**Juan Ignacio Vera**

***Punto 1(Ejercicio 23):***

**class** **Nodo** {

**int** info;

**Nodo** izquierda, derecha;

**Nodo** (**int** valor)

{ info = valor;

izquierda = **null**;

derecha = **null**; }

**void** **setInfo**(**int** valor)

{ info = valor; }

**void** **setIzquierda**(**Nodo** dir)

{ izquierda = dir; }

**void** **setDerecha**(**Nodo** dir)

{ derecha = dir; }

**int** **getInfo**()

{ **return** info; }

**Nodo** **getIzquierda**()

{ **return** izquierda; }

**Nodo** **getDerecha**()

{ **return** derecha; }

**boolean** **esHoja**()

{

**return** (derecha == **null** && izquierda == **null**);

}

**Nodo** **getUnicoHijo**()

{

**if** ( derecha == **null** ) **return** izquierda;

**else** **return** derecha;

}

**boolean** **tieneUnHijo**()

{

**if** (derecha == **null** && izquierda != **null**)

**return** **true**;

**else** **if** (derecha != **null** && izquierda == **null**)

**return** **true**;

**else**

**return** **false**;

}

**Nodo** **buscaMenor**(**Nodo** p)

{

**Nodo** **aux** = p.getIzquierda();

**while** ( aux.getDerecha() != **null**)

aux = aux.getDerecha();

**return** aux;

}

}

**public** **class** **ArbolBinario**

{

**Nodo** raiz;

**Nodo** **getRaiz**()

{**return** **this**.raiz;}

**void** **setRaiz**(**Nodo** p)

{**this**.raiz=p;}

**void** **insertar** (**int** valor)

{

**Nodo** **Nuevo**, **P**, **Ant**;

Nuevo = **new** Nodo(valor);

P = raiz;

Ant = **null**;

**while** (P != **null**)

{Ant = P;

**if** (P.getInfo() > valor)

P = P.getIzquierda();

**else** P = P.getDerecha();

}

**if** ( Ant == **null**)

raiz = Nuevo;

**else**

**if** (Ant.getInfo() > valor)

Ant.setIzquierda(Nuevo);

**else** Ant.setDerecha(Nuevo);

}

**void** **eliminar**(**int** valor)

{

**Nodo** **p** = **this**.raiz;

**Nodo** **t** = **null**;

**while** ( p.getInfo() != valor )

{ t = p;

**if** ( valor > p.getInfo() )

p = p.getDerecha();

**else**

p = p.getIzquierda();

}

**if** ( p.esHoja() )

**if** ( t == **null** ) raiz = **null**;

**else** **if** ( p == t.getIzquierda() )

t.setIzquierda(**null**);

**else** t.setDerecha(**null**);

**else** **if** (p.tieneUnHijo())

**if** (t.getIzquierda() == p)

t.setIzquierda(p.getUnicoHijo());

**else**

t.setDerecha(p.getUnicoHijo());

**else**

{

**Nodo** **q** = p.buscaMenor(p);

p.setInfo(q.getInfo());

t = p;

p = p.getIzquierda();

**while** (p != q)

{

t = p;

p = p.getDerecha();

}

**if** (q==t.getIzquierda())

t.setIzquierda(q.getIzquierda());

**else**

**if** (q.tieneUnHijo())

t.setDerecha(q.getUnicoHijo());

**else**

t.setDerecha(**null**);

}

}

**void** **inorden** ( **Nodo** P)

{

**if** (P != **null**)

{

inorden (P.getIzquierda());

**System**.***out***.println(P.getInfo());

inorden (P.getDerecha());

}

}

**int** **Mayor** ( **Nodo** P)

{

**int** **Mayor**=P.getInfo();

**do**

{

Mayor=P.getInfo();

P=P.getDerecha();

}

**while**(P!=**null**);

**return** Mayor;

}

**boolean** **estaVacio**()

{

**boolean** **bandera**=**true**;

**if**(raiz==**null**)

{bandera=**false**;}

**return** bandera;

}

}

**public** **class** **ColaDePrioridad**

{

**ArbolBinario** elementos;

**ColaDePrioridad**()

{

elementos = **new** ArbolBinario();

}

**int** **suprimir**(**Nodo** r)

{

**int** **sacado** = elementos.Mayor(r);

elementos.eliminar(sacado);

**return** sacado;

}

**void** **insertar**(**int** nuevo)

{

elementos.insertar(nuevo);

}

**boolean** **estaVacia**()

{

**return** elementos.estaVacio();

}

//boolean estaLlena()

//{}

}

El método insertar es mejor con arboles binarios de búsqueda debido a que es menor el recorrido con respecto a los montículos, pero los arboles binarios de búsqueda no es muy bueno con lo que respecta a eliminar ya que hay que hacer un recorrido para buscar al de mayor prioridad

El método suprimir es mejor con montículos ya que no hay que hacer ningún recorrido ya que siempre se eliminara la raíz y después se dejara el inmediato inferior en su lugar, en cambio con el método insertar son mejores los arboles binarios ya que con los montículos hay que hacer un intercambio de valores si se introduce valores mayores, en cambio en los arboles binarios se van acomodando a la derecha de las hojas

***Punto 2:***

***mostrar pares y sumar pares con método recursivo:***

**int** **sumaPares**(**Nodo** P)

{

**int** **suma**=P.getInfo();

**if**(P.getDerecha()==**null** && P.getIzquierda()==**null**)

{

**System**.***out***.println(P.getInfo()+"es par");

**return** suma;

}**else**{

**if**(P.getInfo()%2==0)

{

suma+=sumaPares(P.getIzquierda());

suma+=sumaPares(P.getDerecha());

}**System**.***out***.println(P.getInfo()+"es par");

}

**return** suma;

}

***mostrar pares y sumar pares con iterador:***

**class** **NodoConPunteroPadre** {

**int** info;

**NodoConPunteroPadre** izquierda, derecha,padre;

**NodoConPunteroPadre** (**int** valor)

{ info = valor;

izquierda = **null**;

derecha = **null**;

padre=**null**;}

**NodoConPunteroPadre** **getPadre**() {

**return** padre;

}

**void** **setPadre**(**NodoConPunteroPadre** padre) {

**this**.padre = padre;

}

**void** **setInfo**(**int** valor)

{ info = valor; }

**void** **setIzquierda**(**NodoConPunteroPadre** dir)

{ izquierda = dir; }

**void** **setDerecha**(**NodoConPunteroPadre** dir)

{ derecha = dir; }

**int** **getInfo**()

{ **return** info; }

**NodoConPunteroPadre** **getIzquierda**()

{ **return** izquierda; }

**NodoConPunteroPadre** **getDerecha**()

{ **return** derecha; }

**boolean** **esHoja**()

{

**return** (derecha == **null** && izquierda == **null**);

}

**NodoConPunteroPadre** **getUnicoHijo**()

{

**if** ( derecha == **null** ) **return** izquierda;

**else** **return** derecha;

}

**boolean** **tieneUnHijo**()

{

**if** (derecha == **null** && izquierda != **null**)

**return** **true**;

**else** **if** (derecha != **null** && izquierda == **null**)

**return** **true**;

**else**

**return** **false**;

}

**NodoConPunteroPadre** **buscaMenor**(**NodoConPunteroPadre** p)

{

**NodoConPunteroPadre** **aux** = p.getIzquierda();

**while** ( aux.getDerecha() != **null**)

aux = aux.getDerecha();

**return** aux;

}

}

**public** **class** **ArbolBinarioConPunteroPadre**

{

**public** **class** **ArbolBinario**

{

**NodoConPunteroPadre** raiz;

**void** **setRaiz**(**NodoConPunteroPadre** p)

{**this**.raiz=p;}

**void** **insertar** (**int** valor)

{

**NodoConPunteroPadre** **Nuevo**, **P**, **Ant**;

Nuevo = **new** NodoConPunteroPadre(valor);

P = raiz;

Ant = **null**;

**while** (P != **null**)

{Ant = P;

**if** (P.getInfo() > valor)

P = P.getIzquierda();

**else** P = P.getDerecha();

}

**if** ( Ant == **null**)

raiz = Nuevo;

**else**

**if** (Ant.getInfo() > valor) {

Ant.setIzquierda(Nuevo);

Nuevo.setPadre(Ant);

}**else** {

Ant.setDerecha(Nuevo);

Nuevo.setPadre(Ant);

}

}

**void** **eliminar**(**int** valor)

{

**NodoConPunteroPadre** **p** = raiz;

**NodoConPunteroPadre** **t** = **null**;

**while** ( p.getInfo() != valor )

{ t = p;

**if** ( valor > p.getInfo() )

p = p.getDerecha();

**else**

p = p.getIzquierda();

}

**if** ( p.esHoja() )

**if** ( t == **null** ) raiz = **null**;

**else** **if** ( p == t.getIzquierda() )

t.setIzquierda(**null**);

**else** t.setDerecha(**null**);

**else** **if** (p.tieneUnHijo())

**if** (t.getIzquierda() == p) {

p.getUnicoHijo().setPadre(p.getPadre());

t.setIzquierda(p.getUnicoHijo());

}**else** {

p.getUnicoHijo().setPadre(p.getPadre());

t.setDerecha(p.getUnicoHijo());

}**else**

{

**NodoConPunteroPadre** **q** = p.buscaMenor(p);

p.setInfo(q.getInfo());

t = p;

p = p.getIzquierda();

**while** (p != q)

{

t = p;

p = p.getDerecha();

}

**if** (q==t.getIzquierda()) {

q.getIzquierda().setPadre(q.getPadre());

t.setIzquierda(q.getIzquierda());

}**else**

**if** (q.tieneUnHijo()) {

q.getUnicoHijo().setPadre(q.getPadre());

t.setDerecha(q.getUnicoHijo());

}**else**

t.setDerecha(**null**);

}

}

**void** **inorden** ( **NodoConPunteroPadre** P)

{

**if** (P != **null**) // caso base

{ // caso general

inorden (P.getIzquierda());

**System**.***out***.println(P.getInfo());

inorden (P.getDerecha());

}

}

}

}

**public** **class** **IteradorArbol**

{

**private** **Nodo** siguiente;

**public** **IteradorArbol**(**Nodo** raiz)

{

**this**.siguiente = raiz;

**if**(siguiente == **null**)

**return**;

**while** (siguiente.getIzquierda() != **null**)

**this**.siguiente = siguiente.getIzquierda();

}

**public** **boolean** **tieneSiguiente**()

{

**return** siguiente!=**null**;

}

**public** **Nodo** **actual**()

{

**return** **this**.siguiente;

}

**public** **Nodo** **siguiente**() {

**Nodo** **r**=siguiente;

**if**(siguiente.getDerecha() != **null**) {

**this**.siguiente = siguiente.getDerecha();

**while** (siguiente.getIzquierda() != **null**)

**this**.siguiente = siguiente.getIzquierda();

**return** r;

}

**while**(**true**) {

**if**(siguiente.getPadre() == **null**) {

siguiente = **null**;

**return** r;

}

**if**(siguiente.getPadre().getIzquierda() == siguiente) {

siguiente = siguiente.getPadre();

**return** r;

}

siguiente = siguiente.getPadre();

}

}

}

**public** **class** **MainArbolBinario** {

**public** **static** **void** **main**(**String**[] args)

{

**ArbolBinario** **ab**= **new** ArbolBinario();

ab.insertar(30);

ab.insertar(20);

ab.insertar(40);

ab.insertar(10);

ab.insertar(25);

ab.insertar(35);

ab.insertar(50);

ab.insertar(37);

ab.inorden(ab.getRaiz());

**System**.***out***.println(ab.sumaPares(ab.getRaiz()));

**IteradorArbol** **iterador**=ab.getIteradorArbol();

**while**(iterador.tieneSiguiente())

{

**int** **suma**=0;

**Nodo** **a**=iterador.actual();

**while**(iterador.tieneSiguiente())

{

**if**(a.getInfo()%2==0) {

System.out.println(a.getInfo()+" es par");

suma=suma+a.getInfo();

}

**if**(iterador.tieneSiguiente()) {

a=iterador.siguiente();}

**else**

{**break**;}

}

}

}