Escuela Colombiana de Ingeniería Julio Garavito

Algoritmos y Estructuras de Datos 2024-1

Informe reconstrucción de árboles por recorridos

Andersson David Sánchez Méndez

David Eduardo Salamanca Aguilar

Cristian Santiago Pedraza Rodríguez

22 de abril de 2024

DOCUMENTO TÉCNICO

Requisitos

Especificación

Dados los métodos de preorder e inOrden de un árbol binario, se requiere construir el postOrder del árbol teniendo en cuenta esos dos métodos. Así, lo que se requiere es primero construir el árbol y después si implementar el postOrder. También tienen que cumplirse todas las reglas AVL. Debe considerarse los siguientes aspectos: el factor de equilibrio, las rotaciones en caso de inserción o eliminación, y el balanceo.

Entrada

La única línea de la entrada contiene dos palabras, el recorrido en preorden e inorden respectivamente de un árbol utilizado por Valentina (sólo de referencia con base al ejercicio de la arena).

Salida

Imprima una palabra con el recorrido en postOrden del árbol.

Diseño

Estrategia

1. **Construcción del árbol a partir de los recorridos en preorden e inorden:**

   Los recorridos en preorden e inorden contienen información sobre los elementos del árbol y su posición relativa. El recorrido en preorden proporciona el orden en el que se visitan los nodos (raíz-izquierda-derecha), mientras que el recorrido inorden proporciona la secuencia en la que se visitan los nodos (izquierda-raíz-derecha).

   Estos recorridos fueron necesarios para reconstruir el árbol original. El primer elemento del recorrido en preorden es la raíz del árbol. Se busca este elemento en el recorrido inorden para determinar los elementos en el subárbol izquierdo y derecho.

   Se repite este proceso recursivamente para cada subárbol hasta que todos los nodos sean reconstruidos.

2. **Recorrido en postorden del árbol reconstruido:**

   Después de construir el árbol, se realiza un recorrido en postorden para obtener el recorrido en postorden del árbol.

   El recorrido en postorden visita primero el subárbol izquierdo, luego el subárbol derecho y finalmente la raíz. Este orden asegura que los nodos hoja se visiten antes que sus padres.

La recursión es útil para realizar este recorrido. Primero se visita el subárbol izquierdo en postorden, luego el subárbol derecho en postorden y finalmente se agrega el valor de la raíz al resultado.

3. **Almacenamiento del recorrido en postorden**:

Durante el recorrido en postorden, se almacenan los valores de los nodos en una lista o cadena en el orden correcto.

4. **Retorno del recorrido en postorden**:

Una vez completado el recorrido en postorden, se concatenan los valores almacenados en el orden correcto para obtener el recorrido en postorden como una cadena o lista.

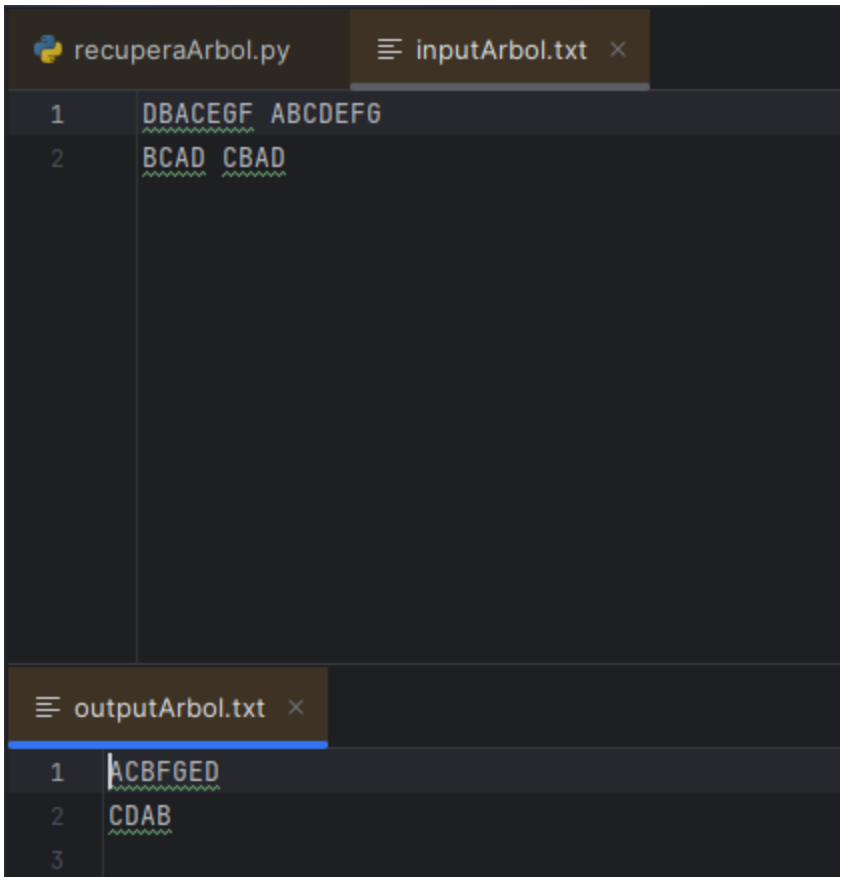
También se hizo una investigación detallada que nos ayudó a la solución de este problema. Hubo 2 temas y estos son:

- **Sucesor óptimo:** este método nos ayudó a reducir el tiempo de ejecución de este algoritmo, es decir, nos evitó una complejidad de alto nivel para este problema pequeño.
- **Bypass:** se involucra en el caso del método delete en el árbol binario para garantizar que un sistema de seguridad informático se evite mediante un error o una característica del sistema, programa,etc.

Casos de prueba

Entrada	Salida
DBACEGF ABCDEFG	ACBFGED
BCAD CBAD	CDAB

CASOS DE LA ARENA DE EJEMPLO



Análisis

Temporal

La complejidad de reconstruir un árbol binario a partir de sus recorridos en preorden e inorden es O(n), donde n es el número de nodos en el árbol. La reconstrucción hace que el árbol se divida recursivamente por recorridos preorden e inorden. Cada nodo se visita exactamente una vez durante ese proceso de reconstrucción, lo que resulta en una complejidad lineal en relación con el número de nodos en el árbol.

Como dato curioso, se puede mejorar la complejidad de este algoritmo implementando tablas hash (diccionarios en Python) para almacenar las posiciones de los elementos del recorrido inorden, reduciendo así la complejidad de la búsqueda de O(n) a O(1) en promedio.

Código

```python
class Node:
    def __init__(self, value):
        self.value = value
        self.left = None
        self.right = None

def build_tree(preorder, inorder):
    if not preorder or not inorder:
        return None
    root_value = preorder[0]
    root = Node(root_value)
    root_index = inorder.index(root_value)
    root.left = build_tree(preorder[1:root_index + 1], inorder[:root_index])
    root.right = build_tree(preorder[root_index + 1:], inorder[root_index + 1:])
    return root

def postorder_traversal(root, result):
    if root:
        postorder_traversal(root.left, result)
        postorder_traversal(root.right, result)
        result.append(root.value)
```

```python
def recupera_el_arbol(preorder, inorder):
    root = build_tree(preorder, inorder)
    result = []
    postorder_traversal(root, result)
    return ''.join(result)

def main():
    for line in stdin:
        preorder, inorder = line.split()
        postorder = recupera_el_arbol(preorder, inorder)
        print(postorder)
```

Documentación

Dentro del código.

Fuentes

Método insertado en el código compartido por el profesor sobre Árboles Binarios AVL y los casos de prueba insertados con el input y output desde Command Prompt.