ESTRUCTURA DE DATOS

DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN Y LA DECISIÓN

MONITOR: ALEJANDRO SALAZAR MEJÍA

alsalazarm@unal.edu.co

2023-2S

TALLER 10: ÁRBOLES AUTOBALANCEADOS Y MONTÍCULOS

En este taller, exploraremos la aplicación fundamental de los árboles binarios de búsqueda: los árboles autobalanceados, junto con otros conceptos como montículos y colas de prioridad. Continuaremos enfocándonos en la recursión como una herramienta poderosa para el diseño de algoritmos. Esta técnica es clave en la resolución de problemas algorítmicos, y este taller te brindará la oportunidad de perfeccionarla.

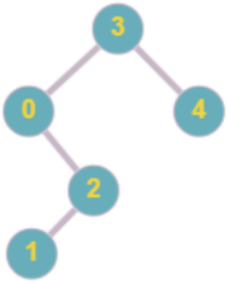
Los talleres son herramientas de estudio, y no tienen asignada una ponderación dentro de las evaluaciones del curso. Se recomienda desarrollar cada problema mediante la presentación de un seudocódigo, y en el caso de diseño de clase mediante diagramas de clase. Sin embargo, si el estudiante puede, solucionar cada problema empleando un lenguaje de programación de alto nivel.

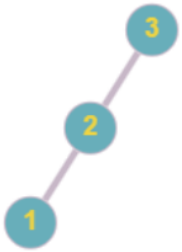
NOTA: Para resolver los ejercicios usando seudocódigo, asuma que ya tiene la implementación de la estructura de datos con los métodos y características que considere necesarias.

1. Dado un Arreglo de números **ordenado,** diseñe un algoritmo **recursivo** que cree un Árbol de Búsqueda Binario **balanceado** con los elementos del Arreglo. Su algoritmo debe correr en $O(n)$, donde *n* es el número de elementos en el arreglo. ***(hint**: el elemento medio del arreglo debería ser la raíz del árbol)*

2. Dado un Árbol Binario de Búsqueda y un intervalo `[low, high]`, pode el árbol de modo que todos sus elementos se encuentren en `[low, high]`. El podado del árbol no debe cambiar la estructura relativa de los elementos que seguirán en el árbol (es decir, cualquier descendiente de un nodo debe seguir siendo descendiente de dicho nodo). Por ejemplo,

`Input:` `[1, 3]` `Output:`





3. Dado un Árbol Binario de Búsqueda y un intervalo `[low, high]`, diseñe y analice un algoritmo que devuelva la suma de los valores de todos los nodos cuyo valor esté el rango `[low, high]`.

4.Resuelva las siguientes preguntas en orden:

- **a)** Dado un Árbol Binario de Búsqueda, cree un algoritmo que **balancee** este árbol usando las operaciones `Left-Rotation` y `Right-Rotation` vistas en clase. Un árbol binario de búsqueda está balanceado si para todos los nodos, la profundidad de sus dos subárboles nunca difiere en más de 1.
- **b)** Consulte sobre los **Árboles AVL,** el primer árbol de búsqueda binario auto-balanceable que se ideó. Consulte sus operaciones básicas y complejidad de estas.

5. Dado un Árbol Binario de Búsqueda, y un número natural *k*, diseñe y analice un algoritmo que devuelva el ***k*-ésimo elemento más pequeño** del árbol dado.

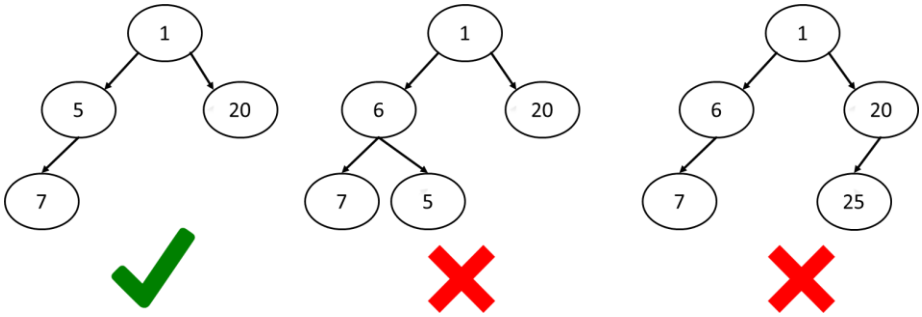
6. Retome el problema 5. de encontrar el ***k*-ésimo elemento más pequeño** de un árbol binario de búsqueda. Consulte cómo es posible ***aumentar*** un Árbol Red-Black para que esta operación se pueda realizar eficientemente. *(**hint**: aumentar quiere decir guardar información adicional en cada nodo sin que se afecte la eficiencia de las demás operaciones insertar, buscar y eliminar)*

7. Recuerde que todo Árboles Red-Black cumple cuatro requisitos: la raíz es negra, los nodos externos son negros, los hijos de un nodo rojo son negros, y todos los nodos externos tienen la misma profundidad negra (*Black Depth*). Con esto en mente, resuelva:

- Considere la siguiente afirmación: "*En un Árbol R-B, el camino más largo desde la raíz hasta una hoja no es más largo que dos veces el camino más corto desde la raíz a una hoja*".
  Piense, ¿por qué esta afirmación es verdadera?
- Consulte la prueba del siguiente **Teorema**: *"Un arboles RED-BLACK con n nodos internos tiene una altura menor o igual a $2\,ln(n+1)$".*
  *(**hint**: Utilice la afirmación del punto anterior. Esta prueba no es difícil de entender y es interesante)*

8. Un **número feo** es un número entero positivo cuyos factores primos son únicamente 2, 3 ó 5. Dado un número entero *n*, devuelva el *n*-ésimo número feo. *(**hint**: utilice una MIN-HEAP!)*

9. Cómo se enseñó en clase, los Montículos se pueden implementar eficientemente con Arreglos. Sin embargo, los Montículos también se pueden ver como **Árboles Binarios Semicompletos** (todos los niveles con completos excepto posiblemente el último, el cual se llena de izquierda a derecha):



Al insertar un elemento se deben garantizar dos cosas:

1. Que se mantenga la estructura del montículo (árbol binario semicompleto).
2. Que se mantenga el orden del montículo (todo padre menor o igual que sus hijos o viceversa).

Para lograrlo es necesario realizar un proceso en dos pasos:

Paso 1: Agregar el nuevo elemento en el último nivel y a la derecha del más a la derecha o, si el último nivel está completo, como hijo izquierdo del más a la izquierda de ese nivel.

Paso 2: En caso de que se incumpla la propiedad de orden se deben reorganizar los elementos, "subiendo" el nuevo elemento por medio de intercambios hasta que quede en el lugar que le corresponde.

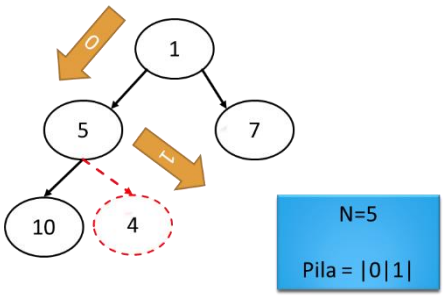
Viendo el Montículo como un Árbol Binario y no como un Arreglo, un algoritmo para el **Paso 1** no es evidente. Considere el siguiente algoritmo para el **Paso 1** y responda:

```
p = new nodo(e);  // e es el elemento a insertar
n++;              // n es el tamaño del montículo
r = new Stack();
if (n == 1) {
    top = p;              // top del montículo
} else {
    a = n;
    while (a>1) {                    (1)
        r.push(a%2);      //r es una pila
        a = a/2;          //división entera
    }
    q = top;
    for (i=0; i<r.size()-1; i++) {   (2)
        if (r.pop() == 0) {
            q = q.left;
        } else {
            q = q.right;
        }
    }
    if (r.pop()==0) {
        q.left = p;
    } else {
        q.right = p;
    }
}
```

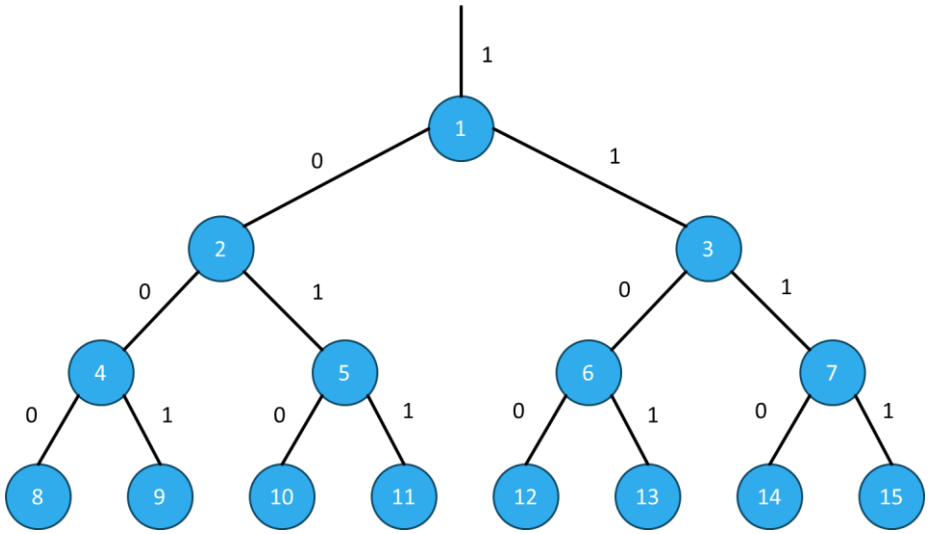
a) Dé una interpretación a lo que está haciendo el ciclo while (1).

b) Note que el clico (2) se encarga de irse desplazando en el montículo ¿por qué llega a la posición correcta?

c) ¿Cuál es la eficiencia de este algoritmo?



Utilice el siguiente diagrama para llegar a las respuestas:



10. Dado un string *s*, ordénelo de forma decreciente en función de la frecuencia de los caracteres. La frecuencia de un carácter es el número de veces que aparece en el string. Devuelve la cadena ordenada. Si hay varias respuestas, devuelve cualquiera de ellas.

Por ejemplo, si la entrada es `aaabbbbccdfffff` la salida debería ser `fffffbbbbaaaccd`, pues f aparece 5 veces, b 4, a 3, c 2 y d 1.

11. Consulte sobre la estructura de datos ***Splay Tree,*** un Árbol binario de búsqueda auto-balanceable, con la propiedad adicional de que a los elementos accedidos recientemente se accederá más rápidamente en accesos posteriores.