Lenguaje de Programación 1

Cola de Prioridades y Montículos

Diego Pinto Roa

Contenido

- Cola de Prioridades
- Construcción de una cola de prioridades
- Montículos
- Ordenación por Montículos (HeapSort)

Colas de Prioridades

Algoritmos y Estructura de Datos.
Una perspectiva en C - Capitulo 12

Aguilar & Martínez

Introducción

- La Cola de Prioridades es una Estructura de Datos utilizada para planificar tareas en aplicaciones informáticas donde la prioridad de la tarea se corresponde con la clave de ordenación.
- También se aplican en los sistemas de simulación donde los sucesos se deben procesar en orden cronológico.
- El común denominador de estas aplicaciones es que siempre se selecciona el elemento mínimo una y otra vez hasta que no quedan más elementos.
- Tradicionalmente se ha designado la prioridad más alta con una clave menor, así los elementos de prioridad tienen más prioridad que los elementos de prioridad 3, por ejemplo.

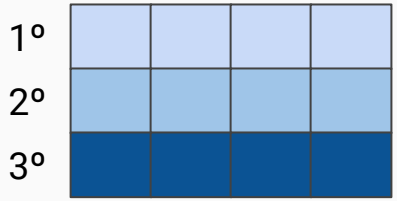
Concepto

El orden en que los elementos de una cola de prioridades son procesados sigue reglas:

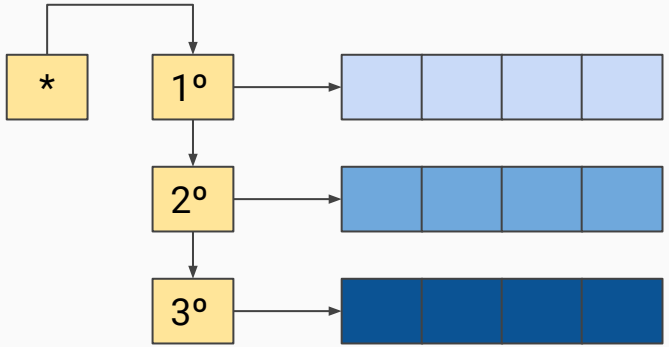
- Se elige la cola no vacía que se corresponde con la mayor prioridad.
- En la cola de mayor prioridad los elementos se procesan según el orden de llegada: primero en entrar, primero en salir.

Implementaciones de cola de 3 prioridades.

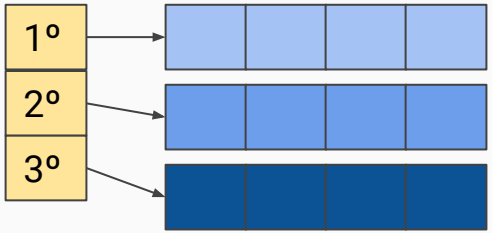
Matriz: codifica tanto la prioridad como las colas.



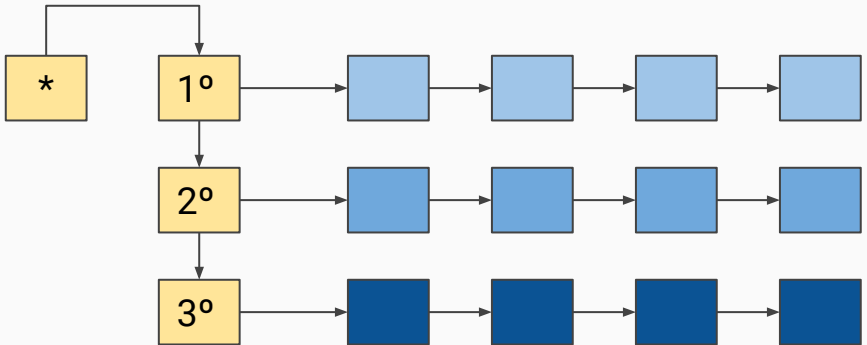
Prioridad con listas enlazadas y Colas con arreglos



Prioridad con arreglos de punteros y las colas con arreglos

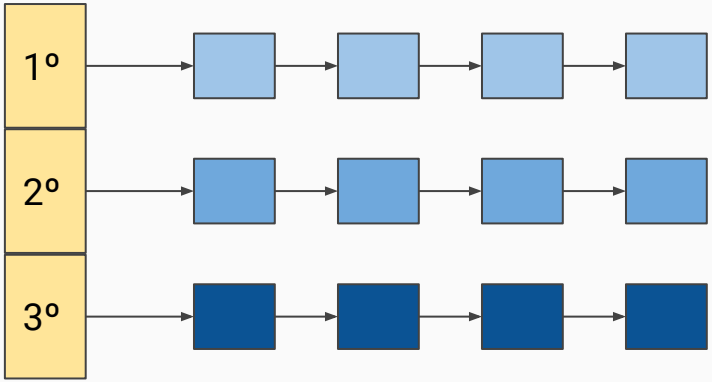


Prioridad con listas enlazadas y Colas con listas



Implementaciones de cola de 3 prioridades.

Prioridad con arreglos y Colas con listas



Funciones de Listas de Colas

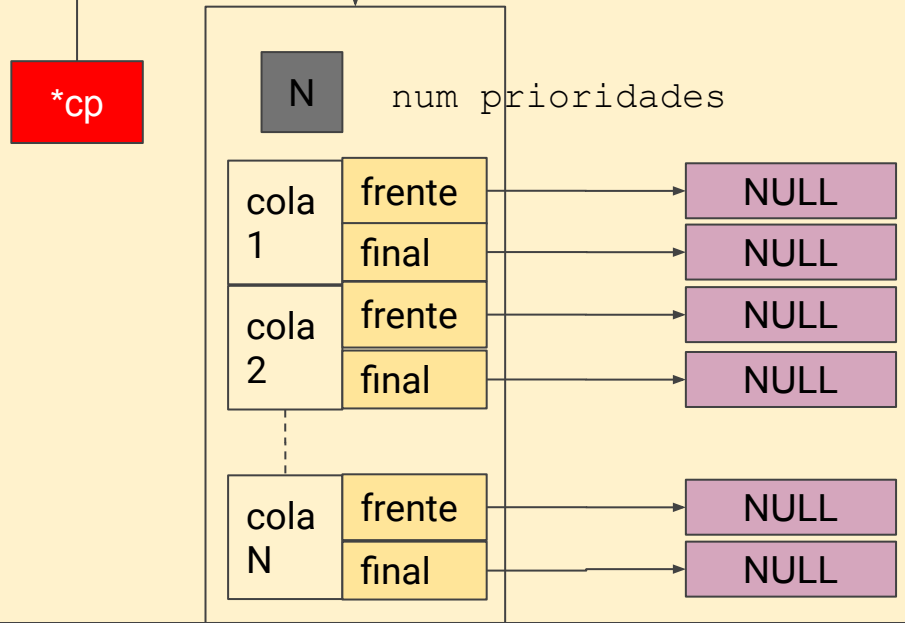
- crearColaPrioridades
- insertarEnPrioridad
- elementoMin
- quitarMin
- colaPrioridadVacia

```
/*archivo cola-prioridad.h*/
#define N 12
#define M 51

struct Tarea{
    int prioridad;
    char nombre[M];
}
typedef struct Tarea TipoDato;

#include "coladinamica.h"

struct cola_prioridades{
    int numprioridades;
    Cola colas[N];
}
typedef struct cola_prioridades ColaPrioridades;

/*Operaciones sobre la Cola de Prioridades*/
void crearColaPrioridades(ColaPrioridad * cp);
void insertarCola(ColaPrioridad * cp, TipoDato t);
TipoDato elementoMin(ColaPrioridad cp);
void quitaMin(ColaPrioridad * cp);
int colaPrioridadVacia(ColaPrioridad cp);
```

Funciones de Listas de Colas

- crearColaPrioridades
- insertarEnPrioridad
- elementoMin
- quitarMin
- colaPrioridadVacia

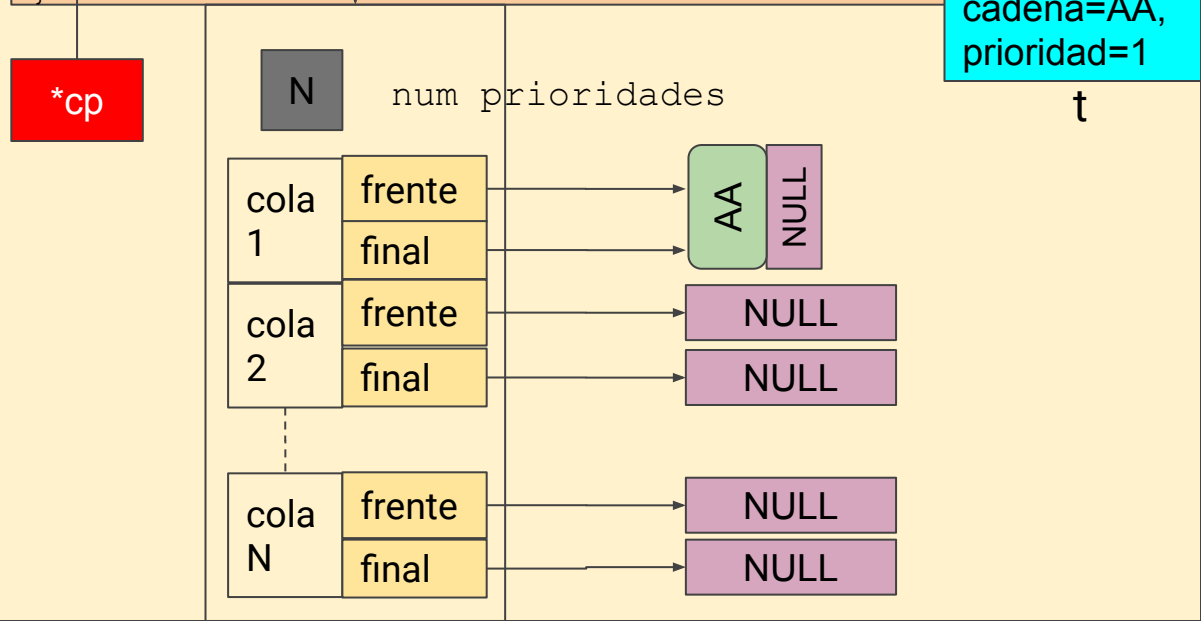
```
/*archivo cola-prioridad.c*/
/*Crea una cola nueva*/
void crearColaPrioridad(ColaPrioridad * cp){
     int j;
     cp -> numprioridades = N;
     for (j = 0; j < N; j++){
          crearCola(&(cp->colas[j]));
     }
}
```



Funciones de Listas de Colas

- crearColaPrioridades
- insertarEnPrioridad
- elementoMin
- quitarMin
- colaPrioridadVacia

```
/*archivo cola-prioridad.c*/
/*Insertar en la cola*/
void insertarColaPrioridad(ColaPrioridad * cp,
Tarea t){
    if (t.prioridad < cp -> numprioridades){
        insertar(&cp->colas[t.prioridad],t);
    }else{
        puts("Prioridad fuera de rango");
    }
}
```



Funciones de Listas de Colas

- crearColaPrioridades
- insertarEnPrioridad
- elementoMin
- quitarMin
- colaPrioridadVacia

/*archivo cola-prioridad.c*/

```
/*quitar de la cola*/
Tarea quitarMin(ColaPrioridad * cp){
    int i = 0, indiceCola = -1;
    /*búsqueda de la primera cola no vacía*/
    do{
        if(!colaVacia(cp->colas[i])){
            indiceCola = i;
            i = cp->numprioridades + 1;
        }else{
            i++;
        }
    }while(i < cp->numprioridades);
    if(indiceCola != -1){
        return quitar(&cp->colas[indiceCola]);
    }else{
        puts("Cola de prioridad vacia");
        exit(1);
    }
}
```

Montículos

Introducción

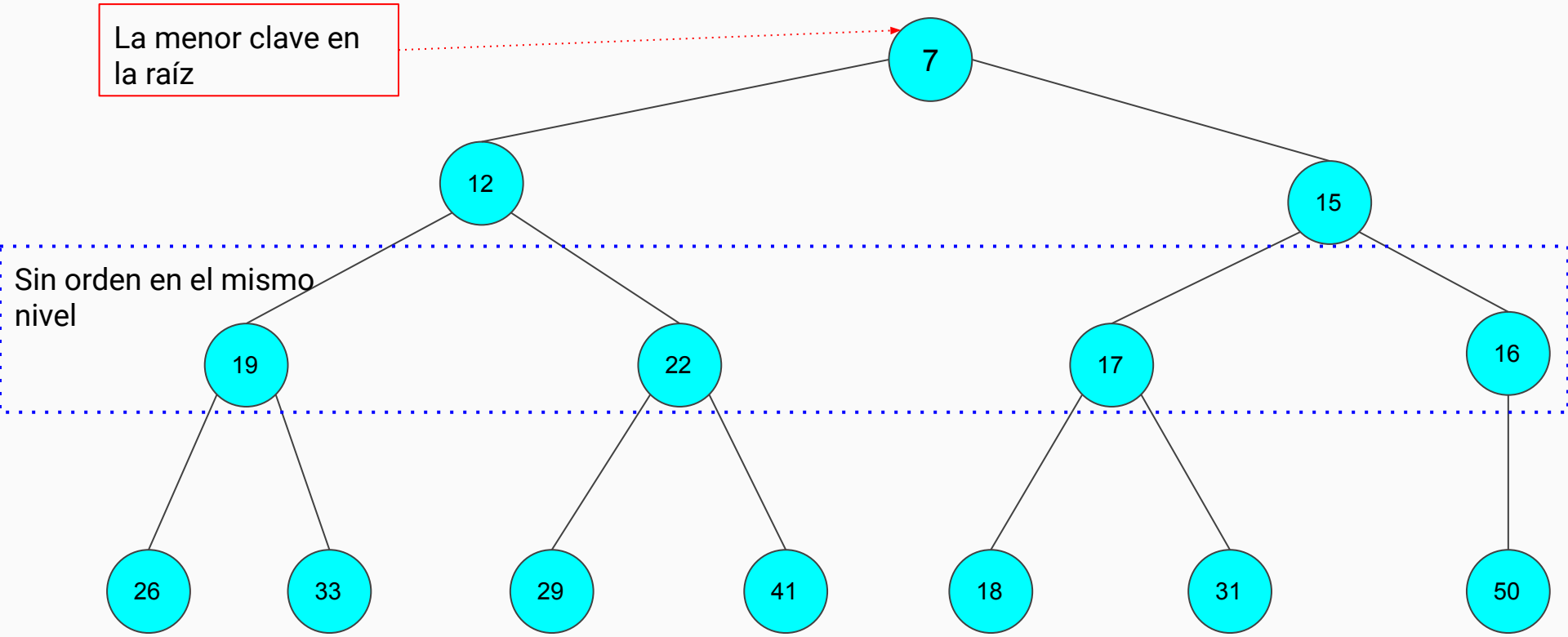
- El montículo binario, simplemente montículo, es la estructura más apropiada para implementar eficientemente las colas de prioridades.
- La idea intuitiva de un montículo es la que mejor explica esta estructura de datos. En la parte más alta del montículo se encuentra el elemento más pequeño.
- Los elementos descendientes de uno dado son mayores en la estructura montículo.
- La estructura montículo garantiza que el tiempo de las operaciones básicas insertar y eliminar mínimo, sea de complejidad logarítmicas.

Definición

Un montículo binario de tamaño `n` se define como un **árbol binario casi completo** de `n` nodos, tal que el contenido de cada nodo es menor o igual que el contenido de su hijos

Un **árbol binario completo** es un árbol con todos los niveles llenos, con la excepción del último nivel que se llena de izquierda a derecha. Esta estructura tiene la propiedad importante de que la altura de un árbol binario completo de `n` nodos es [`log n`].

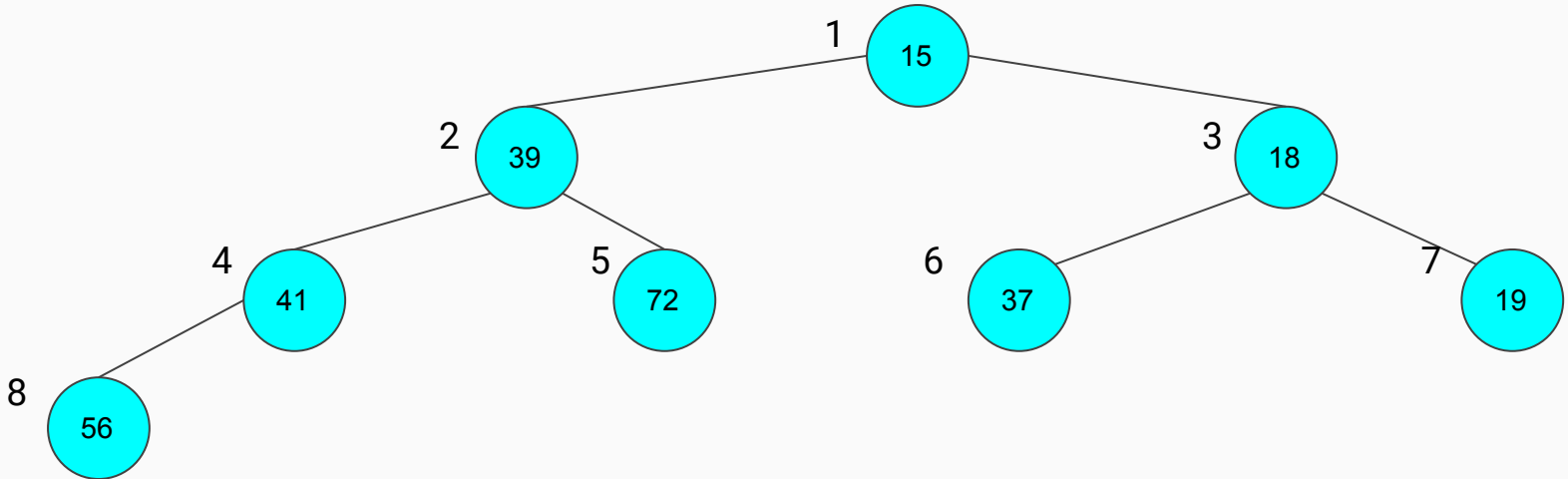
Ejemplo de Montículo



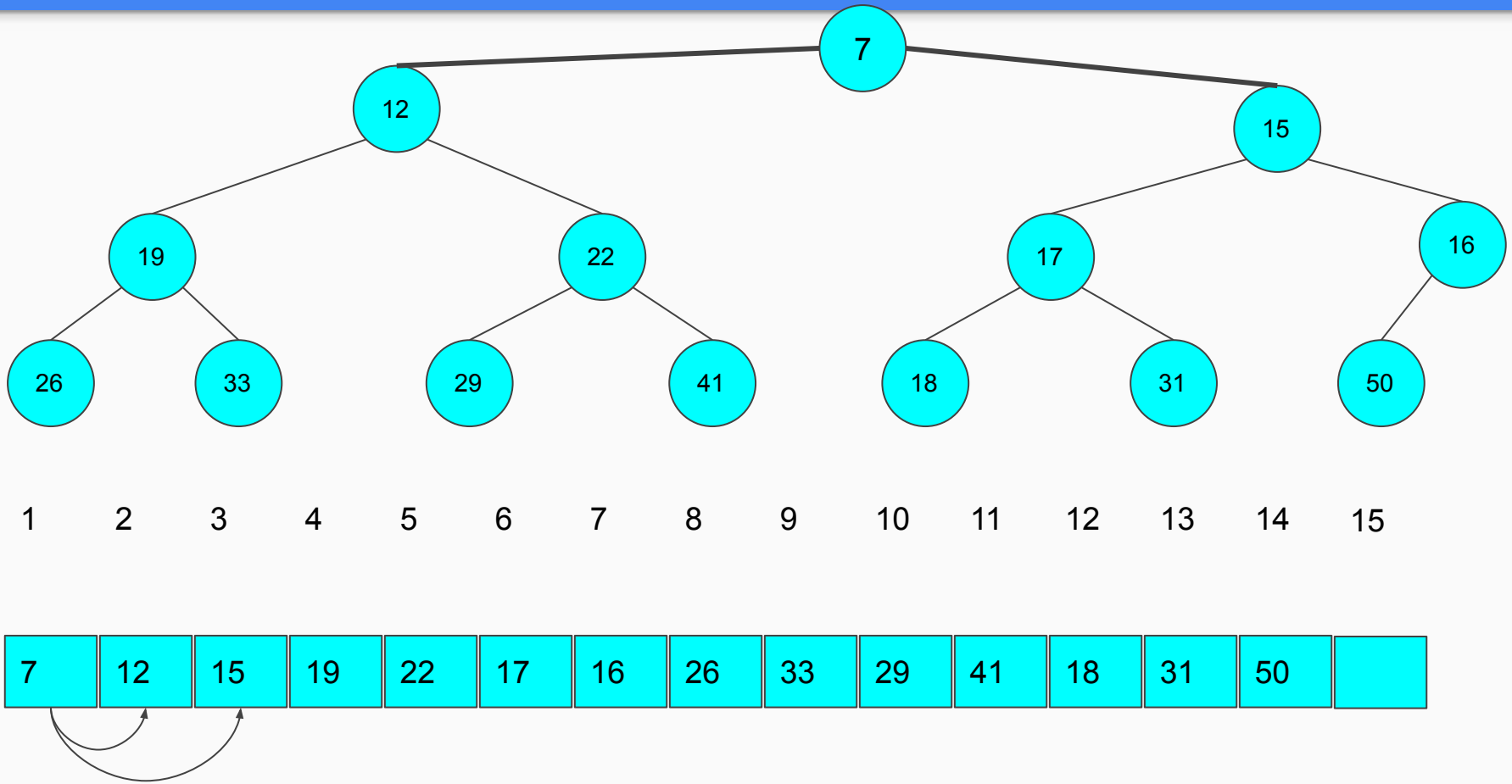
Representación

La forma secuencial de un montículo de `n` elementos implica que si `2*i` es mayor que `n` entonces el nodo `i` no tiene hijo izquierdo (tampoco hijo derecho), y si `2*i+1` es mayor entonces el nodo `i` no tiene hijo derecho.

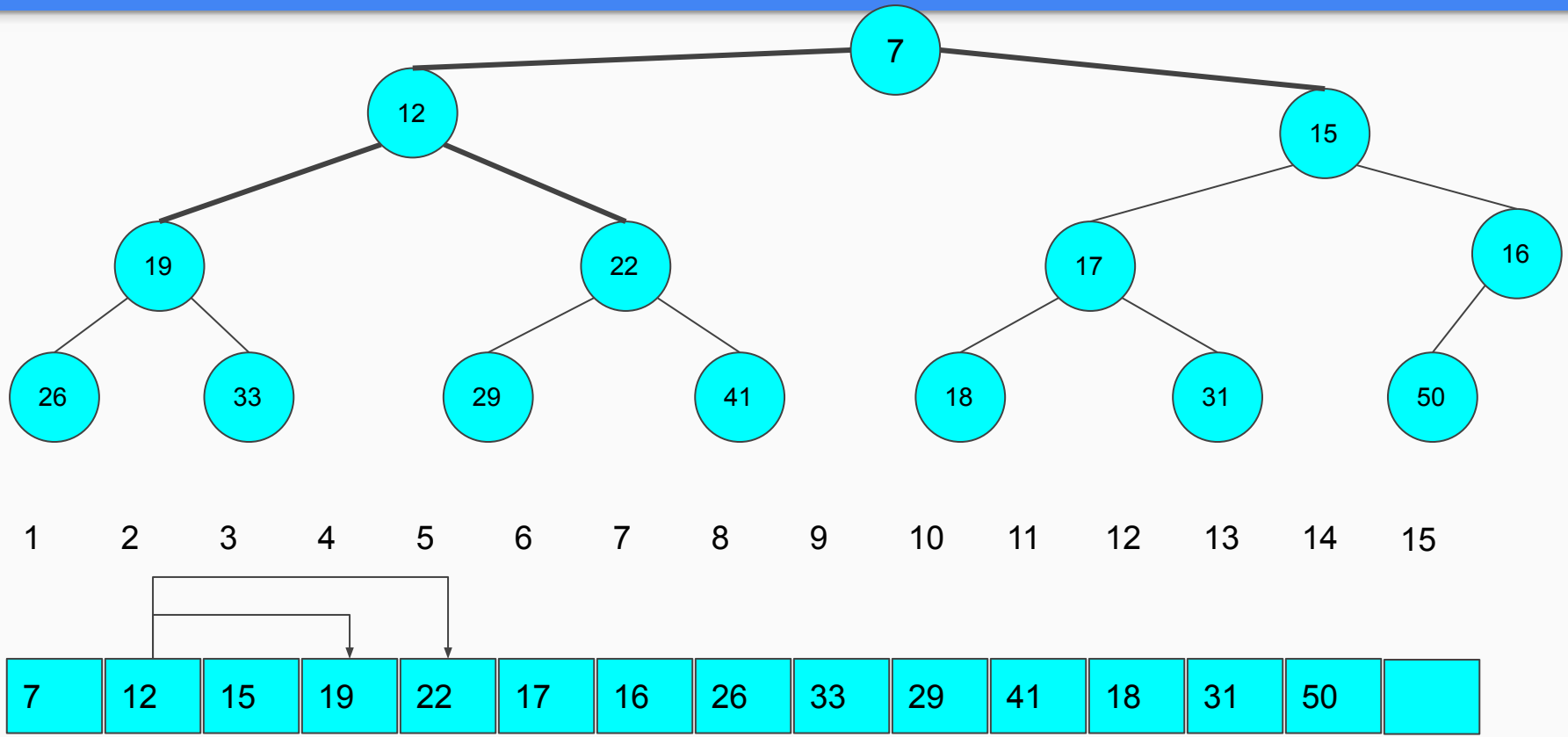
Un nodo que esté en la posición `i` su padre ocupa la posición `[i/2]`, el nodo hijo se ubica en la posición `2*i` y el nodo hijo derecho en `2*i+1`.



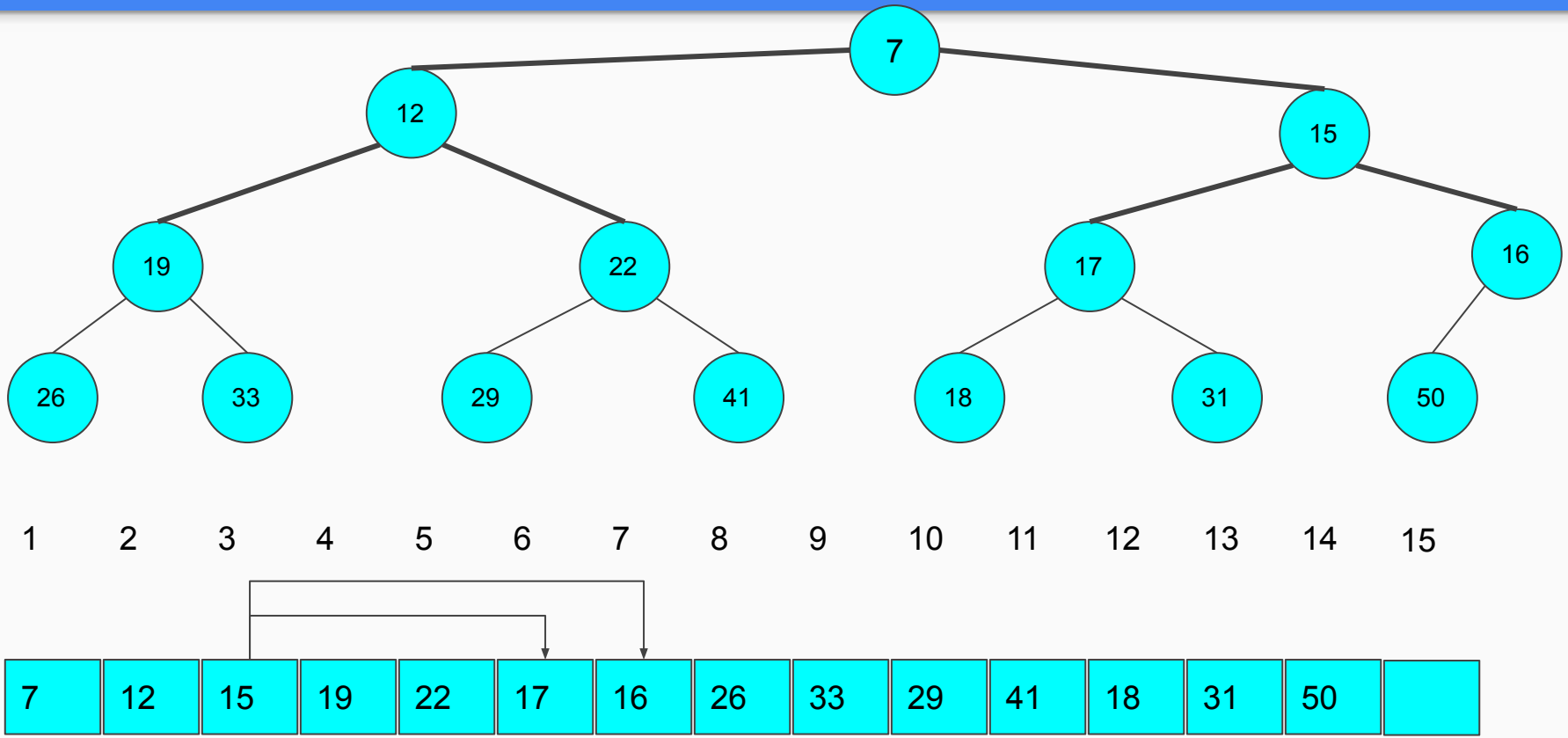
Montículo basado en vector



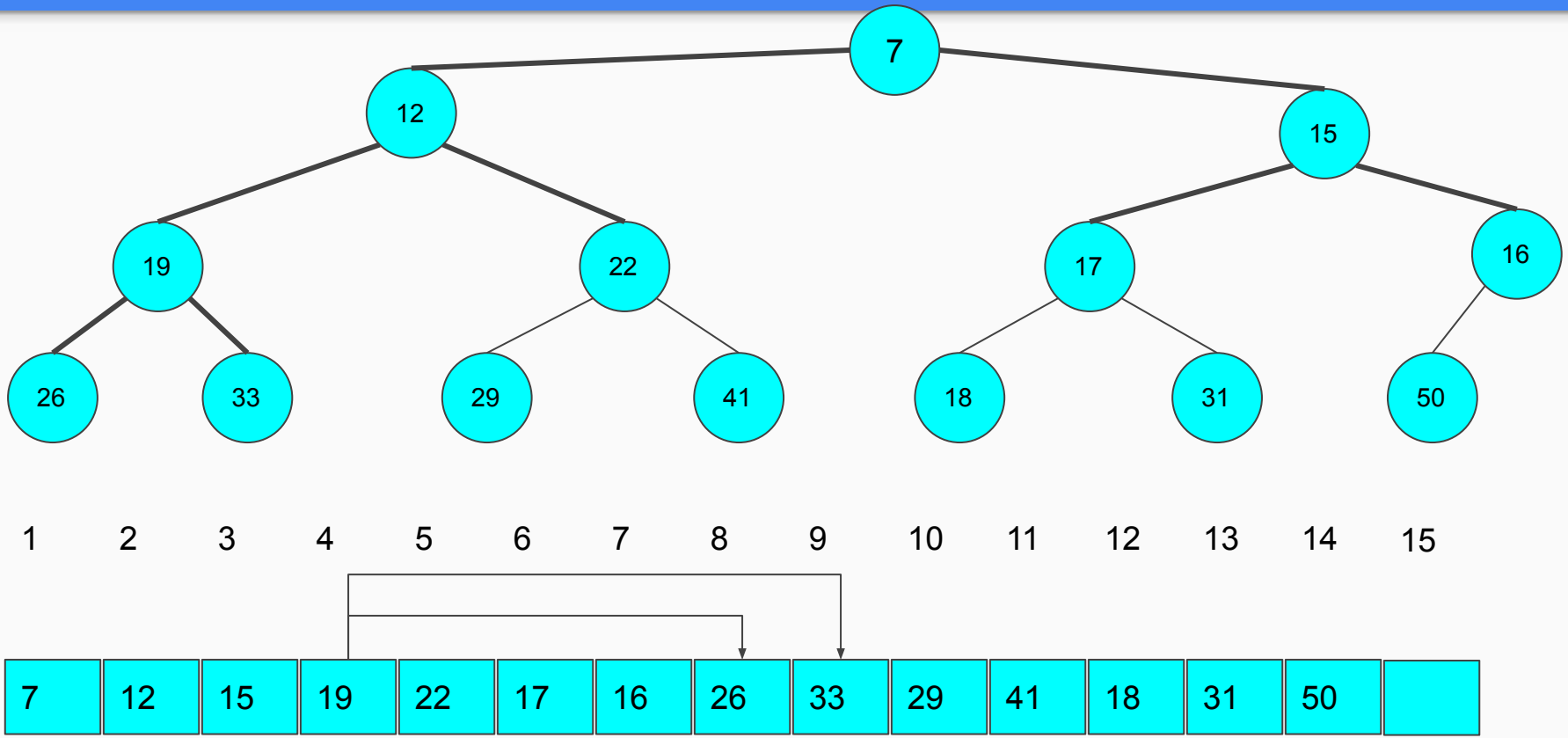
Montículo basado en vector



Montículo basado en vector



Montículo basado en vector



Operaciones

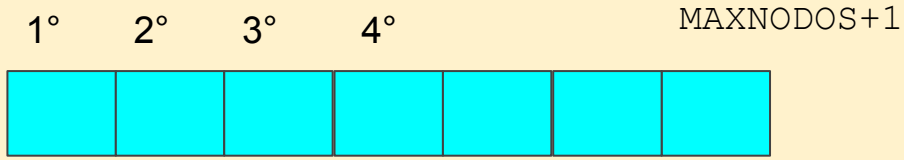
flotar

insertar

buscar mínimo

eliminar mínimo

```
/*archivo montículo.h*/

#define MAXNODOS 1001
struct monticulo{
    int v[MAXNODOS + 1];
    int maxNodos;
    int n;
}
typedef struct monticulo Monticulo;

/*Operaciones sobre el montículo*/
void flotar(Monticulo *mnlo, int i);
void insertarMonticulo(Monticulo *mnlo, int clave);
int buscarMinimo(Monticulo mnlo);
```



Operaciones

crear monticulo

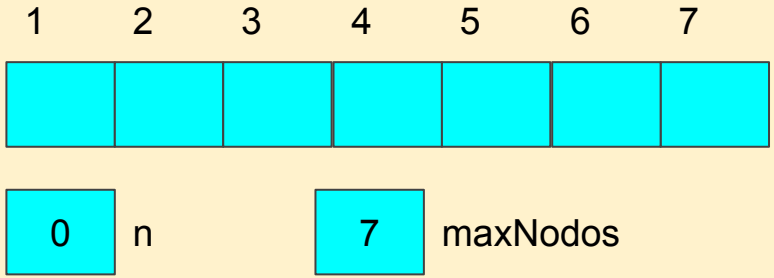
flotar

insertar

buscar mínimo

eliminar mínimo

```
/*archivo montículo.c*/

/*Crear montículo*/
void crearMonticulo(Montículo *mnlo){
    mnlo->n=0;
    mnlo->maxNodos=MAXNODOS;
}
```



Operaciones

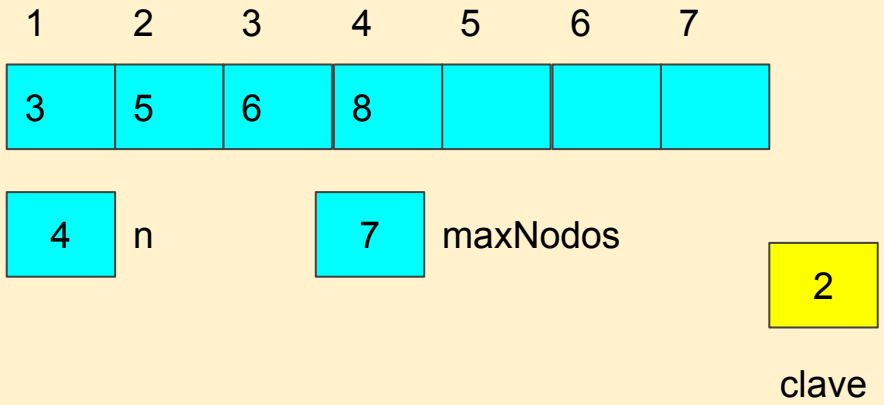
flotar

insertar

buscar mínimo

eliminar mínimo

```
/*archivo montículo.c*/
/*Insertar en el montículo*/
void insertarMonticulo(Montículo *mnlo, int
clave){
     if(mnlo->n == mnlo->maxNodos){
          puts("Montículo lleno\n");
     }
     (mnlo->n)++;
     mnlo->v[mnlo->n]=clave;

     flotar(mnlo,mnlo->n);
}
```



Operaciones

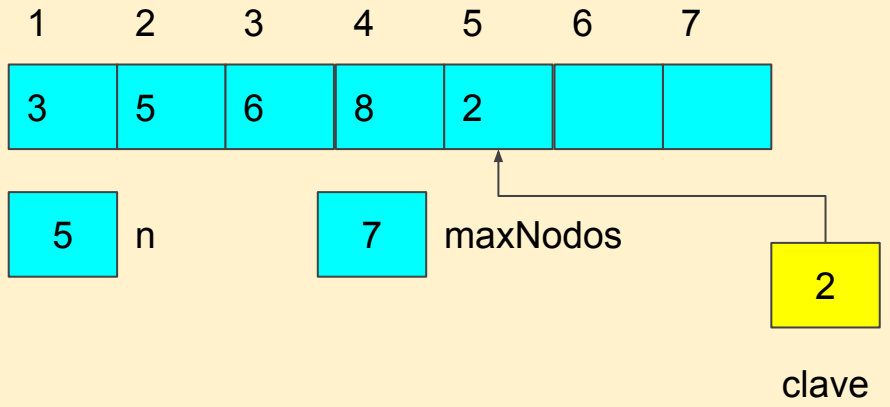
flotar

insertar

buscar mínimo

eliminar mínimo

```
/*archivo montículo.c*/
/*Insertar en el montículo*/
void insertarMonticulo(Montículo *mnlo, int
clave){
     if(mnlo->n == mnlo->maxNodos){
          puts("Montículo lleno\n");
     }
     (mnlo->n)++;
     mnlo->v[mnlo->n]=clave;

     flotar(mnlo,mnlo->n);
}
```



Operaciones

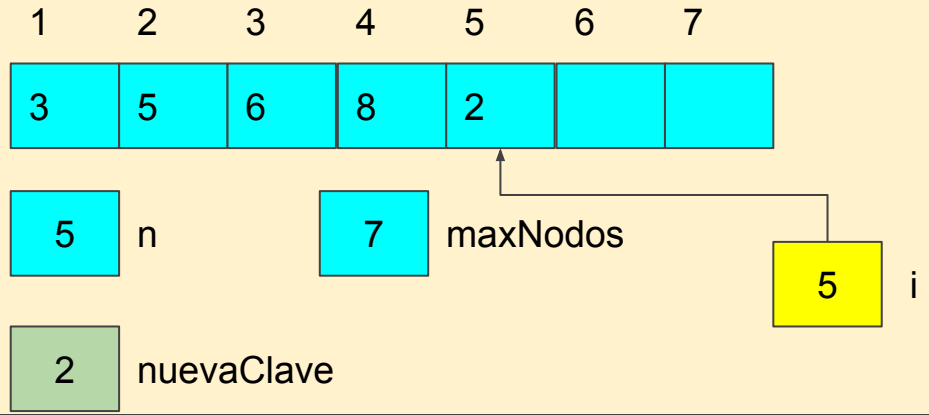
flotar

insertar

buscar mínimo

eliminar mínimo

```
/*archivo montículo.c*/
/*Lleva el menor en la raíz*/
void flotar(Montículo *mnlo, int i){
     int nuevaClave;
     nuevaClave = mnlo->v[i];
     while((i > 1) && (mnlo->v[i/2]> nuevaClave)){
          mnlo->v[i]=mnlo->v[i/2];/*baja el hueco*/
          i = i/2; /*sube un nivel en el árbol*/
     }
     mnlo->v[i] =  nuevaClave; /*situa la clave*/
                               /*en su posición*/
}
```



Operaciones

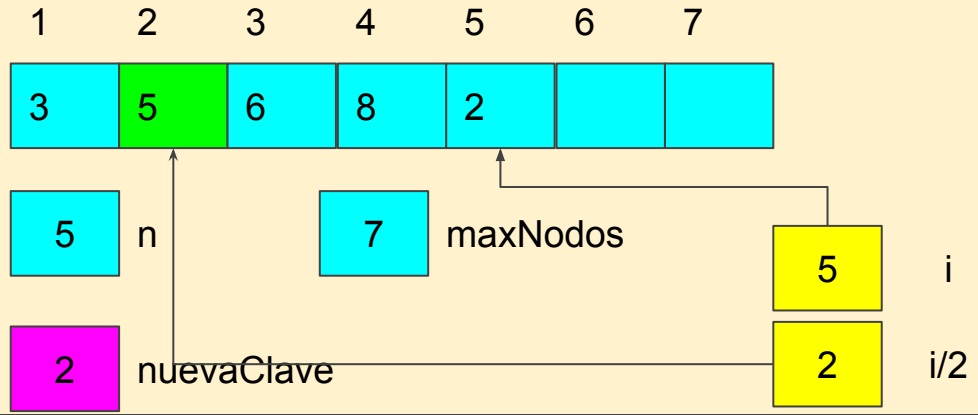
flotar

insertar

buscar mínimo

eliminar mínimo

```
/*archivo montículo.c*/
/*Lleva el menor en la raíz*/
void flotar(Montículo *mnlo, int i){
     int nuevaClave;
     nuevaClave = mnlo->v[i];
     while((i > 1) && (mnlo->v[i/2]> nuevaClave)){
          mnlo->v[i]=mnlo->v[i/2];/*baja el hueco*/
          i = i/2; /*sube un nivel en el árbol*/
     }
     mnlo->v[i] =  nuevaClave; /*situa la clave*/
                               /*en su posición*/
}
```



Operaciones

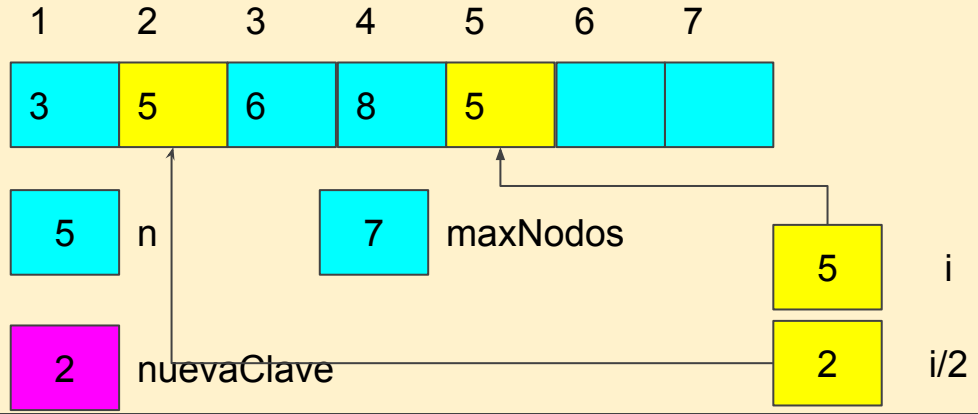
flotar

insertar

buscar mínimo

eliminar mínimo

```
/*archivo montículo.c*/
/*Lleva el menor en la raíz*/
void flotar(Montículo *mnlo, int i){
     int nuevaClave;
     nuevaClave = mnlo->v[i];
     while((i > 1) && (mnlo->v[i/2]> nuevaClave)){
          mnlo->v[i]=mnlo->v[i/2]; /*baja el hueco*/
          i = i/2; /*sube un nivel en el árbol*/
     }
     mnlo->v[i] =  nuevaClave; /*situa la clave*/
                               /*en su posición*/
}
```



Operaciones

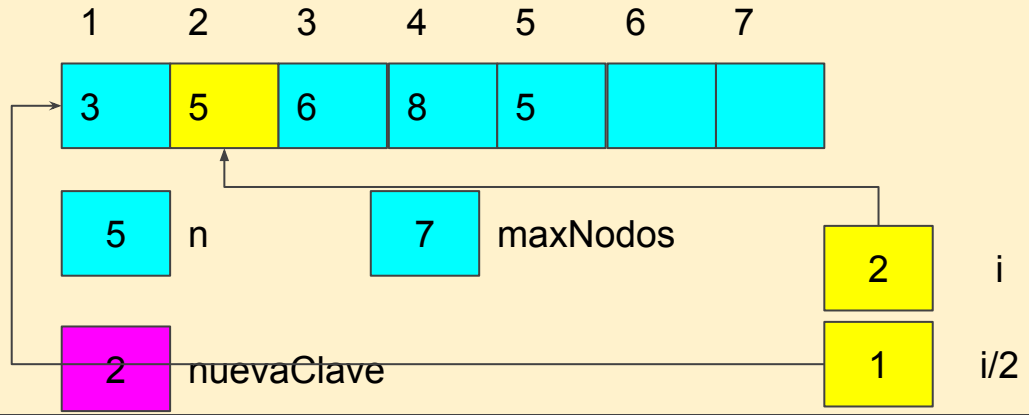
flotar

insertar

buscar mínimo

eliminar mínimo

```
/*archivo montículo.c*/
/*Lleva el menor en la raíz*/
void flotar(Montículo *mnlo, int i){
     int nuevaClave;
     nuevaClave = mnlo->v[i];
     while((i > 1) && (mnlo->v[i/2]> nuevaClave)){
          mnlo->v[i]=mnlo->v[i/2];/*baja el hueco*/
          i = i/2; /*sube un nivel en el árbol*/
     }
     mnlo->v[i] =  nuevaClave; /*situa la clave*/
                               /*en su posición*/
}
```



Operaciones

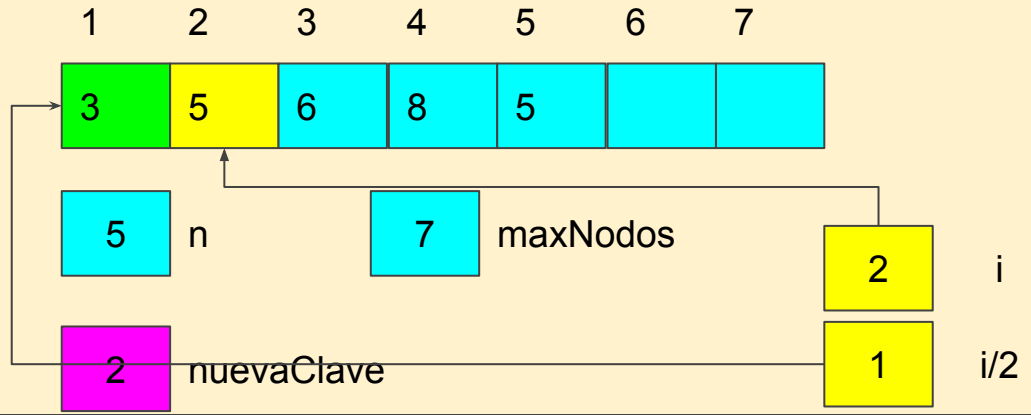
flotar

insertar

buscar mínimo

eliminar mínimo

```
/*archivo montículo.c*/
/*Lleva el menor en la raíz*/
void flotar(Montículo *mnlo, int i){
     int nuevaClave;
     nuevaClave = mnlo->v[i];
     while((i > 1) && (mnlo->v[i/2]> nuevaClave)){
          mnlo->v[i]=mnlo->v[i/2];/*baja el hueco*/
          i = i/2; /*sube un nivel en el árbol*/
     }
     mnlo->v[i] =  nuevaClave; /*situa la clave*/
                               /*en su posición*/
}
```



Operaciones

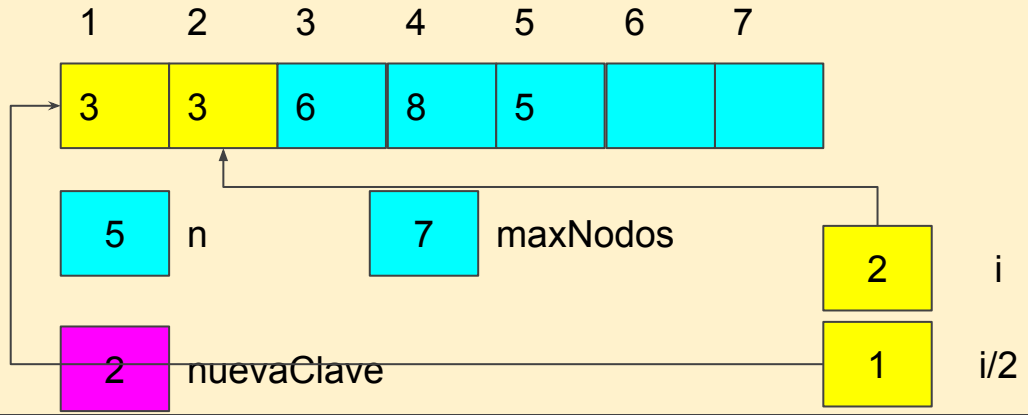
flotar

insertar

buscar mínimo

eliminar mínimo

```
/*archivo montículo.c*/
/*Lleva el menor en la raíz*/
void flotar(Montículo *mnlo, int i){
     int nuevaClave;
     nuevaClave = mnlo->v[i];
     while((i > 1) && (mnlo->v[i/2]> nuevaClave)){
          mnlo->v[i]=mnlo->v[i/2] ;/*baja el hueco*/
          i = i/2; /*sube un nivel en el árbol*/
     }
     mnlo->v[i] =  nuevaClave; /*situa la clave*/
                               /*en su posición*/
}
```



Operaciones

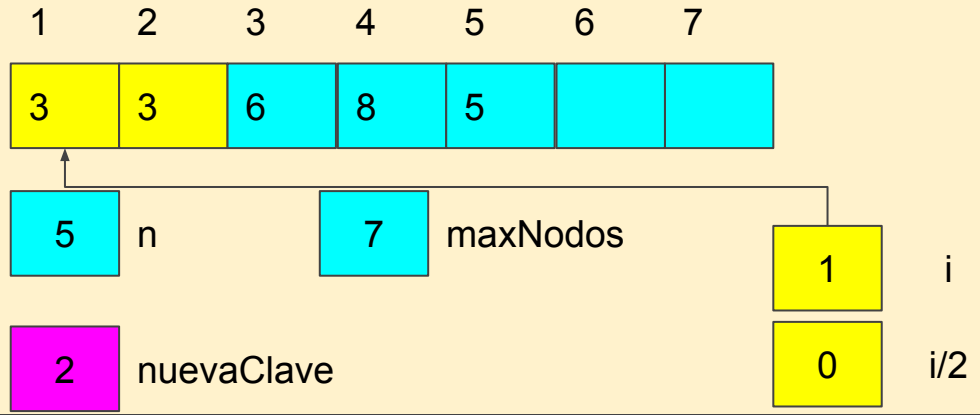
flotar

insertar

buscar mínimo

eliminar mínimo

```
/*archivo montículo.c*/
/*Lleva el menor en la raíz*/
void flotar(Montículo *mnlo, int i){
     int nuevaClave;
     nuevaClave = mnlo->v[i];
     while((i > 1) && (mnlo->v[i/2]> nuevaClave)){
          mnlo->v[i]=mnlo->v[i/2];/*baja el hueco*/
          i = i/2; /*sube un nivel en el árbol*/
     }
     mnlo->v[i] =  nuevaClave; /*situa la clave*/
                               /*en su posición*/
}
```



Operaciones

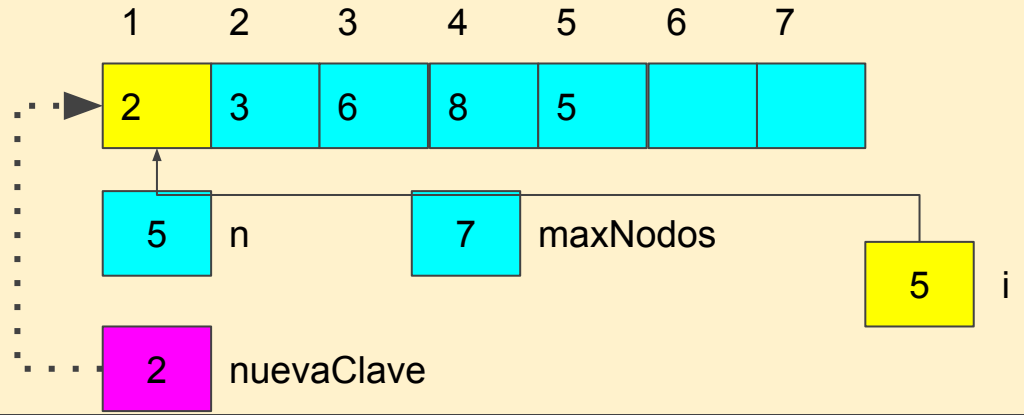
flotar

insertar

buscar mínimo

eliminar mínimo

```
/*archivo montículo.c*/
/*Lleva el menor en la raíz*/
void flotar(Montículo *mnlo, int i){
     int nuevaClave;
     nuevaClave = mnlo->v[i];
     while((i > 1) && (mnlo->v[i/2]> nuevaClave)){
          mnlo->v[i]=mnlo->v[i/2];/*baja el hueco*/
          i = i/2; /*sube un nivel en el árbol*/
     }
     mnlo->v[i] =  nuevaClave; /*sitúa la clave*/
                               /*en su posición*/
}
```



Operaciones

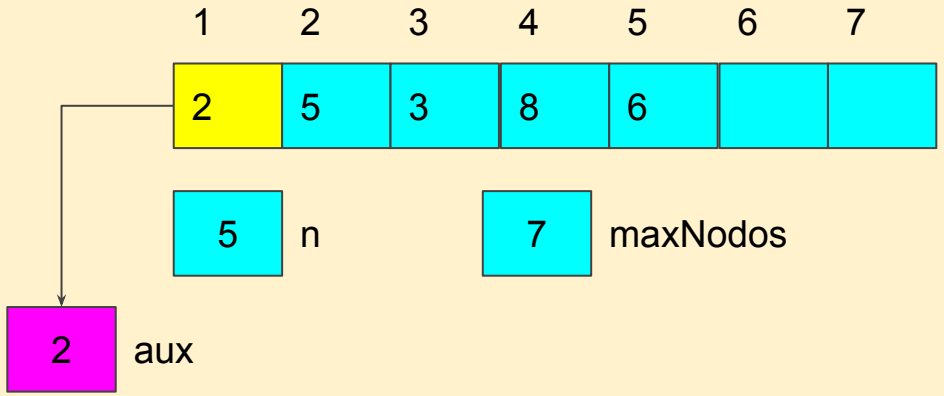
flotar

insertar

buscar mínimo

eliminar mínimo

```
/*archivo montículo.c*/

/*Lleva el menor en la raíz*/
int buscarMinimo(Montículo *mnlo){
     int aux = 0;
     if (!esVacio(mnlo)){
          aux = mnlo -> v[1];
     }
     return aux;
}
```



Operaciones

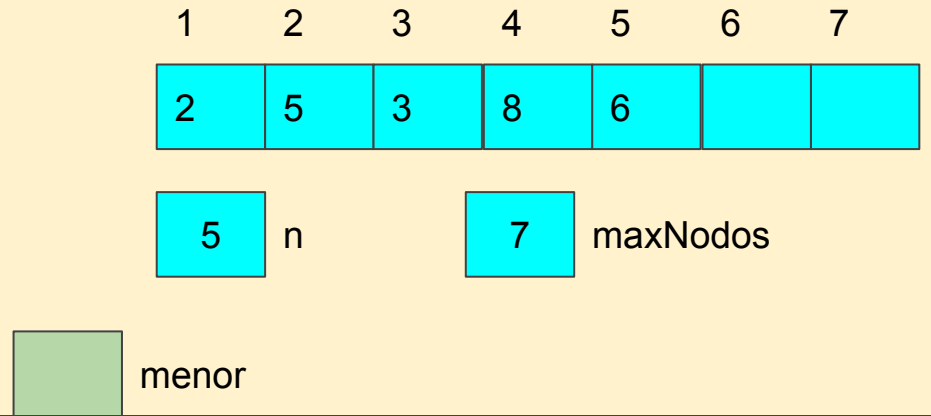
flotar

insertar

buscar mínimo

eliminar mínimo

```
/*archivo monticulo.c*/
/*Elimina el elemento de la raíz*/
int eliminarMinimo(Montículo *mnlo){
     int menor = -1;
     if(!esVacio(mnlo)){
          menor = buscarMínimo(mnlo);
          mnlo->v[1]=mnlo->v[mnlo->n];
          (mnlo->n)--;

          criba(mnlo->v, 1 , mnlo->n);
     }
     return menor;
}
```



Operaciones

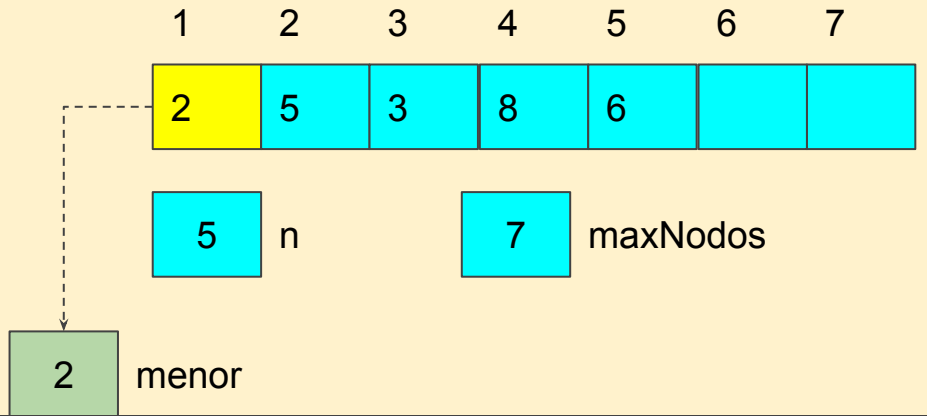
flotar

insertar

buscar mínimo

eliminar mínimo

```
/*archivo monticulo.c*/
/*Elimina el elemento de la raíz*/
int buscarMinimo(Montículo *mnlo){
     int menor = -1;
     if(!esVacio(mnlo)){
          menor = buscarMínimo(mnlo);
          mnlo->v[1]=mnlo->v[mnlo->n];
          (mnlo->n)--;

          criba(mnlo->v, 1 , mnlo->n);
     }
     return menor;
}
```



Operaciones

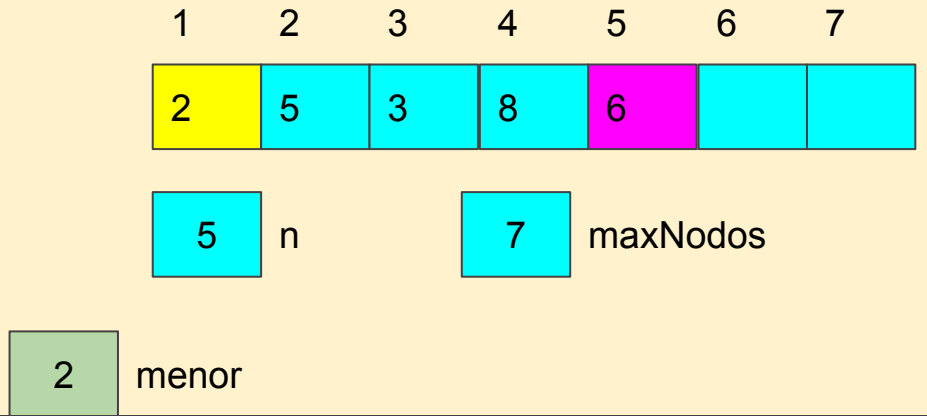
flotar

insertar

buscar mínimo

eliminar mínimo

```
/*archivo monticulo.c*/
/*Elimina el elemento de la raíz*/
int buscarMinimo(Montículo *mnlo){
     int menor = -1;
     if(!esVacio(mnlo)){
          menor = buscarMínimo(mnlo);
          mnlo->v[1]=mnlo->v[mnlo->n];
          (mnlo->n)--;

          criba(mnlo->v, 1 , mnlo->n);
     }
     return menor;
}
```



Operaciones

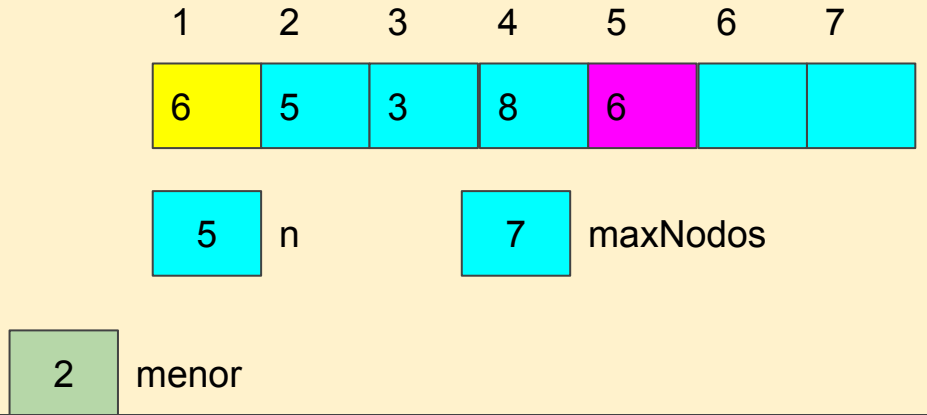
flotar

insertar

buscar mínimo

eliminar mínimo

```
/*archivo monticulo.c*/
/*Elimina el elemento de la raíz*/
int buscarMinimo(Montículo *mnlo){
     int menor = -1;
     if(!esVacio(mnlo)){
          menor = buscarMínimo(mnlo);
          mnlo->v[1]=mnlo->v[mnlo->n];
          (mnlo->n)--;

          criba(mnlo->v, 1 , mnlo->n);
     }
     return menor;
}
```



Operaciones

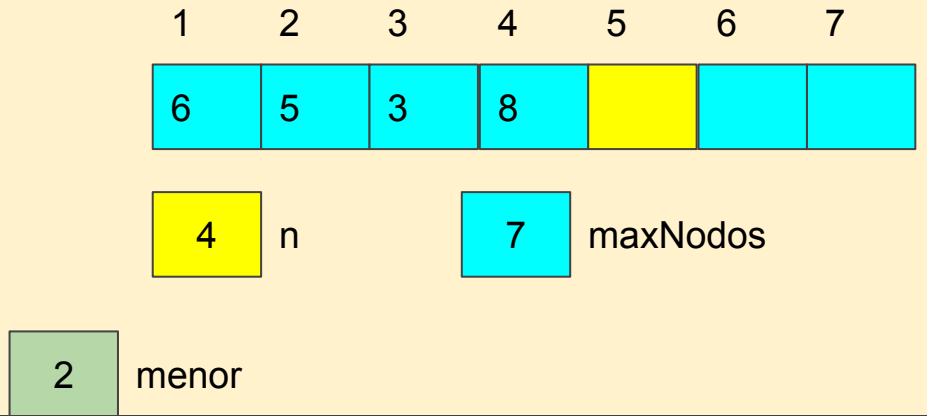
flotar

insertar

buscar mínimo

eliminar mínimo

```
/*archivo monticulo.c*/
/*Elimina el elemento de la raíz*/
int buscarMinimo(Montículo *mnlo){
     int menor = -1;
     if(!esVacio(mnlo)){
          menor = buscarMínimo(mnlo);
          mnlo->v[1]=mnlo->v[mnlo->n];
          (mnlo->n)--;

          criba(mnlo->v, 1 , mnlo->n);
     }
     return menor;
}
```



Operaciones

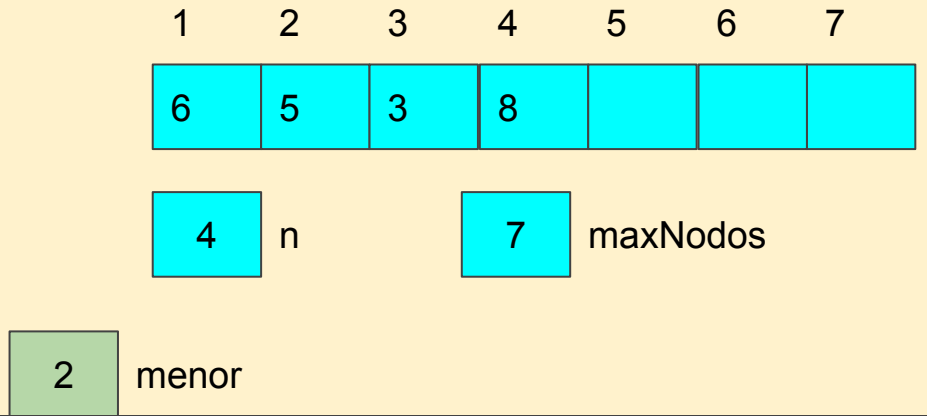
flotar

insertar

buscar mínimo

eliminar mínimo

```
/*archivo monticulo.c*/
/*Elimina el elemento de la raíz*/
int buscarMinimo(Montículo *mnlo){
     int menor = -1;
     if(!esVacio(mnlo)){
          menor = buscarMínimo(mnlo);
          mnlo->v[1]=mnlo->v[mnlo->n];
          (mnlo->n)--;

          criba(mnlo->v, 1, mnlo->n);
     }
     return menor;
}
```



Operaciones

flotar

insertar

buscar mínimo

eliminar mínimo

```
/*archivo montículo.c*/
/*Ordena el montículo*/
void criba(int v[], int primero, int ultimo){
    int esMonticulo = 0, hijo;

    while( (primero <= ultimo/2) && !esMonticulo){
        if(2*primero == ultimo){
            hijo = 2*primero;
        }else{
            if(v[2*primero] < v[2*primero +1]){
                hijo = 2*primero;
            }else{
                hijo = 2*primero + 1;
            }
            if(v[hijo] < v[primero]){
                swap(&v[primero],&v[hijo]);
                primero = hijo;
            }else{
                esMonticulo = 1;
            }
        }
    }
}
```

Operaciones

flotar

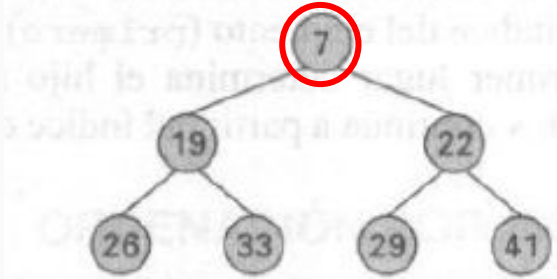
insertar

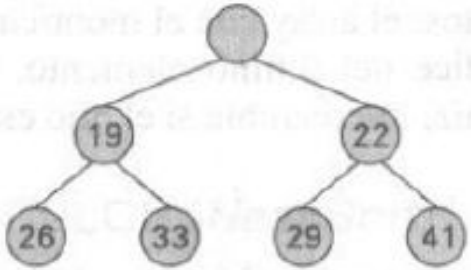
buscar mínimo

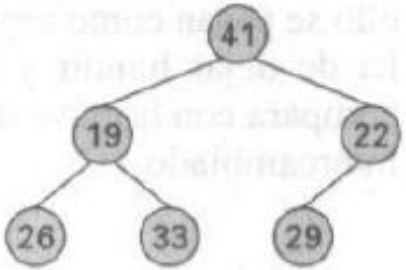
eliminar mínimo

```
/*archivo montículo.c*/
/*Ordena el montículo*/
void criba(int v[], int primero, int ultimo){
    int esMonticulo = 0, hijo;

    while( (primero <= ultimo/2) && !esMonticulo){
        if(2*primero == ultimo){
            hijo = 2*primero;
        }else{
            if(v[2*primero] < v[2*primero +1]){
                hijo = 2*primero;
            }else{
                hijo = 2*primero + 1;
            }
            if(v[hijo] < v[primero]){
                swap(&v[primero],&v[hijo]);
                primero = hijo;
            }else{
                esMonticulo = 1;
            }
        }
    }
}
```

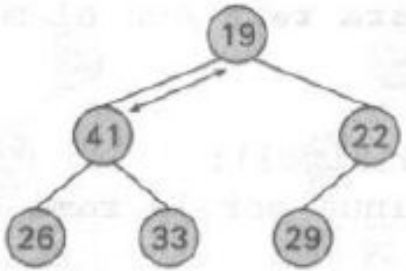
Ejemplo de funcionamiento de la Criba

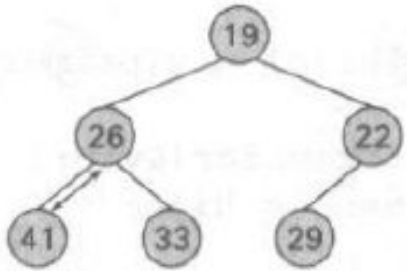




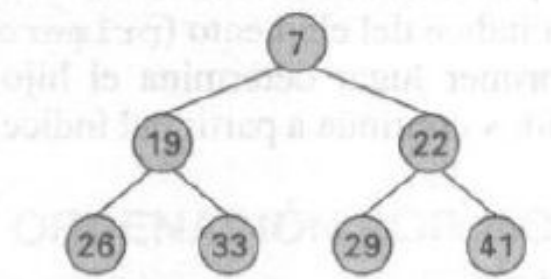


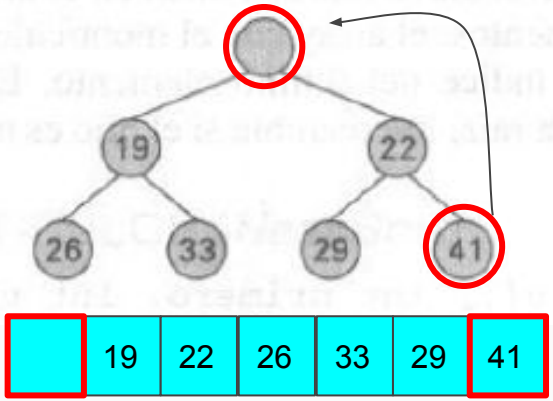
7	19	22	26	33	29	41

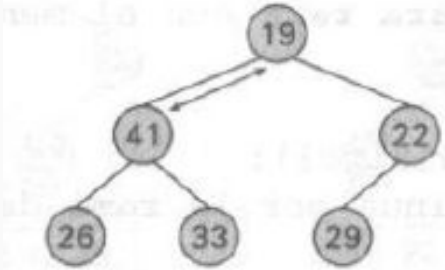


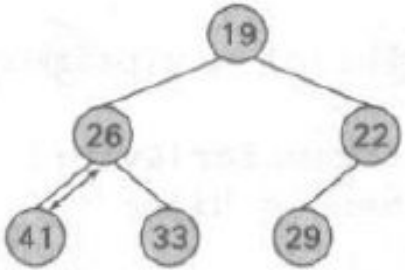


Ejemplo de funcionamiento de la Criba

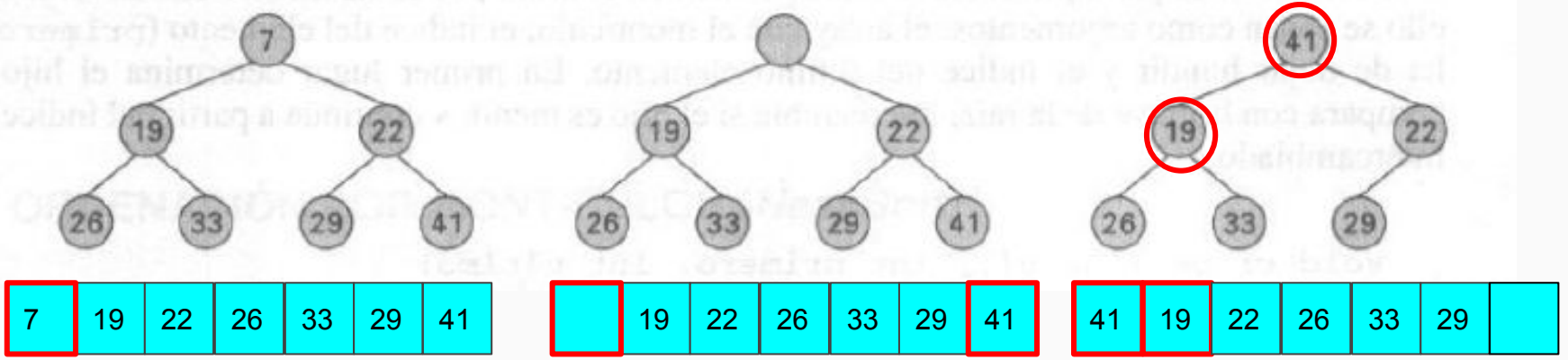


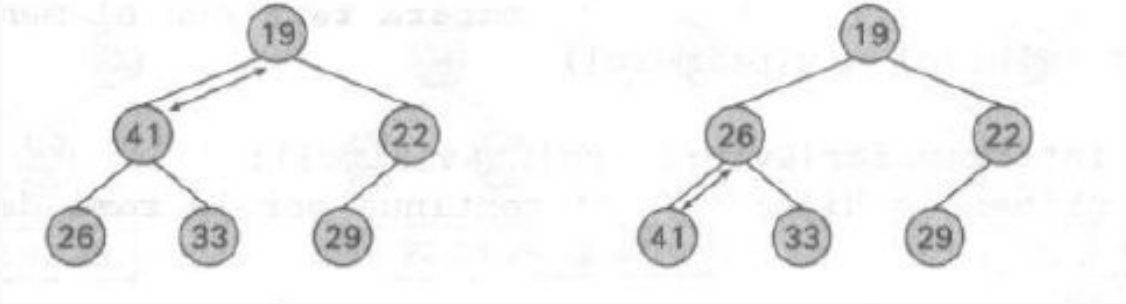




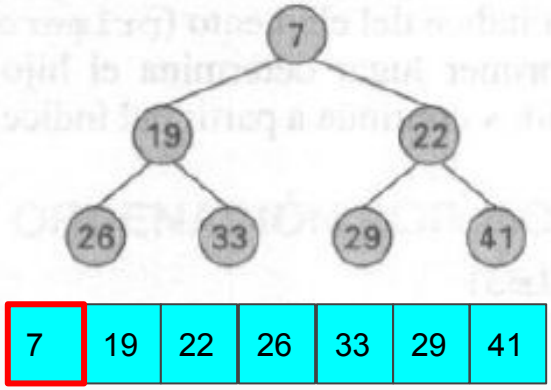


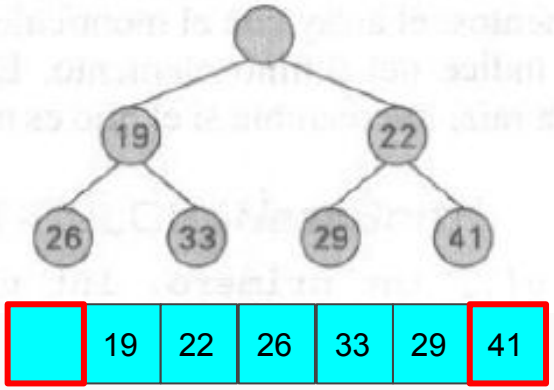
Ejemplo de funcionamiento de la Criba

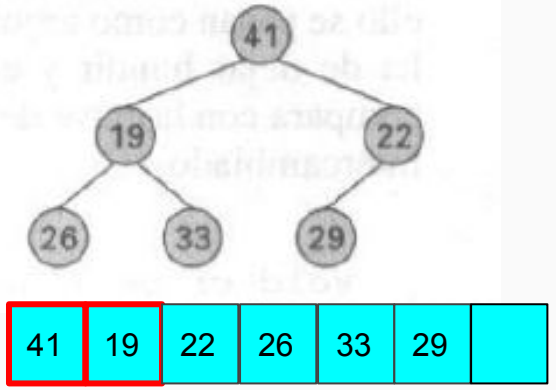


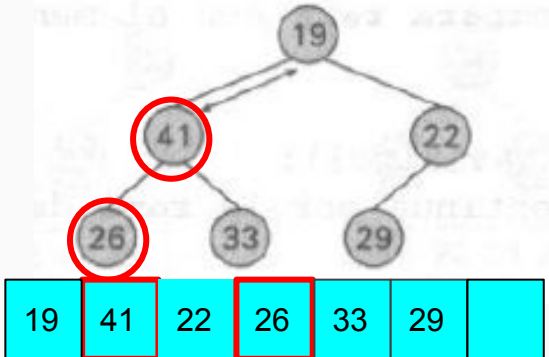


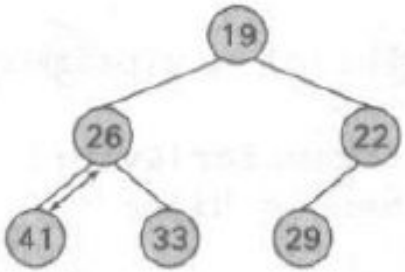
Ejemplo de funcionamiento de la Criba



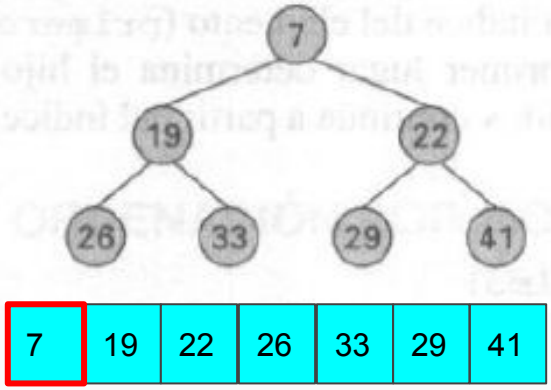


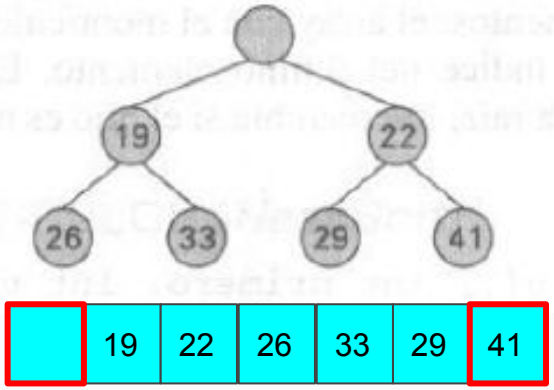


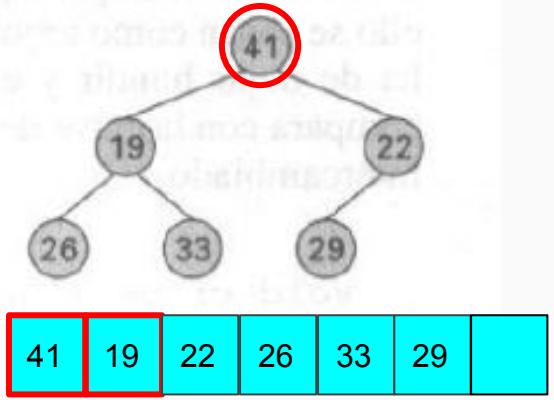


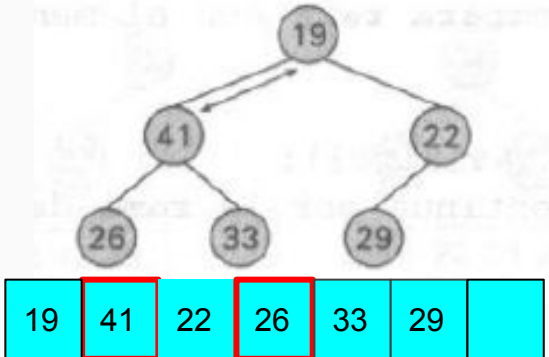


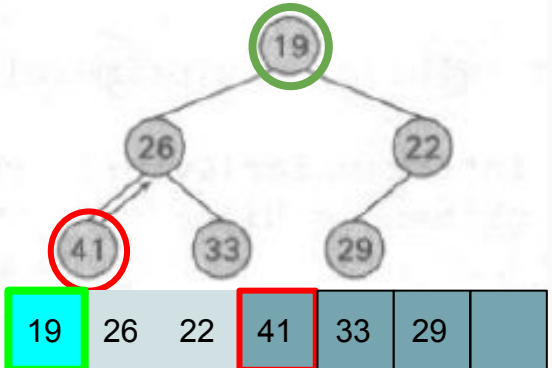
Ejemplo de funcionamiento de la Criba











Aplicación del Montículos: Ordenamiento

Paso 1: cada elemento del vector A no ordenado se inserta en montículo M y ordenar montículo

Paso 2: cada elemento mayor del montículo M quitar e insertar en vector A