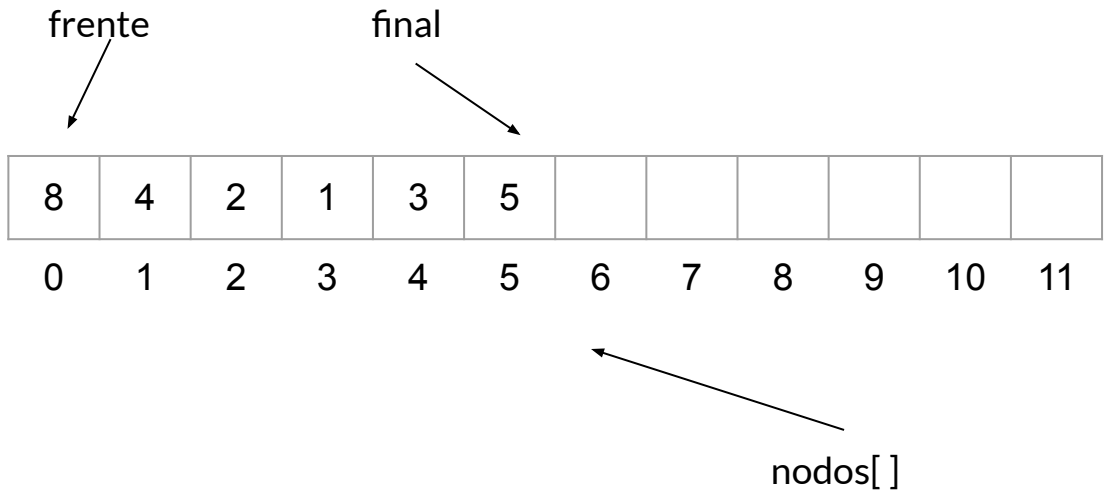
Hacemos nodos[frente] = d;

Lista circular basada en arreglo

- void push_back(T d);
- Insertar al final.

capacidad = 12

Si la **lista** está **llena** (tam == capacidad) entonces llamamos el método **resize**



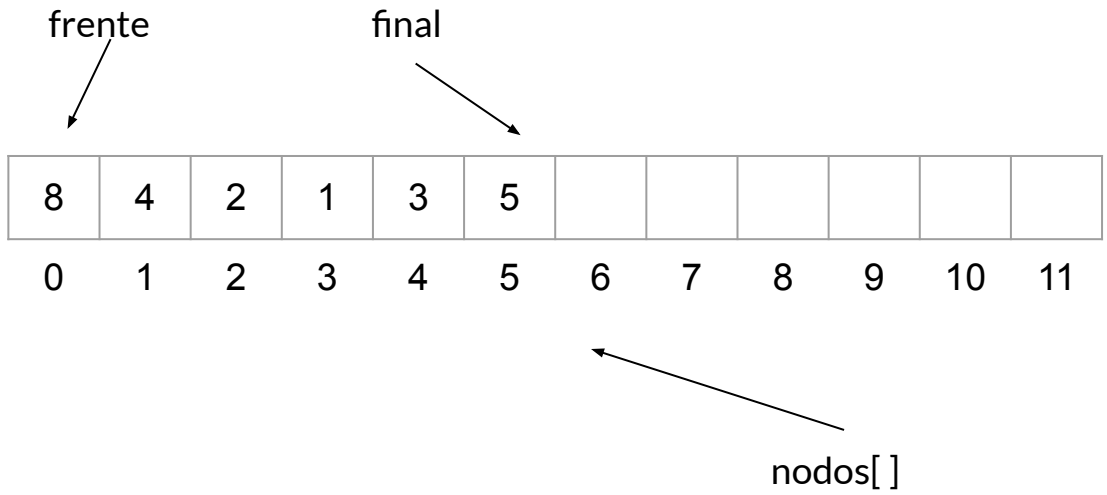
Lista circular basada en arreglo

- void push_back(T d);
- Insertar al final.

capacidad = 12

Si la **lista** está **llena** (tam == capacidad) entonces llamamos el método **resize**

Calculamos la nueva posición final como final = siguiente(final) = 6



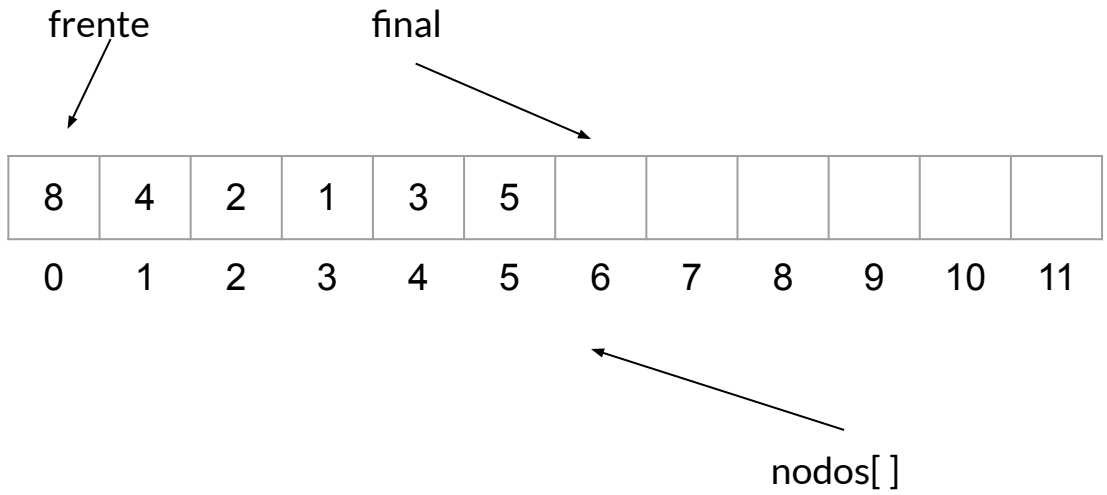
Lista circular basada en arreglo

- void push_back(T d);
- Insertar al final.

capacidad = 12

Si la **lista** está **llena** (tam == capacidad) entonces llamamos el método **resize**

Calculamos la nueva posición final como final = siguiente(final) = 6



Lista circular basada en arreglo

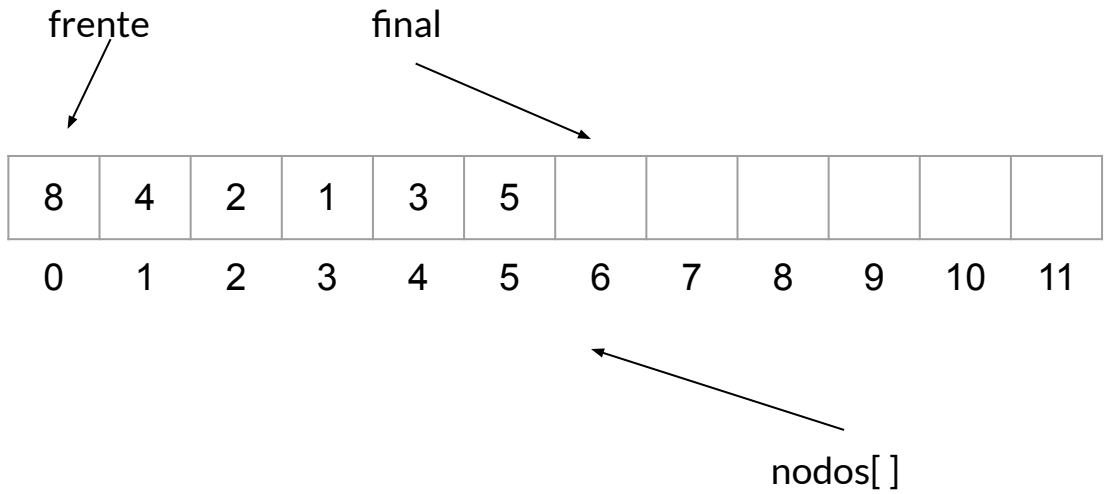
- void push_back(T d);
- Insertar al final.

capacidad = 12

Si la **lista** está **llena** (tam == capacidad) entonces llamamos el método **resize**

Calculamos la nueva posición final como final = siguiente(final) = 6

Hacemos nodos[final] = d;



Lista circular basada en arreglo

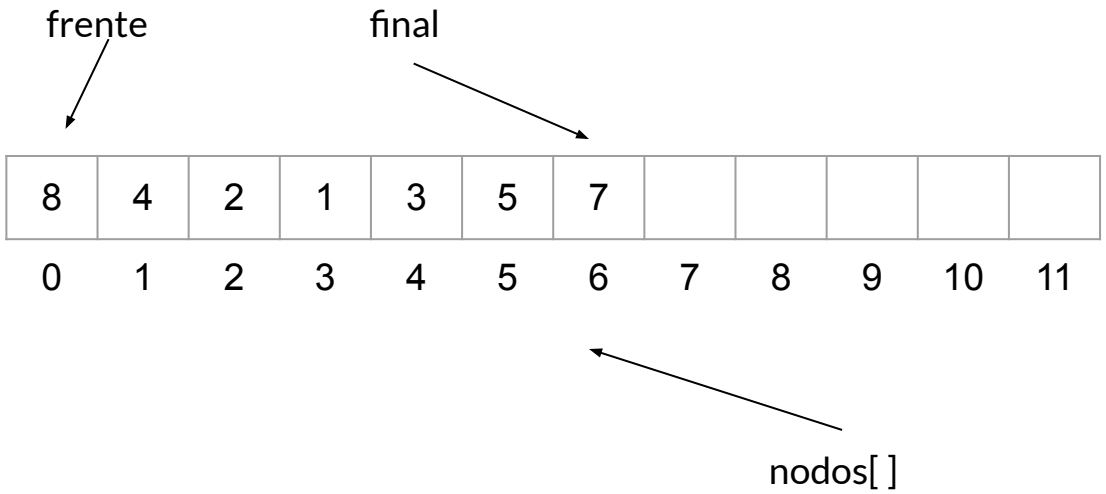
- void push_back(T d);
- Insertar al final.

capacidad = 12

Si la **lista** está **llena** (tam == capacidad) entonces llamamos el método **resize**

Calculamos la nueva posición final como final = siguiente(final) = 6

Hacemos nodos[final] = d;

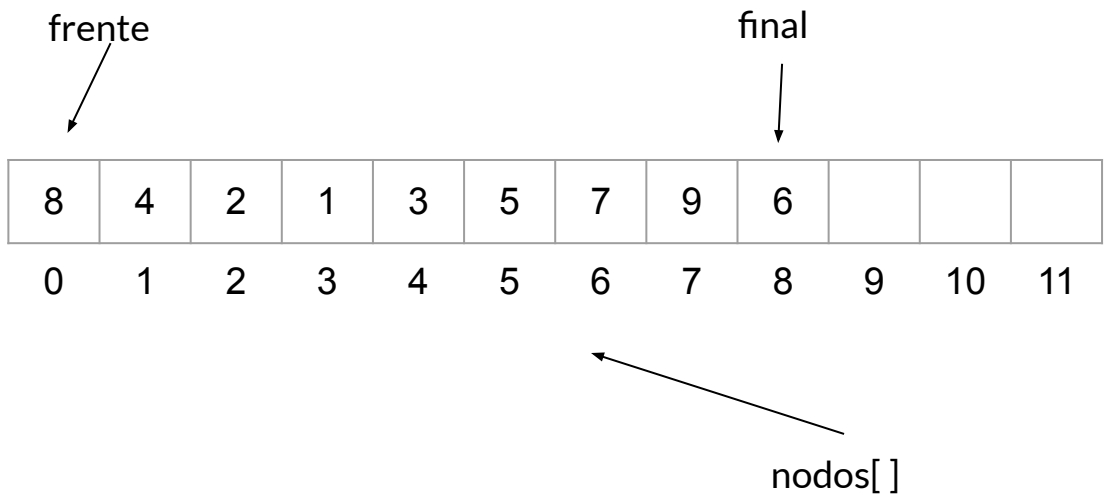


Lista circular basada en arreglo

- void push(T d, int i);
- Insertar en la i-ésima posición..

capacidad = 12

int i = 3;



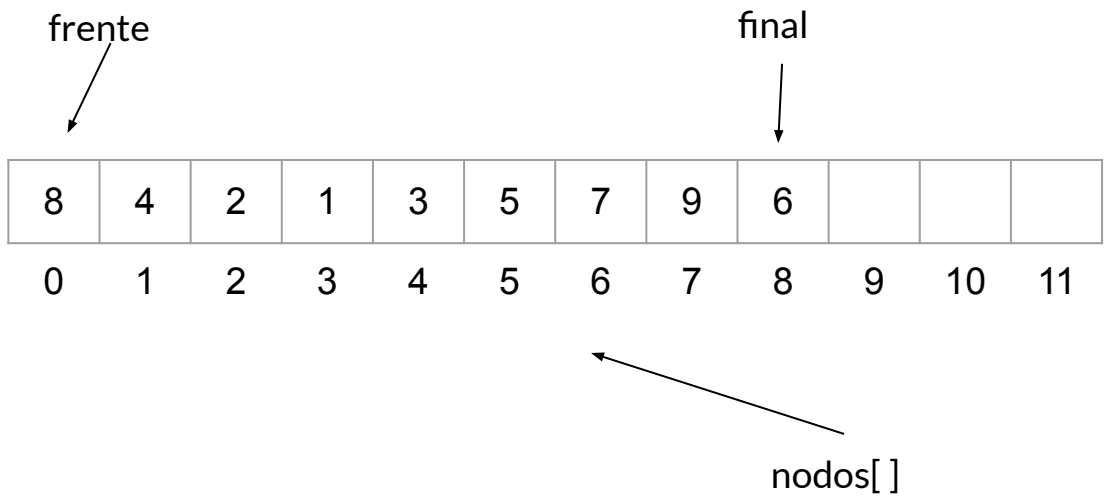
Lista circular basada en arreglo

- void push(T d, int i);
- Insertar en la i-ésima posición..

capacidad = 12

int i = 3;

Si la **lista** está **llena** (tam == capacidad) entonces llamamos el método **resize**



Lista circular basada en arreglo

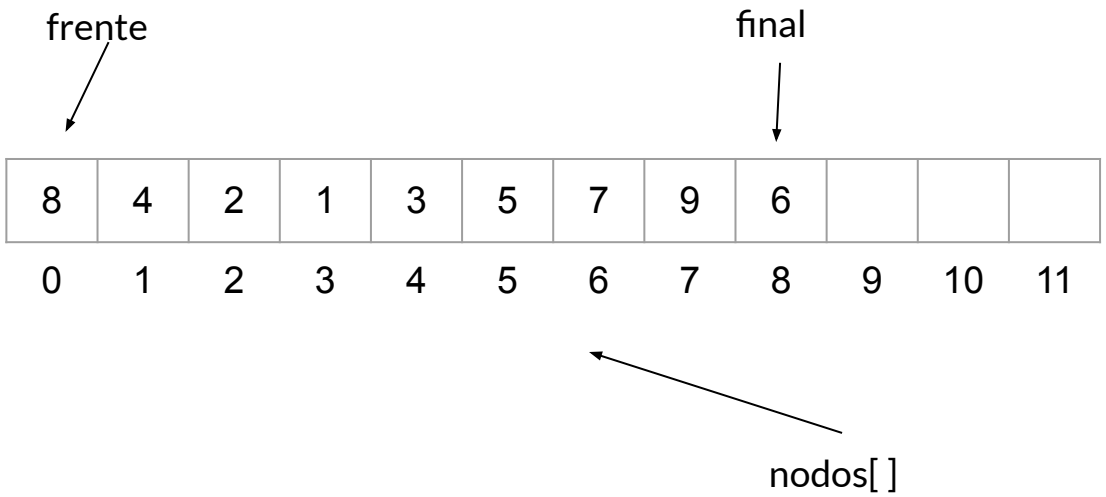
- void push(T d, int i);
- Insertar en la i-ésima posición..

capacidad = 12

int i = 3;

Si la **lista** está **llena** (tam == capacidad) entonces llamamos el método **resize**

Movemos los elementos del arreglo a partir de la **posición i**, una posición a la **derecha**



Lista circular basada en arreglo

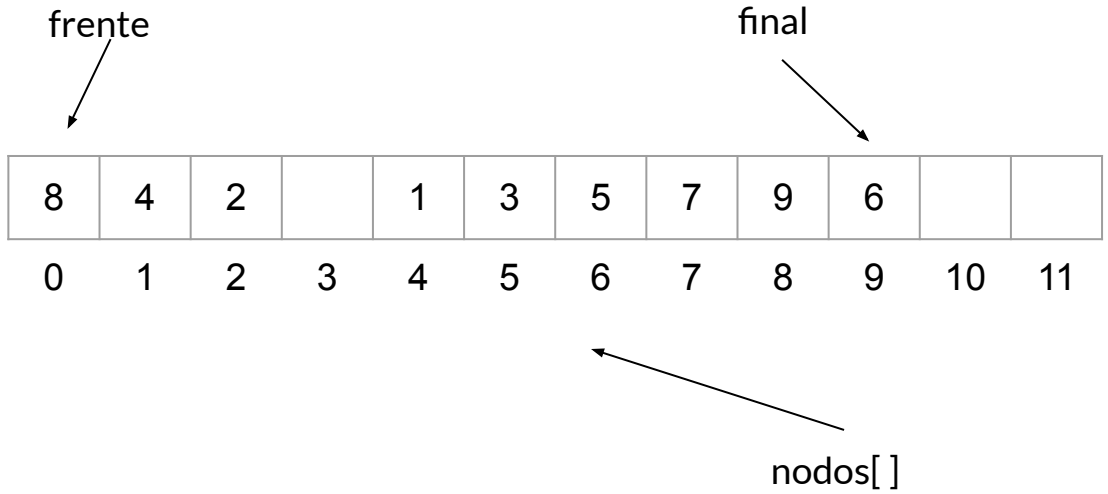
- void push(T d, int i);
- Insertar en la i-ésima posición..

capacidad = 12

int i = 3;

Si la **lista** está **llena** (tam == capacidad) entonces llamamos el método **resize**

Movemos los elementos del arreglo a partir de la **posición i**, una posición a la **derecha**



Lista circular basada en arreglo

- void push(T d, int i);
- Insertar en la i-ésima posición..

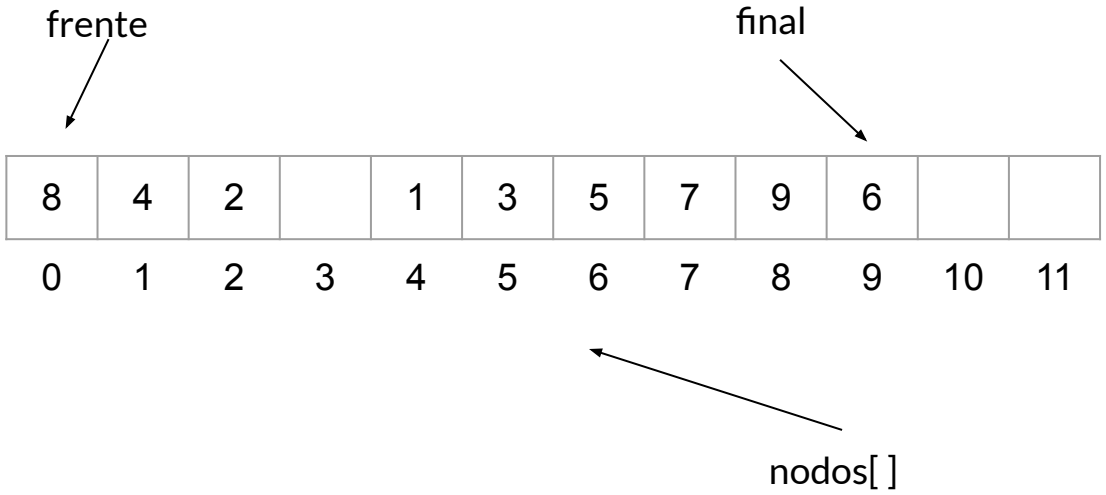
capacidad = 12

int i = 3;

Si la **lista** está **llena** (tam == capacidad) entonces llamamos el método **resize**

Movemos los elementos del arreglo a partir de la **posición i**, una posición a la **derecha**

Hacemos nodos[i] = d;



Lista circular basada en arreglo

- void push(T d, int i);
- Insertar en la i-ésima posición..

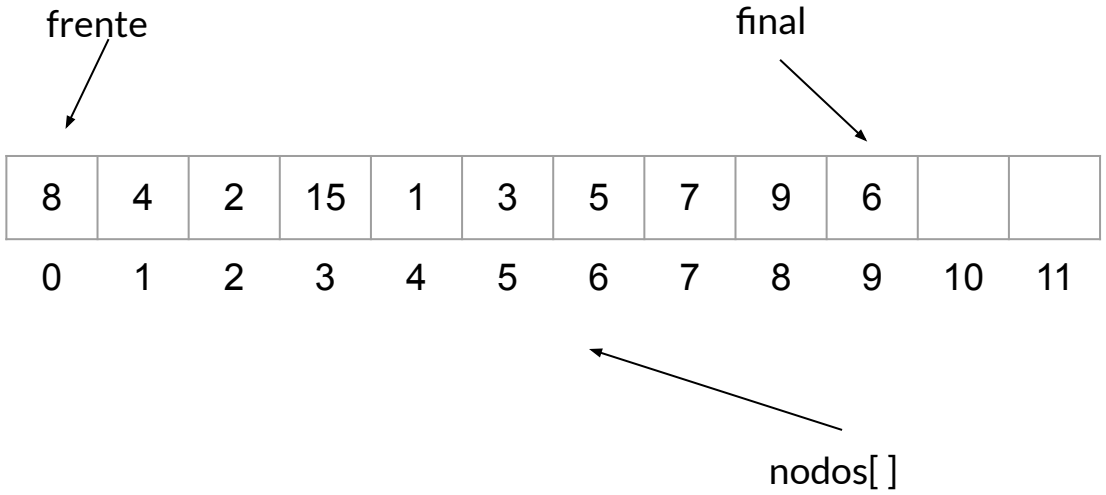
capacidad = 12

int i = 3;

Si la **lista** está **llena** (tam == capacidad) entonces llamamos el método **resize**

Movemos los elementos del arreglo a partir de la **posición i**, una posición a la **derecha**

Hacemos nodos[i] = d;



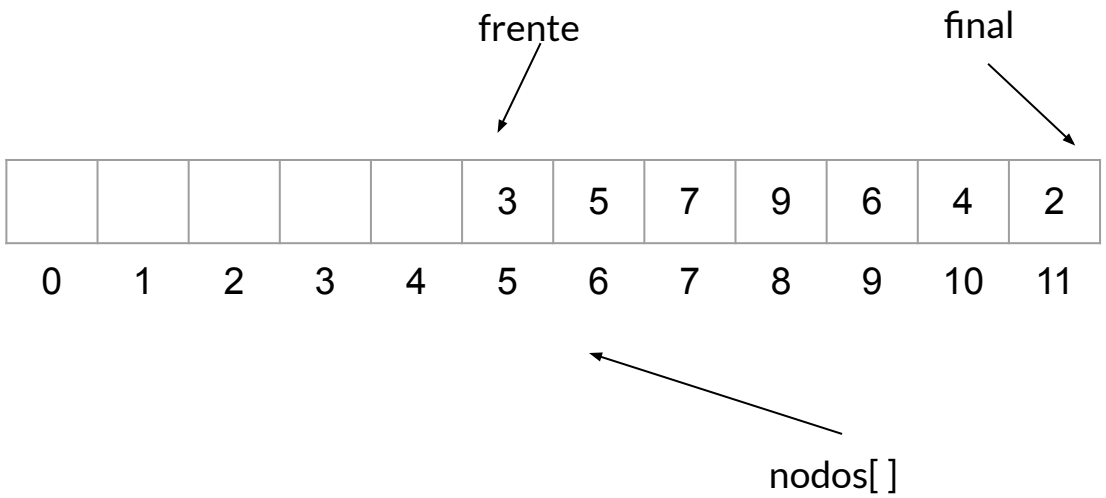
Lista circular basada en arreglo

- void push(T d, int i);
- Insertar en la i-ésima posición..

capacidad = 12

int i = 3;

Si la **lista** está **llena** (tam == capacidad) entonces llamamos el método **resize**



Lista circular basada en arreglo

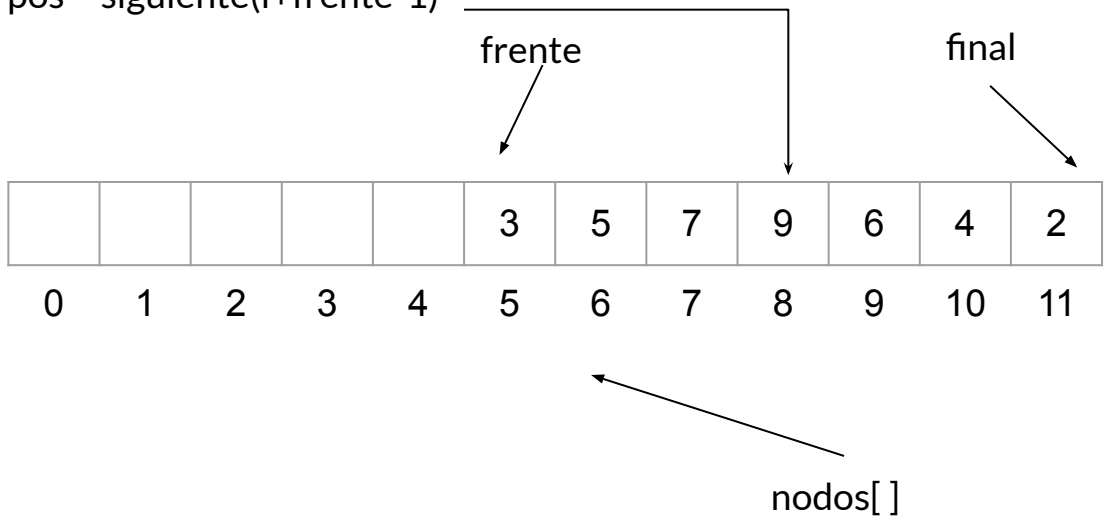
- void push(T d, int i);
- Insertar en la i-ésima posición..

capacidad = 12

int i = 3;

pos = siguiente(i+frente-1)

Si la **lista** está **llena** (tam == capacidad) entonces llamamos el método **resize**



Lista circular basada en arreglo

- void push(T d, int i);
- Insertar en la i-ésima posición..

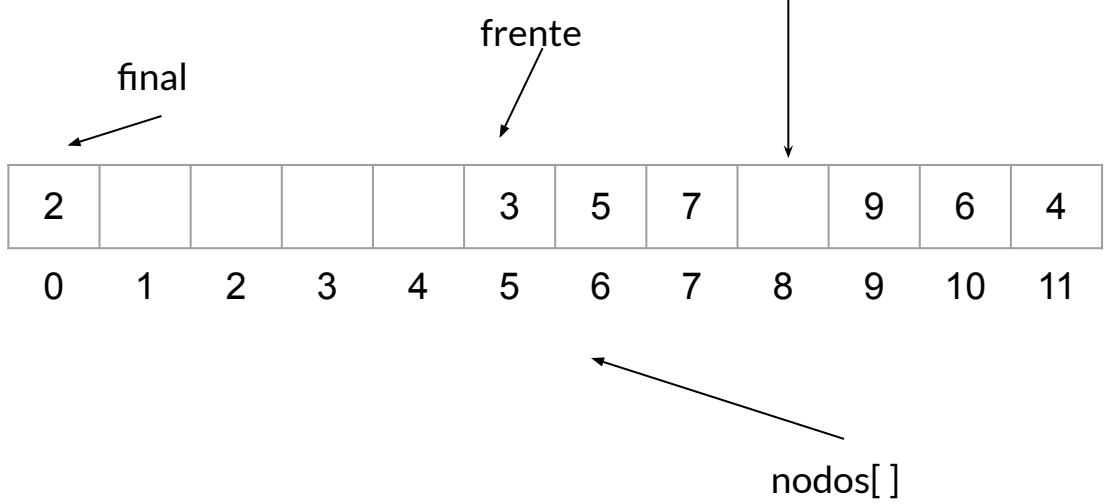
capacidad = 12

int i = 3;

pos = siguiente(i+frente-1)

Si la **lista** está **llena** (tam == capacidad) entonces llamamos el método **resize**

Movemos los elementos del arreglo a partir de la **posición i**, una posición a la **derecha**



Lista circular basada en arreglo

- void push(T d, int i);
- Insertar en la i-ésima posición..

capacidad = 12

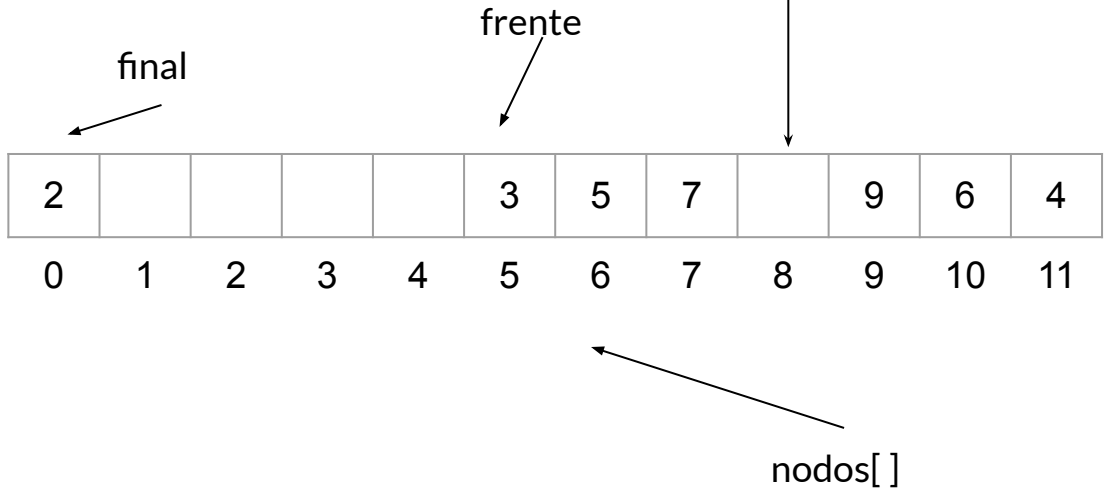
int i = 3;

pos = siguiente(i+frente-1)

Si la **lista** está **llena** (tam == capacidad) entonces llamamos el método **resize**

Movemos los elementos del arreglo a partir de la **posición i**, una posición a la **derecha**

Hacemos nodos[pos] = d;



Lista circular basada en arreglo

- void push(T d, int i);
- Insertar en la i-ésima posición..

capacidad = 12

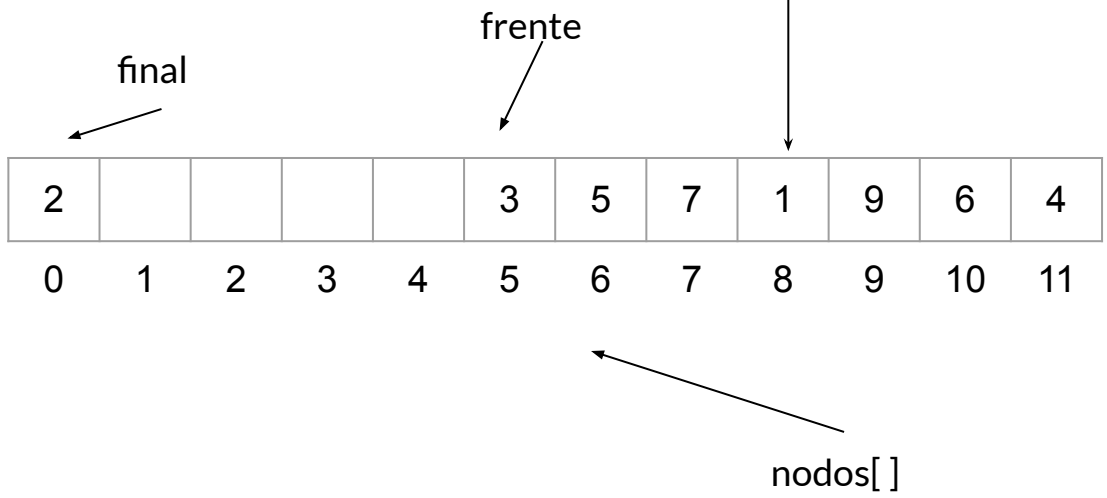
int i = 3;

pos = siguiente(i+frente-1)

Si la **lista** está **llena** (tam == capacidad) entonces llamamos el método **resize**

Movemos los elementos del arreglo a partir de la **posición i**, una posición a la **derecha**

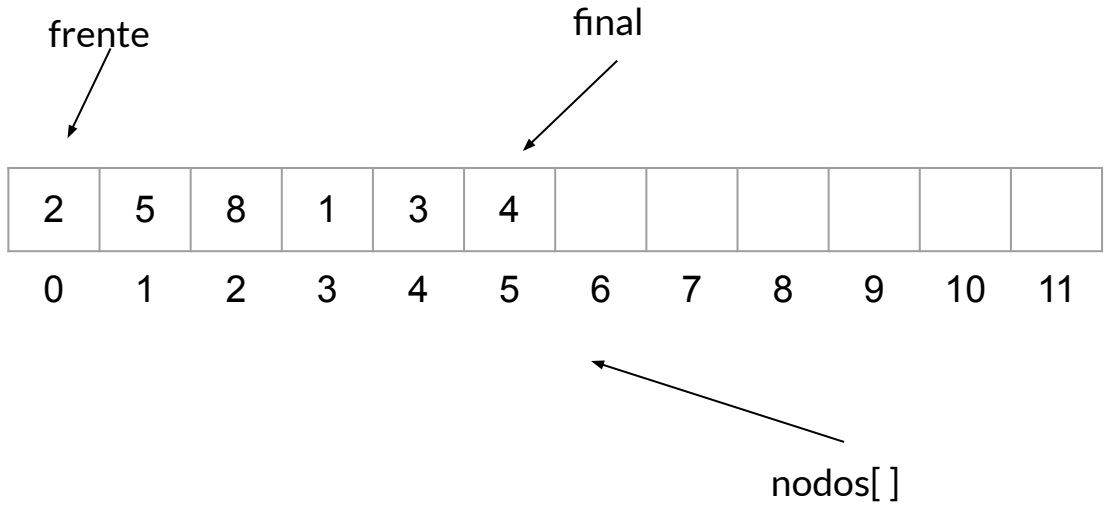
Hacemos nodos[pos] = d;



Lista circular basada en arreglo

- void pop_front( );
- Eliminar al frente.

capacidad = 12

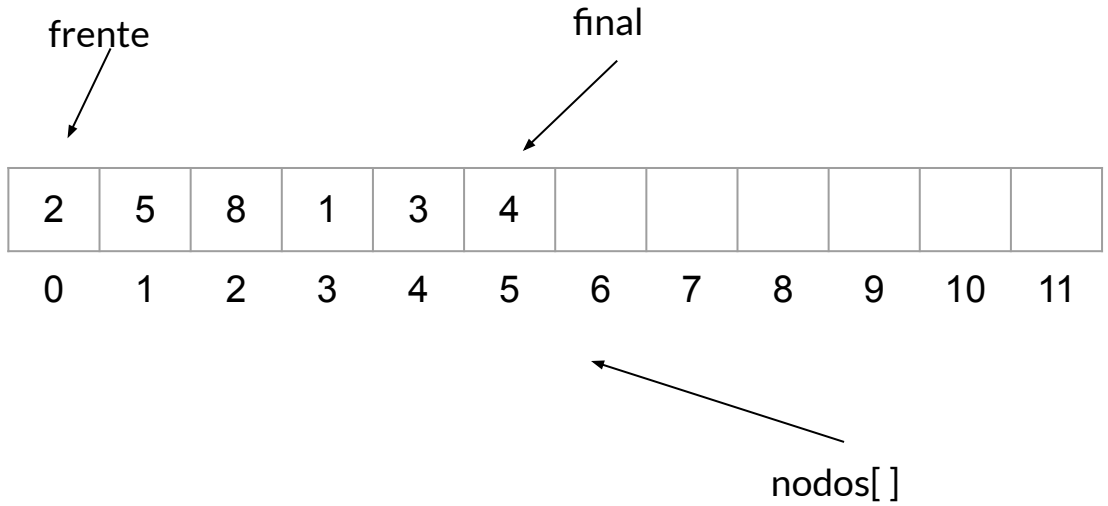


Lista circular basada en arreglo

- void pop_front( );
- Eliminar al frente.

frente = siguiente(frente);

capacidad = 12



Lista circular basada en arreglo

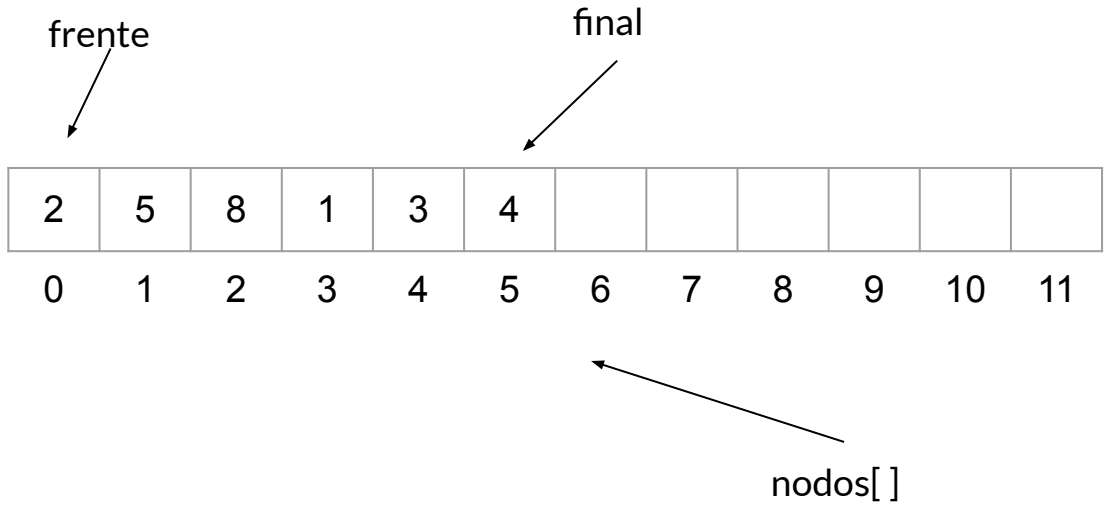
- void pop_front( );
- Eliminar al frente.

capacidad = 12

frente = siguiente(frente);

frente = siguiente(0);

frente = 1;



Lista circular basada en arreglo

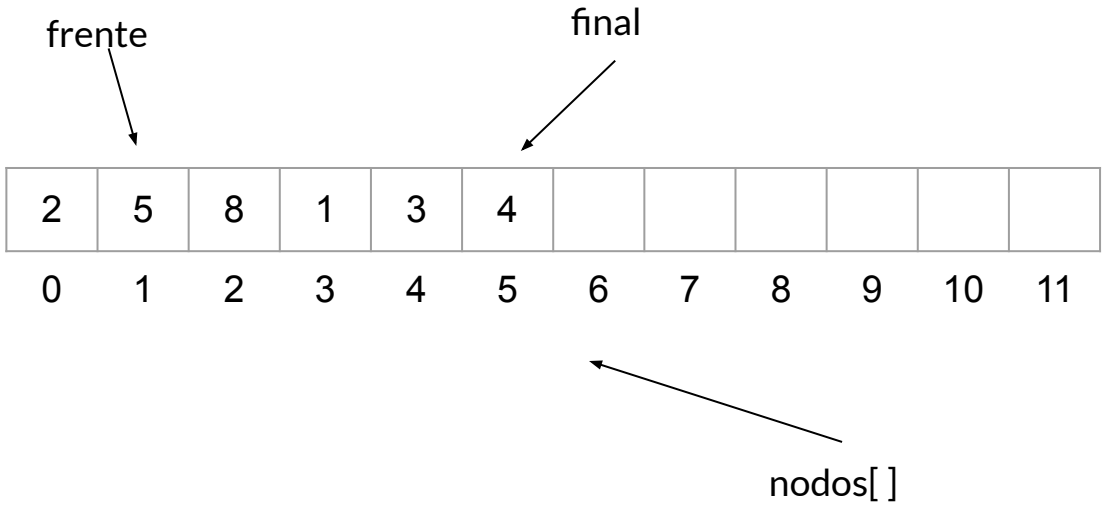
- void pop_front( );
- Eliminar al frente.

capacidad = 12

frente = siguiente(frente);

frente = siguiente(0);

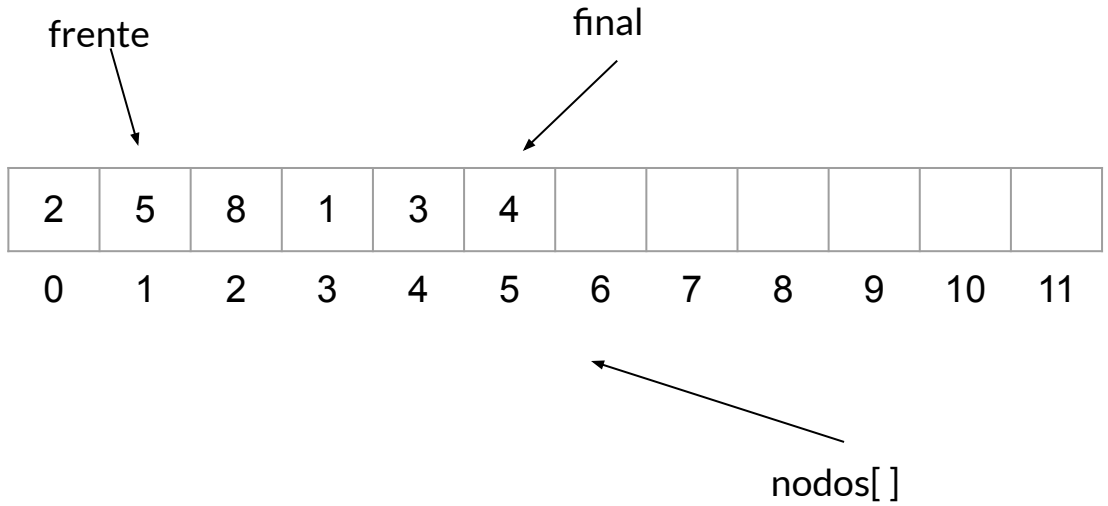
frente = 1;



Lista circular basada en arreglo

- void pop_back( );
- Eliminar al final.

capacidad = 12



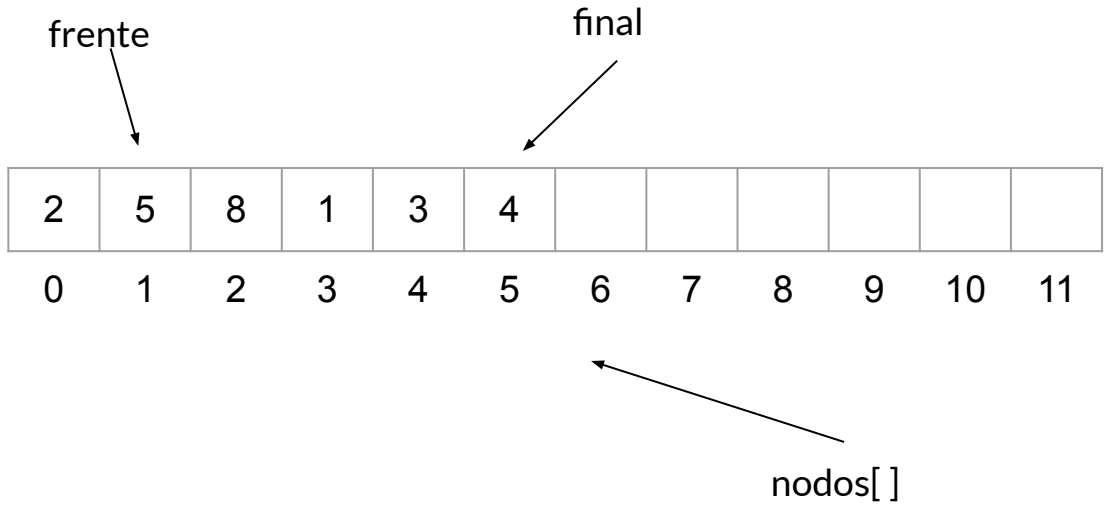
Lista circular basada en arreglo

- void pop_back( );
- Eliminar al final.

final = final - 1;

final = 4;

capacidad = 12



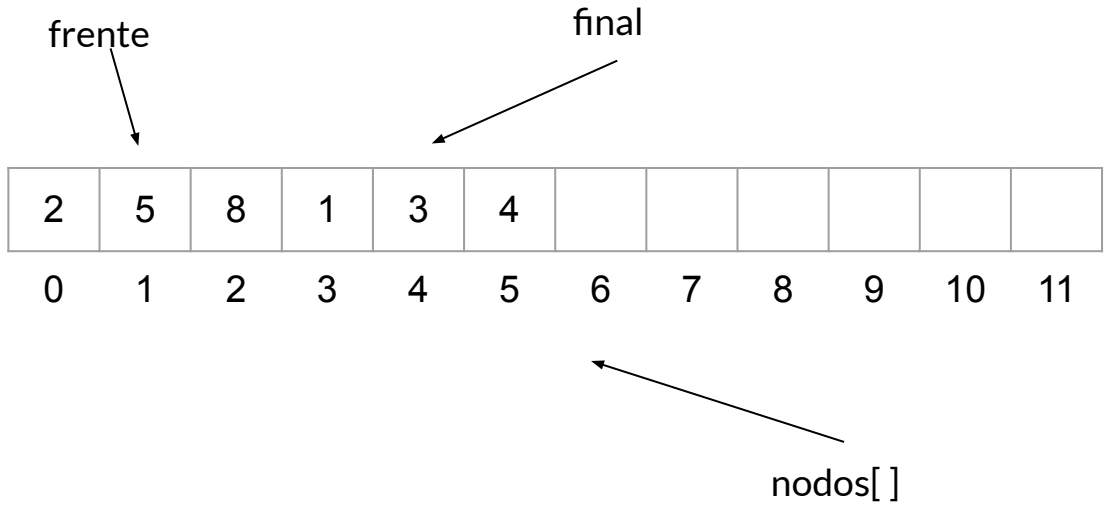
Lista circular basada en arreglo

- void pop_back( );
- Eliminar al final.

final = final - 1;

final = 4;

capacidad = 12

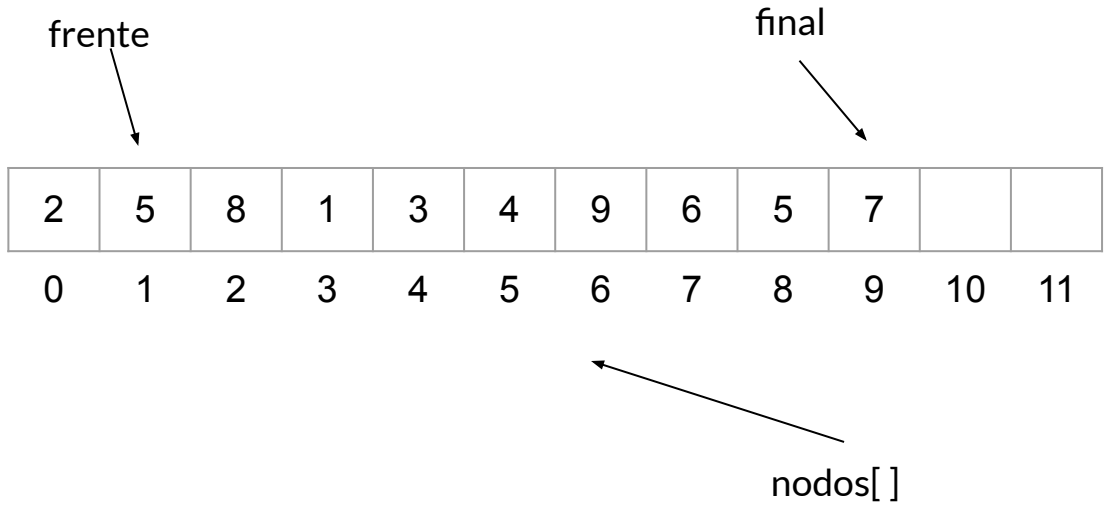


Lista circular basada en arreglo

- void pop(int i);
- Eliminar la posición i.

capacidad = 12

int i = 2;

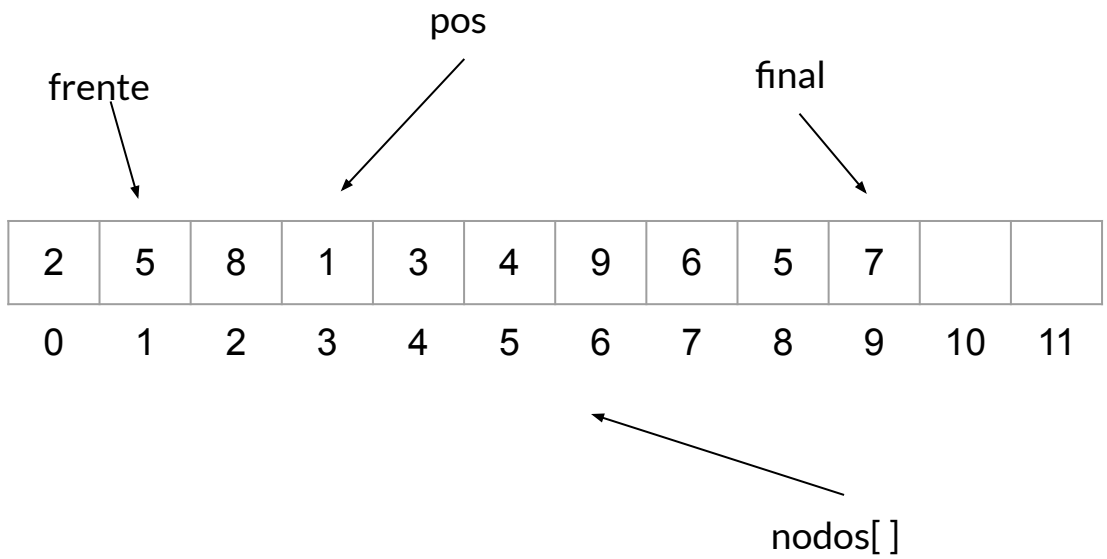


Lista circular basada en arreglo

- void pop(int i);
- Eliminar la posición i.

capacidad = 12

int i = 2;
int pos = siguiente(i + frente - 1);



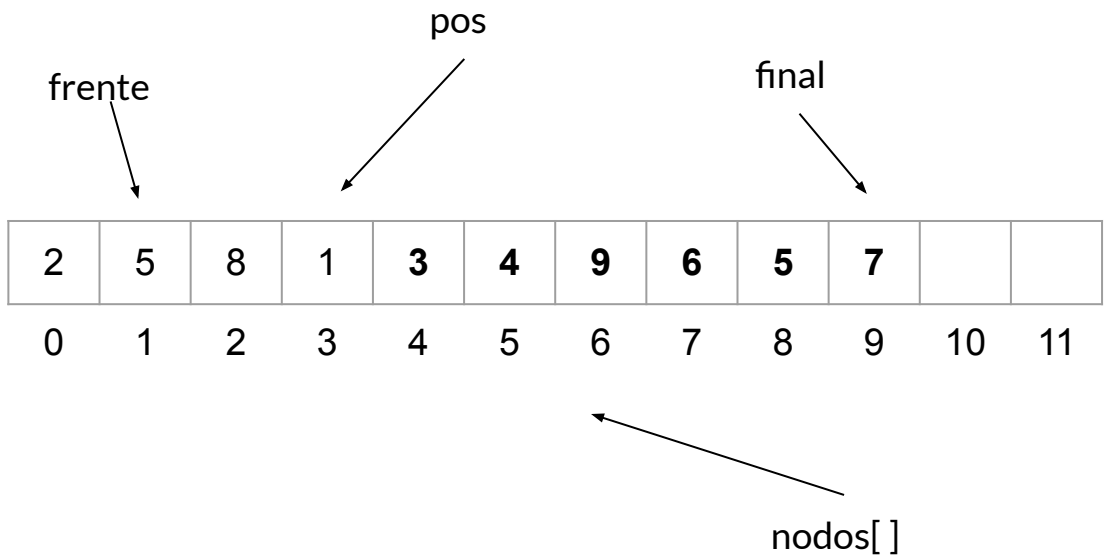
Lista circular basada en arreglo

- void pop(int i);
- Eliminar la posición i.

capacidad = 12

int i = 2;
int pos = siguiente(i + frente - 1);

Movemos todos los elementos desde **siguiente(pos)** hasta **final**, una posición a la **izquierda**



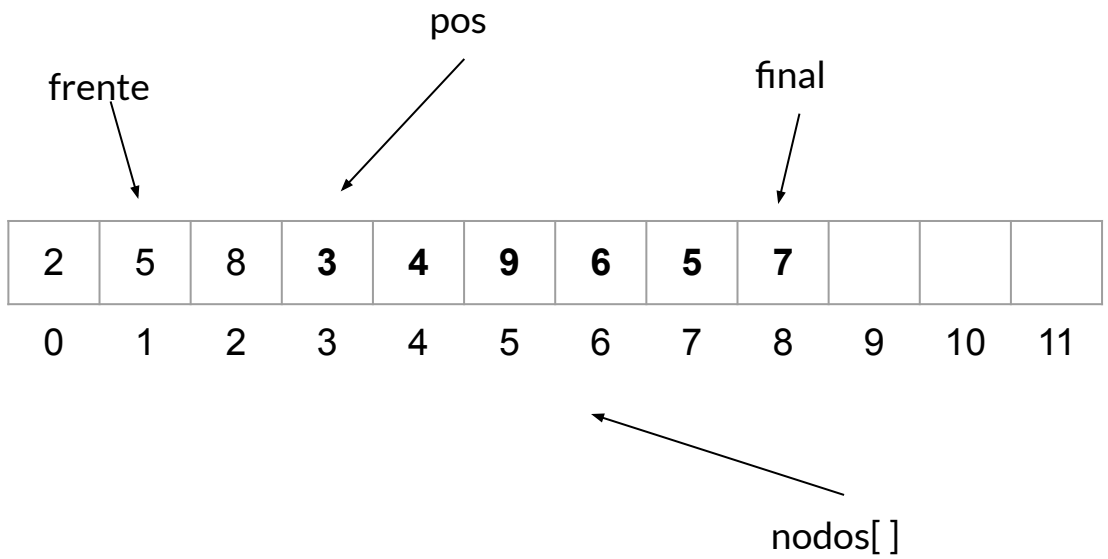
Lista circular basada en arreglo

- void pop(int i);
- Eliminar la posición i.

capacidad = 12

int i = 2;
int pos = siguiente(i + frente - 1);

Movemos todos los elementos desde **siguiente(pos)** hasta **final**, una posición a la **izquierda**



Lista circular basada en arreglo

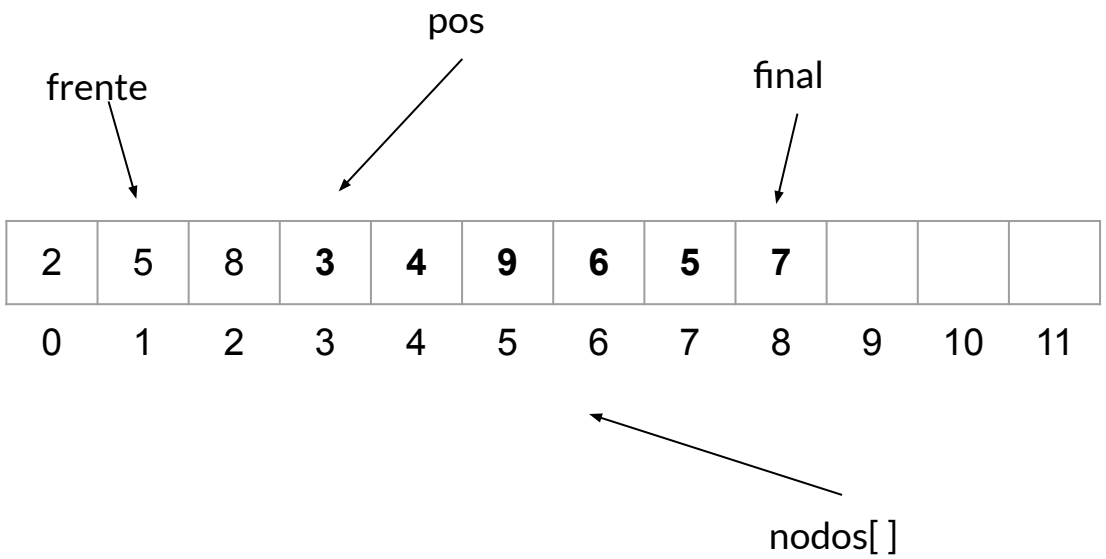
- void pop(int i);
- Eliminar la posición i.

capacidad = 12

int i = 2;
int pos = siguiente(i + frente - 1);

Movemos todos los elementos desde **siguiente(pos)** hasta **final**, una posición a la **izquierda**

La posición **anterior** (izquierda) puede ser calculada como: **(pos - 1) % capacidad**

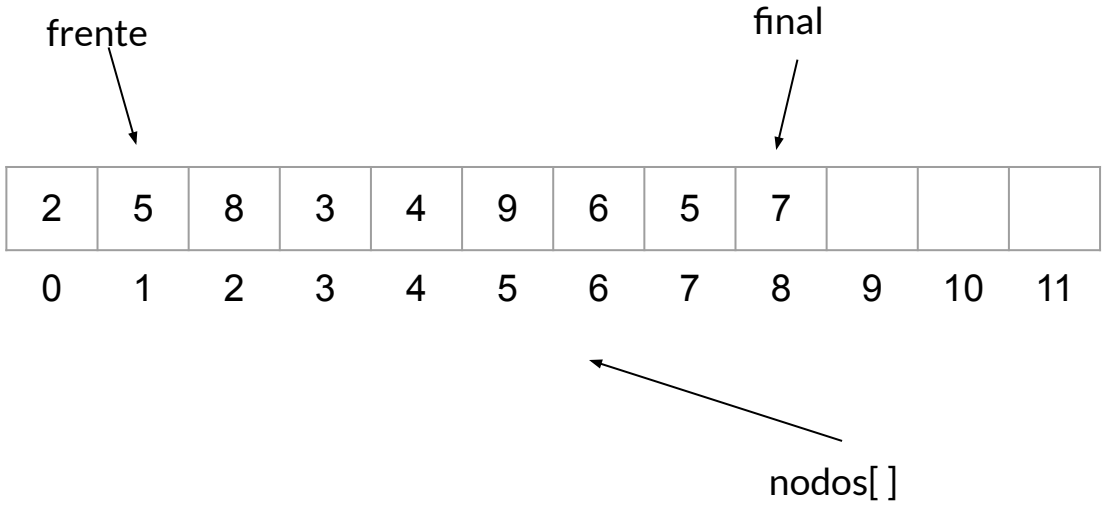


Lista circular basada en arreglo

- T get(int i);
- Devuelve el dato en la posición i-ésima.

capacidad = 12

int i = 2;



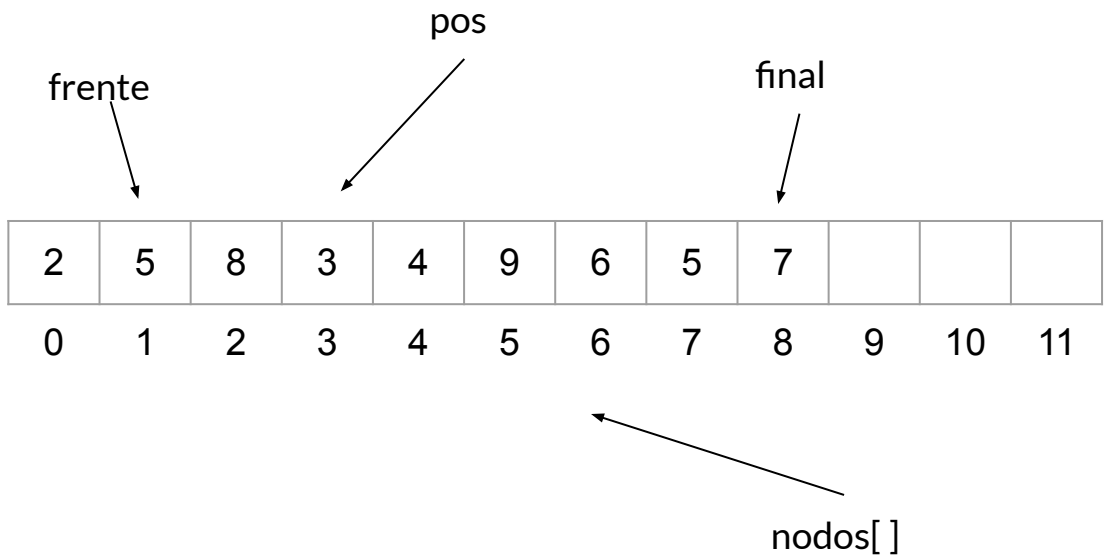
Lista circular basada en arreglo

- T get(int i);
- Devuelve el dato en la posición i-ésima.

int i = 2;

int pos = siguiente(i + frente - 1);

capacidad = 12



Lista circular basada en arreglo

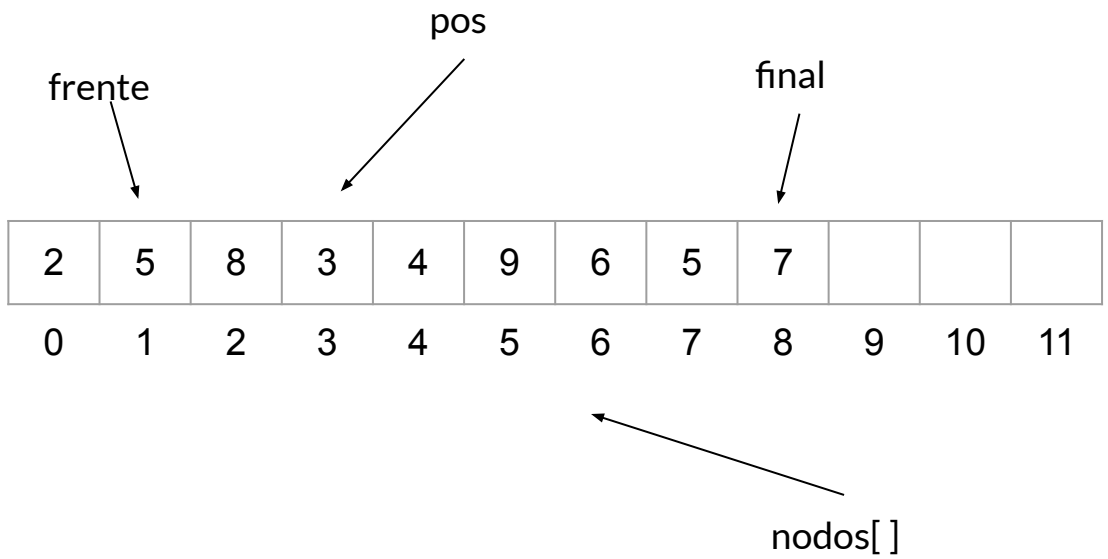
- T get(int i);
- Devuelve el dato en la posición i-ésima.

int i = 2;

int pos = siguiente(i + frente - 1);

return nodos[pos];

capacidad = 12

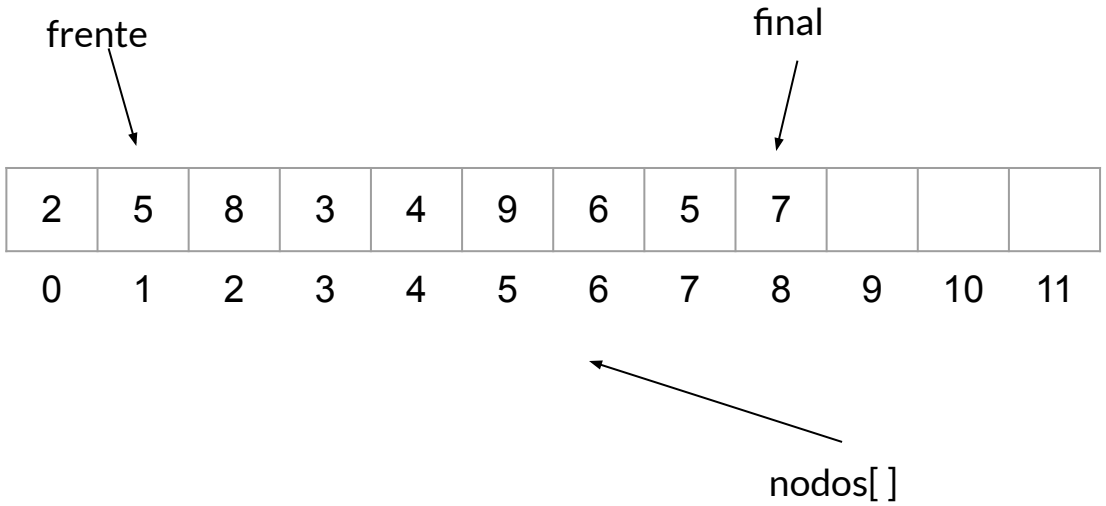


Lista circular basada en arreglo

- void set(T d, int i);
- Actualiza el dato en la posición i-ésima.

capacidad = 12

int i = 2;

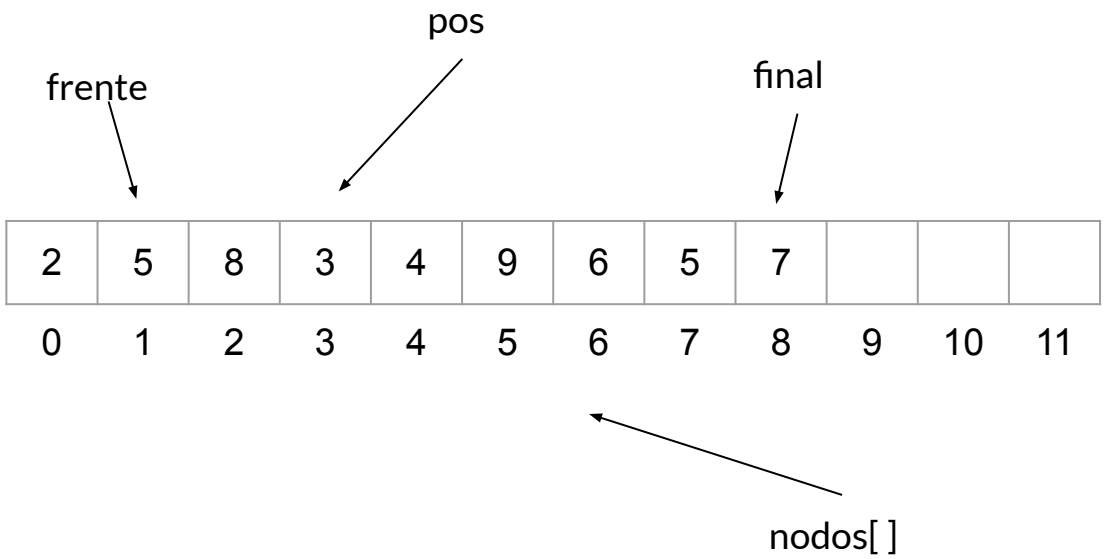


Lista circular basada en arreglo

- void set(T d, int i);
- Actualiza el dato en la posición i-ésima.

int i = 2;

int pos = siguiente(i + frente - 1);



Lista circular basada en arreglo

- void set(T d, int i);
- Actualiza el dato en la posición i-ésima.

int i = 2;

int pos = siguiente(i + frente - 1);

nodos[pos] = d;