Segundo parcial

NOMBRE: Aneth Michelle Tamariz Moreno

EE: Estructuras de Datos

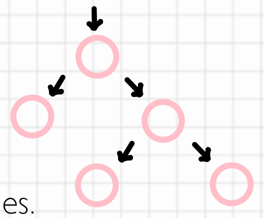
PERIODO: AGO 2022 – ENE 2023

Árbol binario de búsqueda

CARACTERÍSTICAS

Representa una colección de datos del mismo tipo.
Es una estructura NO LINEAL;

- Todos los elementos excepto el primero, tienen un antecesor.
- Todos los elementos tiene como máximo dos sucesores.



Es un caso particular de árboles generales.

aplicaciones

En los compiladores para representar estructuras sintácticas.

En los manejadores de Bases de Datos como índices (B, B+)

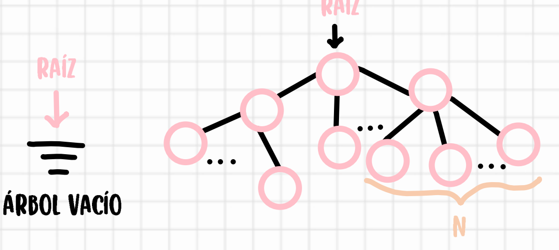
En los videojuegos para simular estrategias. (ajedrez, gato, ect.)

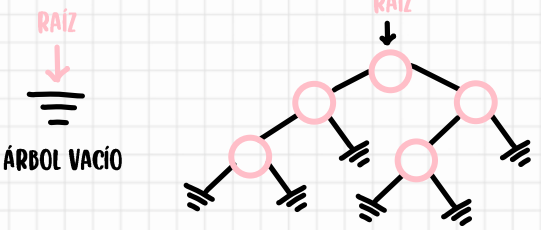
Mantener secuencias ordenadas y facilitar una búsqueda.

CONCEPTOS De árboles

1. ARBOL GENERAL

Es una colección de datos del mismo tipo tal que:
1) Está vacío o bien,
2) Contiene un elemento llamado raíz y N colecciones que también son árboles.





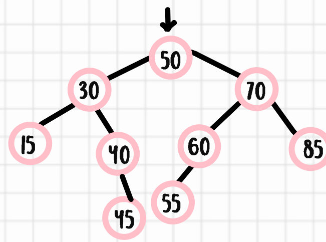
2. ARBOL BINARIO

Es una colección de datos del mismo tipo tal que:
1) Está vacío o bien,
2) Contiene un elemento llamado raíz y 2 colecciones que también son árboles binarios (árbol izquierdo y árbol derecho.)

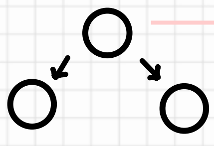
Árbol binario de búsqueda

3 ARBOL BINARIO DE BUSQUEDA

Es un árbol binario en el que cada nodo es mayor que todos los de su árbol izquierdo y menor que todos los del derecho.



Relación padre hijo

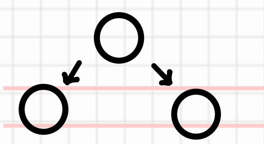


PADRE (NODO QUE APUNTA A OTRO LADO)

HIJO (NODO QUE ES APUNTADO POR OTRO)

RAÍZ: PRIMER NODO QUE SE CREA Y ES POR DONDE SE INICIA EL ACCESO. TODO ÁRBOL NO VACÍO TIENE UNA SOLA RAÍZ.

Relación de hermanos



NODOS QUE TIENEN EL MISMO PADRE.

NODOS ANCESTROS. VIENEN CONECTADOS HACIA ARRIBA.

NODOS SUCESORES. TIENEN CONEXIÓN HACIA ABAJO.

NODO TERMINAL U HOJA. NODO QUE NO TIENE HIJOS.

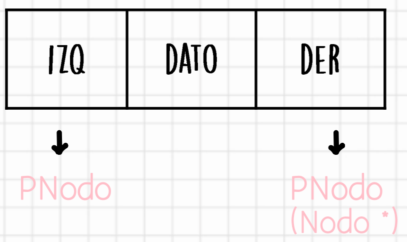
NODO INTERNO. NO ES NI RAÍZ, NI HOJA.

- EL NIVEL DE LA RAÍZ ES 1
- EL NIVEL DE CUALQUIER NODO ES UNO MÁS QUE SU PADRE.

ALTURA/PROFUNDIDAD. ES EL NIVEL MÁXIMO DE SUS NODOS.

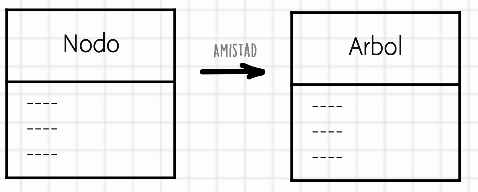
CAMINO. CUALQUIER SECUENCIA ENTRE DOS NODOS. (RAMAS DESDE LA RAÍZ HASTA UN NODO HOJA)

Representación de un Nodo



(int x) -> (int &x)

(Nodo * p) -> (Nodo *&p)
-> (Pnodo & p)




```
Class Nodo{
    int dato;
    Nodo * izq;
    Nodo * der;
    friend class Arbol; // De esta manera se ahorran setters y getters.

    public:
      Nodo ( );
      Nodo (int x);
};


type def int entero; // Ejemplo de cómo asignar un nombre alternativo a tipos existentes.
entero x;

type def Nodo* pNodo;

class Arbol{
    pNodo raiz;

    public :
      Arbol( );
      ~Arbol( );
      int estaVacio( );
      int getDato();
      pNodo getRaiz( );
                // Paso por referencia porque son los únicos que permiten modificar el árbol.
      void agregaNodo(int x, pNodo &p);
      void agregar(int x);
      void eliminarNodo(int x, pNodo &p);
      void eliminar(int x);
// complementarios
      int getMenor(pNodo p);
      int getMayor(pNodo p);
      void mostrar(pNodo p);
      int esHoja(int x, pNodo p);
      int altura(int x, pNodo p);
};
```

Árbol binario de búsqueda

TEMA 1

TEMA 2

TEMA 3

```
Nodo :: Nodo ( ){
    dato = 0;
    izq = NULL;
    der = NULL;
}

Nodo :: Nodo ( ){
    dato = x;
    izq = NULL;
    der = NULL;
}

Arbol :: Arbol ( ){
    raiz = NULL;
}

pNodo Arbol :: getRaiz( ){
    return raíz;
}

int Arbol :: estaVacio( ){
    return raíz == NULL;
}

int Arbol :: getDato( ){
    if(estaVacio( )){
        cout << "Arbol vacio en getDato( )"<<endl;
        return -1
    }
    return raiz -> dato;
}

void Arbol :: agregaNodo(int x, pNodo &p){
    if(p == NULL)
        p = new Nodo(x);
    else if( x < p-> dato)
        agregaNodo(x, p -> izq);
    else if ( x > p -> dato)
        agregaNodo(x, p -> der);
    else{
        //El dato ya existe
    }
}

void Arbol :: agregar (int x){
            agregaNodo(x, raiz);
}
```

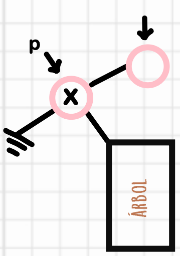
Árbol binario de búsqueda

```
int Arbol :: estax (int x, pNodo p){
    if( p == NULL)
       return 0;
    else if (p -> dato == x)
       return 1;
    else if( x < p -> dato)
       return estax(x, p -> izq);
    else
       return estax(x, p -> der);
}

int Arbol :: getMenor (pNodo p)
    if (estaVacio( )){
       cout << "Arbol vacio en getMenor";
       return -1;
    }
    if (p -> izq == NULL)
       return p -> dato;
    else
       return getMenor(p -> izq);
}
```

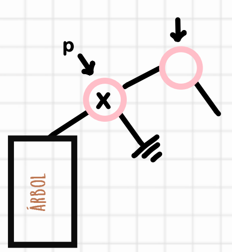
ELIMINAR nodo

CASO 1



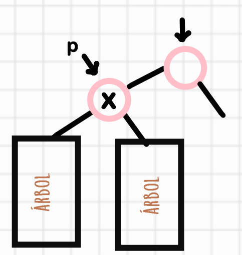
Cuando el dato a eliminar no tiene hijo izquierdo.

CASO 2



Sin hijo derecho.

CASO 3



Ambos hijos.

Tres pasos.

1. Apuntador auxiliar q (apuntando al mismo nodo que p)
2. p toma el valor de la parte derecha. (p = dirección del árbol)
3. delete q.

Árbol binario de búsqueda

```
void Arbol :: eliminaNodo ( int x, pNodo &p){
    if (p == NULL)
        return;
    if (p -> dato == x){
        if(p -> izq == NULL){
            pNodo q;
            q = p;
            p = p -> der;
            delete q;
        }
        else if (p -> der == NULL){
            pNodo q;
            q = p;
            p = p -> izq;
            delete q;
        }
        else {
            int m = getMenor( p -> der);
            p -> dato = m;
            eliminaNodo(m, p-> der);
        }
    }

    else if ( x < p -> dato)
        eliminaNodo(x, p -> izq);
    else
        eliminaNodo(x, p -> der);
}

void Arbol :: eliminar(int x){
    eliminaNodo(x, raíz);
}

Arbol :: ~Arbol( ){
    while(!estaVacio( )){
        eliminar(getDato( ));
    }
}
```

RECORRIDO DE UN árbol binario DE BÚSQUEDA

RECORRIDO EN PREORDEN

1. Visitar la raíz
2. Recorrer en preOrden el árbol izquierdo
3. Recorrer en preOrden el árbol derecho.

$\underbrace{50}_{R}\ \underbrace{25\ 5\ 10\ 30}_{AI}\ \underbrace{75\ 60\ 53\ 100}_{AD}$

RECORRIDO EN INORDEN

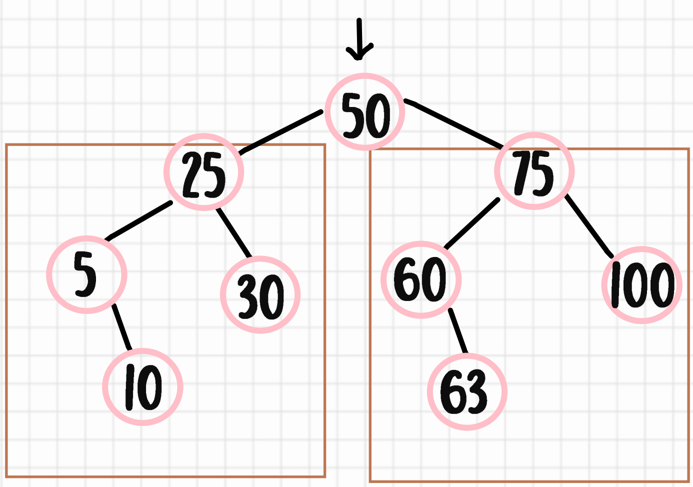
1. Recorrer en InOrden el árbol izquierdo
2. Visitar la raíz
3. Recorrer en InOrden el árbol derecho

$\underbrace{5\ 10\ 25\ 30}_{AI}\ \underbrace{50}_{R}\ \underbrace{53\ 60\ 75\ 100}_{AD}$

RECORRIDO EN POSORDEN

1. Recorrer en posOrden el árbol izquierdo
2. Recorrer en posOrden el árbol derecho
3. Visitar la raíz

$\underbrace{10\ 5\ 30\ 25}_{AI}\ \underbrace{53\ 60\ 100\ 75}_{AD}\ \underbrace{50}_{R}$



Recorrido de un árbol binario de búsqueda

TEMA 1

TEMA 2

TEMA 3

```
void Arbol:: preOrden( pNodo p){
    if ( p != NULL){
       cout << p -> dato << " ";
       preOrden( p -> izq );
       preOrden( p -> der);
    }
}

void Arbol::inOrden(pNodo p){
    if(p!=NULL){
       inOrden(p->izq);
       cout<<"["<<p->nombre<<", "<<p->frecuencia<<"]"<<endl;
       inOrden(p->der);
    }
}

void Arbol::posOrden(pNodo p){
    if(p!=NULL){
       posOrden(p->izq);
       posOrden(p->der);
       cout<<"["<<p->nombre<<", "<<p->frecuencia<<"]"<<endl;
    }
}
```

```
intArbol :: nNodos(pNodo p){
    if (p == NULL)
       return 0;
    else
       return 1 + nNodos(p -> izq) + nNodos(p -> der);
}

int Arbol :: esHoja(int x, pNodo p){
    if (p == NULL)
       return 0;
    else (p -> dato == x){
       if (p -> izq == NULL && p -> der == NULL)
          return 1;
       else
          return 0;
    }
    else if( x < p -> dato)
       return esHoja(x, p -> izq );
    else
       return esHoja(x, p -> der);
}
```

Árbol binario de búsqueda

```
int Arbol :: nivel (int x, pNodo p){
    if(!estaX (x, p))
        return 0;
    if ( x == p -> dato)
        return 1;
    else if ( x < p -> dato)
        return 1 + nivel (x, p -> izq);
    else
        return 1 + nivel (x, p -> der);
}

int Arbol :: altura (pNodo p){
    if( p == NULL)
        return 0;
    int ai = altura( p -> izq );
    int ad = altura ( p -> der );
    if ( ai > ad )
        return ai + 1;
    else
        return ad + 1;
}
```

TEMA 1

TEMA 2

TEMA 3

Métodos de ordenación

TEMA 1
TEMA 2
TEMA 3

OPERACION ORDENAR

Consiste en reagrupar o reacomodar un conjunto de datos en un orden determinado (Ascendente o descendente)

Métodos de ordenación

Simples
Directos

- CORTOS Y FÁCILES DE PROGRAMAR
- SON DE COMPLEJIDAD CUADRÁTICA
- SE APLICA A CONJUNTOS PEQUEÑOS

Complejos

- MÁS LARGOS DE PROGRAMAR
- SON DE COMPLEJIDAD LOGARÍTMICA
- SE APLICA A CONJUNTOS GRANDES

ANÁLISIS DE COMPLEJIDAD

Mide la eficiencia de un algoritmo en términos del número de comparaciones y el número de intercambios.

DIRECTOS

- Intercambios (Burbuja) – Bubble sort
- Selección – Selection sort
- Inserción – Insertion sort

COMPLEJOS

- Shell sort
- Mezcla – Merge sort
- Rápido – Quick sort
- Montículos – Heap sort

LINEALES

- Bucket sort
- Counting sort
- Radix sort

MÉTODO DE ORDENACIÓN POR INTERCAMBIOS (BURBUJA)

Consiste en comparar y ordenar parejas adyacentes del conjunto (intercambiarlas si es necesario), de tal manera que el dato menor se mueva a la primera posición. De la misma manera se ordena el segundo y así sucesivamente hasta ordenar el penúltimo dato.

Se hacen n - 1 pasos.

Este algoritmo obtiene su nombre de la forma con la que suben por la lista los elementos durante los intercambios, como si fueran unas "burbujitas". La más grande es la última que se ordena.

n = 6

i j j j j j
A: 55 22 44 11 66 33

11 44 33 66
11 22
11 55

i empieza en el primer dato, mientras que la j empieza en el último. j es el índice que hace las comparaciones.

PASO 1:

i j j j j
A: 11 55 22 44 33 66

22 55 33 44

Comparaciones: n - 1
En cada paso se va reduciendo el número de comparaciones.

* Podemos asegurar que el 11 está en su lugar

PASO 2:

i j j j
A: 11 22 55 33 44 66

33 55

* En este paso 2 podemos asegurar que 11 y 22 están en su lugar

PASO 3:

i j j
A: 11 22 33 55 44 66

44 55

No podemos saber con exactitud cuántas intercambios hará, pero se puede estimar.

[0, (n -1)]

PASO 4:

i j
A: 11 22 33 44 55 66

PASO 5:

i
j
A: 11 22 33 44 55 66

El algoritmo aunque esté ordenado, ejecuta todo el procedimiento.

ANÁLISIS DE COMPLEJIDAD

n = 6

Número de Comparaciones

P1	5	
P2	4	
P3	3	15 comparaciones
P4	2	
P5	1	

NumComp = $\frac{(n-1)((n-1)+1)}{2} = \frac{(n-1)n}{2} = \frac{n^2-n}{2}$

NumComp = (n-1) + (n-2) + . . . + 1

Gauss, príncipe de las matemáticas ideó una fórmula después de que su profesor como castigo les ordenó sumar todos los números del 1 al 100, por lo que se dio cuenta que al sumar 1 + 100, 2 + 99, 3+ 98 y así consecutivamente el resultado siempre era 101 en esos 50 pares. A lo que obtuvo la siguiente formula para la suma aritmética del 1 a n:

$$\sum_{k=1}^{n} k = \frac{n(n+1)}{2}$$

Si n = 1000, bubble haría 499500 comparaciones.

Número de Intercambios.

- No se puede determinar con exactitud por lo que se deberá
- Calcular un promedio

NumInt = $\frac{mejor\ caso + peor\ caso}{2} = \frac{0 + \frac{n^2-n}{2}}{2} = \frac{n^2-n}{4}$

Si n = 1000, el bubble sort hace un promedio de 249750 intercambios

	MEJOR CASO	CASO PROMEDIO	PEOR CASO
N. COMP n= 1000	$\frac{n^2-n}{2}$ 499,500 comparaciones	$\frac{n^2-n}{2}$ 499,500 comparaciones	$\frac{n^2-n}{2}$ 499,500 comparaciones
N. INT n= 1000	0 0 intercambios	$\frac{n^2-n}{4}$ 249,750 intercambios	$\frac{n^2-n}{2}$ 499,500 intercambios
TOTAL n= 1000	$\frac{n^2-n}{2}$ 499,500 operaciones	$\frac{n^2-n}{2} + \frac{n^2-n}{4}$ 749,250 operaciones	n^2-n 999,999 operaciones

PSEUDOCÓDIGO

```
void ordIntercambio ( int a [ ], int n ) INICIO
   PARA i <- o, n-2   HACER
                nC ++
     PARA j <- n – 1, i + 1 HACER
       Si a [ j ]  <  a [ j – 1 ] Entonces
                nI ++
         aux <- a [ j ];
         a [ j ] <- a [ j - 1 ];
         a [ j - 1 ] <- aux;
       FIN_SI
     FIN_PARA
   FIN_PARA
FIN_ordIntercambio
```

Ordenación por selección

SELECTION SORT

Consiste en localizar la posición del dato menor en intercambiarlo con el de la primera posición.
Después localizar la posición del siguiente menor e intercambiarla con el segundo dato, y así sucesivamente hasta intercambiar el último menor buscando con el penúltimo dato.
Se hacen n - 1 pasos

$n = 6$

	pm i	pm j	pm j	pm j	j	j
A:	55	22	44	11	66	33
	11			55		

PASO 1:

		pm i	j	j	j	j
A:	11	22	44	55	66	33

* En este paso, el número 11 se encuentra ordenado

PASO 2:

			pm i	j	j	j
A:	11	22	44	55	66	33
			44			33

PASO 3:

				pm i	j	j
A:	11	22	33	55	66	44
				44		55

PASO 4:

					pm i	j
A:	11	22	33	44	66	55
					55	66

PASO 5:

A:	11	22	33	44	55	66

ANÁLISIS DE COMPLEJIDAD

$n = 6$

Número de Comparaciones

P1	n- 1 = 5	
P2	n- 2 = 4	
P3	n- 3 = 3	15 comparaciones
P4	n- 2 = 2	
P5	n- 1 = 1	

$$\text{NumComp} = \frac{(n-1)((n-1)+1)}{2} = \frac{(n-1)n}{2} = \frac{n^2-n}{2}$$

Si n = 1000, selection sort haría 499,500 comparaciones.

Número de Intercambios.

- Siempre hace n – 1 intercambios
- Constante, hay una ligera ventaja en intercambios porque no hace movimientos cuando ya están ordenados.

$$\text{NumInt} = n - 1$$

Si n = 1000, el selection sort hace 999 intercambios

	MEJOR CASO	PROMEDIO	PEOR CASO
Burbuja	499500 op	749250 op	999000 op
Selección	500499 op	500499 op	500499 op
Inserción	999 op	500000 op	999999 op

PSEUDOCÓDIGO

```
void ordSeleccion ( int a [ ], int n ) INICIO

    PARA i <- 0, n-2  HACER

        pm <- i;
        PARA j <- i+1, n - 1 HACER
          SI a [ j ]  <  a [ pm ] ENTONCES

             pm <- j;

          FIN_SI
        FIN_PARA
             aux <- a [ i ];
             a [ i ] <- a [ pm ];
             a [ pm ] <- aux;
    FIN_PARA
FIN_ordSeleccion
```

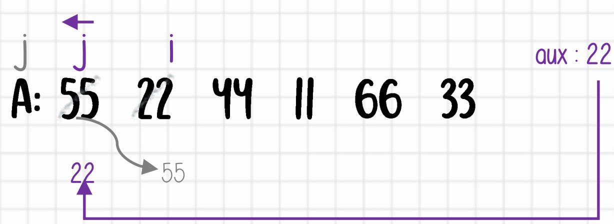
MÉTODO DE ORDENACIÓN POR INSERCIÓN

El primer dato se considera ordenado.
A partir del segundo dato se hace lo siguiente:

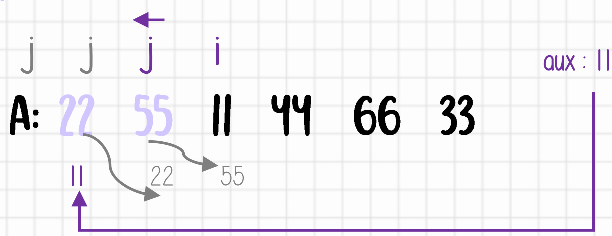
- Se busca el lugar correcto con el conjunto izquierdo, moviendo los datos que sean mayores un lugar a la derecha.
- Para posteriormente mover (o insertar) el dato a ordenar en el lugar del último dato que se movió a la derecha.

- Si ninguno se movió, entonces el dato; ya está en su lugar.
- Se hacen $n - 1$ pasos

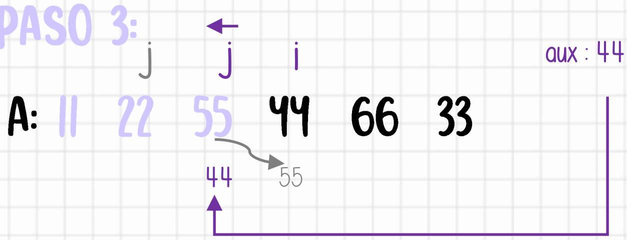
$n = 6$



PASO 2:



PASO 3:



* Se inserta en j+1 (mientras sea j > -1)

Un intercambio es cuando se mueve un dato de una posición a otra.
Mientras que el número de comparaciones siempre se hace con dos datos (se comparan valores, NO posiciones)

PASO 4:

j i aux : 66

A: 11 22 44 55 **66** **33**

66

PASO 5:

j j j j i aux : 33

A: 11 22 44 55 66 **33**

33 44 55 66

PASO 6:

A: 11 22 33 44 55 66

	MEJOR CASO	CASO PROMEDIO	PEOR CASO
N. COMP Si n= 1000	$n - 1$ 999 comparaciones	$\frac{n^2+n-2}{4}$ 250250 comparaciones	$\frac{n^2-n}{2}$ 499,500 comparaciones
N. INT Si n= 1000	0 0 intercambios	$\frac{n^2-n}{4}$ 249,750 intercambios	$\frac{n^2-n}{2}$ 499,500 intercambios
TOTAL Si n= 1000	$n-1$ 999 operaciones	$\frac{n^2-1}{2}$ 500,000 operaciones	n^2-n 999,999 operaciones