Taller_2

May 26, 2023

Taller árboles - UNCode

Integrantes:

- Andrés Felipe Infante Hernández
- Manuel Camilo Rincon Blanco
- Joshua Cardona Toro
- Camilo Chitivo Cerinza

0.1 Viaje

Dificultad punto:

Todo el planteamiento y definición de árbol avl era bastante confuso.

Solución:

Mediante bastante ayuda de documentación, investigación e información en la web se plantearon las diferentes funciones para poder balancear el arbol y de esta forma quedará con la estructura de árbol avl.

Dificultad:

Buscar el ancestro en común del nodo de salida y el nodo de llegada.

Solución:

Mediante los padres se creo un array en dónde se iban guardando los ancestros para después hallar el común, con esto ya solo había que contar la cantidad de ancestros y no olvidar sumarle uno para tener en cuenta el nodo de llegada.

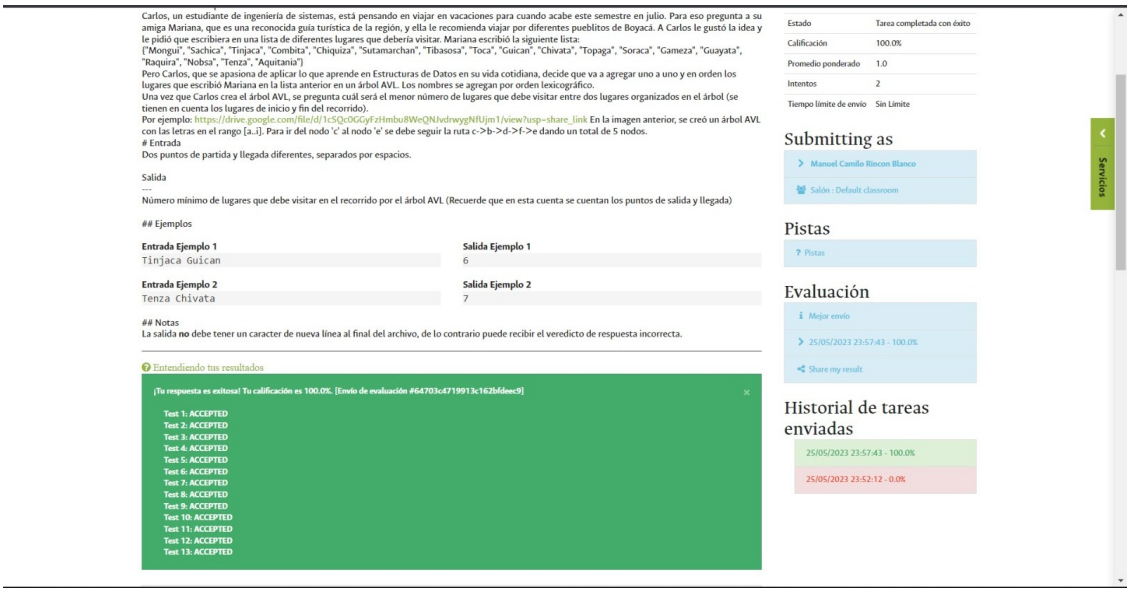
0.1.1 Códigos

```
class Nodo:
    def _init_(self, valor):
        self.valor = valor
        self.izquierdo = None
        self.derecho = None
        self.altura = 1


class ArbolAVL:
    def _init_(self):
```

```
        self.raiz = None

    def obtener_altura(self, nodo):
        if nodo is None:
            return 0
        return nodo.altura

    def obtener_factor_equilibrio(self, nodo):
        if nodo is None:
            return 0
        return self.obtener_altura(nodo.izquierdo) - self.obtener_altura(nodo.derecho)

    def actualizar_altura(self, nodo):
        altura_izquierdo = self.obtener_altura(nodo.izquierdo)
        altura_derecho = self.obtener_altura(nodo.derecho)
        nodo.altura = max(altura_izquierdo, altura_derecho) + 1

    def rotacion_izquierda(self, nodo):
        nuevo_raiz = nodo.derecho
        nodo.derecho = nuevo_raiz.izquierdo
        nuevo_raiz.izquierdo = nodo
        self.actualizar_altura(nodo)
        self.actualizar_altura(nuevo_raiz)
        return nuevo_raiz

    def rotacion_derecha(self, nodo):
        nuevo_raiz = nodo.izquierdo
        nodo.izquierdo = nuevo_raiz.derecho
        nuevo_raiz.derecho = nodo
        self.actualizar_altura(nodo)
        self.actualizar_altura(nuevo_raiz)
        return nuevo_raiz

    def insertar(self, valor):
        self.raiz = self._insertar(self.raiz, valor)

    def _insertar(self, nodo, valor):
        if nodo is None:
            return Nodo(valor)

        if valor < nodo.valor:
            nodo.izquierdo = self._insertar(nodo.izquierdo, valor)
        else:
            nodo.derecho = self._insertar(nodo.derecho, valor)

        self.actualizar_altura(nodo)

        factor_equilibrio = self.obtener_factor_equilibrio(nodo)
```

```
        # Caso de rotación izquierda-izquierda
        if factor_equilibrio > 1 and valor < nodo.izquierdo.valor:
            return self.rotacion_derecha(nodo)

        # Caso de rotación derecha-derecha
        if factor_equilibrio < -1 and valor > nodo.derecho.valor:
            return self.rotacion_izquierda(nodo)

        # Caso de rotación izquierda-derecha
        if factor_equilibrio > 1 and valor > nodo.izquierdo.valor:
            nodo.izquierdo = self.rotacion_izquierda(nodo.izquierdo)
            return self.rotacion_derecha(nodo)

        # Caso de rotación derecha-izquierda
        if factor_equilibrio < -1 and valor < nodo.derecho.valor:
            nodo.derecho = self.rotacion_derecha(nodo.derecho)
            return self.rotacion_izquierda(nodo)

        return nodo

    def buscar(self, valor):
        return self._buscar(self.raiz, valor)

    def _buscar(self, nodo, valor):
        if nodo is None or nodo.valor == valor:
            return nodo

        if valor < nodo.valor:
            return self._buscar(nodo.izquierdo, valor)
        else:
            return self._buscar(nodo.derecho, valor)

    def obtener_camino_minimo(self, origen, destino):
        nodo_origen = self.buscar(origen)
        nodo_destino = self.buscar(destino)

        if nodo_origen is None or nodo_destino is None:
            return 0

        ancestros_origen = self._obtener_ancestros(nodo_origen)
        ancestros_destino = self._obtener_ancestros(nodo_destino)

        # Encontrar el primer ancestro común
        min_lugares = float('inf')
        for i in range(len(ancestros_origen)):
            for j in range(len(ancestros_destino)):
                if ancestros_origen[i] == ancestros_destino[j]:
```

```
                    distancia = i + j
                    min_lugares = min(min_lugares, distancia)

        return min_lugares + 1  # Sumar 1 para contar los puntos de partida y llegada

    def _obtener_ancestros(self, nodo):
        ancestros = []
        while nodo is not None:
            ancestros.append(nodo)
            nodo = self._obtener_padre(nodo)
        return ancestros

    def _obtener_padre(self, nodo):
        return self._buscar_padre(self.raiz, nodo)

    def _buscar_padre(self, nodo_actual, nodo):
        if nodo_actual is None or nodo_actual == nodo:
            return None

        if (nodo_actual.izquierdo is not None and nodo_actual.izquierdo == nodo) or \
                (nodo_actual.derecho is not None and nodo_actual.derecho == nodo):
            return nodo_actual

        if nodo.valor < nodo_actual.valor:
            return self._buscar_padre(nodo_actual.izquierdo, nodo)
        else:
            return self._buscar_padre(nodo_actual.derecho, nodo)


# Crear el árbol AVL
arbol = ArbolAVL()

# Lista de lugares proporcionada por Mariana
lugares = ["Mongui", "Sachica", "Tinjaca", "Combita", "Chiquiza", "Sutamarchan", "Tibasosa", "
           "Chivata", "Topaga", "Soraca", "Gameza", "Guayata", "Raquira", "Nobsa", "Tenza", "A

# Insertar los lugares en el árbol AVL
for lugar in lugares:
    arbol.insertar(lugar)

# Leer los puntos de partida y llegada
punto_partida, punto_llegada = input().split()

# Obtener el mínimo número de lugares a visitar
min_lugares = arbol.obtener_camino_minimo(punto_partida, punto_llegada)

# Imprimir el resultado
print(min_lugares, end="")
```



0.2 Galápagos

Dificultad:

Hubo un poco de confusión sobre cómo manejar los nodos -1

Solución:

En vez de no hacer nodos cuando se recibe -1, se añaden al árbol pero se ignoran a la hora de sumar la cantidad de insectos

Dificultad:

Se intentó hacer una función recursiva para realizar la suma de insectos pero era muy difícil asegurarse de que se añadieran solo las necesarias

Solución:

Se añadió todos los insectos a una lista y se tomaron solo la cantidad dada por la segunda línea

Dificultad:

Se intentó disminuir el tiempo de ejecución comparando las tres maneras de recorrer el árbol al mismo tiempo, lo cual no funcionó

Solución: Se separó las funciones y se realizó cada recorrido por separado, comparando solo al final.

Dificultad:

fue un poco complejo el planteamiento del algoritmo para cada función de inorder, preorder y postorder.

Solución:

Mediante recursividad se uso un bucle usando la función dentro de la misma función con una condición de que está no se cumpliera si el nodo estaba vacío, lo que indicaba que se llegaba a una

hoja, para asegurarse de que quedará bien el orden de cada recorrido.

```
class Node:
    def _init_(self, data):
        self.data = data
        self.izquierda = None
        self.derecha = None
# Lectura de la entrada
num_insectos = list(map(int, input().split()))
num_nodes = int(input())

# Construcción del árbol binario completo
cola = []
raiz = Node(num_insectos[0])
cola.append(raiz)
i = 1

while cola:
    node = cola.pop(0)
    if i < len(num_insectos):
        node.izquierda = Node(num_insectos[i])
        cola.append(node.izquierda)
        i += 1

    if i < len(num_insectos ):
        node.derecha = Node(num_insectos[i])
        cola.append(node.derecha)
        i += 1

def sumaPreorder(raiz,lista):
    if raiz:
        if(raiz.data!=-1):
            lista.append(raiz.data)
        sumaPreorder(raiz.izquierda,lista)
        sumaPreorder(raiz.derecha,lista)
lista = []
sumaPreorder(raiz,lista)
sumatoria = 0
for i in range (num_nodes):
    sumatoria +=lista[i]
a = sumatoria
def sumaInorder(raiz,lista):
    if raiz:
        sumaInorder(raiz.izquierda,lista)
        if(raiz.data!=-1):
            lista.append(raiz.data),
        sumaInorder(raiz.derecha,lista)
lista = []
```

```
sumaInorder(raiz,lista)
sumatoria = 0
for i in range (num_nodes):
    sumatoria +=lista[i]
b = sumatoria

def sumaPostorder(raiz,lista):
    if raiz:
        sumaPostorder(raiz.izquierda,lista)
        sumaPostorder(raiz.derecha,lista)
        if(raiz.data!=-1):
            lista.append(raiz.data)
lista = []
sumaPostorder(raiz,lista)
sumatoria=0
for i in range (num_nodes):
    sumatoria +=lista[i]
c = sumatoria
if(max(a,b,c)==a):
    print("Preorder "+ str(a),end="")
if(max(a,b,c)==b):
    print("Inorder "+ str(b),end="")
if(max(a,b,c)==c):
    print("Postorder "+ str(c),end="")
```

Galapagos

Enunciado del problema
Un grupo de estudiantes de botánica están interesados en innovar. Para eso, deciden hacer un viaje a la isla Wolf (que es una de las Islas Galápagos que solo se puede visitar para quienes tienen fines de investigación o quieren hacer buceo profesional, no para turistas). Este viaje es naturalmente muy costoso y deben sacarle el mayor provecho. Van a ir a investigar sobre una nueva especie de insecto que vive en las copas de los árboles, pero como la isla tiene más de 1.5 kilómetros cuadrados, deben maximizar el proceso de toma de muestras. El proceso de la toma de muestras está organizado en n zonas distribuidas a través de las islas, donde cada zona tiene una cantidad esperada de insectos a recolectar i.
La idea entonces es recibir la lista de las n cantidades de insectos esperados por zona y agregarlos en level order (https://www.geeksforgeeks.org/create-a-tree-in-level-order/). En la contrucción de este árbol encontrará valores en el rango [-1, 25]. El valor -1 es utilizado para representar NULL(es decir, que ese nodo no existe y por ahí no puede hacerse la expedición). Los demás valores diferentes de -1 son valores que sí hacen referencia a la cantidad esperada de insectos.
La expedicción comienza en el cráter del volcán Wolf, que es la primera zona en la lista anterior y representa la raíz del árbol binario creado anteriormente. Como no se tienen los recursos para pasar por cada uno de los nidos de insectos solo se puede pasar por a nodos. Entonces uno de los estudiantes propone 3 maneras de realizar la expedición: Recorrido inorder, preorder y postorder. El estudiante propone visitar los primeros a nodos en cada recorrido, pero no cuenta con un computador en la Isla para realizar los cálculos.
Su misión entonces es ayudar al estudiante a encontrar el mejor recorrido que puede hacer (Inorder, preorder o postorder) y la cantidad esperada de insectos que podrá recolectar en dicho recorrido.
Tenga en cuenta además que si hay dos recorridos que tienen el mismo número de insectos esperados, se tendrá en cuenta el siguiente criterio de preferencia: Preorder - Inorder - Postorder.
Por ejemplo: https://drive.google.com/file/d/18TxsYbe6RYUnZjSADTQU-DKKwiau_mrL/view?usp=share_link
Restricciones
- El árbol binario de entrada es un árbol binario completo con niveles entre 3 y 12.
- Los valores de cada nodo están entre 0 y 25 con la restricción de que -1 representa NULL.

Entrada
En la primera línea va la lista ordenada de posibles insectos por zona (el primer elemento es la raíz del árbol). Hay suficientes elementos por cada entrada para tener un árbol binario completo en su último nivel. En la segunda línea va el número de nodos a que se deben visitar. |
Salida
Inicialmente va la salida dice el mejor recorrido a realizar, con la primera letra en mayúscula ("Preorder", "Inorder", "Postorder") Luego, separado por un espacio, va la cantidad máxima de insectos que se pueden recolectar en la expedición

Ejemplos

Entrada Ejemplo 1
```
4 20 10 -1 16 4 0
4
```

Salida Ejemplo 1
```
Preorder 50
```

Notas
La salida **no** debe tener un caracter de nueva línea al final del archivo, de lo contrario puede recibir el veredicto de respuesta incorrecta.

Entendiendo tus resultados

Tu respuesta contiene algunos errores. Tu calificación actual es 93.33%. [Envío de evaluación #64713f7d19913c162bfdf3e6]

Test 1: ACCEPTED
Test 2: ACCEPTED
Test 3: ACCEPTED
Test 4: ACCEPTED
Test 5: ACCEPTED
Test 6: WRONG_ANSWER
Test 7: ACCEPTED
Test 8: ACCEPTED
Test 9: ACCEPTED
Test 10: ACCEPTED
Test 11: ACCEPTED
Test 12: ACCEPTED
Test 13: ACCEPTED
Test 14: ACCEPTED
Test 15: ACCEPTED

Información

Fecha de entrega	Sin fecha de envío
Estado	La tarea ha fallado
Calificación	93.33%
Promedio ponderado	1.0
Intentos	2
Tiempo límite de envío	Sin Límite

Submitting as

Cardona Toro Joshua

Salón : Default classroom

Pistas

Pistas

Evaluación

Mejor envío

26/05/2023 18:23:41 - 93.33%

Share my result

Historial de tareas enviadas

26/05/2023 18:23:41 - 93.33%

26/05/2023 17:54:29 - 6.67%