



TRABAJO PRACTICO

Grupo N°8

Cattaneo Luna Valentino

Maestromey Mateo

Roldan Lautaro

Roldan Lucas

El enunciado que nos tocó desarrollar fue el siguiente:

"Elabore un algoritmo que identifique y devuelva, en forma de lista, el valor resultante de elevar a la potencia enésima los nodos de un árbol binario que cuenten con exactamente un hijo. Calcule la complejidad del algoritmo."

Para realizar lo propuesto en el enunciado, decidimos trabajar con el código de "Árbol Binario" y "Nodos" que desarrollamos en la materia de Algoritmos y Estructuras de Datos II. Al conocer el funcionamiento de cada método, nos resultó más fácil entender qué debíamos programar para cumplir con la consigna.



DISEÑO

Al momento de diseñar el código para cumplir con la consigna, dividimos el funcionamiento en tres funciones:

- `tieneSoloUnHijo(nodo)`
- `potenciar(numero, potencia)`
- `potenciarHijosUnicos(arbol, potencia)`

tieneSoloUnHijo(nodo)

Esta función se encarga de verificar si el nodo que está siendo evaluado en ese momento cumple con la condición de tener solo un “hijo”, ya sea izquierdo o derecho.

Por lo tanto, lo que retorna esta función es un Boolean.

La complejidad temporal de esta función es $O(1)$.

```
public static boolean tieneSoloUnHijo(NodoArbol n) { // O(1)
    return (n.getNodoIzq().EsVacio() && !n.getNodoDer().EsVacio()) ||
           (!n.getNodoIzq().EsVacio() && n.getNodoDer().EsVacio());
}
```

potenciar(numero, potencia)

Esta función recibe dos parámetros: uno es el número a elevar y el otro es su exponente. Su tarea es elevar el número a la potencia pasada como parámetro. Por lo tanto, esta función retorna un Integer. Su complejidad temporal es de $O(\log(\text{potencia}))$.

```
public static Integer potenciar(int numero, int potencia) { //  $O(\log(\text{potencia}))$ 
    return (int) Math.pow(numero, potencia);
}
```

potenciarHijosUnicos(ArbolB ,potencia)

Esta función recibe dos parámetros: uno es el árbol binario y el otro es el exponente al cual se quieren elevar los valores de los nodos con un solo "hijo".

Esta función fue desarrollada utilizando la técnica de divide y conquista.

Primero, verifica que el árbol no esté vacío.

Si el árbol no está vacío, obtiene la raíz y aplica la función tieneSoloUnHijo(nodo).

Si esto se cumple, se obtiene el valor del nodo y se utiliza la función potenciar(numero, potencia) con el valor del nodo y la potencia pasada como parámetro.

```
public static ArrayList<Integer> potenciarHijosUnicos(ArbolB a, int potencia) { //O(n)
    ArrayList<Integer> lista = new ArrayList<>();

    if(!a.EsVacio()){

        if (tieneSoloUnHijo(a.getNodo())){
            lista.add( potenciar(a.Raiz(),potencia));
        }

        lista.addAll(potenciarHijosUnicos( (ArbolB) a.HijoIzquierdo(), potencia));
        lista.addAll(potenciarHijosUnicos( (ArbolB) a.HijoDerecho(), potencia));

    }
    return lista;
}
```

potenciarHijosUnicos(ArbolB ,potencia)

Una vez potenciado el valor, se agrega a una lista de enteros.

Después, la función se llama recursivamente dos veces, una para cada nodo "hijo" que tiene la raíz, repitiendo el proceso hasta llegar al final del árbol binario.

Esta función retorna un ArrayList<Integer>.

Todo este proceso tiene una complejidad temporal de $O(n)$.

```
public static ArrayList<Integer> potenciarHijosUnicos(ArbolB a, int potencia) { //O(n)
    ArrayList<Integer> lista = new ArrayList<>();

    if(!a.EsVacio()){

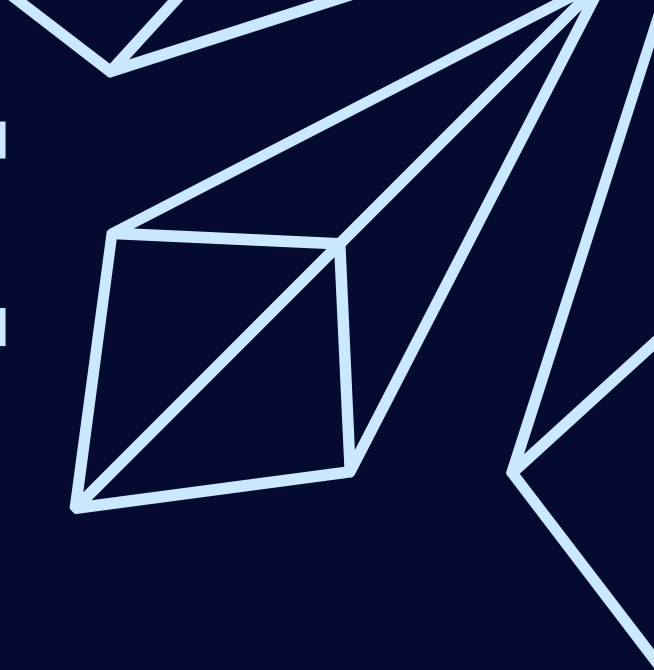
        if (tieneSoloUnHijo(a.getNodo())){
            lista.add( potenciar(a.Raiz(),potencia));
        }

        lista.addAll(potenciarHijosUnicos( (ArbolB) a.HijoIzquierdo(), potencia));
        lista.addAll(potenciarHijosUnicos( (ArbolB) a.HijoDerecho(), potencia));

    }

    return lista;
}
```

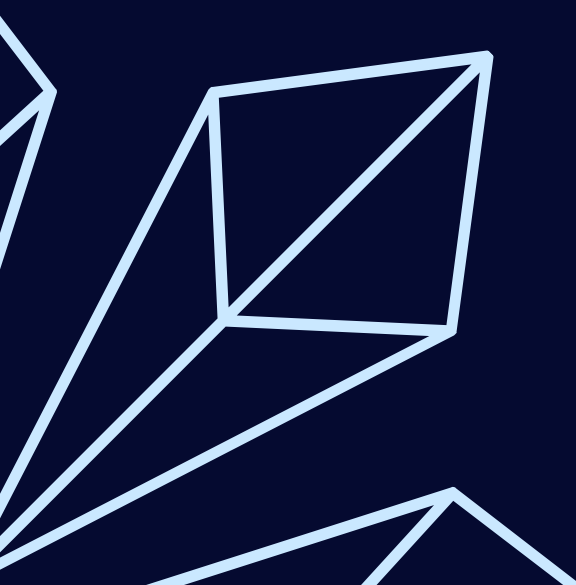

ESTRUCTURAS DE DATOS



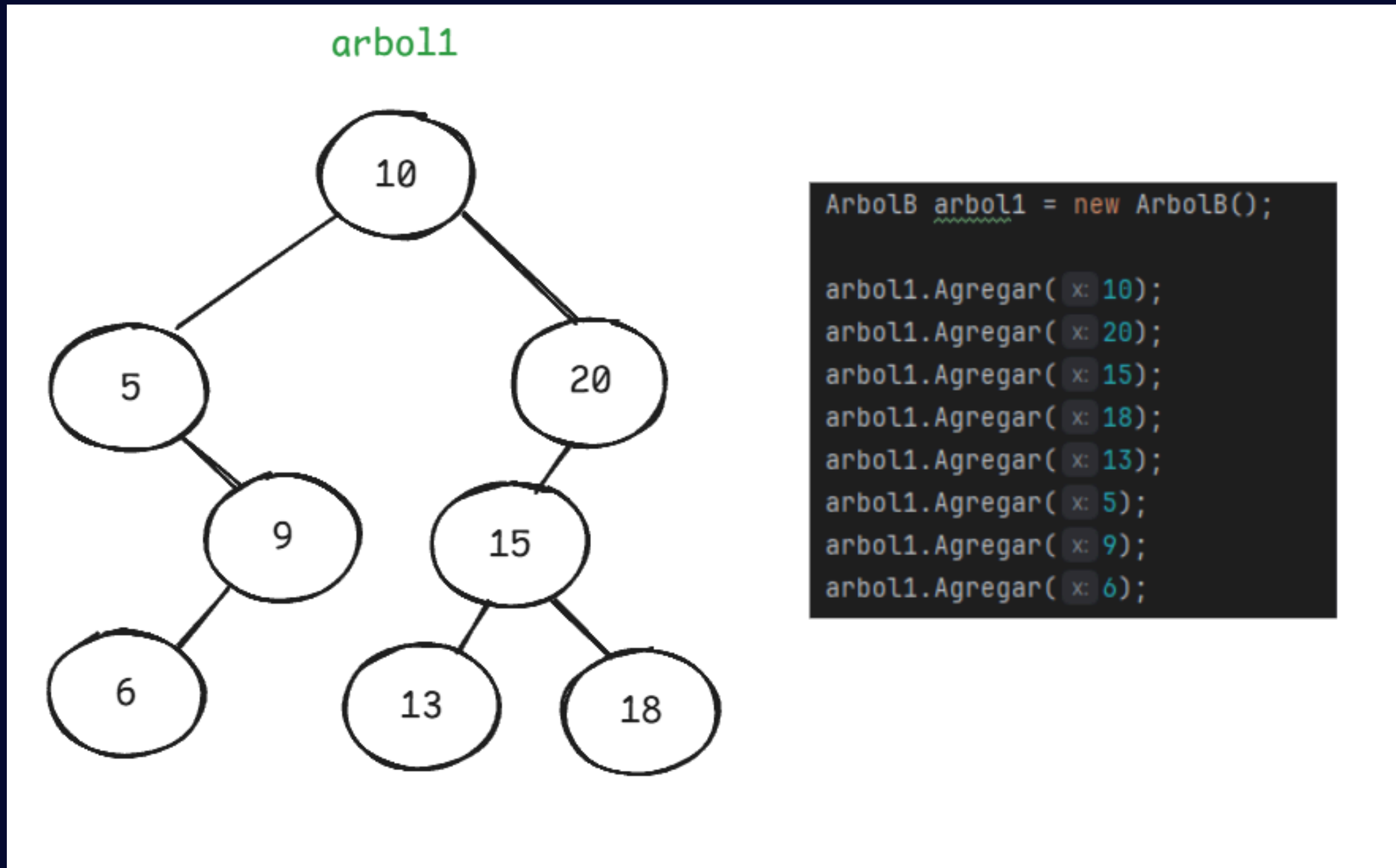
Como se mencionó anteriormente, utilizamos dos estructuras de datos:

- ArbolB
- NodoArbol

Estas clases cuentan con métodos básicos como getters, setters, agregar, eliminar, balancear, etc. Sin embargo, los métodos que resultaron fundamentales para desarrollar las tres funciones y cumplir con la consigna fueron:

- EsVacio(): Indica si el árbol está vacío. Método de la clase ArbolB.
 - HijoIzquierdo(): Retorna el hijo izquierdo de la raíz utilizando el método getNodeIzquierdo() de la clase NodoArbol. Método de ArbolB.
 - HijoDerecho(): Retorna el hijo derecho de la raíz utilizando el método getNodeDerecho() de la clase NodoArbol. Método de ArbolB.
- 

PASO A PASO PASO A PASO

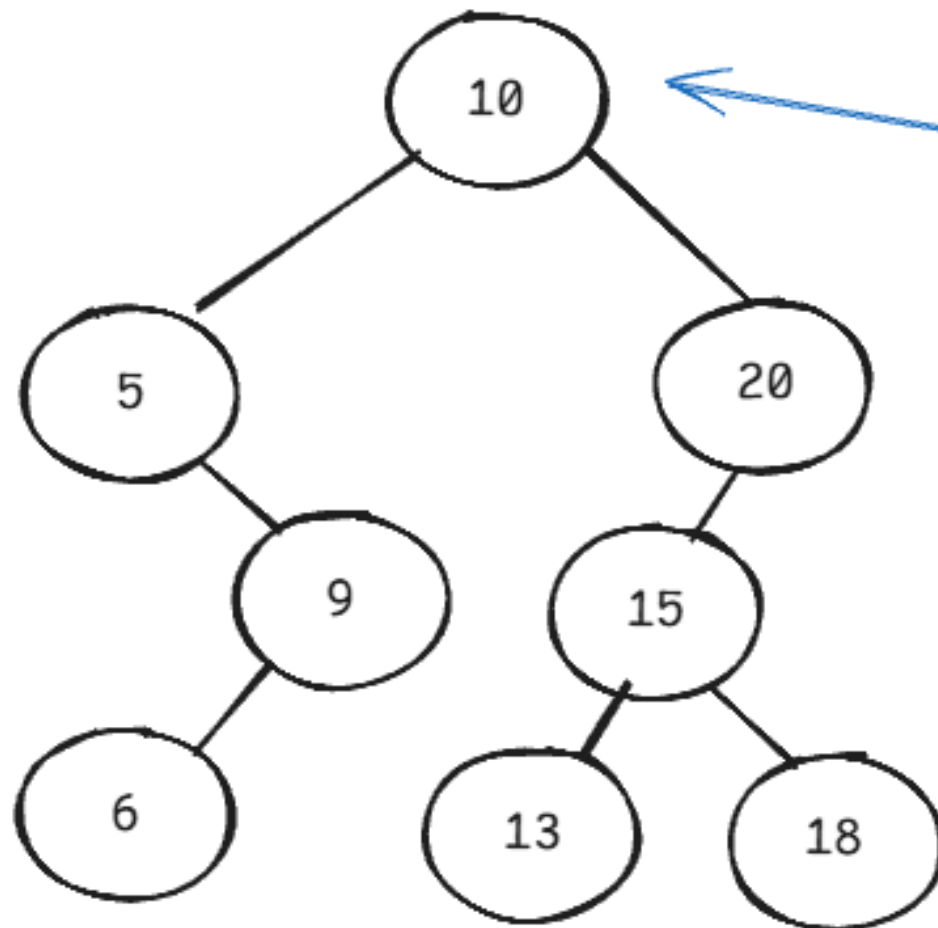


PASO A PASO PASO A PASO

listaPotenciada1[]

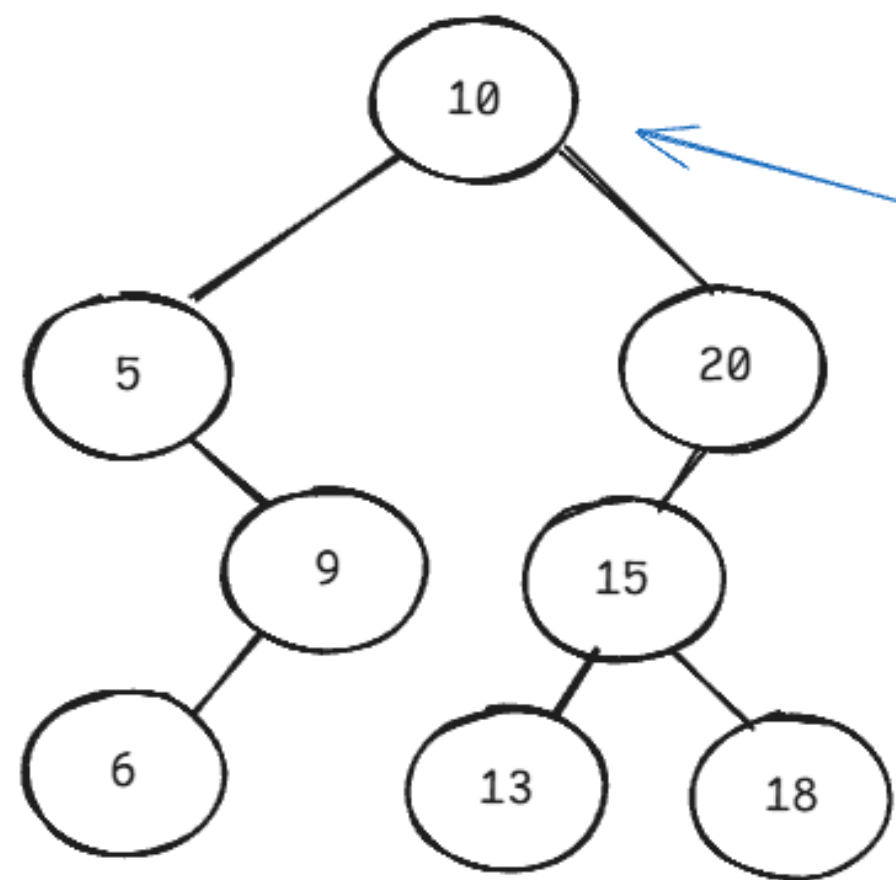
```
ArrayList<Integer> listaPotenciada1 = potenciarHijosUnicos(arbol1, potencia: 2);  
System.out.println(listaPotenciada1);
```

pregunta si la raiz == null



```
public static ArrayList<Integer> potenciarHijosUnicos(ArbolB a, int potencia) { //O(n)  
    ArrayList<Integer> lista = new ArrayList<>();  
  
    if(!a.EsVacio()){  
        if (tieneSoloUnHijo(a.getNodo())){  
            lista.add( potenciar(a.Raiz(),potencia));  
        }  
  
        lista.addAll(potenciarHijosUnicos( (ArbolB) a.HijoIzquierdo(), potencia));  
        lista.addAll(potenciarHijosUnicos( (ArbolB) a.HijoDerecho(), potencia));  
    }  
    return lista;  
}
```

PASO A PASO PASO A PASO



en esta instancia la condicion no se cumple

usa la funcion para ver si el nodo tiene un solo hijo

```
public static ArrayList<Integer> potenciarHijosUnicos(ArbolB a, int potencia) { //O(n)
    ArrayList<Integer> lista = new ArrayList<>();

    if(!a.EsVacio()){
        if (tieneSoloUnHijo(a.getNodo())){
            lista.add( potenciar(a.Raiz(),potencia));
        }

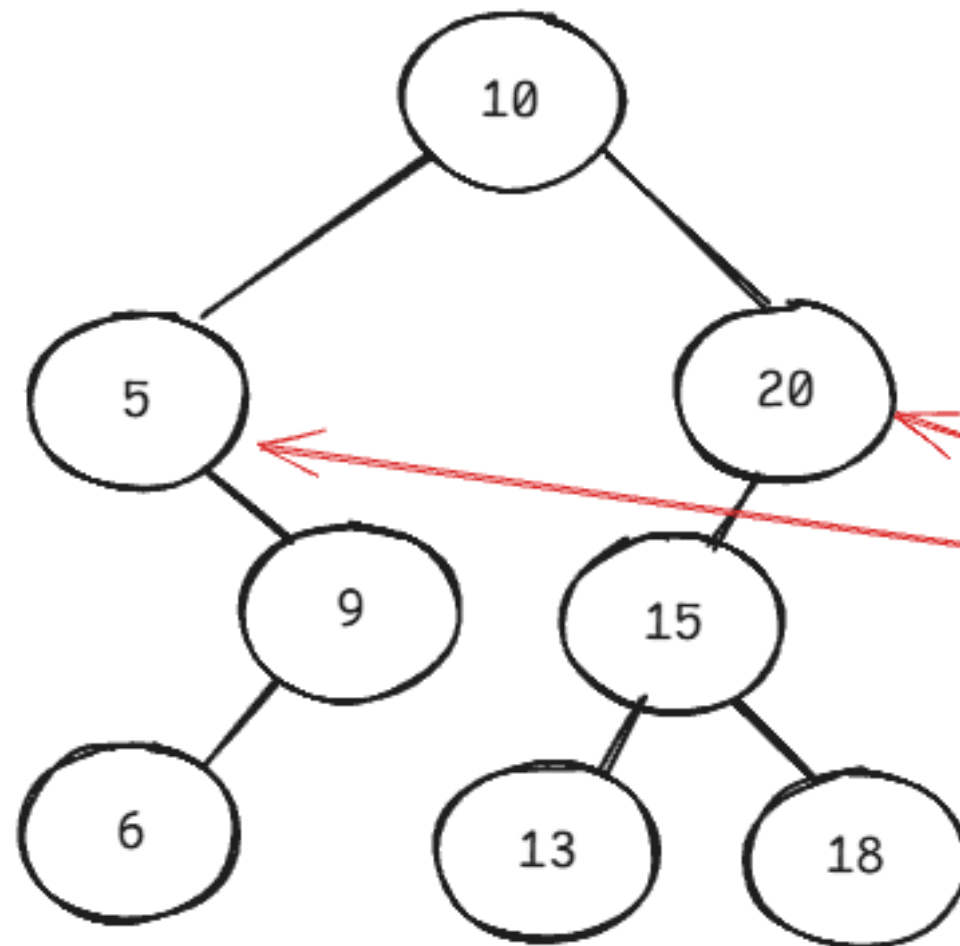
        lista.addAll(potenciarHijosUnicos( (ArbolB) a.HijoIzquierdo(), potencia));
        lista.addAll(potenciarHijosUnicos( (ArbolB) a.HijoDerecho(), potencia));
    }

    return lista;
}
```

```
public static boolean tieneSoloUnHijo(NodoArbol n) { // 0(1)
    return (n.getNodoIzq().EsVacio() && !n.getNodoDer().EsVacio()) ||
           (!n.getNodoIzq().EsVacio() && n.getNodoDer().EsVacio());
}
```

PASO A PASO PASO A PASO

aplica divide y conquista pasando a ejecutar la funcion a partir de los hijos de la raiz principal



```
public static ArrayList<Integer> potenciarHijosUnicos(ArbolB a, int potencia) { //O(n)
    ArrayList<Integer> lista = new ArrayList<>();

    if(!a.EsVacio()){

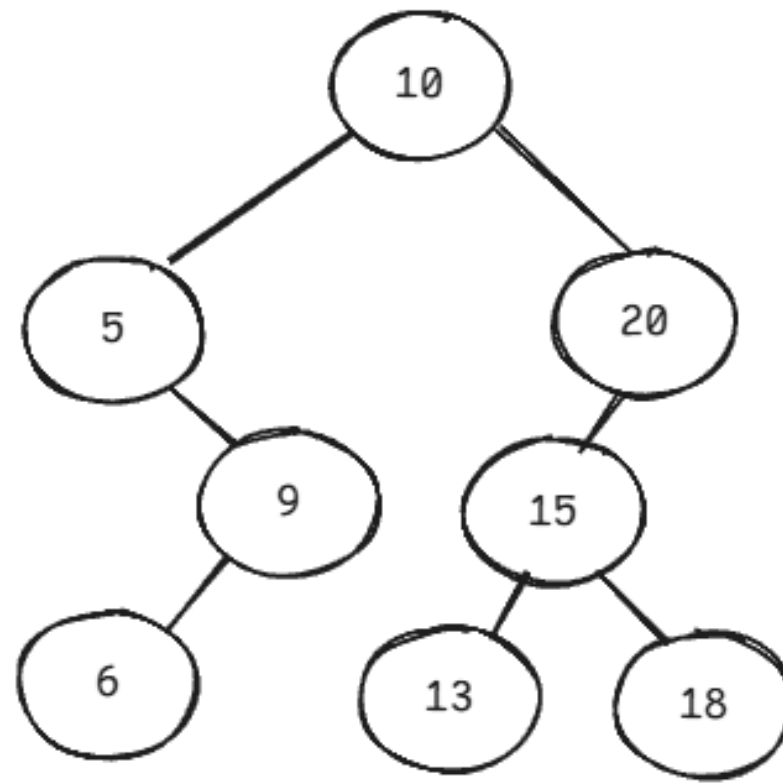
        if (tieneSoloUnHijo(a.getNodo())){
            lista.add( potenciar(a.Raiz(),potencia));
        }

        lista.addAll(potenciarHijosUnicos( (ArbolB) a.HijoIzquierdo(), potencia));
        lista.addAll(potenciarHijosUnicos( (ArbolB) a.HijoDerecho(), potencia));

    }
    return lista;
}
```


PASO A PASO PASO A PASO

repite el mismo procedimiento que hizo en la raiz principal



```
public static ArrayList<Integer> potenciarHijosUnicos(ArbolB a, int potencia) { //O(n)
    ArrayList<Integer> lista = new ArrayList<>();

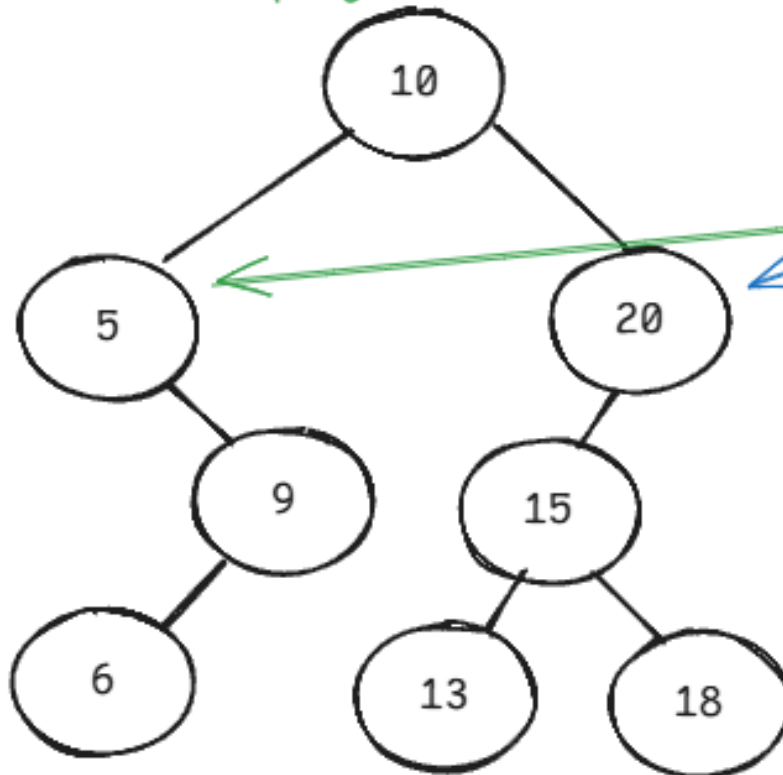
    if(!a.EsVacio()){

        if (tieneSoloUnHijo(a.getNodo())){
            lista.add( potenciar(a.Raiz(),potencia));
        }

        lista.addAll(potenciarHijosUnicos( (ArbolB) a.HijoIzquierdo(), potencia));
        lista.addAll(potenciarHijosUnicos( (ArbolB) a.HijoDerecho(), potencia));
    }

    return lista;
}
```

pregunta si la raiz == null
pregunta si la raiz == null



```
public static ArrayList<Integer> potenciarHijosUnicos(ArbolB a, int potencia) { //O(n)
    ArrayList<Integer> lista = new ArrayList<>();

    if(!a.EsVacio()){

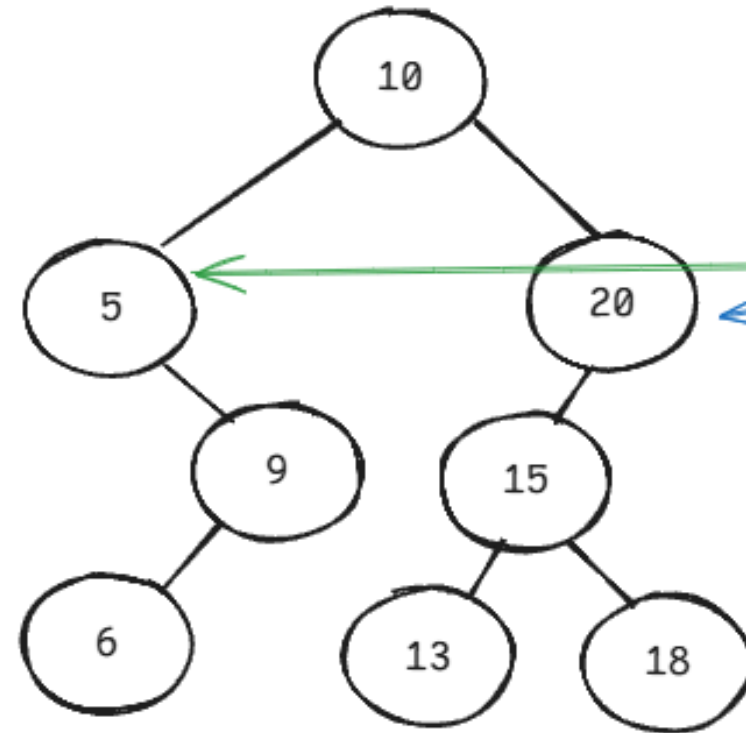
        if (tieneSoloUnHijo(a.getNodo())){
            lista.add( potenciar(a.Raiz(),potencia));
        }

        lista.addAll(potenciarHijosUnicos( (ArbolB) a.HijoIzquierdo(), potencia));
        lista.addAll(potenciarHijosUnicos( (ArbolB) a.HijoDerecho(), potencia));
    }

    return lista;
}
```

PASO A PASO PASO A PASO

usa la funcion para ver si el nodo tiene un solo hijo
usa la funcion para ver si el nodo tiene un solo hijo



EN AMBAS INSTANCIAS SE CUMPLE LA CONDICION

```
public static ArrayList<Integer> potenciarHijosUnicos(ArbolB a, int potencia) { //O(n)
    ArrayList<Integer> lista = new ArrayList<>();

    if(!a.EsVacio()){
        if (tieneSoloUnHijo(a.getNodo())){
            lista.add( potenciar(a.Raiz(),potencia));
        }

        lista.addAll(potenciarHijosUnicos( (ArbolB) a.HijoIzquierdo(), potencia));
        lista.addAll(potenciarHijosUnicos( (ArbolB) a.HijoDerecho(), potencia));
    }

    return lista;
}
```

```
public static boolean tieneSoloUnHijo(NodoArbol n) { // 0(1)
    return (n.getNodoIzq().EsVacio() && !n.getNodoDer().EsVacio()) ||
           (!n.getNodoIzq().EsVacio() && n.getNodoDer().EsVacio());
}
```

A CONTINUACION SE MUESTRA LO QUE PASA CON EL NODO (5) (hace lo mismo en el nodo (20))

obtiene el valor del nodo (5)

lo que obtiene lo eleva a la potencia pasada como parametro (2)

$5 \times 2 = 25$

agrega el 25 a la lista

listaPotenciada1[25]

```
public static Integer potenciar(int numero, int potencia) { // 0( log(potencia) )
    return (int) Math.pow(numero, potencia);
}
```

PASO A PASO PASO A PASO

SE SIGUE CON LA EJECUCION DEL CODIGO HASTA LLEGAR A LOS ULTIMOS NODOS
UNA VEZ LLEGADO AL FINAL SE RETORNA LA LISTA A LA QUE SE LE FUERON
AGREGANDO LOS VALORES POTENCIADOS
CON ESTE EJEMPLO DE ARBOL BINARIO LA LISTA TERMINA QUEDANDO ASI:

```
listaPotenciada1[ 25, 81, 400 ]
```




GRACIAS