**TALLER ALGORITMOS DE ESTRUCTURAS DE DATOS AVANZADAS.**

**Daniel David Valencia Ovallos - 1151670**

**PRESENTADO A:**

**ING. MILTON JESÚS VERA CONTRERAS.**

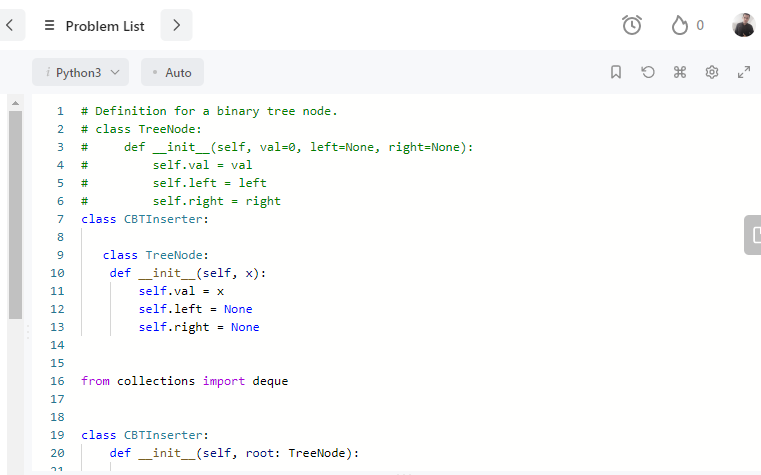
**UNIVERSIDAD FRANCISCO DE PAULA SANTANDER**

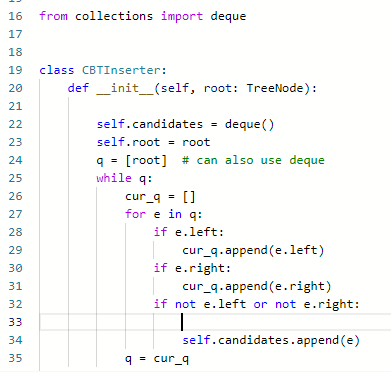
**INGENIERÍA DE SISTEMAS**

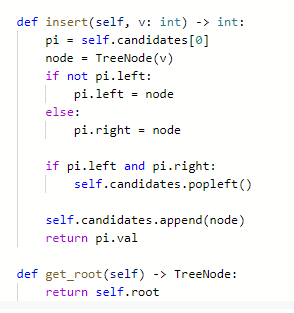
**SAN JOSÉ DE CÚCUTA**

**2023 – 1**

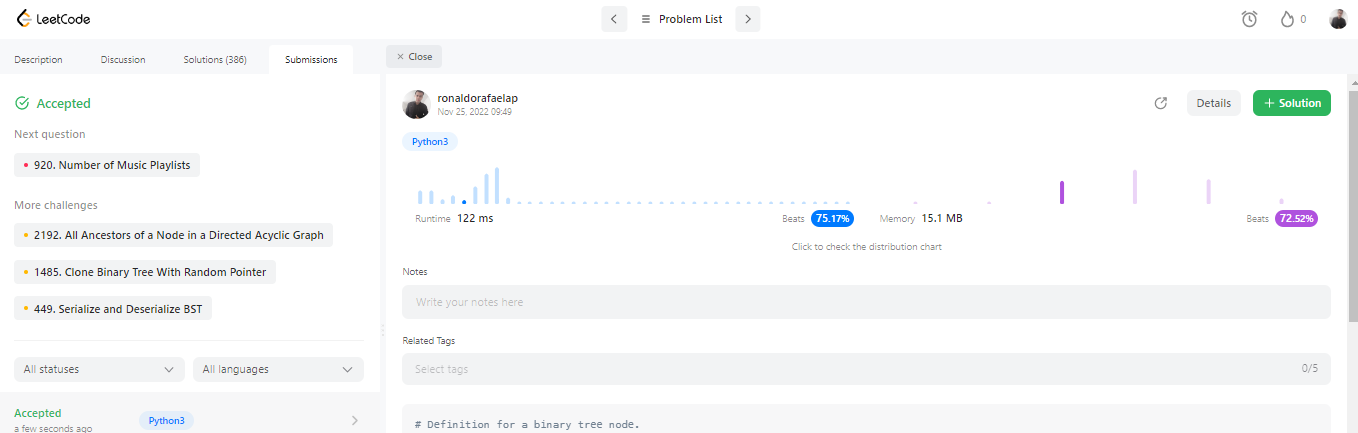
**Ejercicio 1.**









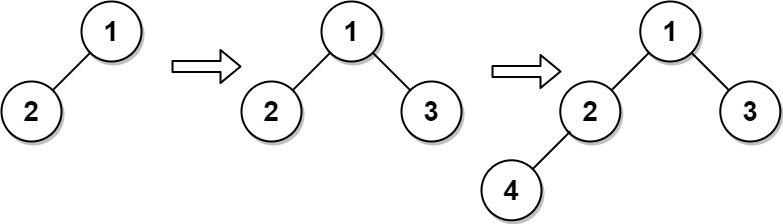


**Análisis de Complejidad**

**Complejidad de tiempo:** El preprocesamiento es O(N), donde N es el número de nodos en el árbol. Cada operación de inserción a partir de entonces es O(1)O(1).

**Complejidad espacial:** O(N) cuando el tamaño del árbol durante la operación de inserción actual.

**Solución Gráfica.**

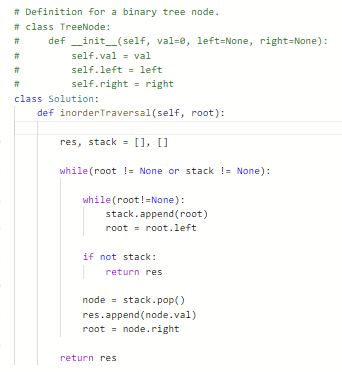
****

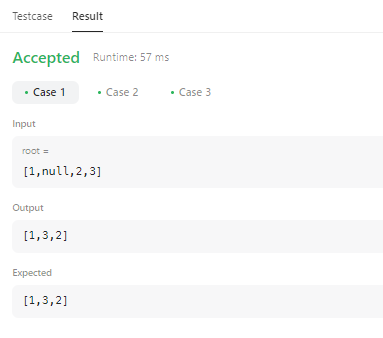
Considere todos los nodos numerados primero por nivel y luego de izquierda a derecha. Llame a esto el "orden numérico" de los nodos.

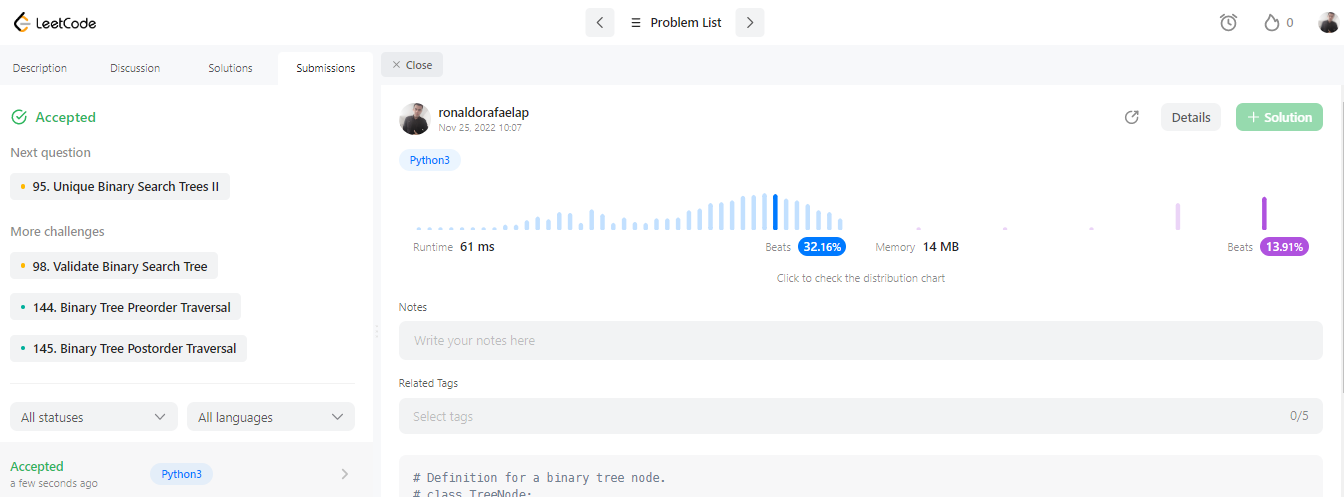
En cada paso de inserción, queremos insertar en el nodo con el número más bajo (que todavía tiene 0 o 1 hijos).

Al mantener una deque (cola de dos extremos) de estos nodos en orden numérico, podemos resolver el problema. Después de insertar un nodo, ese nodo ahora tiene el número más alto y no tiene hijos, por lo que va al final de la deque. Para obtener el nodo con el número más bajo, saltamos desde el principio del deque.

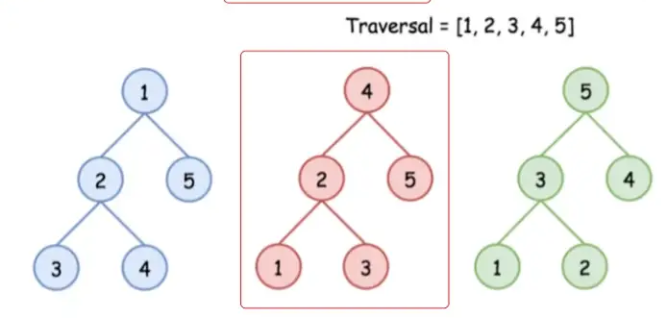
**Ejercicio 2.**







**Solución Gráfica.**

****

En el problema, se menciona claramente que tenemos que hacer el recorrido a través de la forma Inorder.

Primero entendamos qué es Inorder.

Atraviesa el subárbol izquierdo,

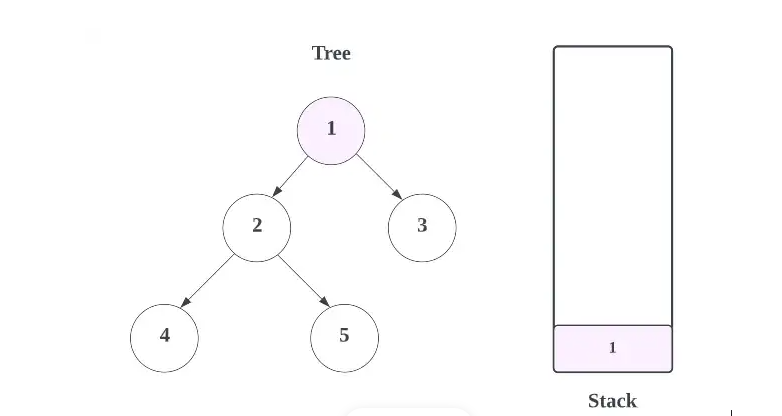
Visita la raíz.

Atraviesa el subárbol derecho,

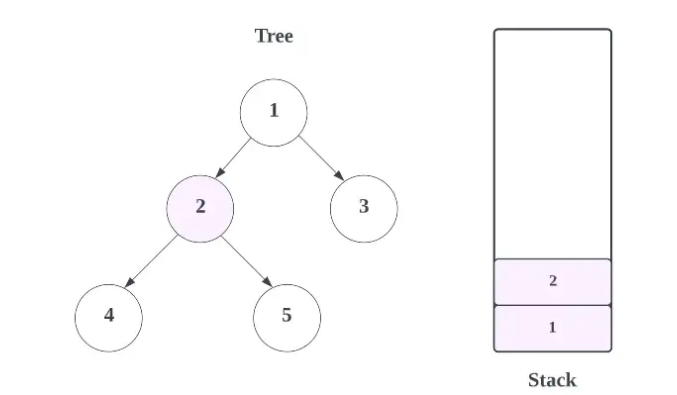
→ Por ejemplo, cuando la entrada es [1,2,3,4,5] (como el gráfico a continuación)

La salida debe ser [4,2,5,1,3]

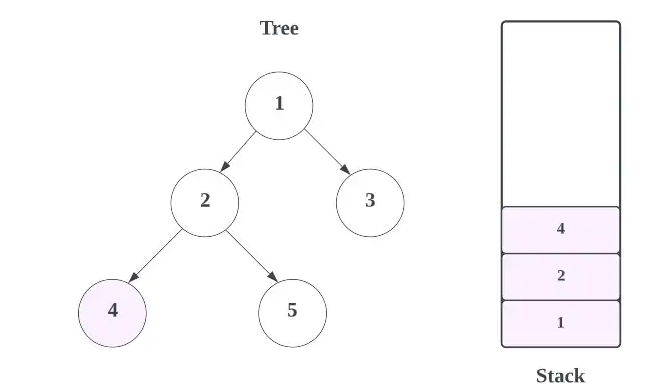
Vamos a entender cómo se calculará.

****

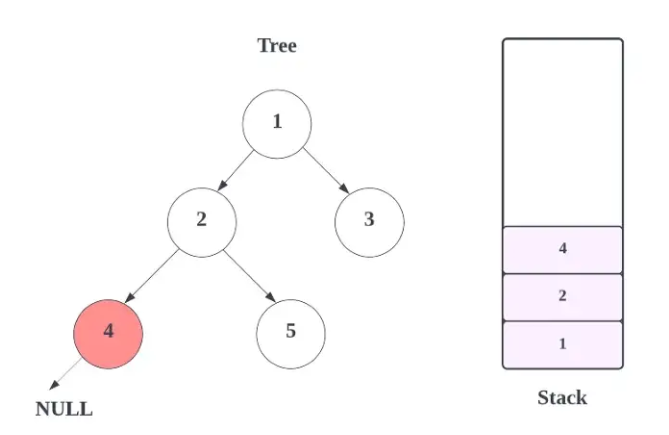
Primero, intentaremos atravesar el nodo izquierdo del árbol y agregar esos valores a la pila hasta que nos quedemos vacíos (NULL).

****

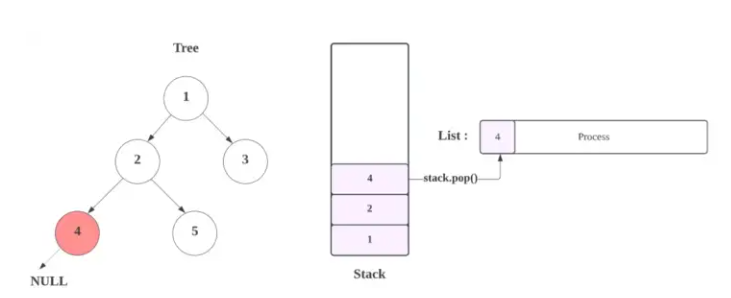
El último es.



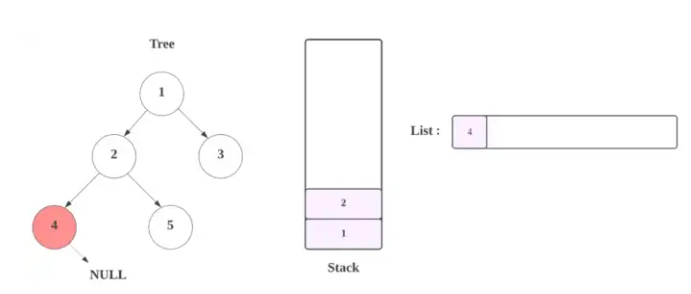
Ahora, aquí en 4, si vamos a su lado izquierdo, nos quedaremos vacíos (NULL) ya que no hay ningún nodo a la izquierda de 4.



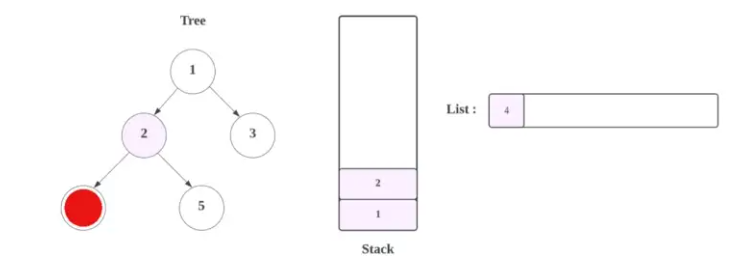
Ahora, como el nodo actual (4) no tiene ningún nodo izquierdo, pondremos el valor del nodo actual en la lista, eliminaremos los valores de la pila y agregaremos ese valor a la lista.



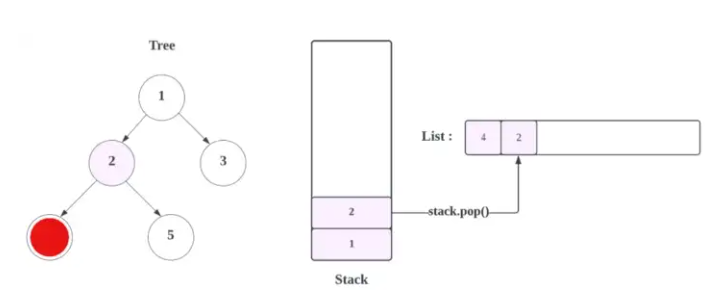
Ahora, verifique el lado derecho del nodo actual (que es 4), que también es nulo.



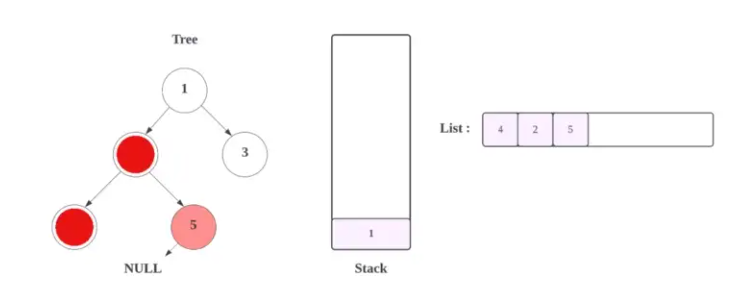
Ahora, extraemos el valor de la pila y lo agregamos a la lista.



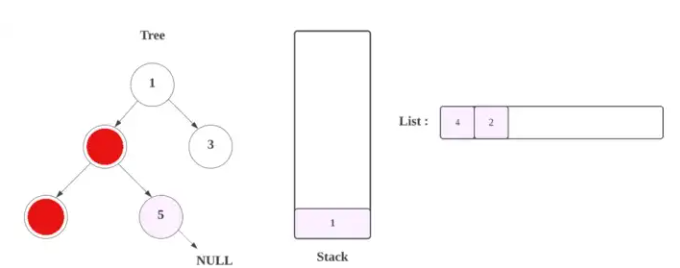
Entonces, el resultado sería.



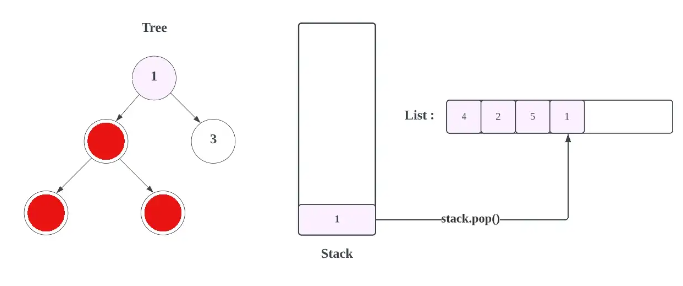
Ahora, verificaremos el lado derecho de 2, que es 5, colocaremos todos los nodos izquierdos del nodo actual (que es 5) en la pila, pero 5 no tiene nodos izquierdos. Ponemos el valor del nodo superior (es decir, 5) en la lista.



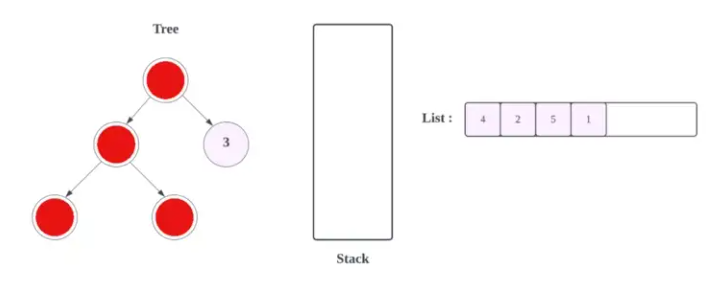
Compruebe el nodo derecho del nodo actual, que también es nulo.



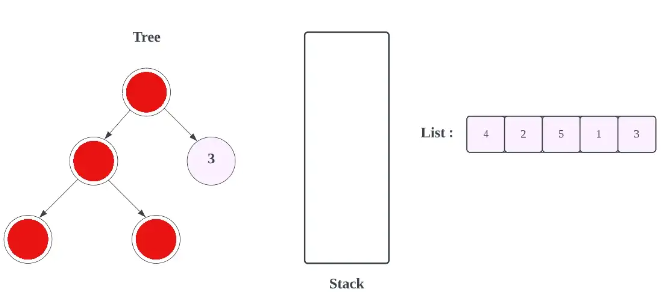
Ahora, extraemos el valor de la pila y lo agregamos a la lista.



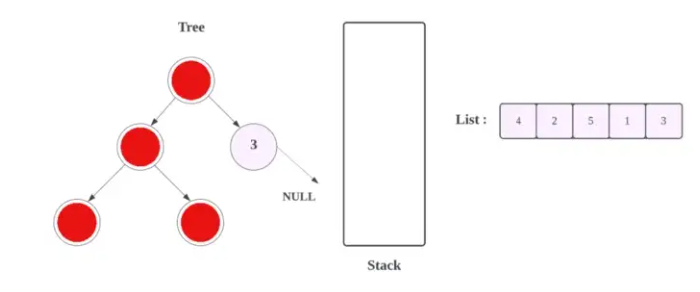
Ahora, verifique el lado derecho del nodo actual (1), que es 3.



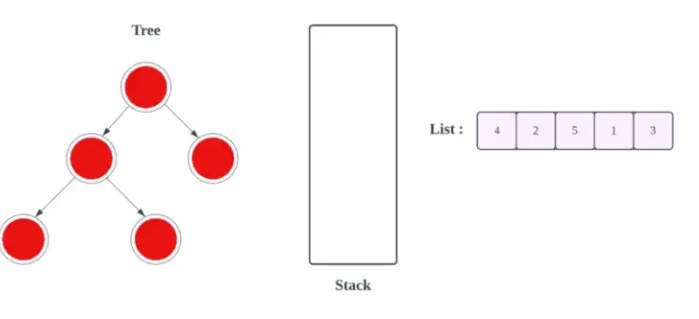
Ahora, coloque todos los nodos izquierdos del nodo actual (que es 3) en la pila, pero 3 no tiene nodos izquierdos.



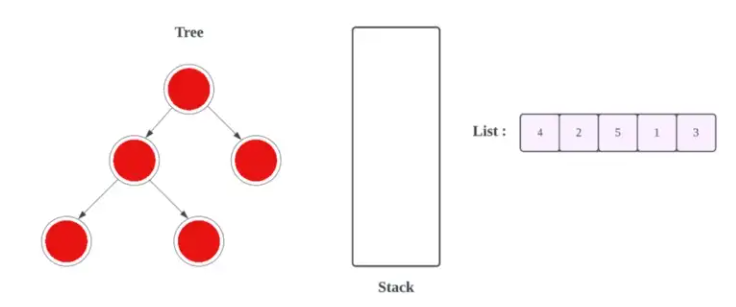
Ahora, el siguiente paso, agregaremos el valor del nodo actual a la lista.



Luego verifique todo el lado derecho del nodo actual, que también es nulo.



Después.



Aquí, tanto el nodo como la pila son nulos y están vacíos. No queda ningún nodo para agregar a la lista.

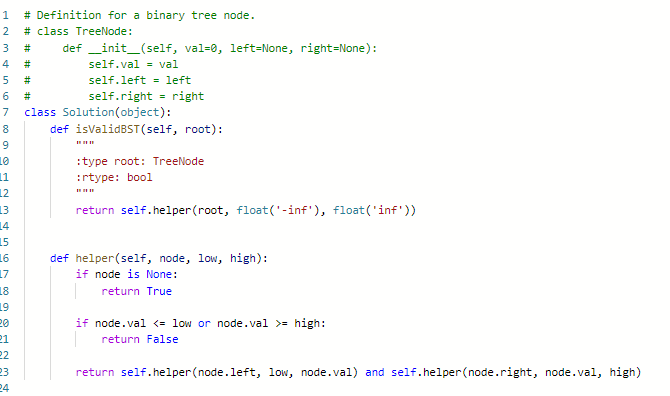
**Complejidad del tiempo.**

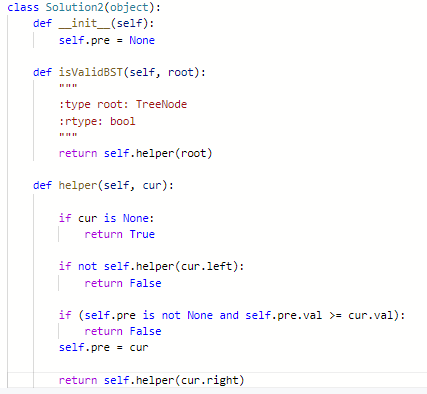
Aquí, estamos atravesando todos los nodos del árbol una vez, por lo que la complejidad del tiempo total es O(n).

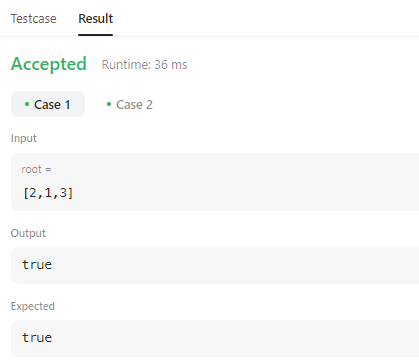
**Complejidad espacial.**

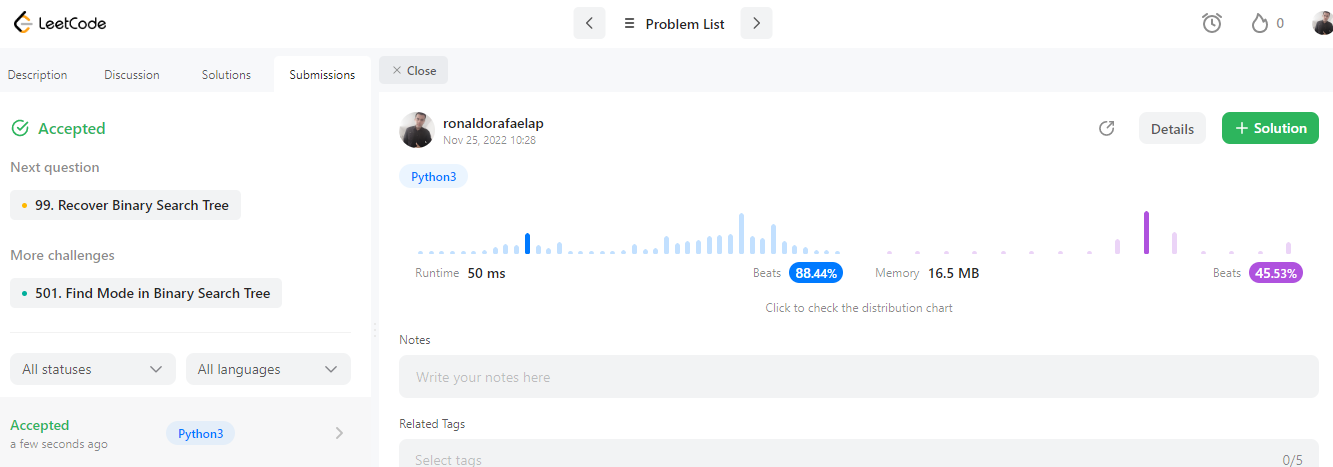
Dado que hemos usado dos matrices adicionales como res y stack, la complejidad del espacio será O(2n) => O(n).

**Ejercicio 3.**





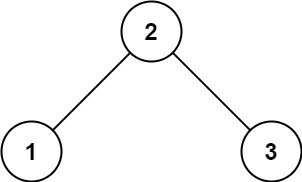


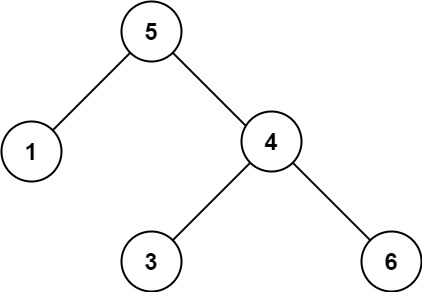


**Complejidad temporal:** *O(n)*

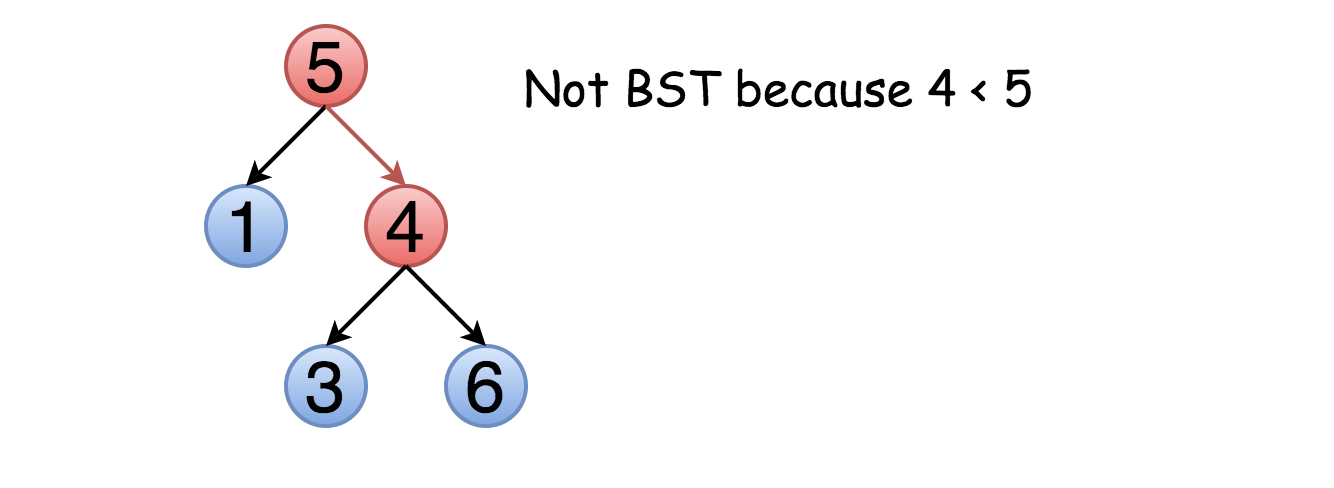
**Complejidad Espacial:** *O(n)*

**Solución Gráfica.**

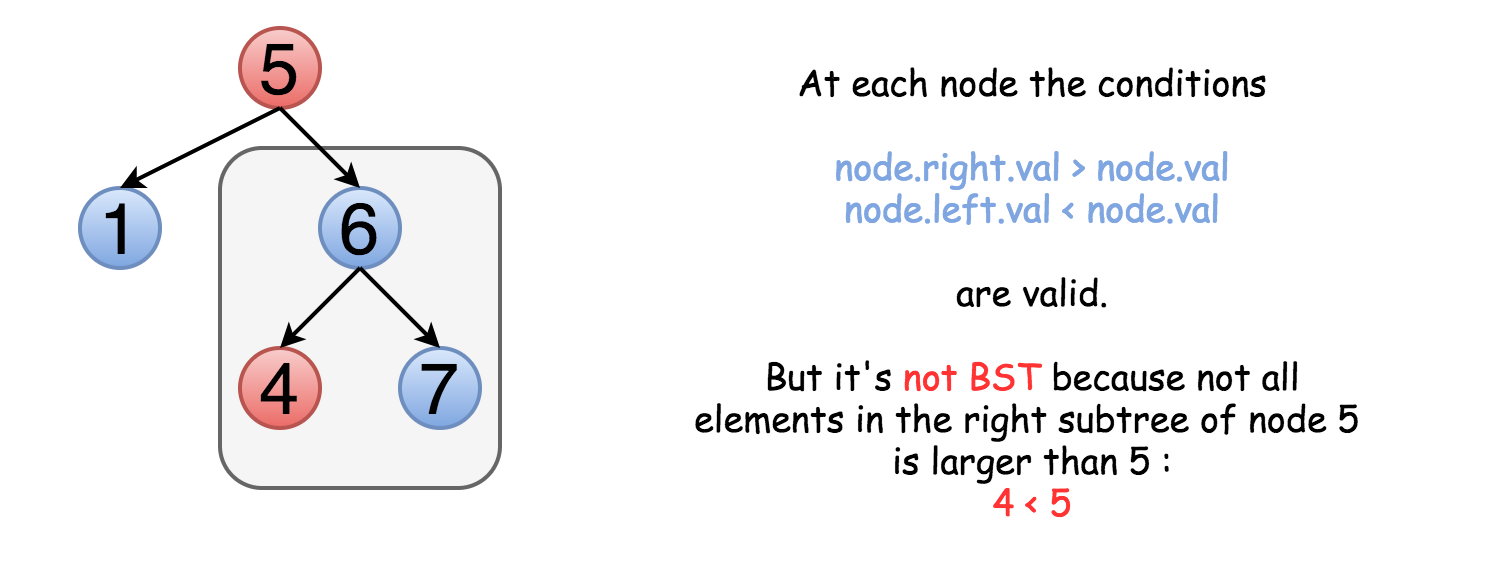
****

****

A primera vista, el problema es trivial. Recorrimos el árbol y verificamos en cada paso si node.right.val > node.val y node.left.val < node.val. Este enfoque incluso funciona para algunos árboles.

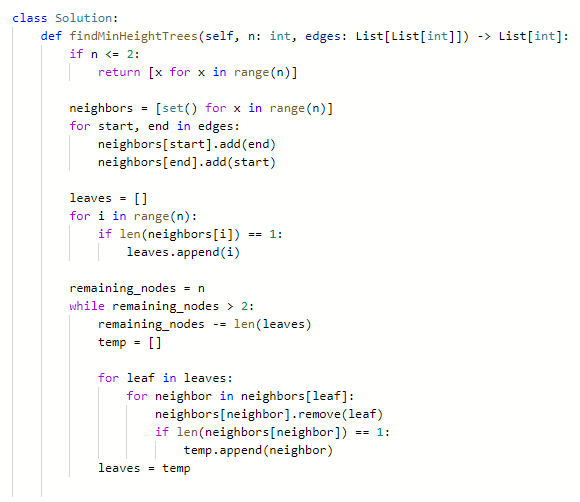
****

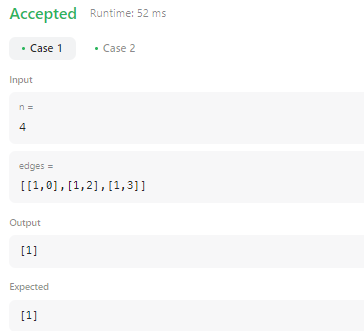
El problema es que este enfoque no funcionará para todos los casos. No solo el hijo derecho debe ser más grande que el nodo, sino todos los elementos en el subárbol derecho. Aquí hay un ejemplo :

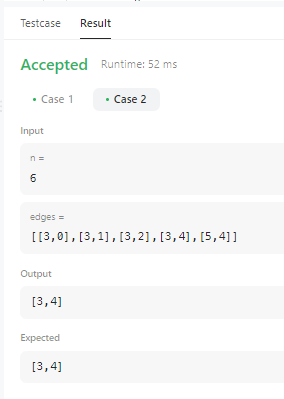


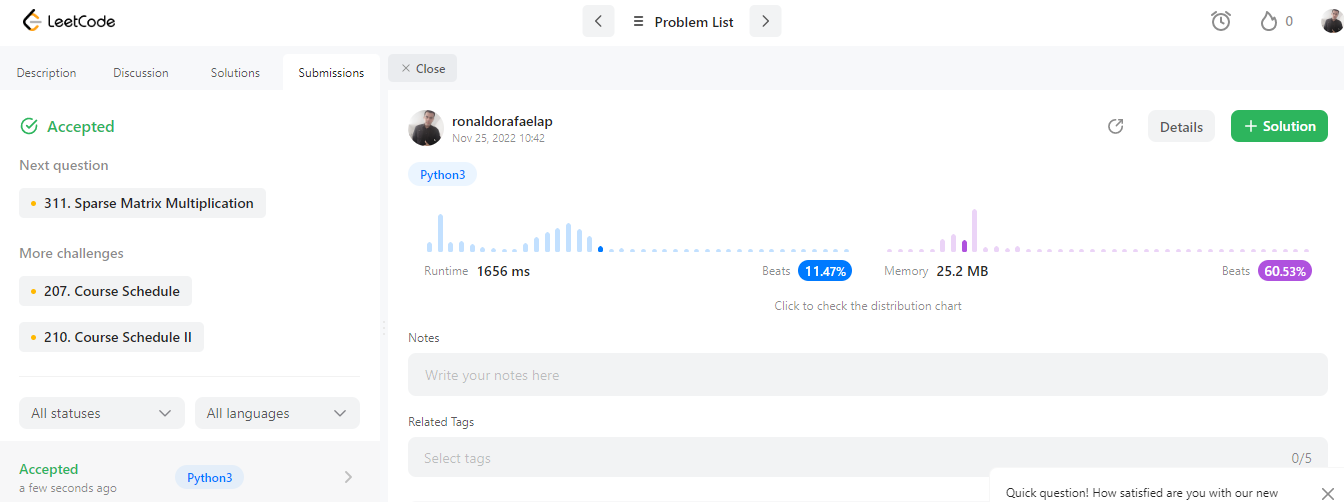
Eso significa que uno debe mantener los límites superior e inferior para cada nodo al atravesar el árbol, y comparar el valor del nodo no con los valores de los hijos sino con estos límites.

**Ejercicio 4.**





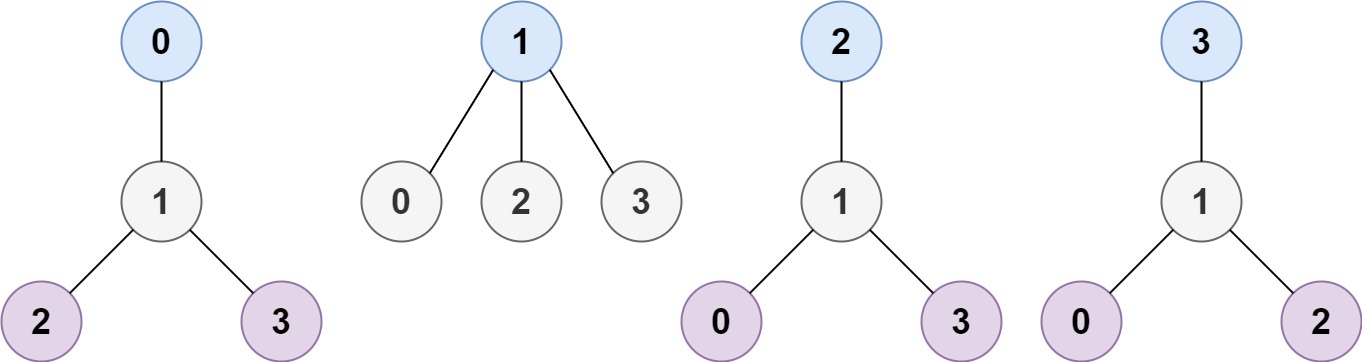


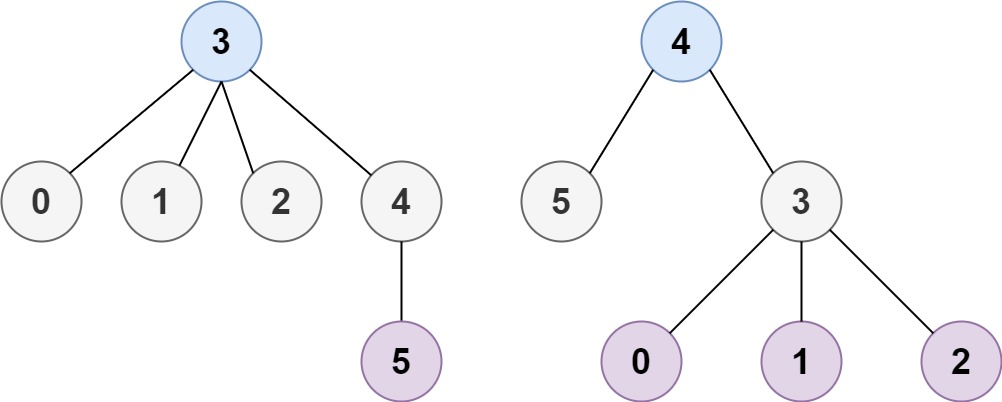


**Complejidad temporal:**

**Complejidad Espacial:**

**Solución Gráfica.**

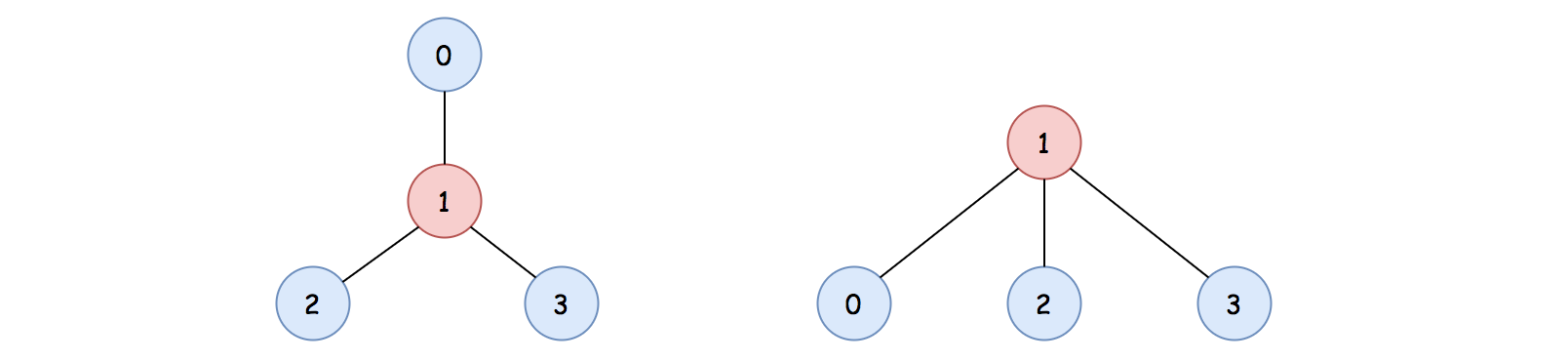




Tenga en cuenta que normalmente podría haber varias rutas para conectar los nodos en un gráfico. Sin embargo, en nuestro caso, dado que el gráfico de entrada puede formar un árbol desde cualquier nodo, como se especifica en el problema, sólo podría haber una ruta entre dos nodos cualesquiera. Además, no habría ningún ciclo en el gráfico. Como resultado, no habría ambigüedad en la definición anterior de distancia.

La altura de un árbol se puede definir como la distancia máxima entre la raíz y todos los nudos de sus hojas.

Con las definiciones anteriores, podemos reformular el problema cómo encontrar los nodos que en general están cerca de todos los demás nodos, especialmente los nodos hoja.



Por ejemplo, en el gráfico anterior, está claro que el nodo con el valor 1 es el centroide del gráfico. Si elegimos el nodo 1 como raíz para formar un árbol, obtendríamos un árbol con la altura mínima, en comparación con otros árboles que se forman con cualquier otro nodo.

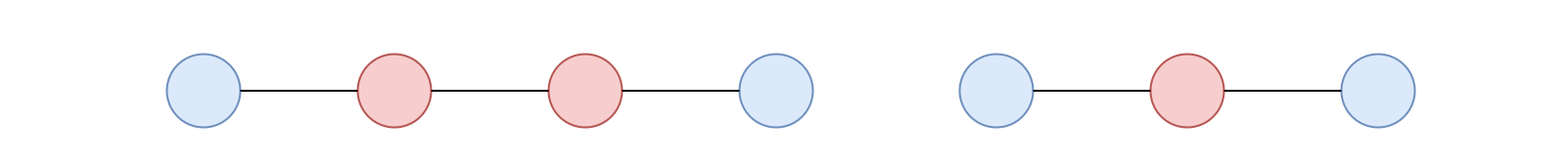
Antes de continuar, aquí hacemos una afirmación que es esencial para el algoritmo.

Para el gráfico similar a un árbol, el número de centroides no es más de 2.

Si los nodos forman una cadena, es intuitivo ver que se cumple la declaración anterior, que se puede dividir en los siguientes dos casos:

Si el número de nodos es par, entonces habría dos centroides.

Si el número de nodos es impar, entonces solo habría un centroide.



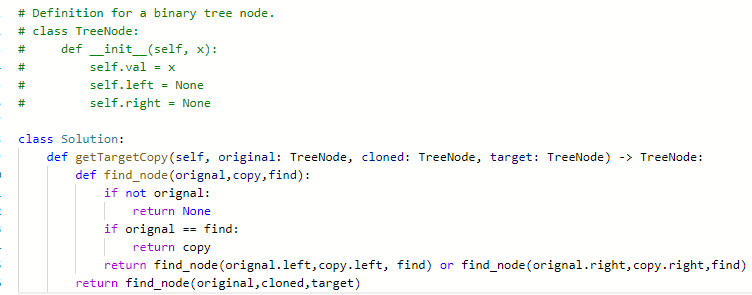
Para el resto de los casos, podríamos probar por contradicción. Supongamos que tenemos 3 centroides en el gráfico, si eliminamos todos los nodos que no son centroides en el gráfico, entonces los 3 nodos centroides deben formar un triángulo, de la siguiente manera:



Debido a que estos centroides son igualmente importantes entre sí, y también deberían estar igualmente cerca entre sí. Si falta alguno de los bordes del triángulo, los 3 centroides se reducirían a un solo centroide.

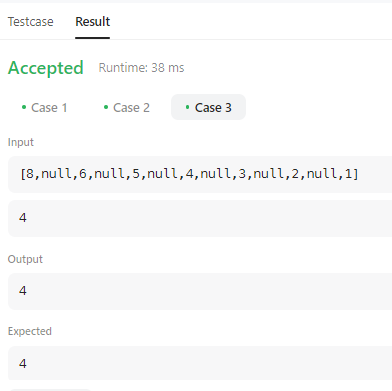
Sin embargo, la forma del triángulo forma un ciclo que se contradice con la condición de que no hay ciclo en nuestro gráfico de árbol. De manera similar, para cualquiera de los casos que tengan más de 2 centroides, estos deben formar un ciclo entre los centroides, lo cual se contradice con nuestra condición.

**Ejercicio 5.**







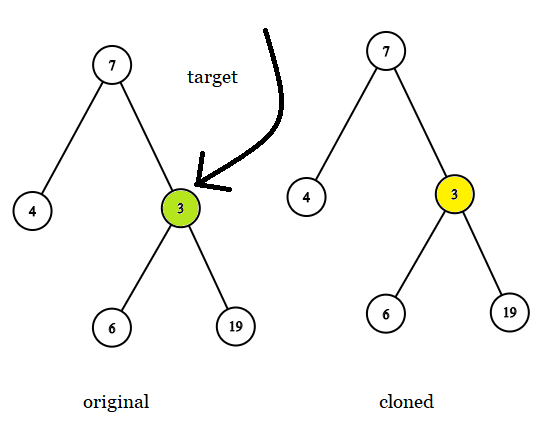


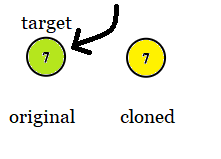
**Análisis de Complejidad**

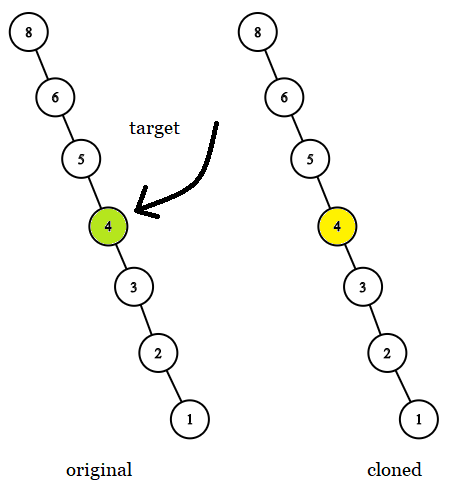
**Complejidad del tiempo:**O(N). Ya que uno tiene que visitar cada nodo, donde NN es un número de nodos.

**Complejidad espacial:** O(N). En el caso del árbol degenerativo (donde el árbol tiene la forma de una lista enlazada), todos los nodos estarán en la pila de tiempo de ejecución mientras se procesa el nodo más profundo. Si el árbol está equilibrado, la complejidad del espacio estará más cerca de O(logN), pero recuerda que para los propósitos del análisis de complejidad, consideramos principalmente el peor de los casos.

**Solución Gráfica.**

****

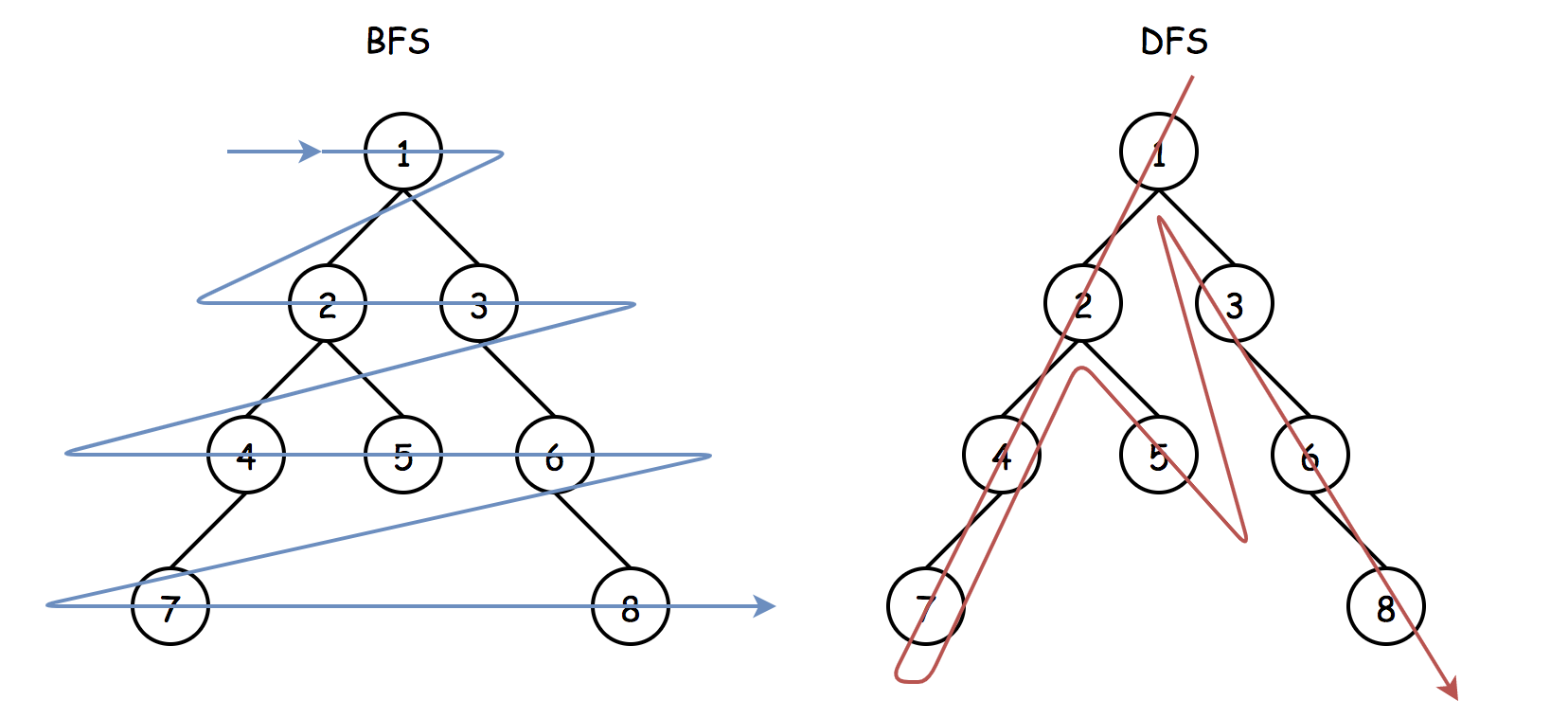
****

****

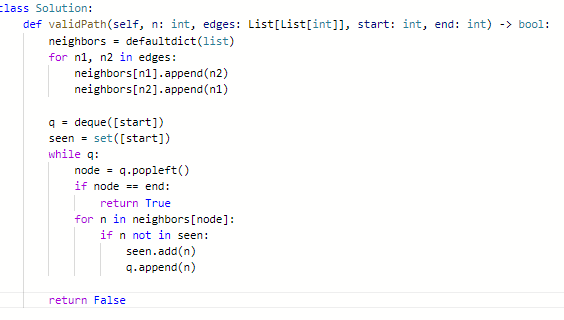
Recorramos ambos árboles en paralelo y, una vez que se identifique el nodo de destino en el primer árbol, devolvamos el nodo correspondiente del segundo árbol.

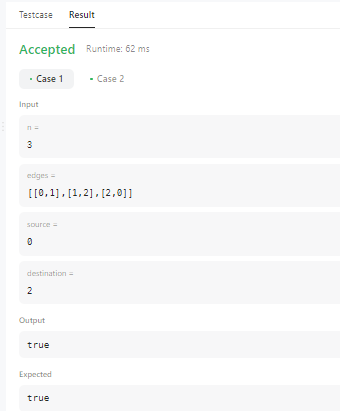
Hay dos formas de atravesar el árbol: búsqueda primero en profundidad DFS y búsqueda primero en amplitud BFS.

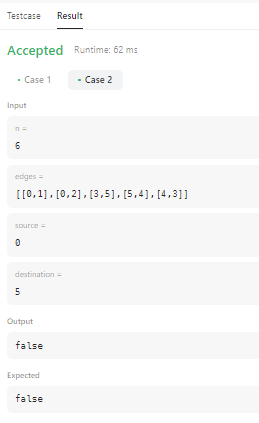
Ambos comienzan desde la raíz y bajan, ambos usan estructuras adicionales, ¿cuál es la diferencia? Así es como se ve a gran escala: BFS atraviesa nivel por nivel, y DFS primero va a las hojas.

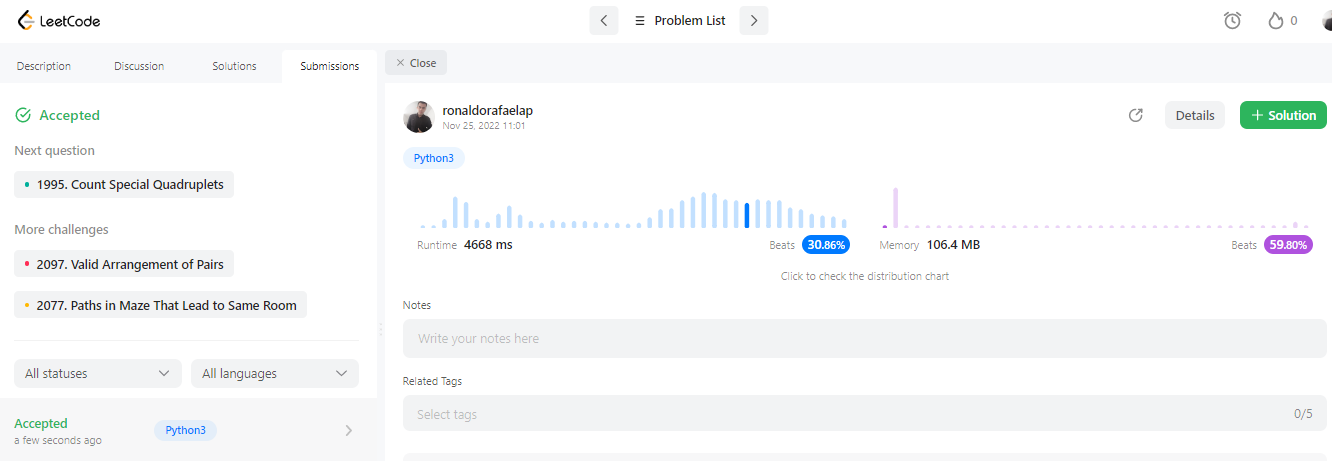


**Ejercicio 6.**





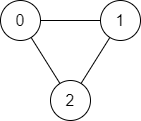


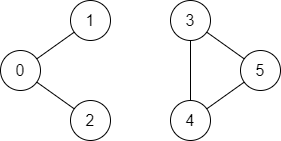


**Complejidad temporal:** O(n + m)O(n+m)

**Complejidad Espacial:**  O(n + m)O(n+m).

**Solución Gráfica.**

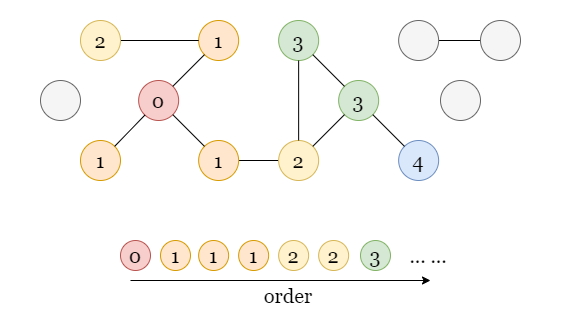
****

****

Si no está muy familiarizado con el cruce de BFS, le sugerimos que lea nuestra Tarjeta de exploración de Leetcode y tenga algunos conocimientos de antemano.

En BFS, exploramos los nodos en el orden de su profundidad. Suponiendo que el nodo inicial tiene una profundidad de 0, exploraremos todos los nodos en la profundidad actual (d) antes de pasar a todos los nodos en la siguiente profundidad (d + 1).

Aquí hay un ejemplo del orden en el que visitamos los nodos usando BFS, el nodo inicial está coloreado en rojo y tiene una profundidad de 0. Los números representan la profundidad de cada nodo. Independientemente de la estructura específica, siempre visitamos el nodo de profundidad = 0, luego todos los nodos de profundidad = 1, todos los nodos de profundidad = 2 y así sucesivamente.

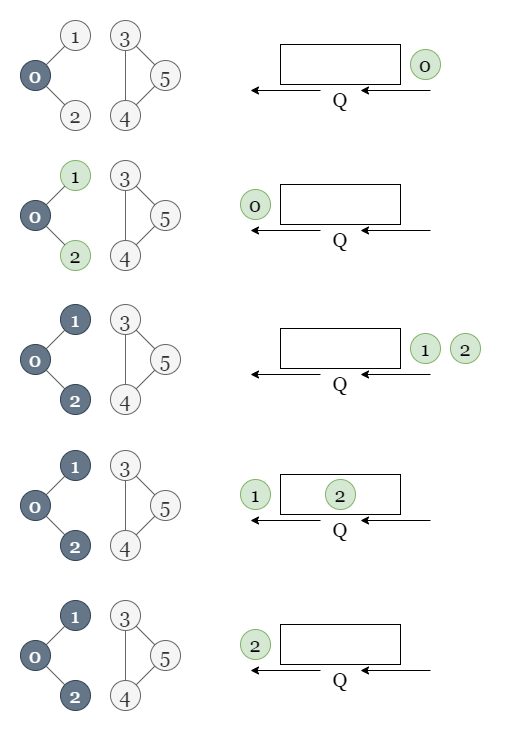


Volviendo a este problema, comenzamos con el origen del nodo con profundidad = 0, luego marcamos todos sus nodos vecinos no visitados con profundidad = 1 para ser visitados pronto, una vez que visitamos un nodo con profundidad = 1, marcamos todos sus nodos vecinos no visitados con profundidad = 2 también.

Podemos usar una cola cola como contenedor para almacenar todos los nodos a visitar sin mezclar el orden. Dado que la operación en la cola se realiza en el orden Primero en entrar, Primero en salir (FIFO), nos permite explorar todos los nodos con la profundidad actual, antes de pasar a los nodos con mayor profundidad.

Una vez que agregamos un nodo a la cola, lo marcamos inmediatamente como visitado para evitar que otros nodos lo agreguen nuevamente a la cola más adelante.

Si encontramos el destino del nodo durante el proceso, significa que existe una ruta desde el origen hasta el destino. De lo contrario, indica que no podemos encontrar ese camino.

****

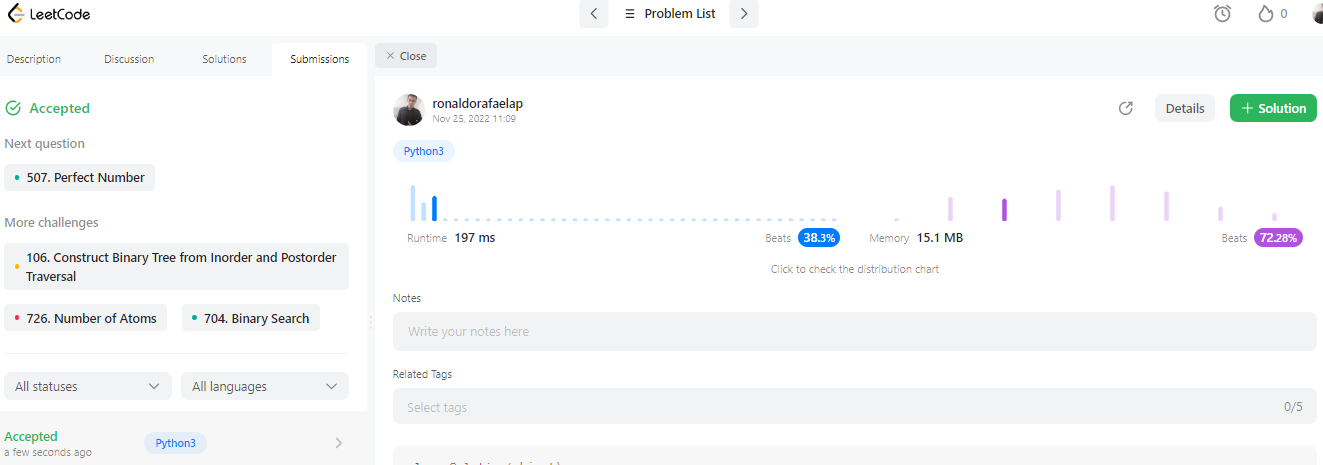
Como se muestra en la figura anterior, el nodo 0 se visita mientras que el nodo 5 no se visita. Por lo tanto, no hay un camino de 0 a 5.

**Ejercicio 7.**









**Link Repositorio:** <https://github.com/valencia321/Taller-Estructuras-de-Datos-Avanzadas>