TUGAS MANDIRI
PERANCANGAN ANALISIS ALGORITMA
AVL Tree

Dosen Pengampu:

Disusun Oleh:

Novita Sari Siregar
22343008

PRODI INFORMATIKA
DEPARTEMAN TEKNIK ELEKTRONIKA
FAKULTAS TEKNIK

UNIVERSITAS NEGERI PADANG

2024

KATA PENGANTAR

Puji dan syukur ke hadirat Tuhan Yang Maha Kuasa atas segala rahmat yang diberikan-Nya sehingga tugas Mandiri yang berjudul “AVL Tree” ini dapat saya selesaikan. Makalah ini saya buat sebagai kewajiban untuk memenuhi tugas Perancangan Analisis Algoritma.

Dalam kesempatan ini, saya menghaturkan terimakasih yang dalam kepada semua pihak yang telah membantu menyumbangkan ide dan pikiran mereka demi terwujudnya makalah ini. Sehingga akhirnya saran dan kritik pembaca yang dimaksud untuk mewujudkan kesempurnaan makalahini sangat saya hargai.

Padang, 7 Maret 2024

Novita Sari Siregar

A. Pengertian AVL Tree

Dalam ilmu komputer, sebuah pohon AVL adalah sebuah pohon biner terurutyang dapatmenyeimbangkan dirinya sendiri. Pada sebuah pohon AVL, tinggi dari dua anak sub pohon dari simpul apapun memiliki perbedaan paling besar 'satu'. *Lookup,* penyisipan *(insertion),* dan penghapusan *(deletion)* semuanya memerlukan O(log*n*) kali dalam kasus biasa dan kasus terburuk.

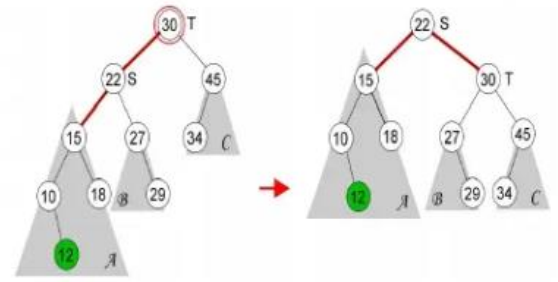
Penambahan *(additions)* dan penghapusan membutuhkan pohon tersebut untukmenyeimbangkan kembali dirinya melaluirotasi pohonsatu kali atau lebih.cara perurutannya yaitu sebelah kiri nilai yang paling rendah sedangkan sebelah kanan nilai paling besar dari nilai utamanya (root),left<root<right.

Operasi insertion dilakukan untuk memepertahan kan height pada avl tree, AVL Tree, Insert suatu node pada AVL sama halnya pada insert node pada binary search tree,dimana node baru diposisikan sebagai leaf. Setelah memasukkan node baru, maka harusdilakukan penyeimbangan kembali pada path dari node yang baru di insert atau pathterdalam. Namun biasanya, path terdalam adalah path dari node yang baru saja di insert.Ada 2 acara menyeimbangkanya yaitu dengan singel rotation dan double rotation.

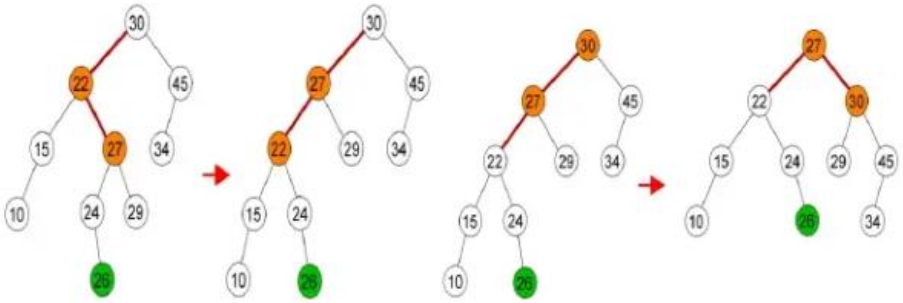
Operasi penghapusan node sama seperti pada Binary Search Tree, yaitu node yangdihapus digantikan oleh node terbesar pada subtree kiri atau node terkecil pada subtreekanan. Jika yang dihapus adalah leaf, maka langsung hapus saja. Namun jika node yangdihapus memiliki child maka childnya yang menggantikannya. Namun setelah operasi penghapusan dilakukan, cek kembali apakah tree sudah seimbang atau belum, jika belummaka harus diseimbangkan kembali. Cara menyeimbangkannya pun sama sepertiinsertion.

B. Cara Kerja AVL Tree

Avl tree bekerja dengan cara insertion dan deletion.Insertion yaitu memasukan node baru, setelah dimasukan node baru maka dilakukan penyemimbangan dengan cara singel rotation dan double rotation.Singel roation rotasi (rotasi 1x) dilakukan apabila searah, left-left atau right-right



Double rotation Double rotasi (rotasi 2x) dilakukan apabila searah, left-right atau right-left



Delection bekerja dengan cara yaitu menghapus node, jika node yang dihapus dalah leaf makalangsung saja, tetapi jika yang di hapus adalah sub tree maka node nya akan menggantikan yangdi hapus, setelah itu dilihat jika tree belum seimbang maka dilakukan dengan cara singel rotationdan double rotation.

Singel rotation, dilakukan jika pada tree terjadi perpindahan secara se arah misalya left-left danright-right.

Double rotation, dilakukan jika pada tree terjadi perpindahan secara dua arah misalnya left-rightdan right-left.

C. Pseudocode dari AVL Tree

```
1. Insert(key, root):
    a. Lakukan operasi penyisipan seperti biasa untuk BST.
    b. Update tinggi dari setiap simpul yang terlibat.
    c. Periksa apakah terjadi ketidakseimbangan (tinggi simpul kiri - tinggi simpul kanan > 1 atau sebaliknya) pada setiap simpul dari simpul baru ke root.
    d. Jika ada ketidakseimbangan, lakukan rotasi sesuai kebutuhan.

2. Rotasi kanan (y):
    a. Simpan simpul kiri dari y sebagai x.
    b. Simpan anak kanan dari x sebagai anak kiri dari y.
    c. Jika y adalah akar, update root dengan x.
    d. Jika y bukan akar, hubungkan x ke orang tua y.
    e. Hubungkan y sebagai anak kanan dari x.
    f. Perbarui ketinggian y dan x.

3. Rotasi kiri (x):
    a. Simpan simpul kanan dari x sebagai y.
    b. Simpan anak kiri dari y sebagai anak kanan dari x.
    c. Jika x adalah akar, update root dengan y.
    d. Jika x bukan akar, hubungkan y ke orang tua x.
    e. Hubungkan x sebagai anak kiri dari y.
    f. Perbarui ketinggian x dan y.
```

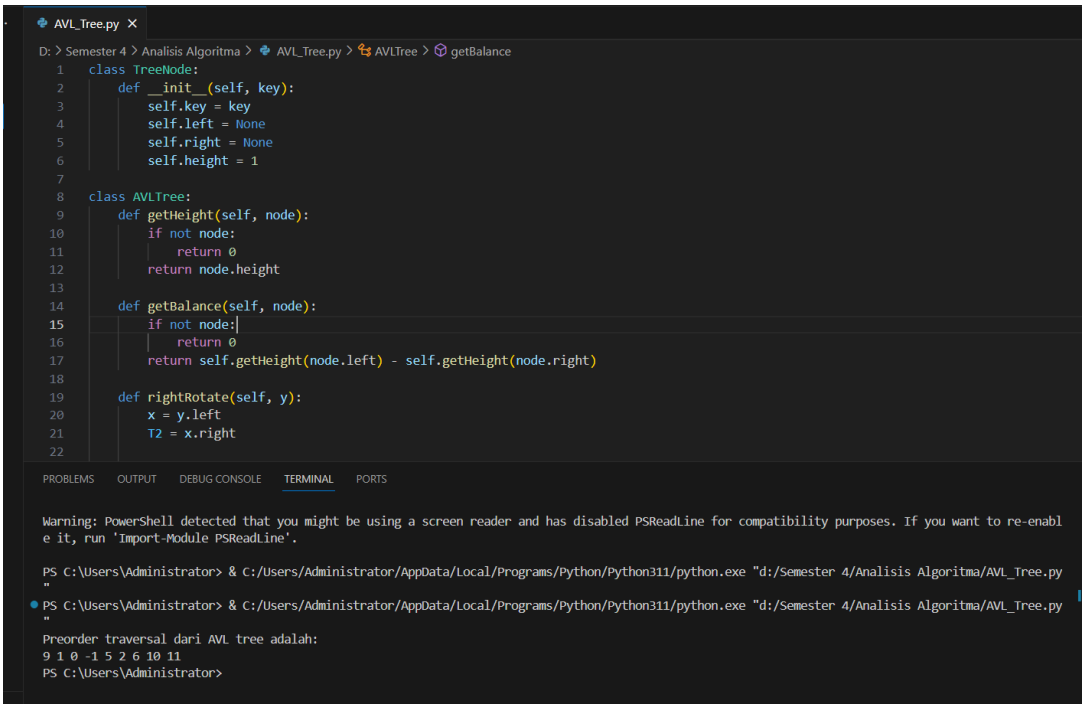
D. Souce Code AVL Tree

```python
class TreeNode:

  def __init__(self, key):
      self.key = key
      self.left = None
      self.right = None
```

```
        self.height = 1

class AVLTree:
    def getHeight(self, node):
        if not node:
            return 0
        return node.height

    def getBalance(self, node):
        if not node:
            return 0
        return self.getHeight(node.left) - self.getHeight(node.right)

    def rightRotate(self, y):
        x = y.left
        T2 = x.right

        x.right = y
        y.left = T2

        y.height = 1 + max(self.getHeight(y.left), self.getHeight(y.right))
        x.height = 1 + max(self.getHeight(x.left), self.getHeight(x.right))

        return x

    def leftRotate(self, x):
        y = x.right
        T2 = y.left

        y.left = x
        x.right = T2

        x.height = 1 + max(self.getHeight(x.left), self.getHeight(x.right))
        y.height = 1 + max(self.getHeight(y.left), self.getHeight(y.right))

        return y

    def insert(self, root, key):
        if not root:
            return TreeNode(key)
        if key < root.key:
            root.left = self.insert(root.left, key)
        else:
            root.right = self.insert(root.right, key)
```

```
        root.height = 1 + max(self.getHeight(root.left),
self.getHeight(root.right))

        balance = self.getBalance(root)

        if balance > 1:
            if key < root.left.key:
                return self.rightRotate(root)
            else:
                root.left = self.leftRotate(root.left)
                return self.rightRotate(root)

        if balance < -1:
            if key > root.right.key:
                return self.leftRotate(root)
            else:
                root.right = self.rightRotate(root.right)
                return self.leftRotate(root)

        return root

    def preOrder(self, root):
        if not root:
            return
        print("{0} ".format(root.key), end="")
        self.preOrder(root.left)
        self.preOrder(root.right)

# Contoh penggunaan:
myTree = AVLTree()
root = None
keys = [9, 5, 10, 0, 6, 11, -1, 1, 2]
for key in keys:
    root = myTree.insert(root, key)

print("Preorder traversal dari AVL tree adalah:")
myTree.preOrder(root)
```

E. Screenshoot Program



F. Analisis Kebutuhan AVL Tree

Seperti yang mungkin Anda asumsikan dari deskripsi pohon AVL sebagai "penyeimbang diri", AVL menyeimbangkan subpohonnya saat sedang dibangun. Kita akan membahasnya nanti, tapi untuk saat ini, pahami saja bahwa pohon AVL SELALU seimbang sempurna.

Sekarang setelah kita mengetahui seperti apa pohon seimbang dan tidak seimbang, mari kita jelajahi manfaat pengoperasian pohon penyeimbangan mandiri AVL dibandingkan dengan Pohon Pencarian Biner tradisional.

Tree type		**Average**	**Worst**
Binary search tree	Space	Θ(n)	O(n)
	Insert	Θ(log n)	O(n)
	Search	Θ(log n)	O(n)
	Delete	Θ(log n)	O(n)

AVL tree	Space	O(n)	O(n)
	Insert	O(log n)	O(log n)
	Search	O(log n)	O(log n)
	Delete	O(log n)	O(log n)

Dalam hal kompleksitas ruang, pohon AVL tidak memberikan manfaat apa pun, seperti yang diharapkan karena kita tidak mengubah berapa banyak data yang disimpan, namun bagaimana data tersebut disusun. Namun, jika menyangkut kompleksitas waktu dari setiap operasi pohon, di situlah pohon AVL jauh lebih unggul daripada pohon biner tradisional. Seperti yang Anda lihat, satu-satunya perbedaan antara dua bagan di atas

adalah skenario terburuk untuk operasi pohon AVL adalah O(log(n)), berbeda dengan skenario terburuk O(n) untuk operasi yang sama persis di pohon biner tradisional.

Mengapa kasus terburuk dan terbaik dari operasi pohon AVL sama persis dengan BST? Hal ini disebabkan oleh aspek "penyeimbangan diri" dari pohon AVL yang menjamin kita mendapatkan pohon yang seimbang setiap saat. Dalam pohon biner seimbang, pencarian, penyisipan, dan penghapusan semuanya memerlukan waktu logaritmik karena jumlah node yang harus dilalui dipotong setengahnya pada setiap tingkat saat kita menelusuri pohon tersebut. Manfaat keseluruhan dari menyimpan data kita di dalam pohon biner adalah untuk membuka kompleksitas runtime besar yang hanya mungkin terjadi di dalam pohon biner seimbang, yang selalu dijamin oleh AVL. Seperti yang kita lihat pada contoh di atas, tanpa penyeimbangan, sebuah pohon bisa menjadi sama buruknya dengan LinkedList, yang berarti kita harus menelusuri semua nilai yang mungkin (O(n) kompleksitas waktu) untuk melakukan operasi apa pun (kasus terburuk).

G. Sumber Referensi

https://medium.com/@natepill/avl-trees-a-complete-guide-and-implementation-addd4516c1d2
https://www.scribd.com/embeds/439382188/content?start_page=1&view_mode=scroll&access_key=key-fFexxf7r1bzEfWu3HKwf
https://youtu.be/hFKCk70r7WE?si=IpOYqmsAEkatxTfx