

# **STRUKTUR DATA (PYTHON)**

## **“Struktur BST dan Pohon Seimbang AVL”**

[@SUARGA] [Pertemuan 14]

# OutLine





# Konsep Pohon Seimbang

- Pohon Seimbang (Balanced Tree) adalah pohon yang tinggi-nya pada sisi kiri dan sisi kanan hampir sama.
- Apabila tidak diatur keseimbangan-nya maka mungkin terjadi suatu pohon dimana semua node ada di sisi kiri sementara sisi kanan-nya kosong, atau sebaliknya.
- Ada 2 contoh struktur pohon yang akan dibahas pada materi hari ini, yaitu:
  1. Binary Search Tree (BST) : memudahkan pencarian
  2. AVL Tree : sangat seimbang

# Konsep Binary Search Tree

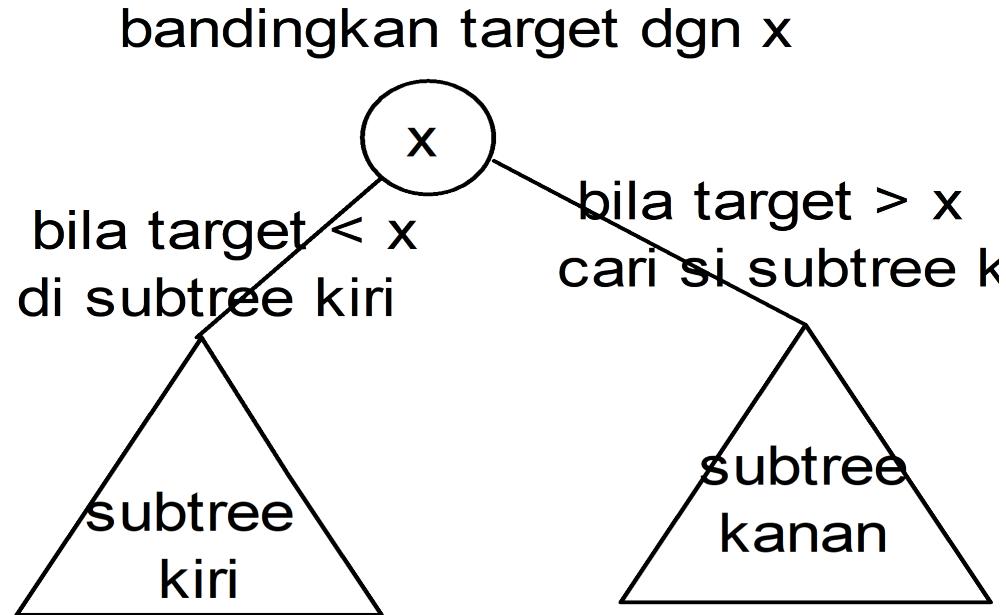
- Suatu Binary Search Tree atau BST adalah pohon biner dimana pada setiap node termuat suatu **kunci** (key), dan juga **data** (value) dan disusun mengikuti aturan pada pohon biner, agar suatu node mudah dicari, yaitu:
  1. Semua kunci yang lebih kecil dari kunci node-V akan disimpan disisi kiri dari node V.
  2. Semua kunci yang lebih besar atau sama dari kunci node-V akan disimpan di sisi-kanan dari node V
  3. Akan diatur agar kunci pada root memiliki nilai yang berada disekitar titik tengah vektor nilai.

# BST Class

- Node dari BST bisa di-bangun dalam Python sebagai berikut:

```
class BST:  
    def __init__( self ):  
        self._root = None  
        self._size = 0  
    def __len__( self ):  
        return self._size  
  
class _BSTNode:  
    def __init__(self, key, value):  
        self.key = key  
        self.value = value  
        self.left = None  
        self.right = None
```

- Pada dasarnya akses ke BST tidak jauh berbeda dari pohon biner, misalnya dalam proses pencarian kunci, maka nilai kunci yang dicari dibandingkan dengan nilai  $x$  pada suatu node. Apabila kunci itu lebih kecil dari  $x$  maka cari di subtree sebelah kiri dan bila lebih besar maka cari di subtree sebelah kanan.



# fungsi \_bstSearch()

- Mencari suatu kunci target dalam BST
- Kode fungsi-nya bisa seperti berikut:

```
def _bstSearch( self, subtree, target ):  
    if subtree is None:  
        return None  
    elif target < subtree.key :  
        return self._bstSearch(subtree.left )  
    elif target > subtree.key :  
        return self._bstSearch(subtree.right )  
    else :  
        return subtree
```

- Berdasarkan pada fungsi `_bstSearch()` dapat dibuat dua fungsi baru yaitu fungsi `_contains_()` untuk memeriksa apakah suatu kunci ada dalam BST, dan juga fungsi `valueOf()` untuk mencari nilai data (value) yang berhubungan dengan suatu kunci (key).

```
def __contains__( self, key ):  
    return self._bstSearch(self._root, key )  
  
def valueof( self, key ):  
    node = self._bstSearch( self._root, key )  
    assert node is not None, "Invalid key... "  
    return node.value
```

- Mencari nilai key yang minimum maupun maksimum sering pula menjadi hal yang penting dalam BST. Nilai minimum diharapkan berada pada sisi-kiri dari root BST, karena bila root memuat key minimum maka pasti node lainnya semua akan berada di sisi kanan, dan sisi-kiri menjadi kosong. Demikian pula dengan nilai key maksimum, diharapkan berada di-sisi kanan dari root. Apabila berada di-posisi root, maka node lainnya semua berada di sisi-kiri.

# mencari maximum dan minimum

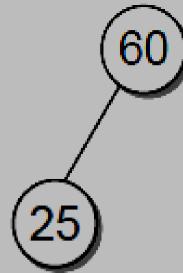
```
def _bstMinimum( self, subtree ):  
    if subtree is None:  
        return None  
    elif subtree.left is None:  
        return subtree  
    else :  
        return self._bstMinimum( subtree.left )  
  
def _bstMaximum( self, subtree ):  
    if subtree is None:  
        return None  
    elif subtree.right is None:  
        return subtree  
    else :  
        return self._bstMaximum( subtree.right )
```

# menyisipkan node: insert()

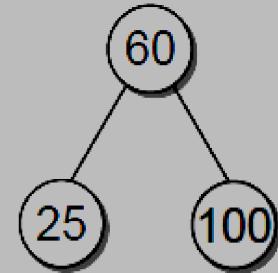
- Memasukkan suatu node ke dalam BST adalah dengan mengikuti aturan yang telah ditetapkan sebelumnya. Node pertama menjadi root selanjutnya bila key lebih kecil dari root maka tempatkan di sisi-kiri dan sebaliknya di sisi-kanan.



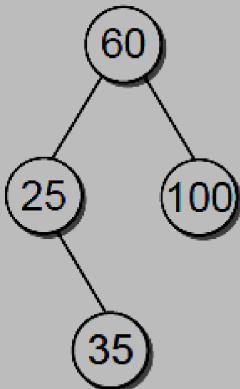
(a) Insert 60.



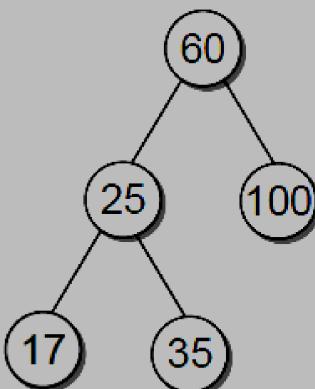
(b) Insert 25.



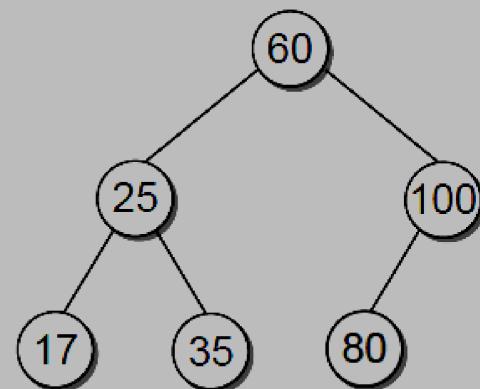
(c) Insert 100.



(d) Insert 35.



(e) Insert 17.



(f) Insert 80.

- Kode untuk menyisipkan atau memasukkan nilai baru ke dalam node adalah sebagai berikut.

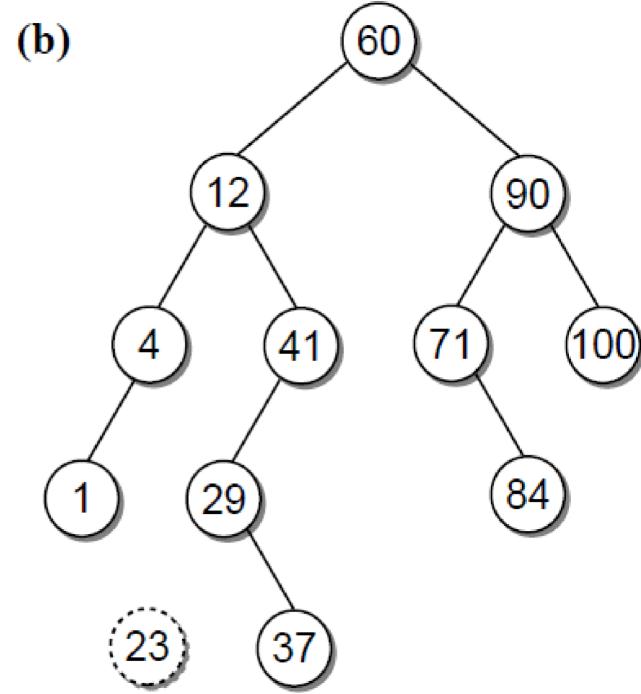
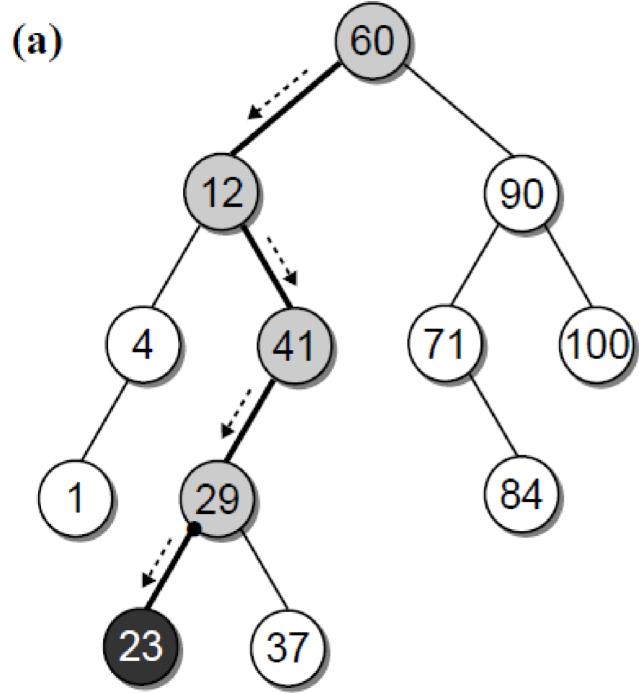
```
def _bstInsert( self, subtree, key, value ) :  
    if subtree is None :  
        subtree = _BSTNode( key, value )  
    elif key < subtree.key :  
        subtree.left = self._bstInsert(  
            subtree.left, key, value )  
    elif key > subtree.key :  
        subtree.right = self._bstInsert(  
            subtree.right, key, value )  
    return subtree
```

- Apabila key sudah ada tetapi data value-nya mau diganti maka kode-nya sebagai berikut:

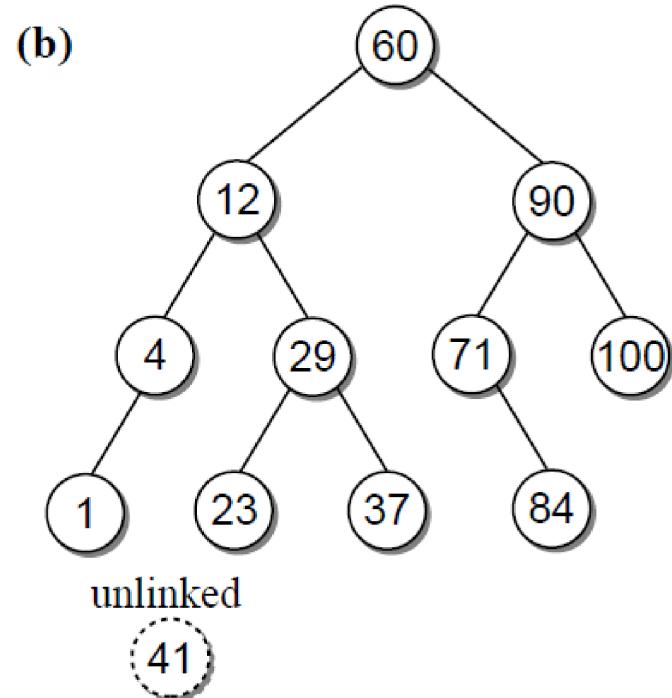
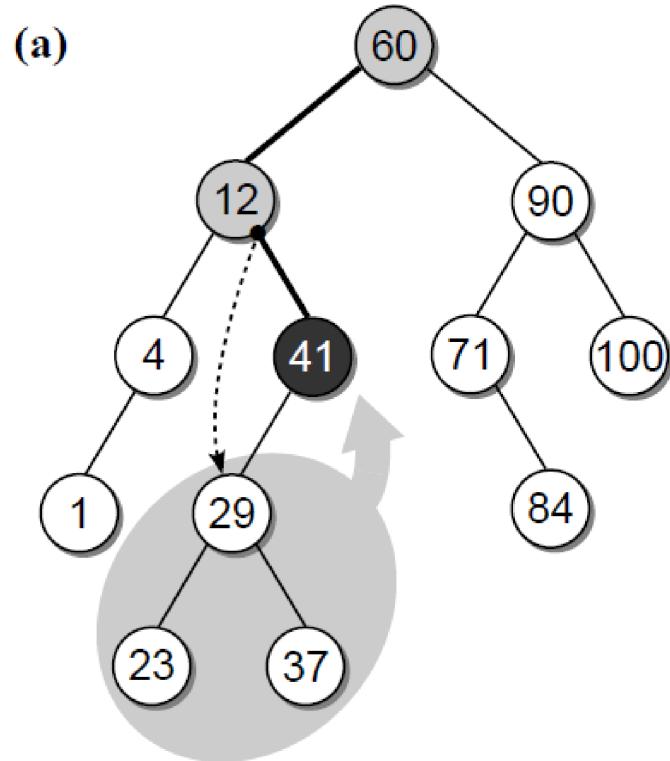
```
def add( self, key, value ):  
    node = self._bstSearch( key )  
    # apa kunci sdh ada  
    if node is not None:  
        # bila sdh ada  
        node.value = value  
        # ganti value-nya  
        return False  
    else:                      # kunci tidak ada  
        self._root = self._bstInsert( self._root, key,  
                                      value )  
        self._size += 1      # masukkan node-nya  
    return True
```

# fungsi remove()

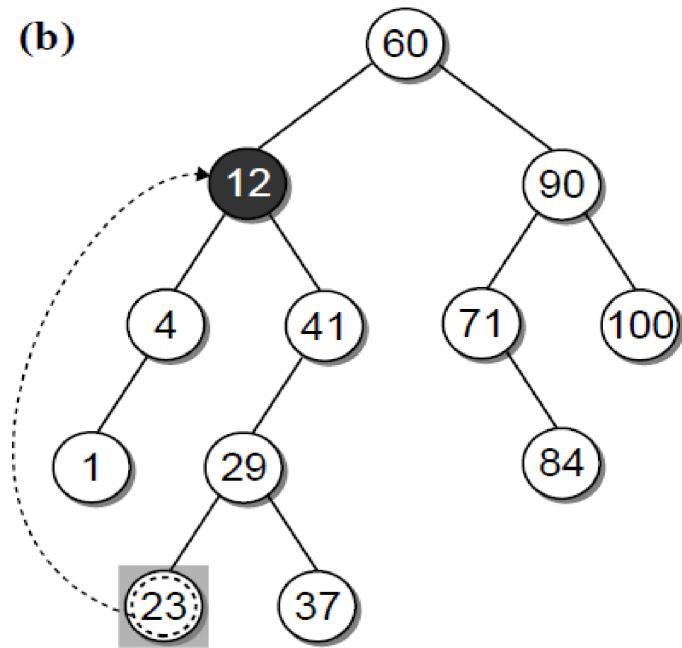
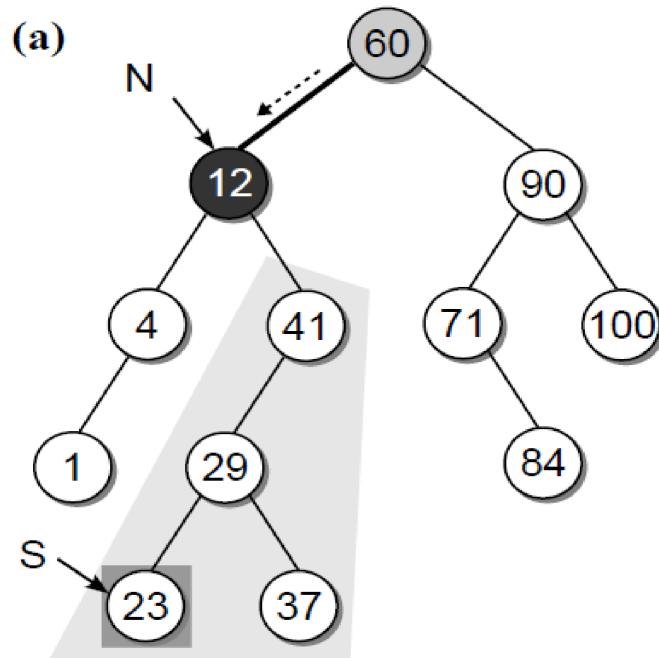
- Hal yang paling sulit dilakukan pada BST adalah menghapus suatu node (remove) berdasarkan suatu kunci. Ada tiga kemungkinan yang terjadi, yaitu:
  1. Node ini tidak memiliki anak, maka bisa langsung dihapus
  2. Node ini memiliki satu anak, maka node ini dihapus kemudian anak-nya ditarik menggantikannya
  3. Node ini memiliki dua anak, maka anak yang berada di-ujung kanan dari subtree kiri diangkat menggantikan node yang dihapus, kalau tidak ada, maka anak yang berada paling kiri dari subtree kanan-nya dipilih sebagai pengganti, kemudian anak sisi-kanannya dipasang dibawahnya.



Kasus 1, node 23 tidak memiliki anak  
node 23 bisa langsung dihapus

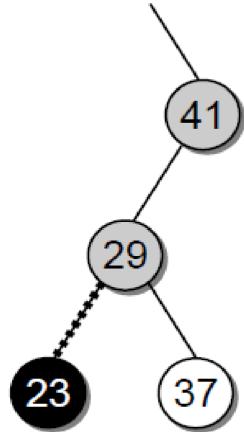


Kasus 2, node 41 memiliki satu anak  
node 29 dapat diangkat menggantikan node 41

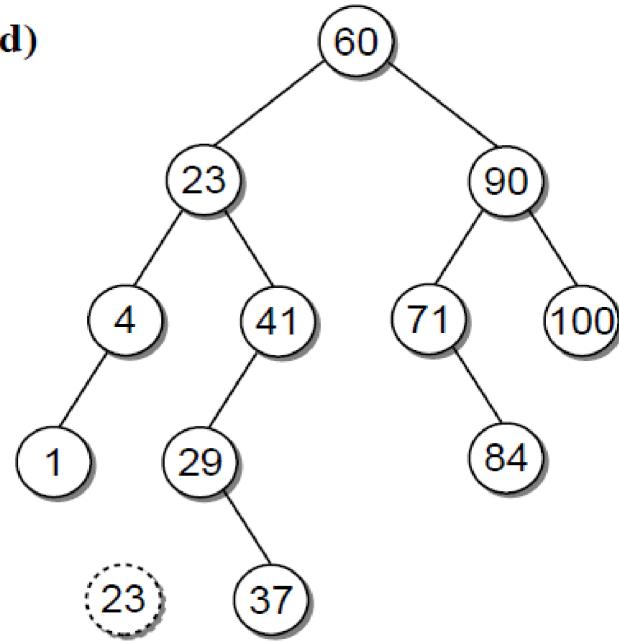


Kasus 3, node 12 memiliki 2 anak  
cari node paling kanan dari turunan sisi-kiri  
atau node paling kiri dari turunan sisi-kanan

(c)



(d)



Karena tidak ada node paling kanan dari sisi-kiri node 12  
maka pilih node paling kiri dari sisi-kanan node 12, yaitu  
node 23 untuk menggantikan node 12

- Kode fungsi remove() dapat ditulis sebagai berikut.

```
def remove( self, key ):  
    assert key in self, "Invalid key."  
    self._root = self._bstRemove( self._root, key )  
    self._size -= 1  
  
    # metoda yang melakukan ‘remove’ dari item recursif.  
def _bstRemove( self, subtree, target ):  
    # mencari item dalam tree.  
    if subtree is None :  
        return subtree  
    elif target < subtree.key :  
        subtree.left = self._bstRemove( subtree.left, target )  
    return subtree
```

```
elif target > subtree.key :  
    subtree.right = self._bstRemove( subtree.right, target )  
    return subtree  
# disini node yang memuat item ditemukan.  
else :  
    if subtree.left is None and subtree.right is None :  
        return None  
    elif subtree.left is None or subtree.right is None :  
        if subtree.left is not None :  
            return subtree.left  
        else :  
            return subtree.right  
    else :  
        successor = self._bstMinimum( subtree.right )  
        subtree.key = successor.key  
        subtree.value = successor.value  
        subtree.right = self._bstRemove( subtree.right,  
                                         successor.key )  
    return subtree
```

# Implementasi BST

```
1 #my_BST.py == implementasi BST @Suarga
2 #definisi class untuk node BST
3 class _BSTNode:
4     def __init__(self, key, value):
5         self.key = key
6         self.value = value
7         self.left = None
8         self.right = None
9
10 #definisi| pohon BST
11 class BST:
12     #memulai BST
13     def __init__( self ):
14         self._root = None
15         self._size = 0
16
17     #menghitung banyaknya elemen/node
18     def __len__( self ):
19         return self._size
20
```

```
21 #mencari sebuah node target
22 def _bstSearch( self, subtree, target ):
23     if subtree is None:
24         return None
25     elif target < subtree.key :
26         return self._bstSearch( subtree.left, target )
27     elif target > subtree.key :
28         return self._bstSearch( subtree.right, target )
29     else :
30         return subtree
31
32 #memeriksa apakah key ada dalam BST
33 def __contains__( self, key ):
34     return self._bstSearch(self._root, key )
35
36 #menampilkan nilai yang berkaitan dengan kunci key
37 def valueOf( self, key ):
38     node = self._bstSearch( self._root, key )
39     assert node is not None, "Invalid key... "
40     return node.value
41
```

```
42 #menampilkan key minimum
43 def _bstMinimum( self, subtree ):
44     if subtree is None:
45         return None
46     elif subtree.left is None:
47         return subtree
48     else :
49         return self._bstMinimum( subtree.left )
50
51 #menampilkan key maksimum
52 def _bstMaximum( self, subtree ):
53     if subtree is None:
54         return None
55     elif subtree.right is None:
56         return subtree
57     else :
58         return self._bstMaximum( subtree.right )
59
60 #ADT tambahan ketika menyisipkan satu Node ke BST
61 def _bstInsert( self, subtree, key, value ):
62     if subtree is None :
63         subtree = _BSTNode( key, value )
64     elif key < subtree.key :
65         subtree.left = self._bstInsert( subtree.left, key, value )
66     elif key > subtree.key :
67         subtree.right = self._bstInsert( subtree.right, key, value )
68     return subtree
```

```
69
70     #ADT untuk memasukkan sebuah Node ke dalam BST
71     def add( self, key, value ):
72         node = self._bstSearch( self._root, key ) # apa kunci sdh ada
73         if node is not None:                  # bila sdh ada
74             node.value = value              # ganti value-nya
75             return False
76         else:                           # kunci tidak ada
77             self._root = self._bstInsert( self._root, key, value )
78             self._size += 1                 # masukkan node-nya
79             return True
80
81     #menghapus sebuah Node
82     def remove( self, key ):
83         assert key in self, "Invalid key."
84         self._root = self._bstRemove( self._root, key )
85         self._size -= 1
86
```

```
87 # ADT tambahan ketika menghapus sebuah key.
88 def _bstRemove( self, subtree, target ):
89     # Search for the item in the tree.
90     if subtree is None :
91         return subtree
92     elif target < subtree.key :
93         subtree.left = self._bstRemove( subtree.left, target )
94         return subtree
95     elif target > subtree.key :
96         subtree.right = self._bstRemove( subtree.right, target )
97         return subtree
98     # We found the node containing the item.
99     else :
100        if subtree.left is None and subtree.right is None :
101            return None
102        elif subtree.left is None or subtree.right is None :
103            if subtree.left is not None :
104                return subtree.left
105            else :
106                return subtree.right
107        else :
108            successor = self._bstMinimum( subtree.right )
109            subtree.key = successor.key
110            subtree.value = successor.value
111            subtree.right = self._bstRemove( subtree.right, successor.key )
112    return subtree
113
```

```
114 #menampilkan isi BST
115 def disp_tree(self, indent_char = '...', indent_delta=2):
116     node = self._root
117     def disp_tree_1(indent, node):
118         if node == None:
119             return None
120         else:
121             disp_tree_1(indent+indent_delta, node.right)
122             print(indent*indent_char+str(node.key)+ '|' + str(node.value))
123             disp_tree_1(indent+indent_delta, node.left)
124     disp_tree_1(0,node)
125
126 def disp_tree_rev(self, indent_char = '...', indent_delta=2):
127     node = self._root
128     def disp_tree_1(indent, node):
129         if node == None:
130             return None
131         else:
132             disp_tree_1(indent+indent_delta, node.left)
133             print(indent*indent_char+str(node.key)+ '|' + str(node.value))
134             disp_tree_1(indent+indent_delta, node.right)
135     disp_tree_1(0,node)
136
```

```
137 def main():
138     bt = BST()
139     bt.add(5, "Purple")
140     bt.add(3, "Black")
141     bt.add(7, "Red")
142     bt.add(2, "Yellow")
143     bt.add(9, "Green")
144     bt.disp_tree()
145     print('tambah beberapa node : ')
146     bt.add(8, "Blue")
147     bt.add(4, "White")
148     bt.disp_tree()
149     print('buang node key 4 : ')
150     bt.remove(4)
151     bt.disp_tree()
152     print('Cari key maksimum : ')
153     r = bt._root
154     node = bt._bstMaximum(r)
155     print(str(node.key) + ' | ' + str(node.value))
156     print('Cari key minimum : ')
157     node = bt._bstMinimum(r)
158     print(str(node.key) + ' | ' + str(node.value))
159
160 main()
```

# Hasil Test

```
===== RESTART: D:/USER/
.....9|Green
.....7|Red
5|Purple
.....3|Black
.....2|Yellow
tambah beberapa node :
.....9|Green
.....8|Blue
.....7|Red
5|Purple
.....4|White
.....3|Black
.....2|Yellow
buang node key 4 :
.....9|Green
.....8|Blue
.....7|Red
5|Purple
.....3|Black
.....2|Yellow
Cari key maksimum :
9|Green
Cari key minimum :
2|Yellow
```

# Balanced Tree / AVL Tree

- Pohon AVL pada prinsipnya adalah pohon BST dengan aturan yang lebih ketat.
- Pada BST akses bisa lebih efisien apabila pohon ini seimbang di sisi-kiri dengan sisi-kanan, karena pada kondisi ini maka tinggi pohon adalah  $(\log n)$ , dimana n adalah jumlah node dalam BST.
- Namun bisa saja terjadi kasus yang sangat buruk (worst case), yaitu ketika akar memiliki kunci yang nilainya minimum atau nilainya maksimum, dalam kondisi ini maka tinggi pohon adalah  $(n)$ .

- Berdasarkan pada kasus wors-case ini maka tiga orang peneliti yaitu: G.M. Adel'son, Velskii, dan Y.M. Landis merancang satu pohon biner dimana keseimbangan selalu di-perhatikan setiap kali satu node baru dimasukkan.
- Pohon biner ini kemudian diberi nama pohon AVL yang diambil dari huruf pertama nama belakang penemunya. Suatu pohon biner disebut seimbang apabila tinggi (height) sisi-kiri dan tinggi sisi-kanan maksimum berbeda satu.

- Pada setiap node dalam AVL di-hubungkan dengan suatu indikator yang disebut “balance factor” atau faktor keseimbangan, yaitu:

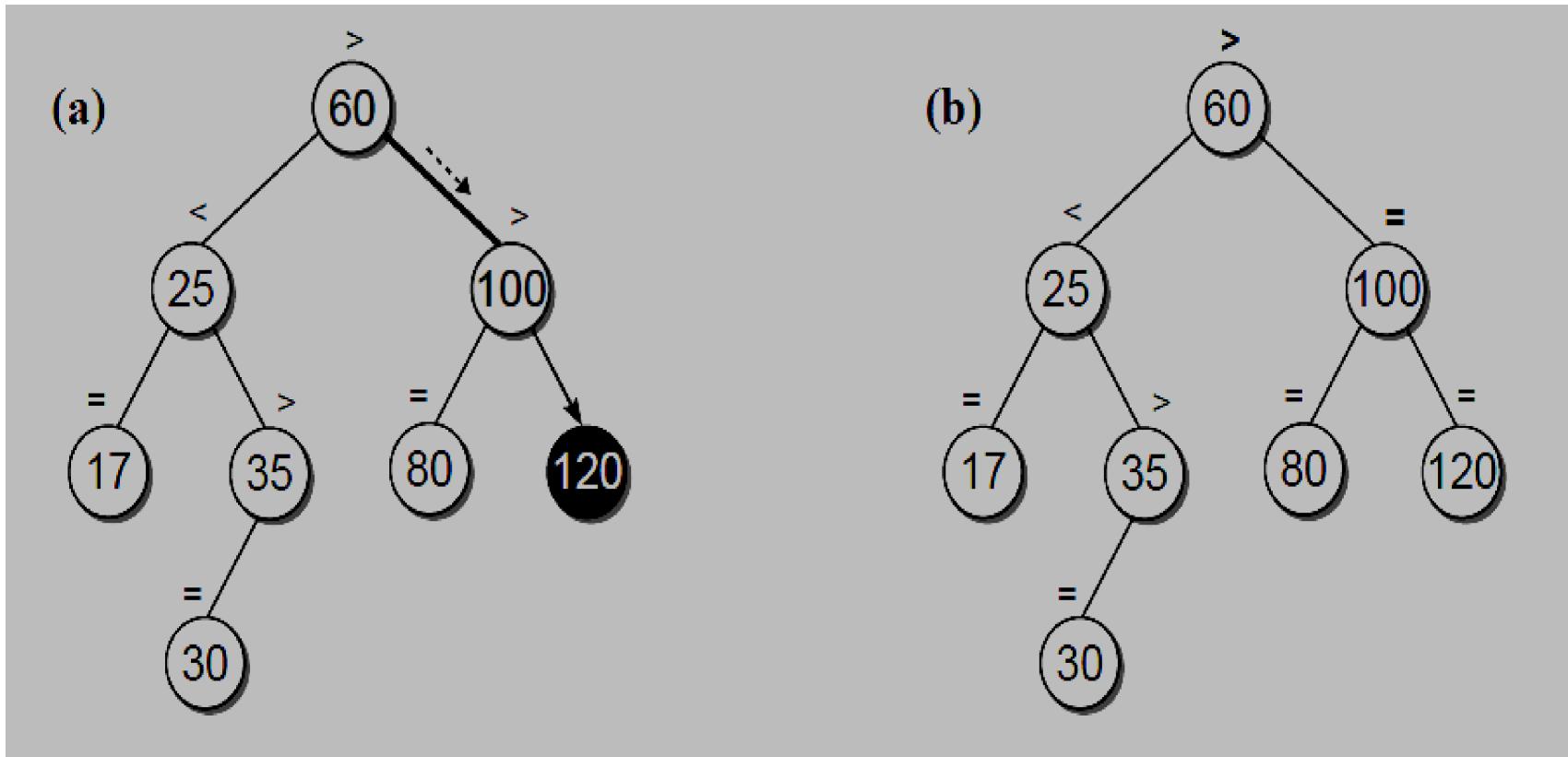
tanda > bila yang dikiri lebih tinggi

tanda = bila sama tinggi

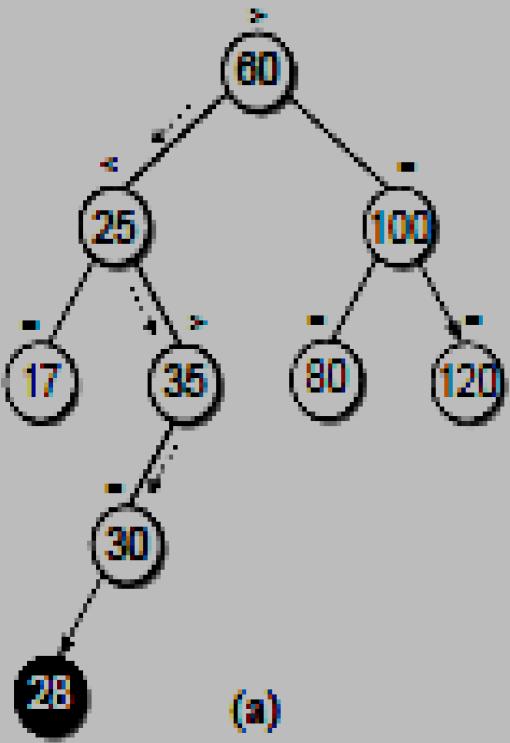
tanda < bila yang dikanan lebih tinggi

- Indikator ini bisa berubah ketika suatu node baru telah ditempatkan, sehingga susunan node dalam AVL bisa diubah menuju keseimbangan kiri dan kanan, agar bisa dijamin tinggi pohon maksimal adalah ( $1.44 \log n$ ).

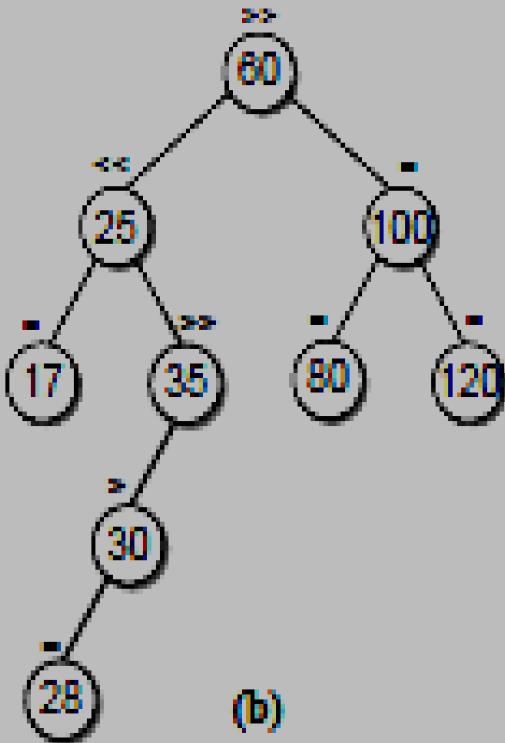
- Sebagai contoh andaikan node 120 akan ditambahkan ke dalam pohon AVL seperti pada gambar (a) berikut ini, maka sesuai dengan aturan pohon biner node 120 ditempatkan di-sisi-kanan node 100. Indikator keseimbangan berubah seperti pada gambar (b). Walau demikian antara sisi-kiri dan sisi-kanan masih tetap berselisih satu tinggi-nya, sehingga masih memenuhi syarat AVL.



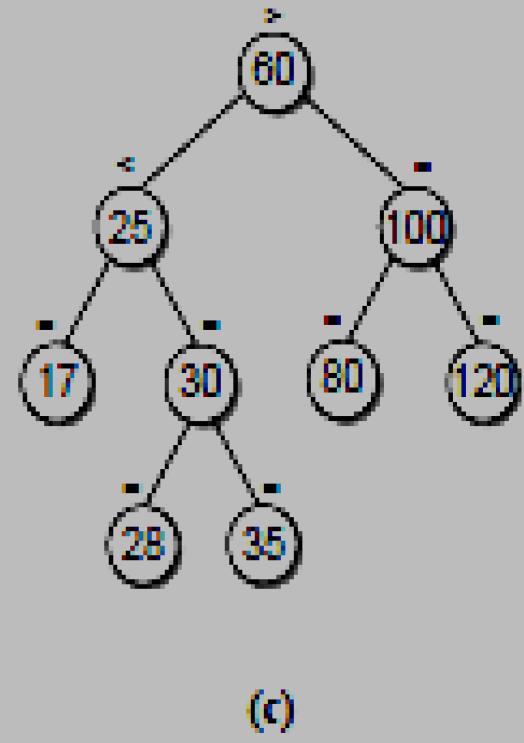
Bagaimana kalau node 28 ditambahkan, node ini ditempatkan di sisi-kiri node 30, seperti pada gambar berikut (a), maka antara sisi-kiri dan sisi-kanan tinggi-nya sudah berbeda dua (b), sehingga AVL ini perlu diseimbangkan lagi menjadi seperti gambar (c).



(a)



(b)



(c)

# Implementasi pohon AVL

```
#####
#AVLTree.py  == @Suarga
import sys

class AVLNode:
    def __init__(self):
        self.data = 0
        self.ddata = ""
        self.left = None
        self.right = None

    def displayNode(self):
        print('{', self.data, ', ', self.ddata, '}')
```

Definisi node dari pohon AVL

```
- class AVLTree:
-     def __init__(self):
-         self.root = None
-
-     def avlinsert(self,data,ddata):
20        newItem = AVLNode()
21        newItem.data = data
22        newItem.ddata = ddata
23        if (self.root == None):
24            self.root = newItem
25        else:
26            self.root = self.RecursiveInsert(self.root, newItem)
```

Fungsi inisialisasi (`__init__()`) dan fungsi untuk menyisip sebuah node baru. (`avlinsert()`)

```
def RecursiveInsert(self, current, newDt):
    if (current == None):
        current = newDt
        return current
    elif (newDt.data < current.data): # go left
        current.left = self.RecursiveInsert(current.left, newDt)
        current = self.balance_tree(current) #balancing after insert
    elif (newDt.data > current.data): # go right
        current.right = self.RecursiveInsert(current.right, newDt)
        current = self.balance_tree(current)

    return current
```

fungsi menyisip node ke posisi-nya (di kiri atau di kanan)

```
40
def balance_tree(self, current):
    b_factor = self.balance_factor(current)
    if (b_factor > 1):
        if (self.balance_factor(current.left) > 0):
            current = self.RotateLL(current)
        else:
            current = self.RotateLR(current)
    elif (b_factor < -1):
        if (self.balance_factor(current.right) > 0):
            current = self.RotateRL(current)
        else:
            current = self.RotateRR(current)
    return current
50
```

fungsi untuk memeriksa keseimbangan AVL dengan menghitung “balance\_factor”, bila tidak seimbang adakan suatu rotasi node:

- RotateLL : rotasi ke arah kiri
- RotateLR : rotasi ke kiri dulu lalu putar ke kanan
- RotateRL : rotasi ke kanan dulu lalu putar ke kiri
- Rotate RR: rotasi ke arah kanan

```
def getHeight(self, current):
    height = 0
    if (current != None):
        l = self.getHeight(current.left)
        r = self.getHeight(current.right)
        m = max(l, r)
        height = m + 1
    return height

60

def balance_factor(self, current):
    l = self.getHeight(current.left)
    r = self.getHeight(current.right)
    b_factor = l - r
    return b_factor
66
```

fungsi untuk menhitung tinggi pohon (getHeight()) dan fungsi yang menghitung faktor keseimbangan (balance\_factor())

```
70 def RotateRR(self, parent):
    pivot = parent.right
    parent.right = pivot.left
    pivot.left = parent
    return pivot

80 def RotateLL(self, parent):
    pivot = parent.left
    parent.left = pivot.right
    pivot.right = parent
    return pivot

83 def RotateLR(self, parent):
    pivot = parent.left
    parent.left = self.RotateRR(pivot)
    return self.RotateLL(parent)

90 def RotateRL(self, parent):
    pivot = parent.right
    parent.right = self.RotateLL(pivot)
    return self.RotateRR(parent)
```

fungsi-fungsi rotasi untuk menyeimbangkan pohon AVL

```
def DisplayTree(self):
    self.InOrderDisplayTree(self.root)
    print()

def InOrderDisplayTree(self, current):
    if (current != None):
        self.InOrderDisplayTree(current.left)
        print('{',current.data,',',current.ddata,'} ')
        self.InOrderDisplayTree(current.right)

def find(self, key):
    ketemu = self.search(key)
    if (ketemu != None):
        print('Ditemukan: ')
    else:
        print(key, 'Tidak ditemukan')
```

Fungsi untuk menampilkan isi pohon (DisplayTree() + InOrderDisplayTree())  
serta fungsi untuk menemukan satu kunci (find())

```
    def search(self, key):
        current = self.root
        while (current.data != key):
            if (key < current.data):
                current = current.left
            else:
                current = current.right
            if (current == None):
                return None
        current.displayNode()
        return current
```

Fungsi search() adalah ADT tambahan untuk pencarian key

```
def delete(self, current, target):
    if (current == None):
        return None
    else:
        if (target < current.data): # left subtree
            current.left = self.delete(current.left, target)
            if (self.balance_factor(current) == -2):
                if (self.balance_factor(current.left) <= 0):
                    current = self.RotateRR(current)
                else:
                    current = self.RotateRL(current)
        elif (target > current.data): # right subtree
            current.right = self.delete(current.right, target)
            if (self.balance_factor(current) == 2):
                if (self.balance_factor(current.right) <= 0):
                    current = self.RotateLL(current)
                else:
                    current = self.RotateLR(current)
        else: # target found
```

Fungsi delete() untuk menghapus sebuah node, yang memanfaatkan fungsi rotasi RR, RL, LL, LR

```
140     else: # target found
141         if (current.right != None):
142             parent = current.right
143             while (parent.left != None):
144                 parent = parent.left
145             current.data = parent.data
146             current.right = self.delete(current.right, parent.data)
147             if (self.balance_factor(current) == 2):
148                 if (self.balance_factor(current.left) <= 0):
149                     current = self.RotateLL(current)
150                 else:
151                     current = self.RotateLR(current)
152             else:
153                 return current.left
154
155     return current
```

```
158     def Delete(self, target):
159         self.delete(self.root, target)
160
161     # display sideways
162     def disp_tree(self, indent_char = '...', indent_delta=2):
163         node = self.root
164         def disp_tree_1(indent, node):
165             if node == None:
166                 return None
167             else:
168                 disp_tree_1(indent+indent_delta, node.right)
169                 print(indent*indent_char+str(node.data))
170                 disp_tree_1(indent+indent_delta, node.left)
171
172         disp_tree_1(0, node)
173
174     # display normally
```

```
# display normally
def display_Tree(self):
    globalStack = []
    globalStack.append(self.root)
    nBlanks = 32
    isRowEmpty = False
    titik = '..'
    print(32*titik)
    while (isRowEmpty == False):
        localStack = []
        isRowEmpty = True
        spasi = ' '
        for j in range(nBlanks):
            sys.stdout.write("%s" % spasi)
        while (len(globalStack) != 0):
            temp = globalStack.pop()
            if (temp != None):
                sys.stdout.write("%d" % temp.data)
                localStack.append(temp.left)
                localStack.append(temp.right)
                if ((temp.left != None) or (temp.right != None)):
                    isRowEmpty = False
            else:
                isRowEmpty = True
    print('')

180
190
```

```
        else:
            sys.stdout.write('---')
            localStack.append(None)
            localStack.append(None)
            for j in range(nBlanks * 2 - 2):
                sys.stdout.write("%s" % spasi)
            print(' ')
            nBlanks = nBlanks//2
            while (len(localStack)!= 0):
                globalStack.append(localStack.pop())
            print(32*titik)

206
def main():
    theTree = AVLTree()
    print('Insert data : ')
210
    theTree.avlinsert(50, "Agus")
    theTree.avlinsert(25, "Beddu")
    theTree.avlinsert(75, "Chaidar")
    theTree.avlinsert(12, "Daud")
    theTree.avlinsert(37, "Erik")
    theTree.avlinsert(43, "Farid")
    theTree.avlinsert(30, "Gugun")
    theTree.avlinsert(33, "Harun")
    theTree.avlinsert(87, "Indah")
    theTree.avlinsert(93, "Jeny")
    theTree.avlinsert(97, "Kiki")
220
```

```
220     theTree.avlinsert(97, "Kiki")
        print('Insert selesai')
        print('Data dibaca inorder :')
        theTree.DisplayTree()
        print('Bentuk AVL : ')
        theTree.display_Tree()
        print('Menghapus node 43')
        theTree.Delete(43)
        print('AVL setelah 43 dihapus:')
        theTree.display_Tree()
        print('AVL tree : sideways')
        theTree.disp_tree()
        print('Mencari node 30')
        theTree.find(30)
        print('Mencari node 55')
        theTree.find(55)
        print('Menghapus node 87')
        theTree.Delete(87)
        print('AVL setelah 87 dihapus:')
        theTree.display_Tree()
        print('Insert node 56:')
        theTree.avlinsert(56, 'Lilik')
        print('AVL setelah 56 di-insert')
        theTree.display_Tree()
```

2-September-2022

Struktur Data-14, @Suarga

# Hasil Test

```
Python Interpreter
*** Remote Interpreter Reinitialized ***
Insert data :
Insert selesai
Data dibaca inorder :
{ 12 , Daud }
{ 25 , Beddu }
{ 30 , Gugun }
{ 33 , Harun }
{ 37 , Erik }
{ 43 , Farid }
{ 50 , Agus }
{ 75 , Chaidar }
{ 87 , Indah }
{ 93 , Jeny }
{ 97 , Kiki }

Bentuk AVL :
.....
      37
    25           87
  12       30       50       93
--   --   --   33   43   75   --
.....
```



Python Interpreter

Ditemukan:

Mencari node 55

55 Tidak ditemukan

Menghapus node 87

AVL setelah 87 dihapus:

.....  
37

25

93

12

30

50

97

--

--

--

33

--

75

--

--

Insert node 56:

AVL setelah 56 di-insert

.....  
37

25

93

12

30

56

97

--

--

--

33

50

75

--

--

>>>