IMPLEMENTACIÓN DE ESTRUCTURAS DE DATOS

Aránzazu Jurío

ALGORITMIA

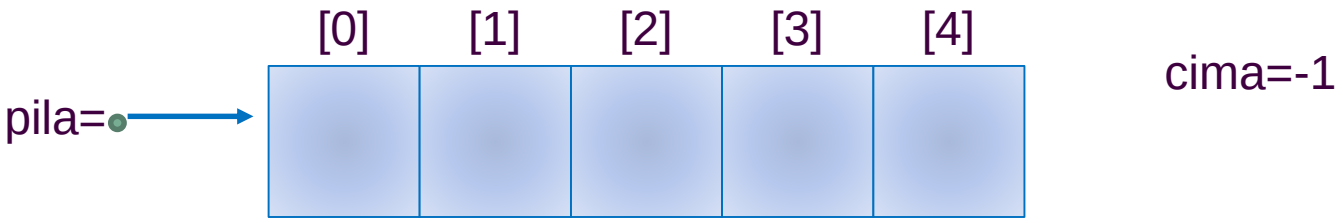
2018/2019

Índice

- Implementación de pilas
- Implementación de árboles binarios de búsqueda

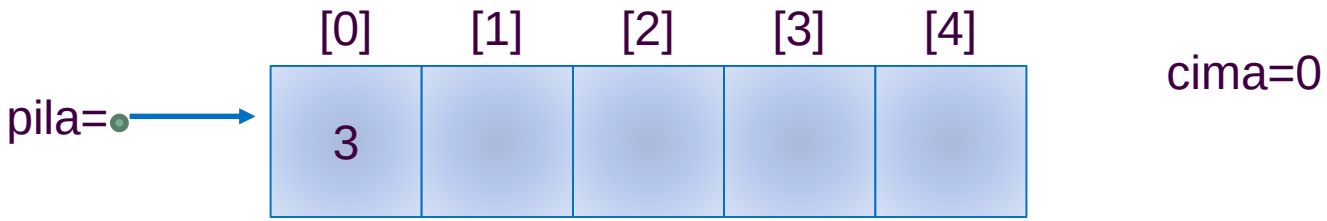
Pilas estáticas

- Pila de como máximo 5 elementos



Pilas estáticas

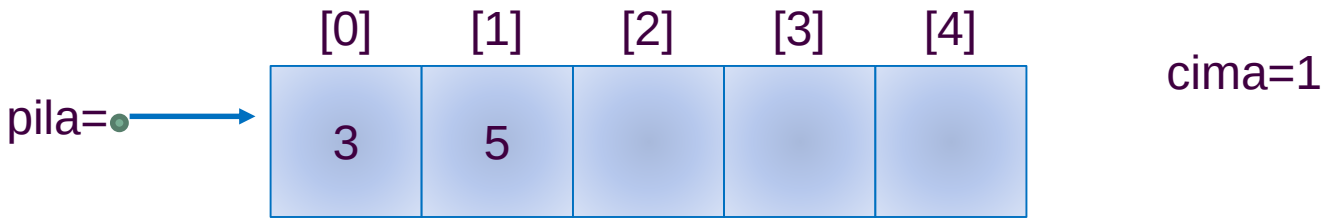
- Pila de como máximo 5 elementos



  - Apilar 3

Pilas estáticas

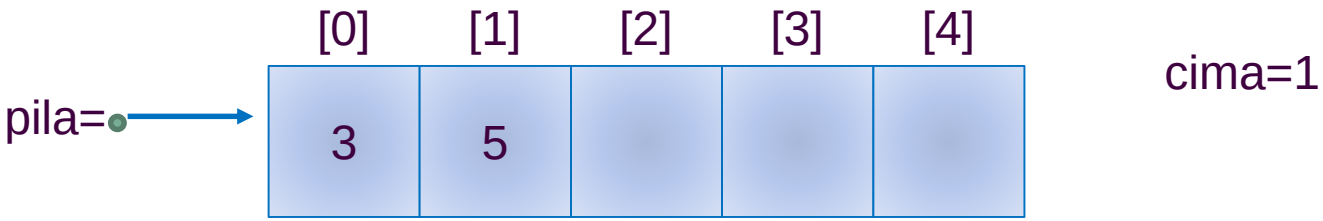
- Pila de como máximo 5 elementos



  - Apilar 3
  - Apilar 5

Pilas estáticas

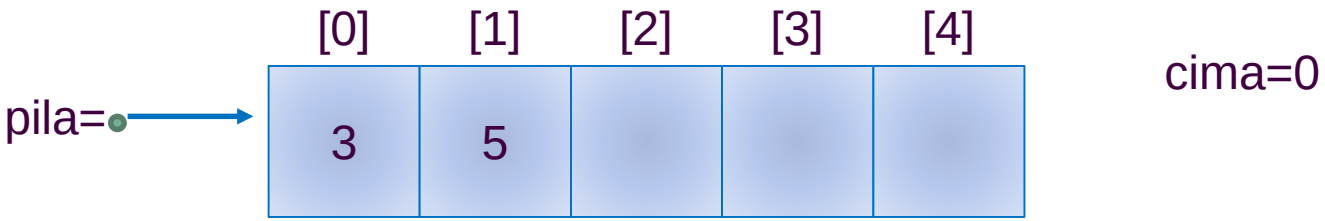
- Pila de como máximo 5 elementos



  - Apilar 3
  - Apilar 5
  - Cima → pila[cima]=5

Pilas estáticas

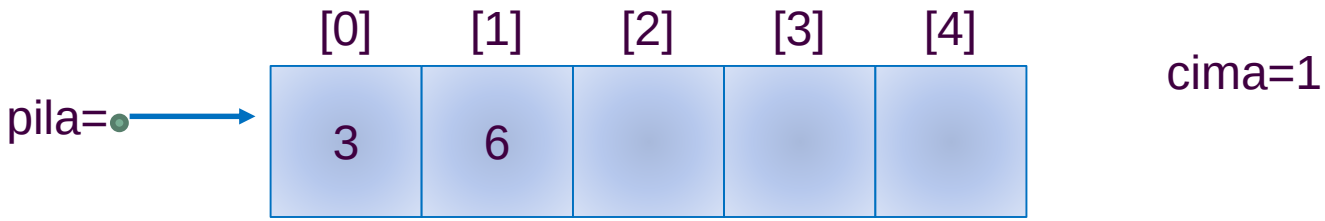
- Pila de como máximo 5 elementos



  - Apilar 3
  - Apilar 5
  - Cima → pila[cima]=5
  - Desapilar

Pilas estáticas

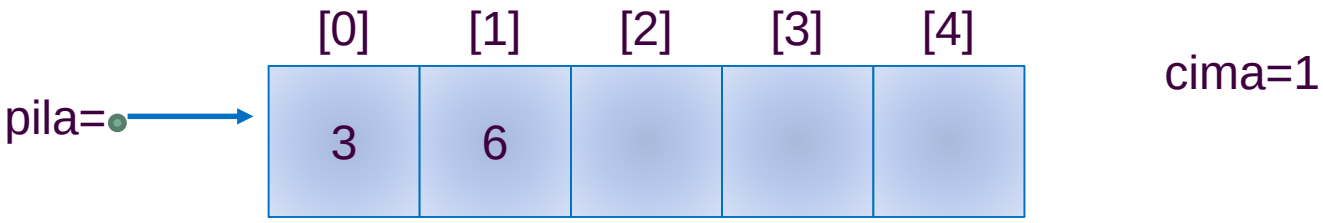
- Pila de como máximo 5 elementos



  - Apilar 3
  - Apilar 5
  - Cima → pila[cima]=5
  - Desapilar
  - Apilar 6

Pilas estáticas

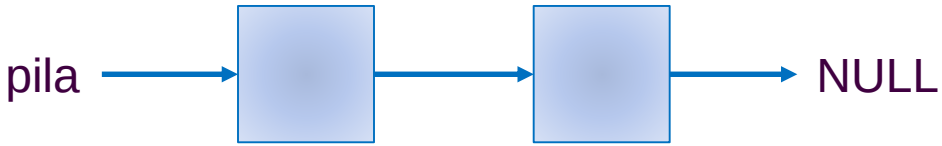
- Pila de como máximo 5 elementos



- ¿Y si no se cuántos elementos va a tener mi pila?
- Reserva de un vector muy grande → ineficiente
- Ideal: que la pila tenga en cada momento el tamaño ajustado a los elementos que contiene

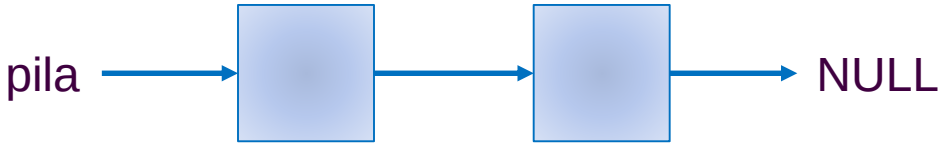
Pilas dinámicas

- La variable pila es un puntero a un elemento
  - Si la pila está vacía, debe apuntar a NULL
- Cada elemento debe estar unido al siguiente elemento de la pila
  - El último debe estar unido a NULL



Pilas dinámicas

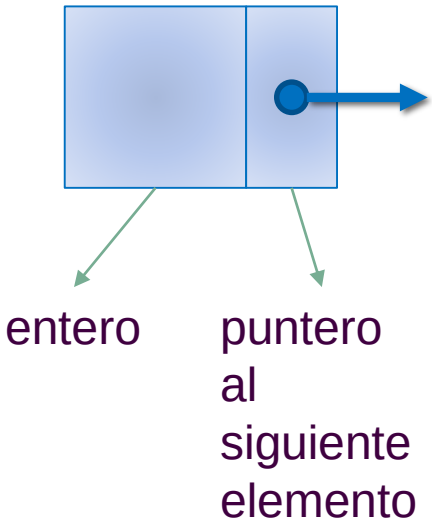
- La variable pila es un puntero a un elemento
  - Si la pila está vacía, debe apuntar a NULL
- Cada elemento debe estar unido al siguiente elemento de la pila
  - El último debe estar unido a NULL



- ¿Con qué elemento debe estar unido pila? ¿El primero o el último introducido?
- ¿Cómo consigo que cada elemento pueda almacenar un número y unirse con el siguiente elemento?

Pilas dinámicas

- Estructura para cada elemento



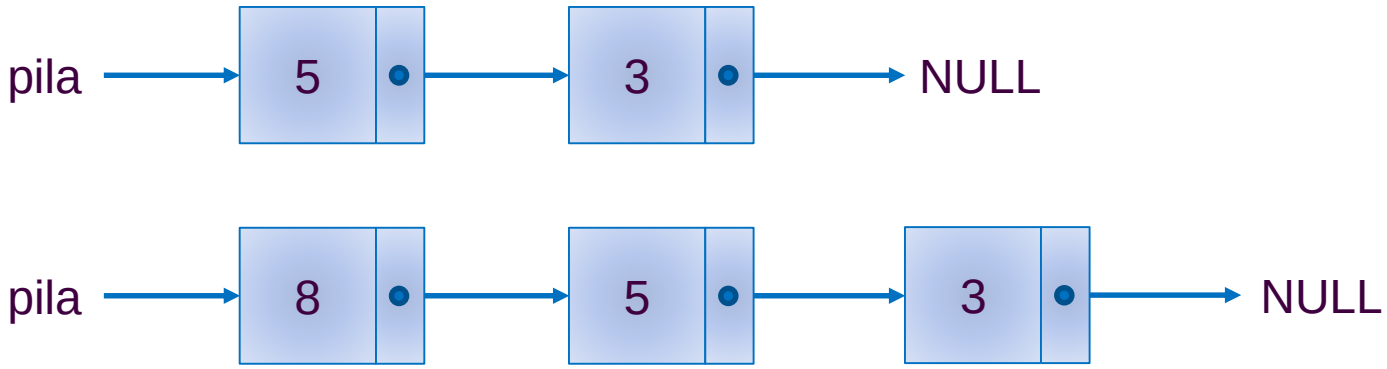

```
typedef struct celda{
        int num;
        struct celda *sig;
}celda;
```

Pilas dinámicas

- Vamos a implementar las siguientes funciones:
  - Apilar
  - Desapilar
  - Cima
  - Es_vacia
  - Vaciar

Pilas dinámicas

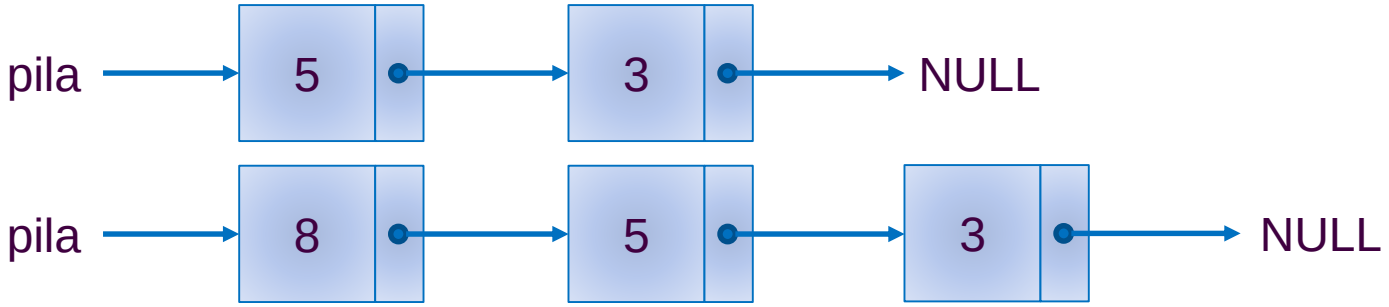
- Apilar un elemento



- ¿Qué tengo que hacer para apilar un elemento?

Pilas dinámicas

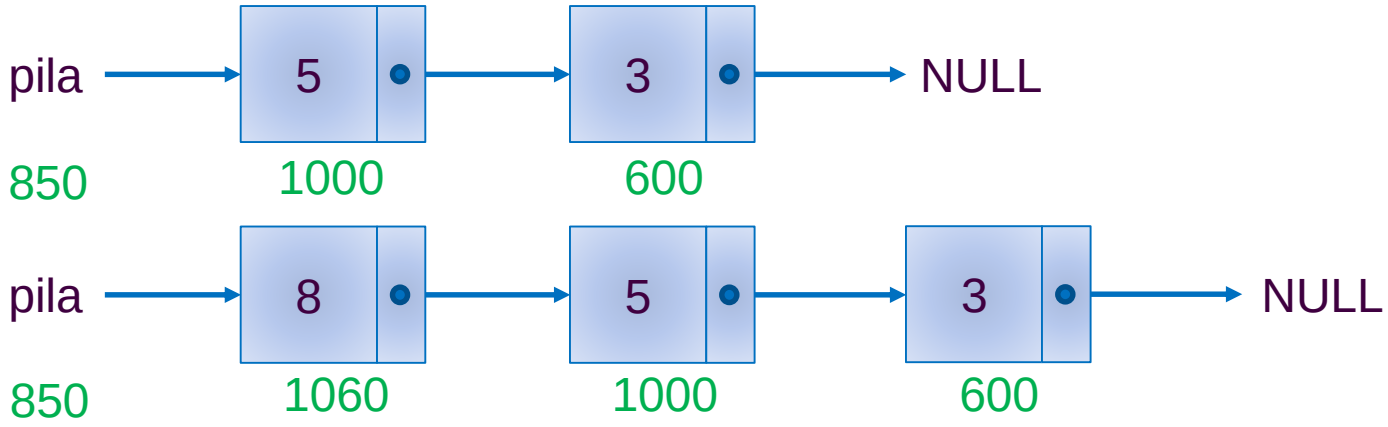
- Apilar un elemento



- ¿Qué tengo que hacer para apilar un elemento?
  - Crear el elemento (reservar memoria y rellenar valor)
  - Unirlo con el anterior primer elemento
  - Hacer que la pila se una a él

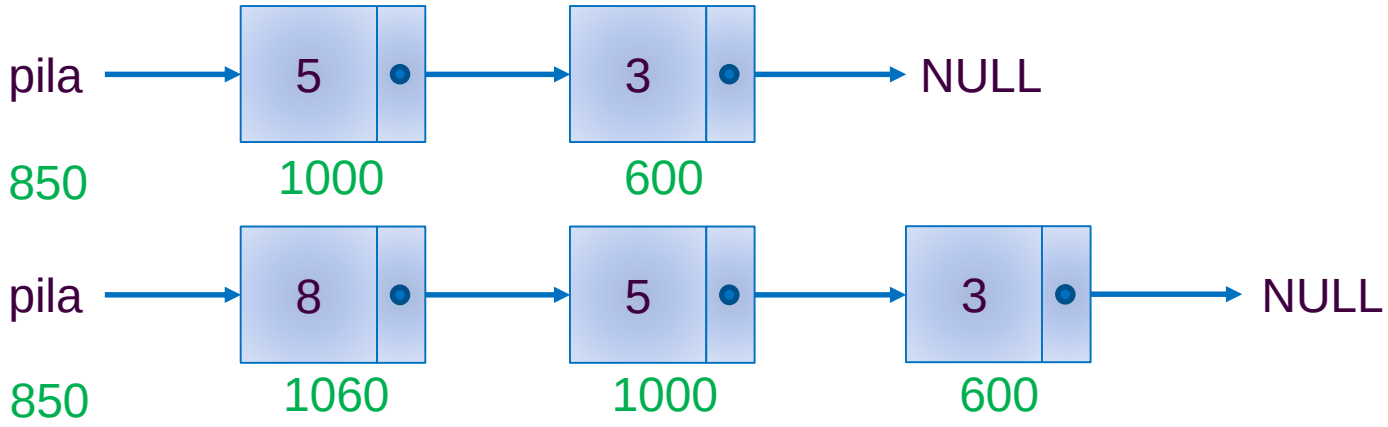
Pilas dinámicas

- Apilar un elemento
  - ¿Puede cambiar el primer elemento de la pila?



Pilas dinámicas

- Apilar un elemento
  - ¿Puede cambiar el primer elemento de la pila?



  - Pasar la pila por referencia

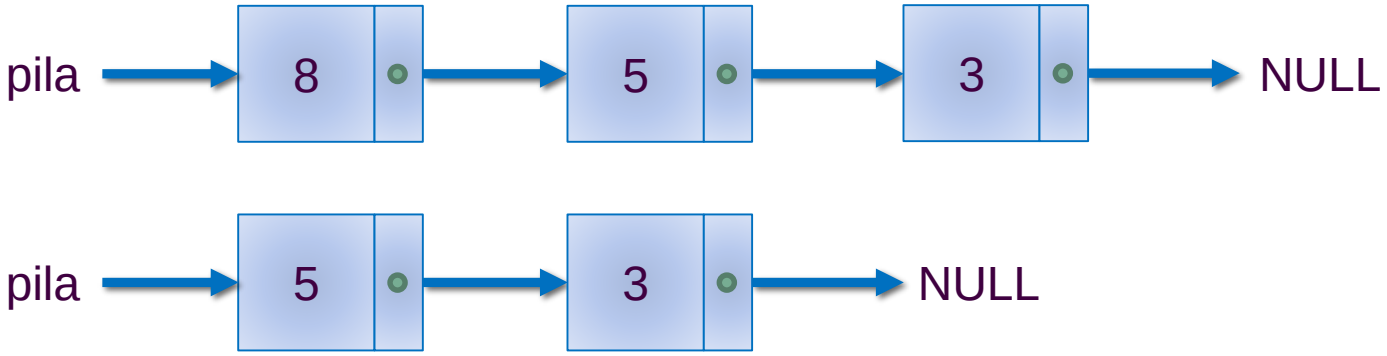
Pilas dinámicas

- Apilar un elemento
- ¿Qué tengo que hacer para apilar un elemento?
  - Crear el elemento (reservar memoria y rellenar valor)
  - Unirlo con el anterior primer elemento
  - Hacer que la pila se una a él

```
void apilar (celda **cabecera, int numero){
        celda *nuevo;
        nuevo=(celda*)malloc(sizeof(celda));
        nuevo->num=numero;
        nuevo->sig=*cabecera;
        *cabecera=nuevo;
        return;
}
```

Pilas dinámicas

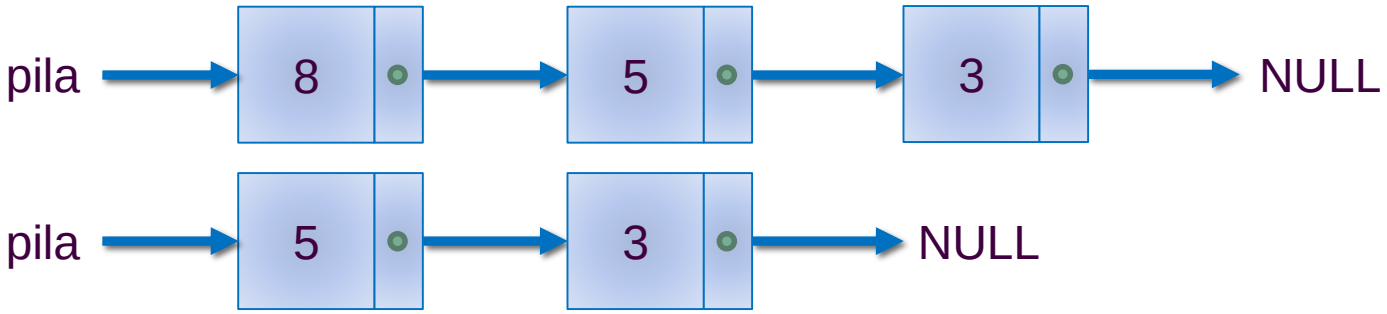
- Desapilar un elemento



- ¿Qué tengo que hacer para desapilar un elemento?

Pilas dinámicas

- Desapilar un elemento



- ¿Qué tengo que hacer para desapilar un elemento?
  - Hacer que la pila se una con el siguiente
  - Liberar el elemento desapilado

Pilas dinámicas

- ¿Qué tengo que hacer para desapilar un elemento?
  - Hacer que la pila se una con el siguiente
  - Liberar el elemento desapilado
- ¿Hay algún caso en el que este procedimiento no funcione?

Pilas dinámicas

- ¿Qué tengo que hacer para desapilar un elemento?
  - Hacer que la pila se una con el siguiente
  - Liberar el elemento desapilado
- ¿Hay algún caso en el que este procedimiento no funcione?
  - Si la pila está vacía

Pilas dinámicas

- Desapilar un elemento
- ¿Qué tengo que hacer para desapilar un elemento?
  - Mostrar el elemento desapilado
  - Hacer que la pila se una con el siguiente
  - Liberar el elemento desapilado
- Si la pila está vacía

```
void desapilar (celda **cabecera){
    celda *aux;
    if(es_vacia(*cabecera))
        printf("No puedes desapilar
una pila vacia\n");
    else{
        aux=*cabecera;
        *cabecera=aux->sig;
        free(aux);
    }
    return;
}
```

Pilas dinámicas

- Cima
- ¿Qué tengo que hacer?
  - Devolver el primer elemento
- ¿Hay algún caso en el que este procedimiento no funcione?
  -
- ¿Modifico el primer elemento?
  -

Pilas dinámicas

- Cima
- ¿Qué tengo que hacer?
  - Devolver el primer elemento
- ¿Hay algún caso en el que este procedimiento no funcione?
  - Si la pila está vacía
- ¿Modifico el primer elemento?
  - No. Paso por valor

Pilas dinámicas

- Cima
- ¿Qué tengo que hacer?
  - Devolver el primer elemento
- ¿Hay algún caso en el que este procedimiento no funcione?
  - Si la pila está vacía
- ¿Modifico el primer elemento?
  - No. Paso por valor

```
int cima (celda *cabecera){
        if(es_vacia(cabecera))
                return -1;
        else
                return cabecera-
>num;
}
```

Pilas dinámicas

- Comprobar si está vacía
  - ¿Qué se debe cumplir siempre que una pila no tenga elementos almacenados?
  - ¿Modifico el primer elemento de la pila?

Pilas dinámicas

- Comprobar si está vacía
  - ¿Qué se debe cumplir siempre que una pila no tenga elementos almacenados?
    - La variable debe apuntar a NULL
  - ¿Modifico el primer elemento de la pila?
    - No. Paso por valor

```
int es_vacia (celda *cabecera){
        if (cabecera==NULL)
                return 1;
        else
                return 0;
}
```

Pilas dinámicas

- Vaciar la pila
  - Sólo se puede acceder al primer elemento de la pila
  - Desapilar elementos hasta que la pila esté vacía

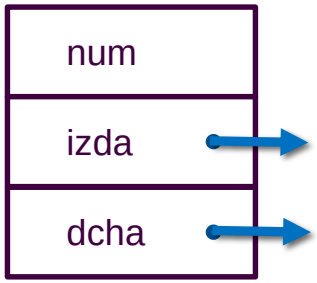
```
void vaciar (celda **cabecera){
        while(es_vacia(*cabecera)==0
)
                desapilar(cabecera);
        return;
}
```

Índice

- Repaso. Estructuras de datos básicas
- Implementación de árboles binarios de búsqueda

Árboles binarios

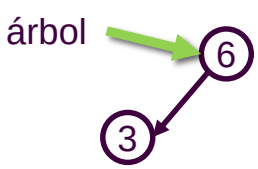
- En cada elemento queremos almacenar el número correspondiente.
- Además, tenemos que unirlo con su hijo izquierdo y su hijo derecho.

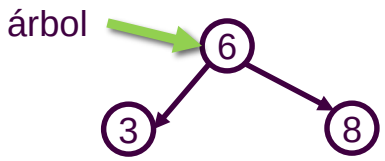



```
typedef struct celda{
    int num;
    struct celda *izda;
    struct celda *dcha;
}celda;
```

Árboles binarios

- Función insertar
  - Argumentos
    - Árbol – puntero al nodo raíz
    - Número a insertar
  - ¿Por valor o por referencia?
    - Si el árbol cambia de valor (el puntero al nodo raíz apunta a otro objeto), se pasa por referencia
    - Si el árbol no cambia (el puntero al nodo raíz sigue apuntando al mismo elemento), se pasa por valor





Árboles binarios

- Función insertar
  - Debe valer para todas las inserciones
  - ¿Y el primer elemento?

Árboles binarios

- Función insertar
  - Debe valer para todas las inserciones
  - ¿Y el primer elemento?

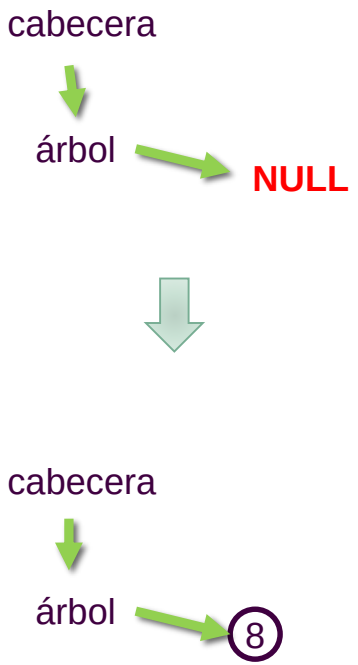
  - El árbol se debe pasar por referencia

```
void insertar(celda **cabecera,int numero)
```

Árboles binarios

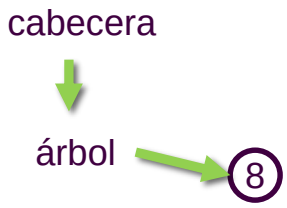
- Función insertar
  - Ejemplo: 8 5 6 3 9 4
  - Insertar 8

Como no hay ningún elemento coloco el 8 como nodo raíz

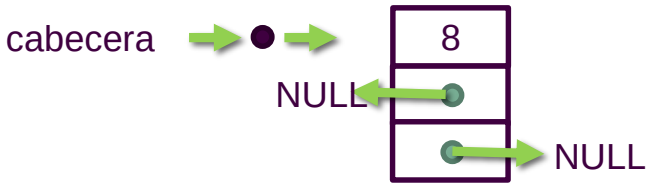


Árboles binarios

- Función insertar
  - Ejemplo: 8 5 6 3 9 4
  - Insertar 8




```
celda *nuevo;
if(*cabecera==NULL){
    nuevo=(celda*)malloc(sizeof(celda));
    nuevo->num=numero;
    nuevo->izda=NULL;
    nuevo->dcha=NULL;
    *cabecera=nuevo;
}
```



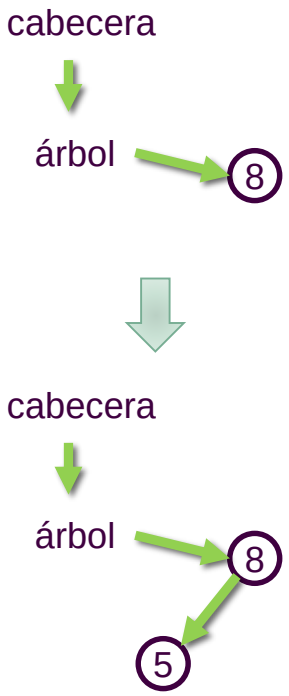
Árboles binarios

- Función insertar
  - Ejemplo: 8 5 6 3 9 4
  - Insertar 5

Comparo el número 5 con el número del nodo raíz

Como es más pequeño, miro el hijo izquierdo

Como no hay elemento, lo coloco ahí



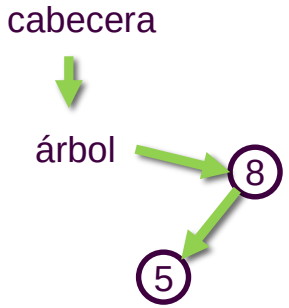
Árboles binarios

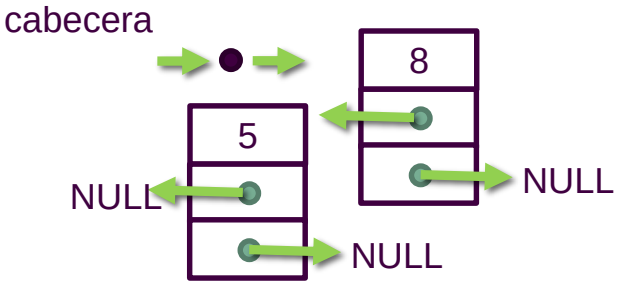
- Función insertar
  - Ejemplo: 8 5 6 3 9 4
  - Insertar 5

Comparo el número 5 con el número del nodo raíz

Como es más pequeño, miro el hijo izquierdo

Como no hay elemento, lo coloco ahí





Árboles binarios

- Función insertar
  - Ejemplo: 8 5 6 3 9 4
  - Insertar 6

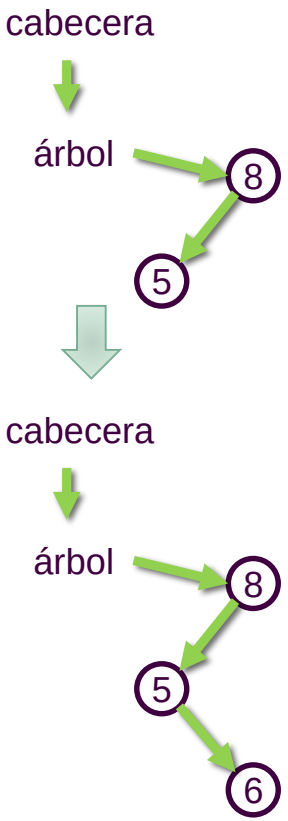
Comparo el número 6 con el número del nodo raíz

Como es más pequeño, miro el hijo izquierdo

Comparo el número 6 con el número al que estoy apuntando ahora

Como es mayor, miro el hijo derecho

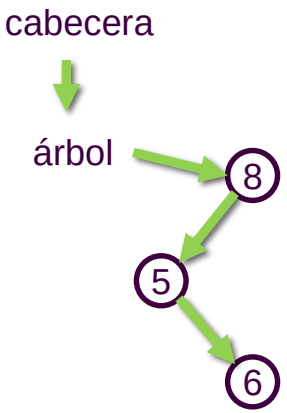
Como no hay elemento, lo coloco ahí

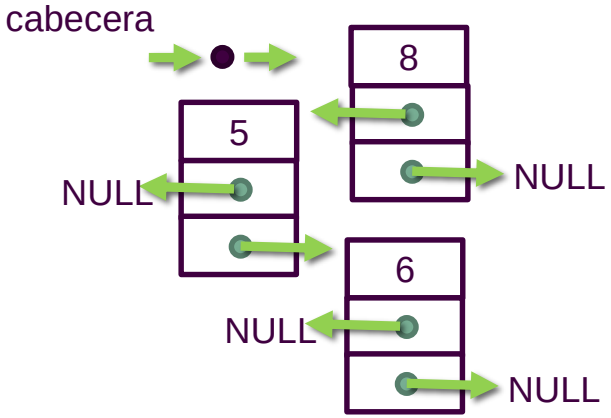


Árboles binarios

- Función insertar
  - Ejemplo: 8 5 6 3 9 4
  - Insertar 6

```
celda *nuevo;
if(*cabecera!=NULL){
    if((*cabecera)->num==numero)
        printf("El numero %d ya estaba en el
                arbol\n",numero);
    else
        if((*cabecera)->num>numero)
            insertar(&((*cabecera)->izda),numero);
        else
            insertar(&((*cabecera)-
>dcha),numero);
}
```





Árboles binarios

- Función profundidad
  - Argumentos
    - Árbol – puntero al nodo raíz
  - ¿Por valor o por referencia?
    - Si el árbol cambia de valor (el puntero al nodo raíz apunta a otro objeto), se pasa por referencia
    - Si el árbol no cambia (el puntero al nodo raíz sigue apuntando al mismo elemento), se pasa por valor

Árboles binarios

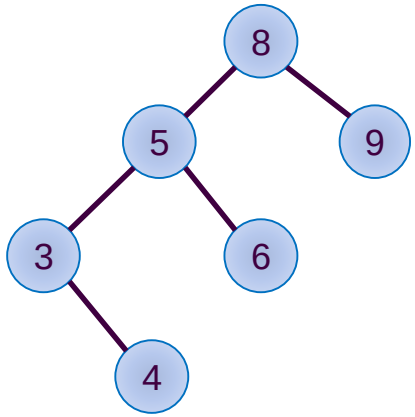
- Función profundidad
  - Argumentos
    - Árbol – puntero al nodo raíz
  - ¿Por valor o por referencia?
    - Si el árbol cambia de valor (el puntero al nodo raíz apunta a otro objeto), se pasa por referencia
    - Si el árbol no cambia (el puntero al nodo raíz sigue apuntando al mismo elemento), se pasa por valor
  - El árbol se debe pasar por valor

Árboles binarios

- Función profundidad
  - Utilizamos también la recursividad
  - Cuando estoy trabajando con un nodo, tengo acceso a:
    - Elemento del nodo
    - Llamar al hijo izquierdo
    - Llamar al hijo derecho
  - ¿En qué orden debo poner las instrucciones?

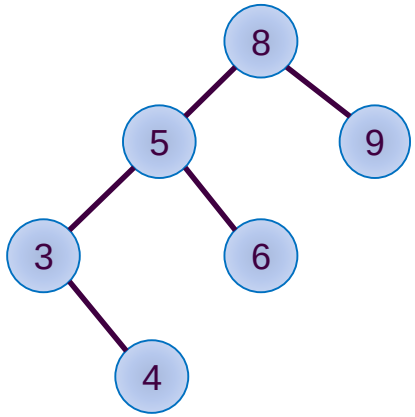
Árboles binarios

- Función profundidad
  - hijo izquierdo – hijo derecho – elemento
    - nodo 8
      - nodo 5
        - nodo 3
          - nodo 4
            - imprimir 4
          - imprimir 3
        - nodo 6
          - imprimir 6
        - imprimir 5
      - nodo 9
        - imprimir 9
      - imprimir 8



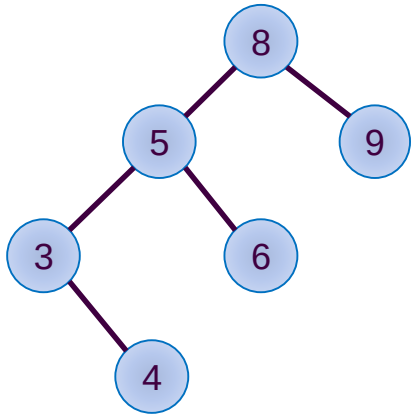
Árboles binarios

- Función profundidad
  - hijo izquierdo – elemento - hijo derecho
    - nodo 8
      - nodo 5
        - nodo 3
          - imprimir 3
          - nodo 4
            - imprimir 4
        - imprimir 5
        - nodo 6
          - imprimir 6
      - imprimir 8
      - nodo 9
        - imprimir 9



Árboles binarios

- Función profundidad
  - elemento - hijo izquierdo - hijo derecho
    - nodo 8
      - imprimir 8
      - nodo 5
        - imprimir 5
        - nodo 3
          - imprimir 3
          - nodo 4
            - imprimir 4
        - nodo 6
          - imprimir 6
      - nodo 9
        - imprimir 9



Árboles binarios

- Función profundidad

```
void profundidad(celda* cabecera){
    if(cabecera==NULL)
        printf("El arbol esta vacio\n");
    else{
        printf("%d  ",cabecera->num);
        if(cabecera->izda!=NULL)
            profundidad(cabecera->izda);
        if(cabecera->dcha!=NULL)
            profundidad(cabecera->dcha);
    }
    return;
}
```