**DataStructures**

Pasos para visualizar de manera adecuada el repositorio:

* Cada clase de java, tiene su ejercicio y en los comentarios hay explicaciones guia, pero para mas información dirigirse al documento que contiene las diapositivas
* Para mejor entendimiento a la hora de entrar a cada ejercicio en su IDE de preferencia, ir a la par con las diapositivas, para poder ejecutar el codigo y verificar que hace de manera correcta lo propuesto en la solución.
* Los ejercicios del 1 al 4 se encuentra en la carpeta "Ejercicios 1-4/CodesLeetcode", y cada uno de estos esta realizado por 1 clase respectiva para cada uno.
* Los ejercicios 6 y 8, se encuentra en la carpeta "Ejercicios 6 y 8/Solution(6\_&\_8)" y estos tienen clases que generan casos prueba y los pone a prueba.
* Por ultimos el Ejercicio 5, 7 y 10, se encuentran por separado y se llaman "Problema 5" , "Problema 7" ... asi respectivamente.
* Las complejidades de los ejercicios, no fueron calculadas ya que se priorizo el aprendizaje de las estructuras de datos, algunas de estas utilizadas recursivamente, y otras de manera iterativas, Solo se asegura que las complejidades eran las aptas para poder ser aprobadas por los respectivos jueces de cada plataforma. (La complejidad mas baja fue: O(nlog(n)), y la cota superior mas alta fue: O(n^2) mas que todo por la lectura de datos y procesamiento de estos)

Estrategias de Solución de los ejercicios:

Ejercicio 1: (<https://leetcode.com/problems/complete-binary-tree-inserter/>)

* Nos piden Diseñar un algoritmo para insertar un nuevo nodo en un árbol binario completo manteniéndolo completo después de la inserción. Es bien sabido que Un árbol binario completo es un árbol binario en el que todos los niveles, excepto posiblemente el último, están completamente llenos y todos los nodos están lo más a la izquierda posible.

​ Para esto se crea una cola de TreeNodes, donde se almacenaran los nodos del árbol binario (Binary Tree), y también se crea una raíz (root) con la que trabajaremos el árbol.

​ El método CBTInserter(TreeNode root), podría considerarse el constructor de la clase, y en esta se inicializa la estructura de datos con la "root" del árbol binario. La estrategia de solución es aplicando un "Deque"(ArrayDeque) que este me permite insertar y eliminar elementos al comienzo y al final de un arreglo de números(en este caso de una cola), donde agrego valores a la cola que son del árbol binario y lo realizo de manera iterativa.

​ En el método insert(int val), tiene como objetivo insertar un TreeNode, en el árbol binario, aplicando la estructura que nos ofrece leetcode de un TreeNode que tiene un constructor para asignar valores(node.val = val)Y el manejo que se realiza en la cola, agregando el nodo que necesita para completar el árbol, y haciendo validaciones para poder remover de la cola los valores que no tengan ni izquierdo ni derecho, para poder ubicarlo en el nodo contrario en el que esta para poder completar el arbol, y quede balanceado.

​ Por ultimo se encuentra el método get\_root(), lo que hace eso solo retornar la raíz.

Ejercicio 2: (<https://leetcode.com/problems/binary-tree-inorder-traversal/>)

* El problema nos dice que dada la raíz de un árbol binario, devuelve el recorrido en orden de los valores de sus nodos. Para esto se recorrera el arbol por el recorrido de inOrder.​ Por ende, simplemente tenemos que visitar los nodos izquierdos, pasar el nodo raíz y terminar de recorrer el nodo derecho. Se hace de manera recursiva para mayor brevedad del código.

Ejercicio 3: (<https://leetcode.com/problems/validate-binary-search-tree/>)

* El problema nos dice que dada la raíz de un árbol binario, determine si es un árbol de búsqueda binario (BST) válido.

​ Se hace el recorrido InOrden, de tal manera que podemos hacer validaciones respecto a un BST, lo cual validamos si el número menor está a la izquierda de este o si es mayor está a la derecha del mismo.

Ejercicio 4: (<https://leetcode.com/problems/minimum-height-trees/>)

"Un árbol es un gráfico no dirigido en el que dos vértices cualesquiera están conectados exactamente por un camino. En otras palabras, cualquier gráfico conexo sin ciclos simples es un árbol."

* El problema nos dice que dado un árbol de n nodos etiquetados de 0 a n - 1, y una matriz de n - 1 aristas donde aristas[i] = [ai, bi] indica que hay una arista no dirigida entre los dos nodos ai y bi en el árbol , puede elegir cualquier nodo del árbol como raíz. Cuando selecciona un nodo x como raíz, el árbol resultante tiene una altura h. Entre todos los árboles enraizados posibles, aquellos con altura mínima (es decir, min (h)) se denominan árboles de altura mínima (MHT).

Devuelve una lista de las etiquetas raíz de todos los MHT. Puede devolver la respuesta en cualquier orden.

La altura de un árbol con raíces es el número de aristas en el camino descendente más largo entre la raíz y una hoja.

​ Para el ejercicio 4, se usan son grafos y se desarrolla por el recorrido por anchura (BFS) y se sabe que un arbol son grafos no dirigidos para esto se crea una lista la cual será la respuesta de lo que nos pide el ejercicio, para manipular esta lista debemos trabajar con un hashmap, ya que no podemos repetir valores en un grafo, y no podemos tener 2 números iguales.

​ Un "edge" son las conexiones o links de un nodo hacia otros. Debemos conocer que son los "degree" en un grafo, porque también se realiza un array de estos.​

​ Que son "degree" en grafos, hay 2 tipos de degrees: indegree y outdegree. Haremos lo siguiente: Suponiendo el (nodo X, este apunta a Y y Z) (nodo Y, apunta a Z) y (nodo Z, apunta a X y Y)indegree, de un vértice o nodo, son la cantidad de nodos que estan apuntando al nodo que se esta evaluando en el caso de X, el único nodo que apunta a X es Z, por ende tan solo X tiene 1 indegree outdegree, ahora sabiendo lo de indegree, en outdegree, son los nodos a los que apunta 1 nodo, en el caso de X, tiene 2 outdegree, porque apunta a Y y Z. Pero en el ejercicio que trabajamos no estan dirigidos, por ende no se aplica indegree y outdegree, solo se trabaja degree ya que no son dirigidos.

​ Sabiendo lo anterior se construye una lista de "edges" y "degrees", y se realiza una comparación según lo que el problema plantea, para agregarlo a una cola que se crea para poder verificar los siguientes casos, que son 2: Cada vez, se elimina los nodos hoja actuales, los nodos de la izquierda tienen 2 casos: (root = raiz) ​

1) solo root.

2) 2 nodos conectados por un "edge", ambos pueden ser root.

​ Ahora se tendrá en cuenta si la cola esta vacía, se hace algo similar al ordenamiento en grafos que es conocido como "topological sort“.Pero para este necesitamos las conexiones a las cuales llamaremos vecinos he iremos recorriendo, ya que en cada iteración trabajaremos con el primer elemento que entro a la cola, y lo iremos descolando (poll).

​ Finalmente ya se podrá obtener en la lista que creamos al principio el resultado, y esto será lo que se retorne en el método.

Para los siguientes ejercicios, la información se encuentra con mas detalle en las diapositivas que estan en el documento de powerpoint subido al repositorio.

Ejercicio 5: (<https://leetcode.com/problems/find-a-corresponding-node-of-a-binary-tree-in-a-clone-of-that-tree/>)

Ejercicio 6: (<https://leetcode.com/problems/find-if-path-exists-in-graph/>)

Ejercicio 7: (<https://leetcode.com/problems/relative-ranks/>)

Ejercicio 8: (<https://codeforces.com/problemset/problem/681/C>)

Ejercicio 9: (<https://atcoder.jp/contests/abc240/tasks/abc240_b?lang=en>)

Ejercicio 10: (<https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&page=show_problem&problem=1998>)