Pertemuan 13

AVL Tree

Learning Outcomes

Pada akhir pertemuan ini, diharapkan mahasiswa akan mampu :

- mendemonstrasikan operasi pada AVL Tree.
- menerapkan AVL Tree pada program aplikasi komputer.

Outline Materi

- Terminologi
- Operasi Insert
  - Single Rotation
  - Double Rotation
- Representasi
- Implementasi

Terminologi

AVL Tree adalah BST dengan ketentuan |height(LS)-height(RS)|<2

Height-Balanced Tree

BST adalah Height-Balanced p-Tree, yang berarti maksimum perbedaan height antara subtree kiri dan kanan adalah p.

AVL Tree adalah Height-Balanced 1-Tree yang berarti maksimum perbedaan tinggi antara subtree kiri dan kanan adalah 1.

Balancing :

- **TallLeft (–):subtree kiri lebih tinggi dari subtr**
- **TallRight(+):subtree kanan lebih tinggi dari subtree kiri.**
- **Balance (0):tinggi subtree kiri&kanan**

balance factor = -1

balance factor = 1

balance factor = 0

Search Path

Path pencarian lokasi untuk dilakukan operasi INSERT, (dimulai dari Root).

Pivot Point

Adalah Node pada Search Path yang balancenya TallLeft atau TallRight dan terletak paling dekat dengan node yang baru.

Operasi Insert

- **Operasi INSERT pada AVL tree harus tetap menghasilkan AVL Tree (tidak mengubah ketentuan AVL Tree).**

- **Setiap penambahan node baru, ada kemungkinan menyebabkan non AVL Tree, untuk itu perlu dilakukan *rebalancing* / *regenerate*.**

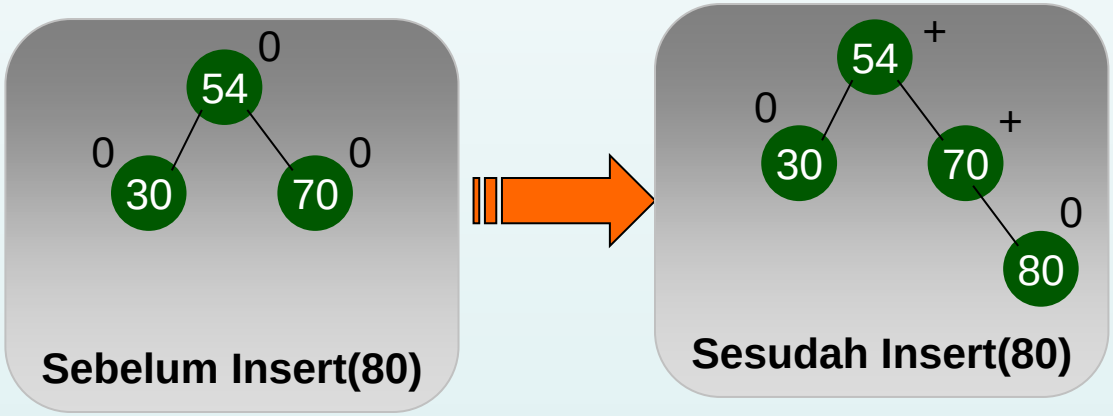
- **Proses *rebalancing* dilakukan dengan cara melakukan rotasi pada subtree.**

- **Penambahan node baru pada kondisi seluruh subtree *balance*, *rebalancing* tidak diperlukan lagi setelah proses INSERT**.

Kasus 1

Tidak ada pivot point dan setiap node adalah balance, maka insert bisa langsung dilakukan sama seperti BST (tanpa rebalancing).

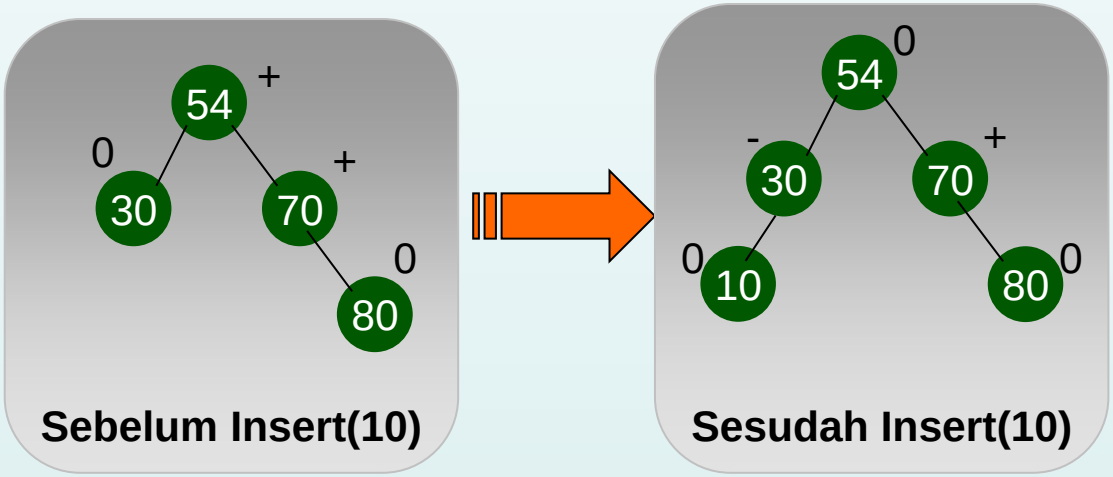
Contoh :



Kasus 2

Ada pivot point tetapi subtree yang akan diinsert lebih pendek, maka insert langsung bisa dilakukan.

Contoh :



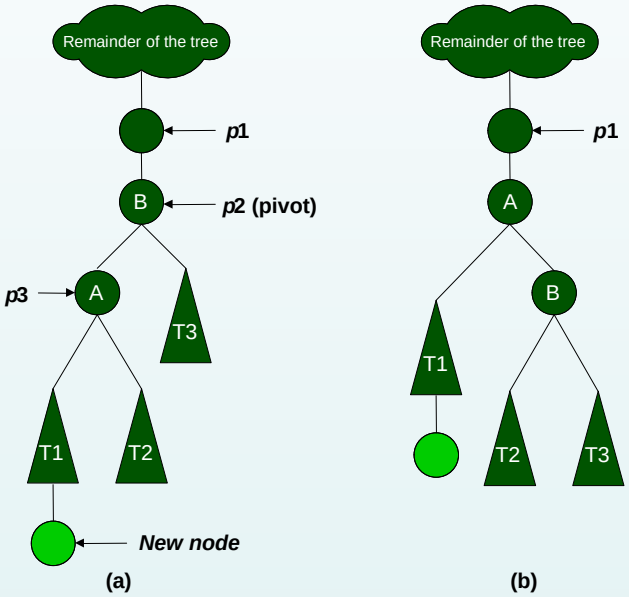
Kasus 3

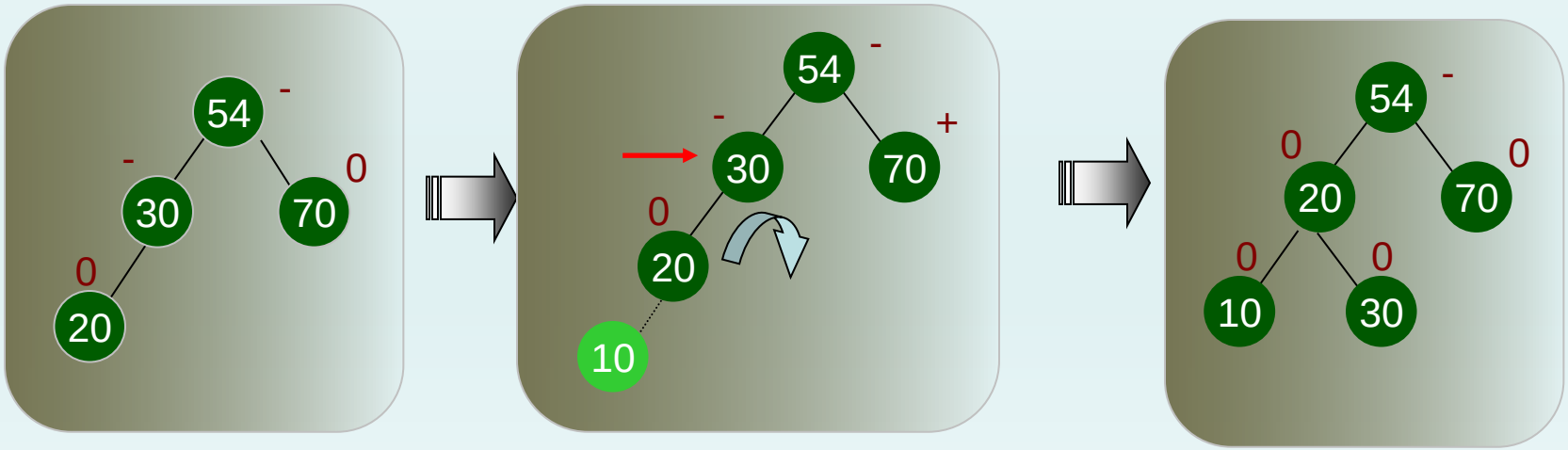
Jika ada pivot point dan subtree yang akan diinsert lebih tinggi, maka TREE harus digenerate, supaya tetap menghasilkan AVL TREE.

Regenerate :

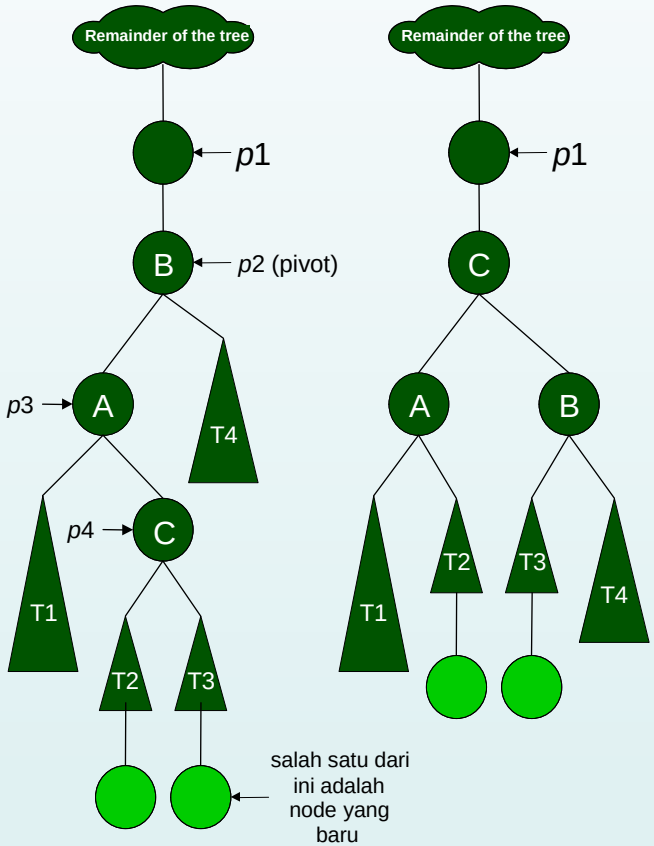
- **Single Rotation**
- **Double Rotation**

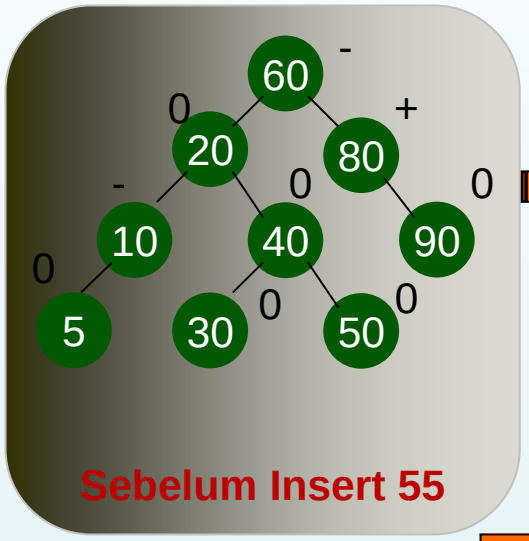
Single Rotation

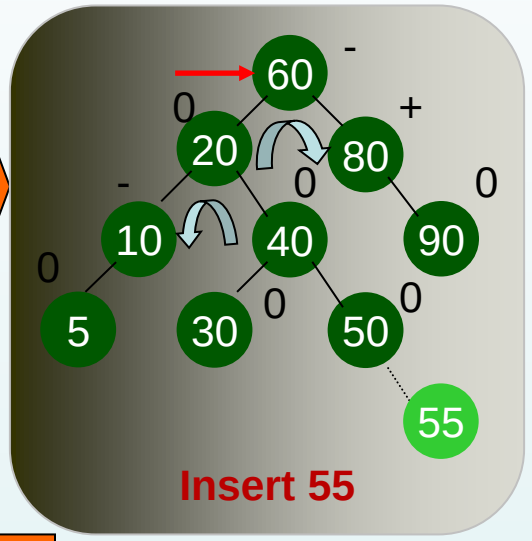


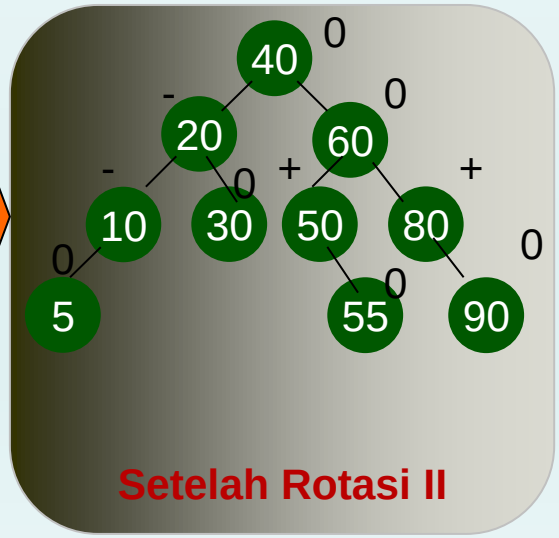


Double Rotation









Implementasi

```
##define IS_FULL(ptr) (!ptr)
##define FALSE = 0
##define TRUE = 1

/*   Representation   */
typedef struct {
     int key;
     } element;

typedef struct tree_node *tree_pointer;
     struct tree_node {
               tree_pointer        left_child;
               element             data;
               short int           bf;
               tree_pointer        right_child;
     };

int unbalanced = FALSE;
tree_pointer root=NULL;
```

Implementasi(2)

```
void avl_insert(tree_pointer *parent, element x, int *unbalanced) {
      if (!parent) {
        *unbalanced = TRUE;
        parent=(tree_pointer) malloc(sizeof(tree_node));
        if (IS_FULL(!parent)) {
                fprintf(stderrr,"The memory is full\n");
                exit(1);
        };
        (*parent)->left_child=(*parent)->right_child=NULL;
        (*parent)->bf=0;
        (*parent)->data=x;
      }
      else if(x.key<(*parent)->data.key){
                …
      }
      else if(x.key>(*parent)->data.key){
                …
      }
      else {
                *unbalanced = FALSE;
                printf("The key is already in the tree");
      }
};
```

```
avl_insert((*parent)->left_child, x, unbalanced);
if (*unbalanced) {
     /* left branch has grown higher */
    switch((*parent)->bf){
    case -1:(*parent)->bf=0;*
                unbalanced=FALSE; break;
    case 0 :(*parent)->bf=1; break;
    case 1 :left_rotation(parent,unbalanced);}}
```

```
avl_insert((*parent)->right_child, x, unbalanced);
if (*unbalanced) {
   /* right branch has grown higher */
   switch((*parent)->bf){
   case 1 :(*parent)->bf=0;
               *unbalanced=FALSE;break;
   case 0 :(*parent)->bf=1;break;
   case 1 :right_rotation(parent,unbalanced); }}
```

Latihan

Jika + berarti Insert, maka buatlah AVL - Tree dari operasi-operasi berikut ini :

+ 76 + 35 + 90 + 50 + 25

+ 150 + 100 + 80 + 40 + 50

+ 125 + 60 + 75 + 140 + 55

+ 140 + 20 + 120 + 66 + 82

Latihan

Jika + berarti Insert, maka buatlah AVL - Tree dari operasi-operasi berikut ini :

+ 16	+ 35	+ 40	+ 50	+ 75
+ 99	+ 90	+ 85	+ 80	+ 60
+ 25	+ 65	+ 55	+ 44	+ 15
+ 14	+ 20	+ 22	+ 66	+ 82

Latihan

Jika + berarti Insert, dan – berarti Delete, maka buatlah AVL - Tree dari operasi - operasi berikut ini :

+ 76 + 35 + 90 + 50 + 25

+ 150 + 100 + 80 + 40 - 50

+ 125 + 60 - 76 + 140 + 55

+ 140 - 25 + 120 - 60 - 80

Selesai