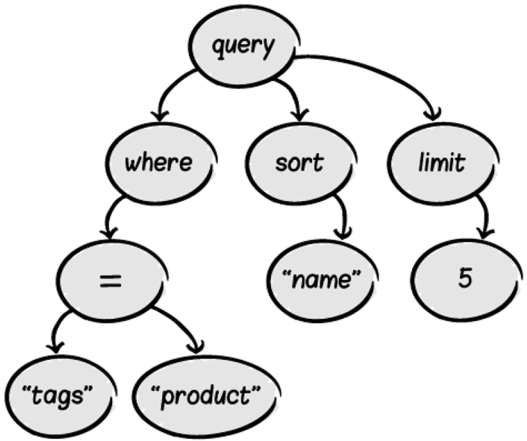
AVL Agacı

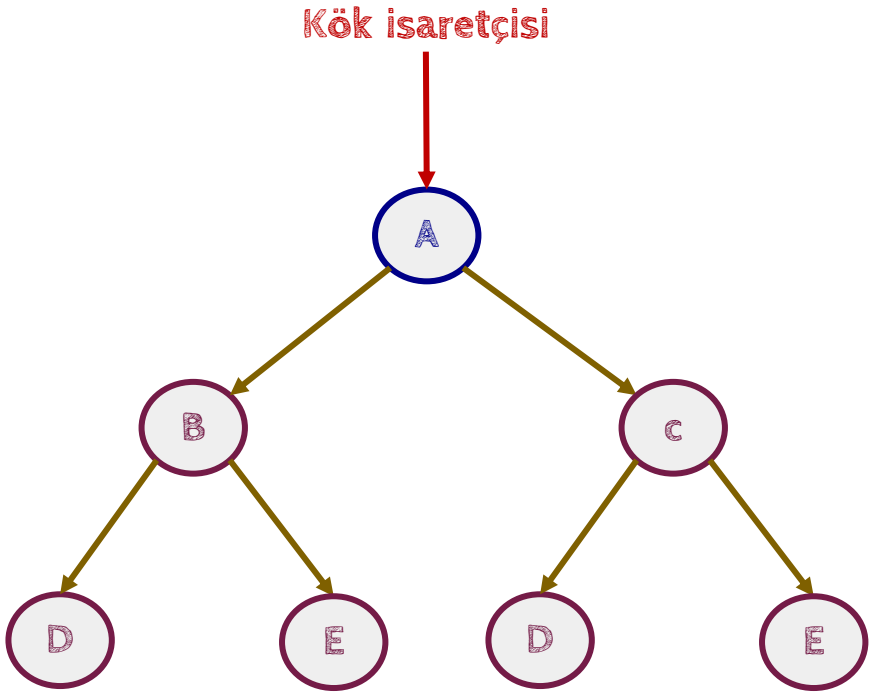


Suhap SAHIN
Onur GÖK

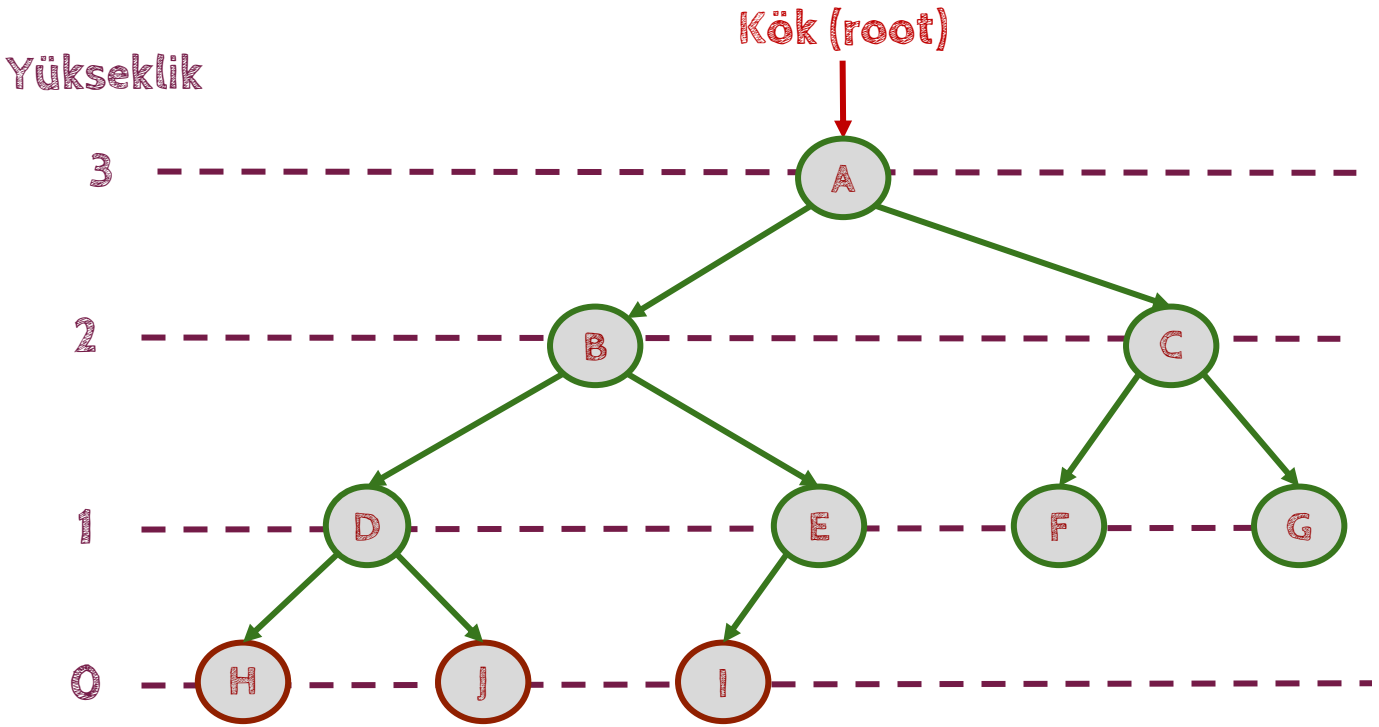
AVL (Adel'son-Vel'skiĭ) Landis Agacı

AVL Agacı:

- Dengeli ikili agaç
- Denge Faktörü



Ağaç Veri Modeli

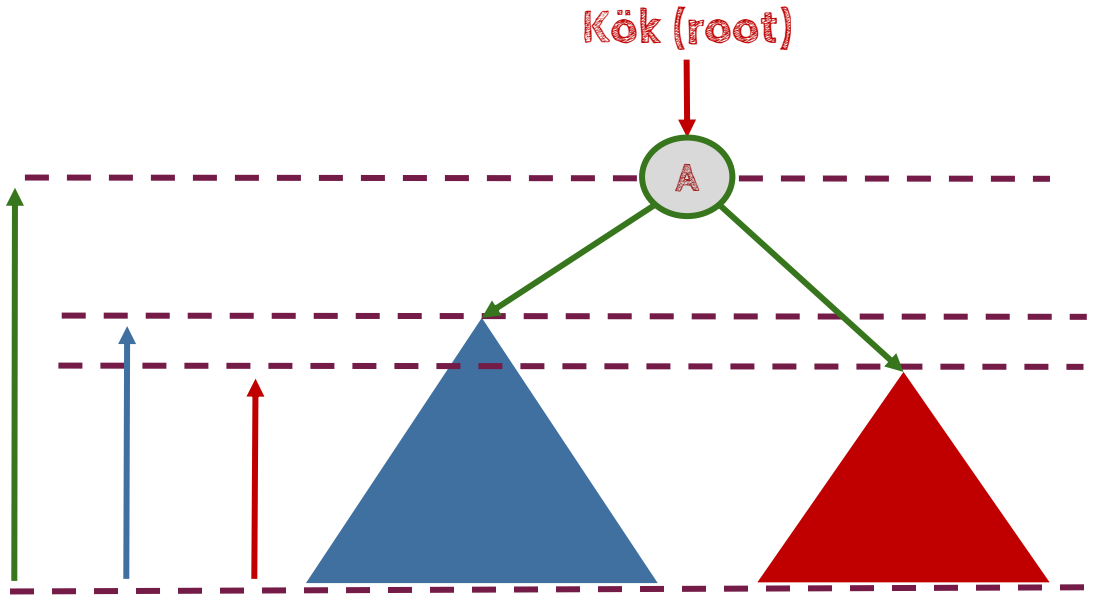


AVL (Adel'son-Vel'skiĭ) Landis Agacı

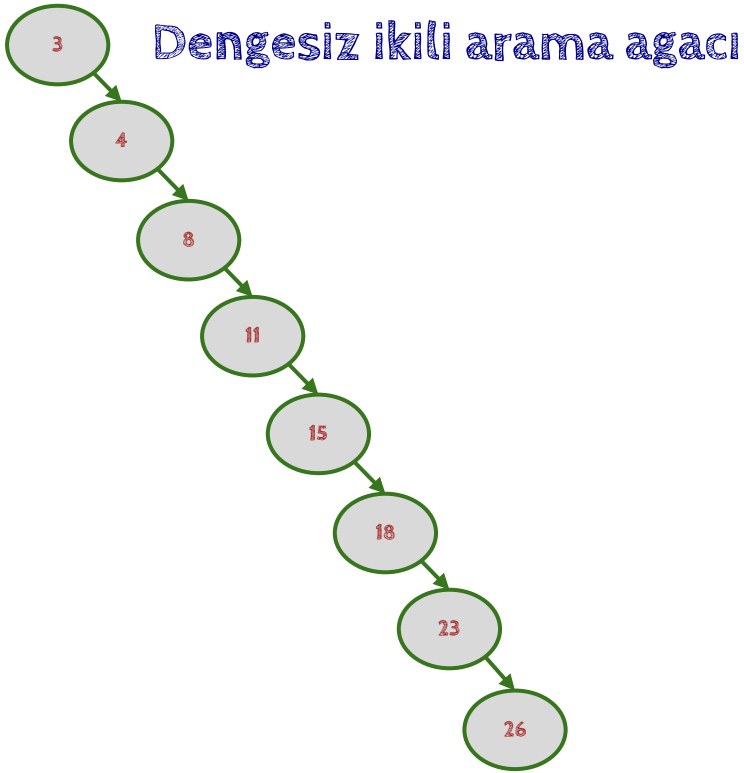
AVL Agacı:

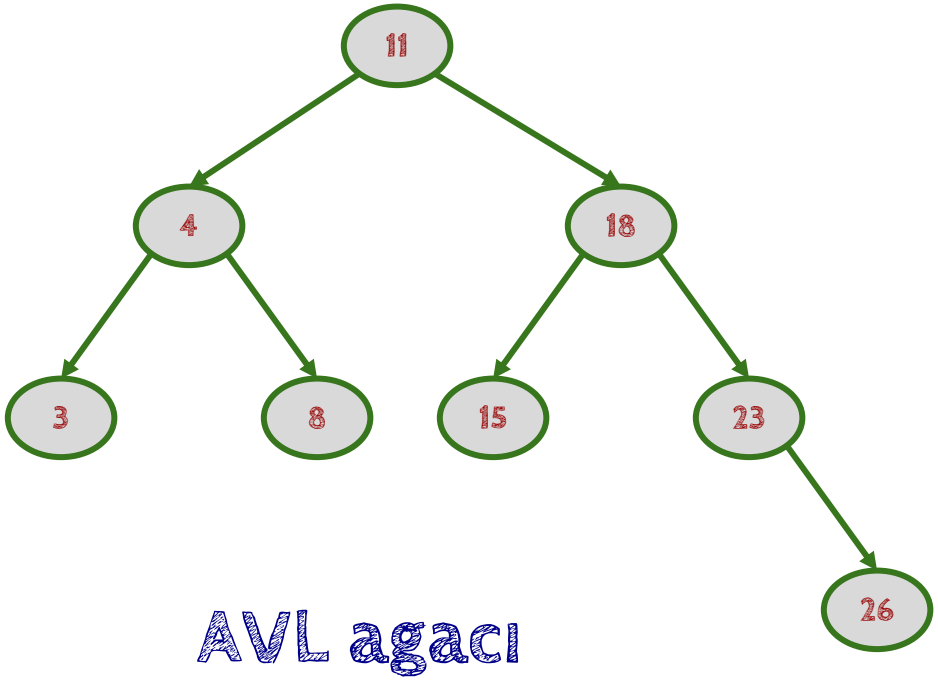
Dengeli ikili agaç

Denge Faktörü

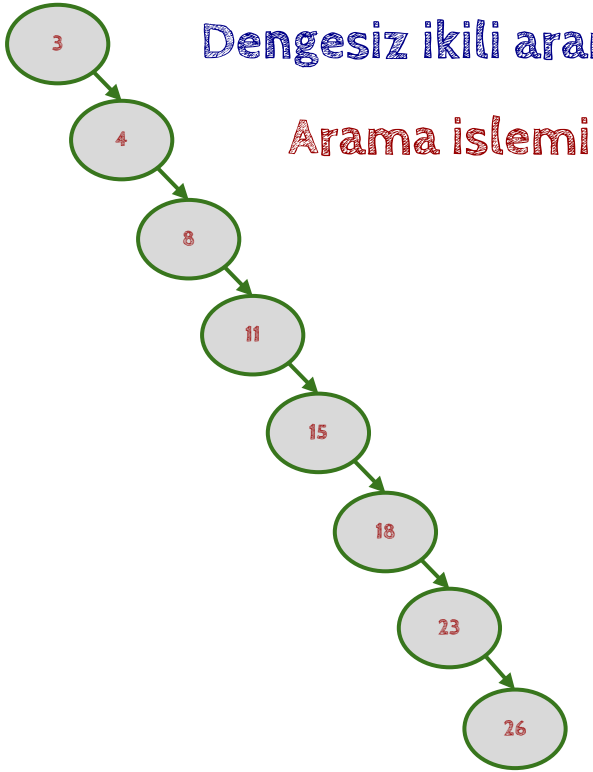


AVL (Adel'son-Vel'skiĭ) Landis Agacı



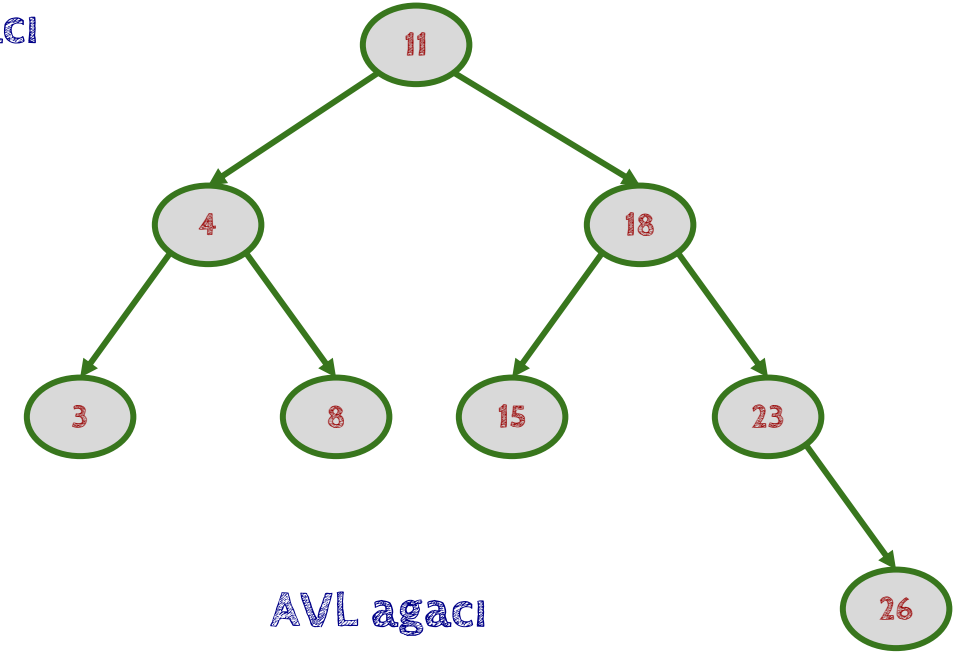


AVL (Adel'son-Vel'skiĭ) Landis Agacı



Dengesiz ikili arama agacı

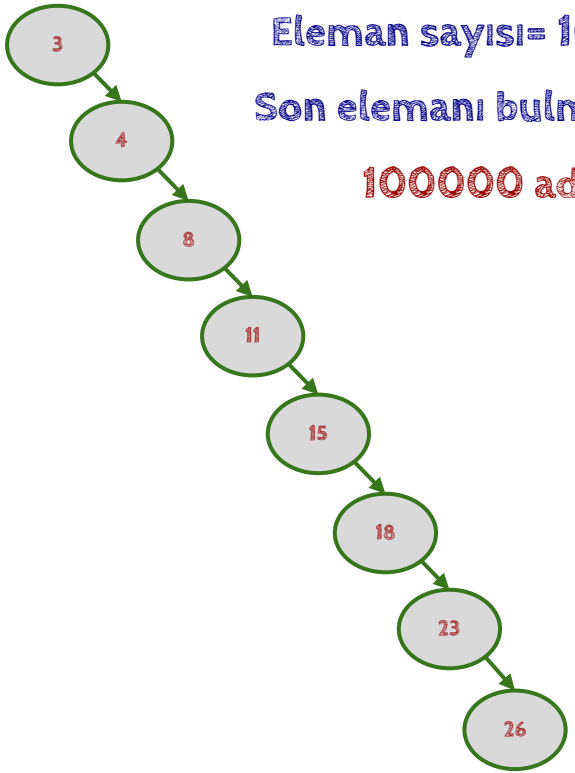
Arama islemi O(n)



AVL agacı

Arama islemi $O(\log_2 n)$

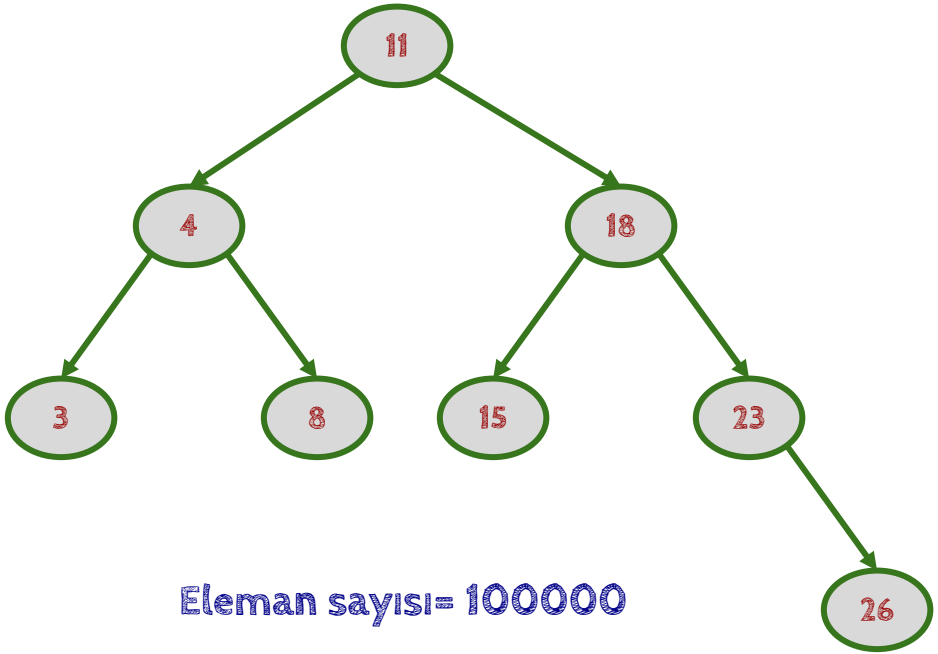
AVL (Adel'son-Vel'skiĭ) Landis Agacı



Eleman sayısı= 100000

Son elemanı bulma islemi

100000 adım



Eleman sayısı= 100000

Köke en uzak elemanı bulma islemi

$\log_2(100000) = 17$ adım

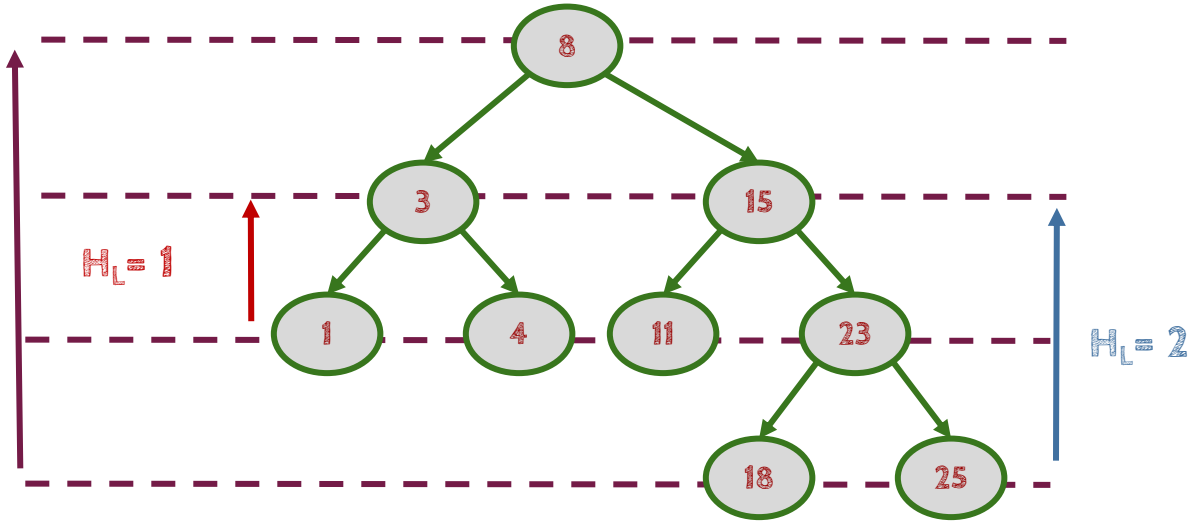
AVL Denge Faktörü

H_L : Sol alt agacın yüksekligi

H_R : Sag alt agacın yüksekligi

denge faktörü = H_L - H_R

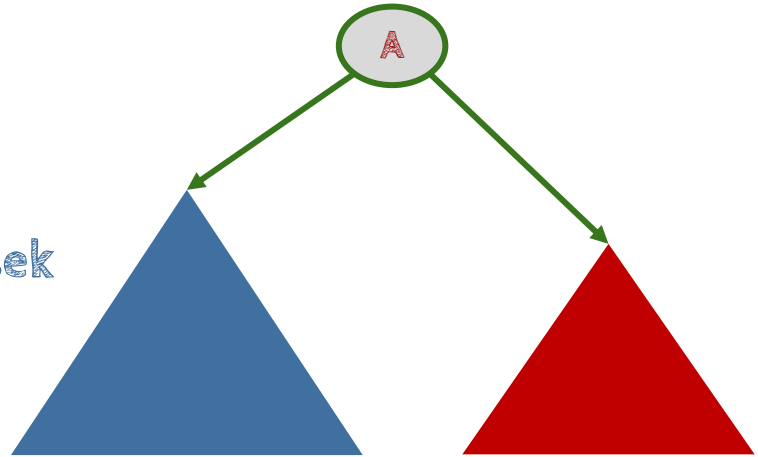
denge faktörü = 1 - 2 = -1



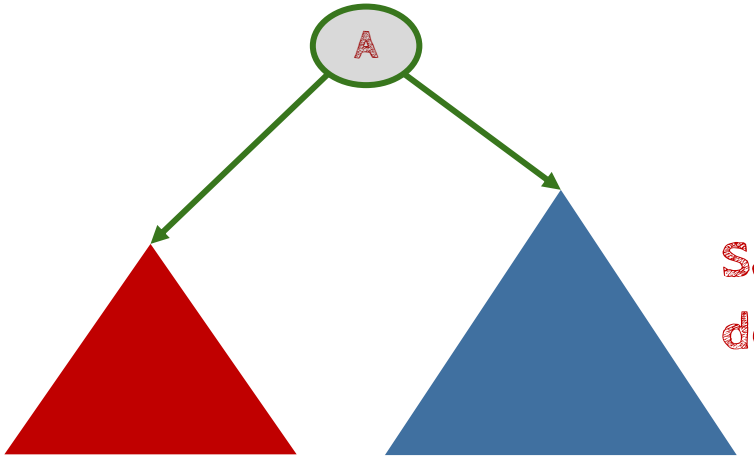
AVL Denge Faktörü

Sol agaç daha yüksek

denge faktörü = +1



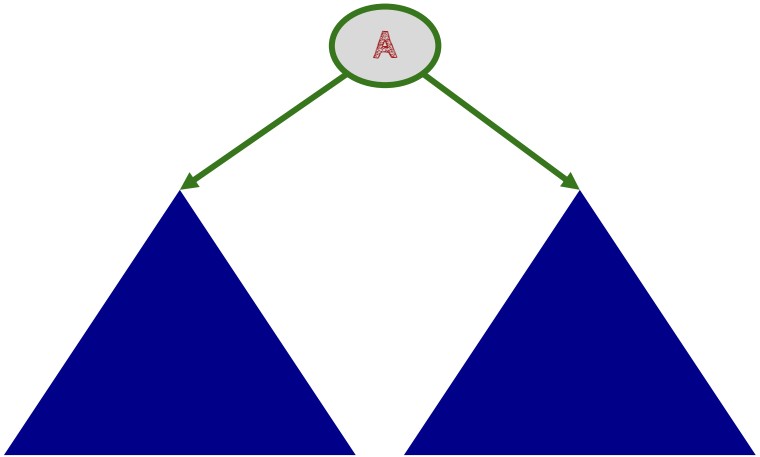
AVL Denge Faktörü



Sag agaç daha yüksek

denge faktörü = -1

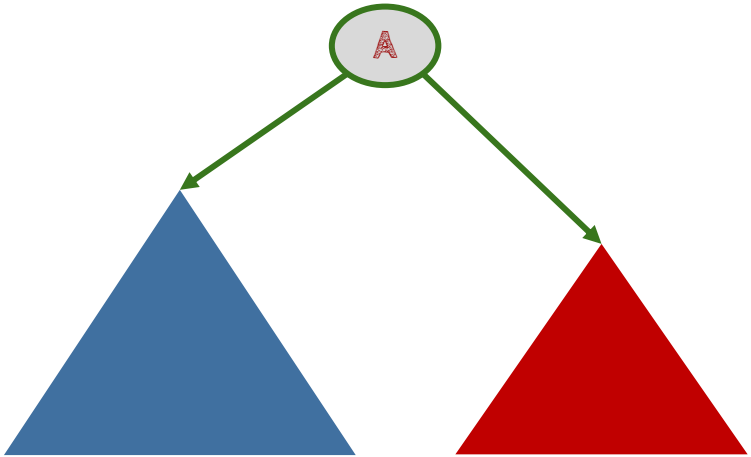
AVL Denge Faktörü



iki taraf esit yükseklikteyse denge faktörü = 0

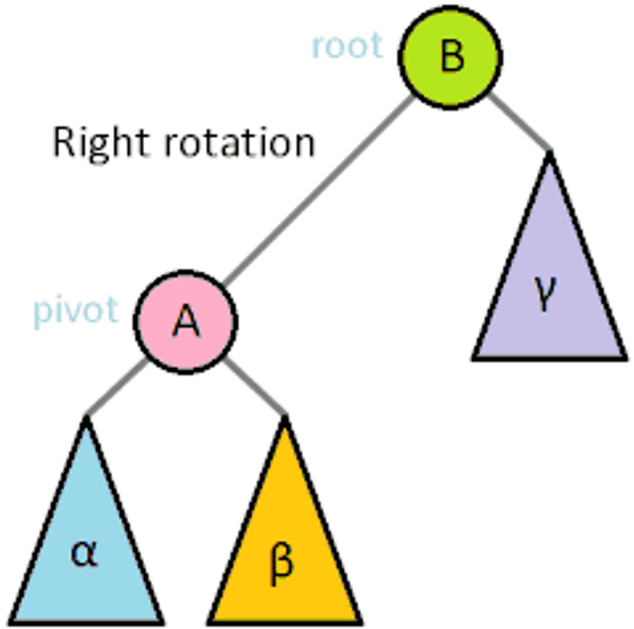
AVL Denge Faktörü

Ekleme veya silme esnasında, herhangi bir dügümün denge faktörü -2 veya +2 olursa dengeleme islemi yapılır



AVL Denge Faktörü

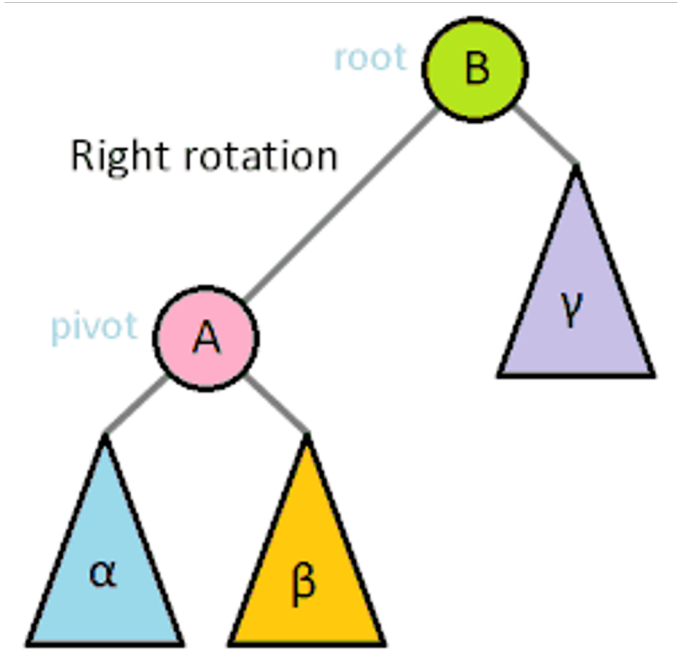
AVL agacı bazı dügümlerin saga veya sola döndürülmesiyle dengeli hale getirilebilir.



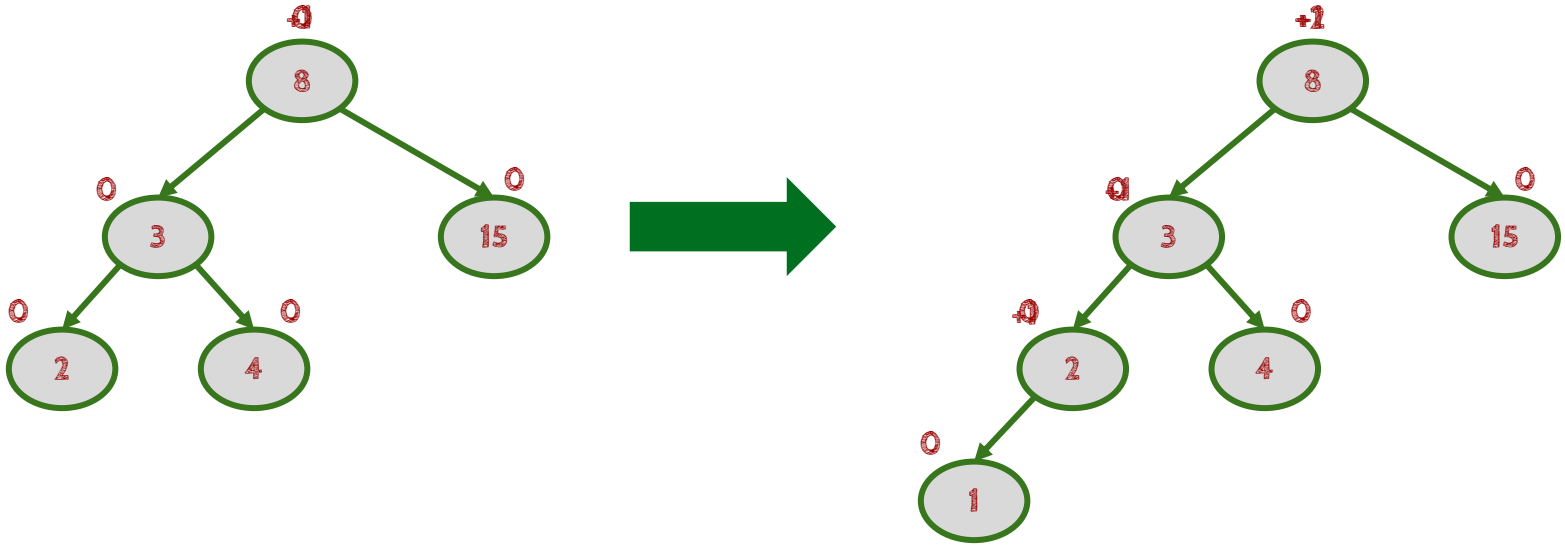
AVL Denge Faktörü

Dengesiz agacı dengeleme isleminde dört durum vardır:

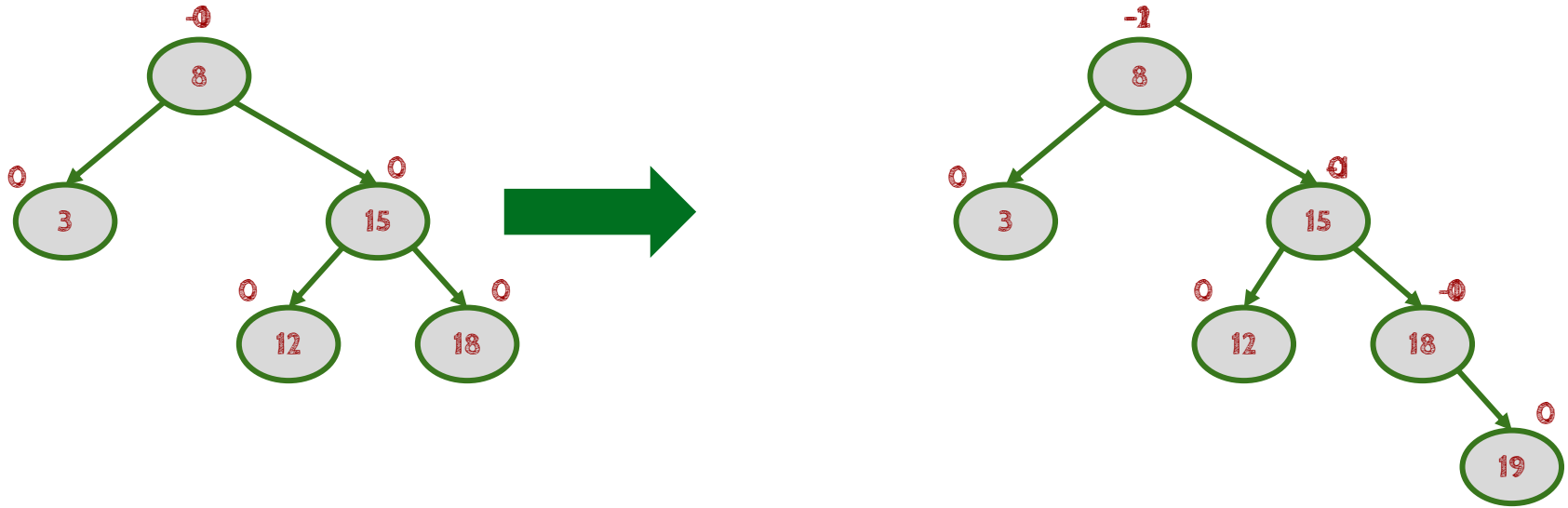
- solun solu
- sagın sagı
- solun sagı
- sagın solu



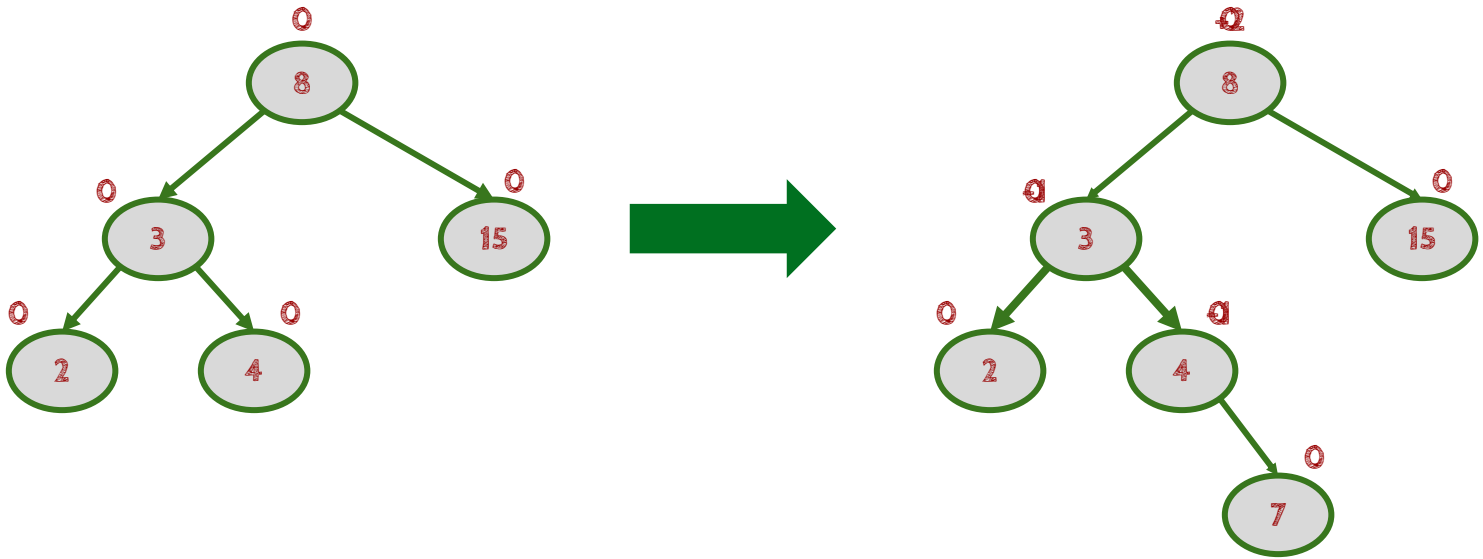
Dengeleme islemi : solun solu



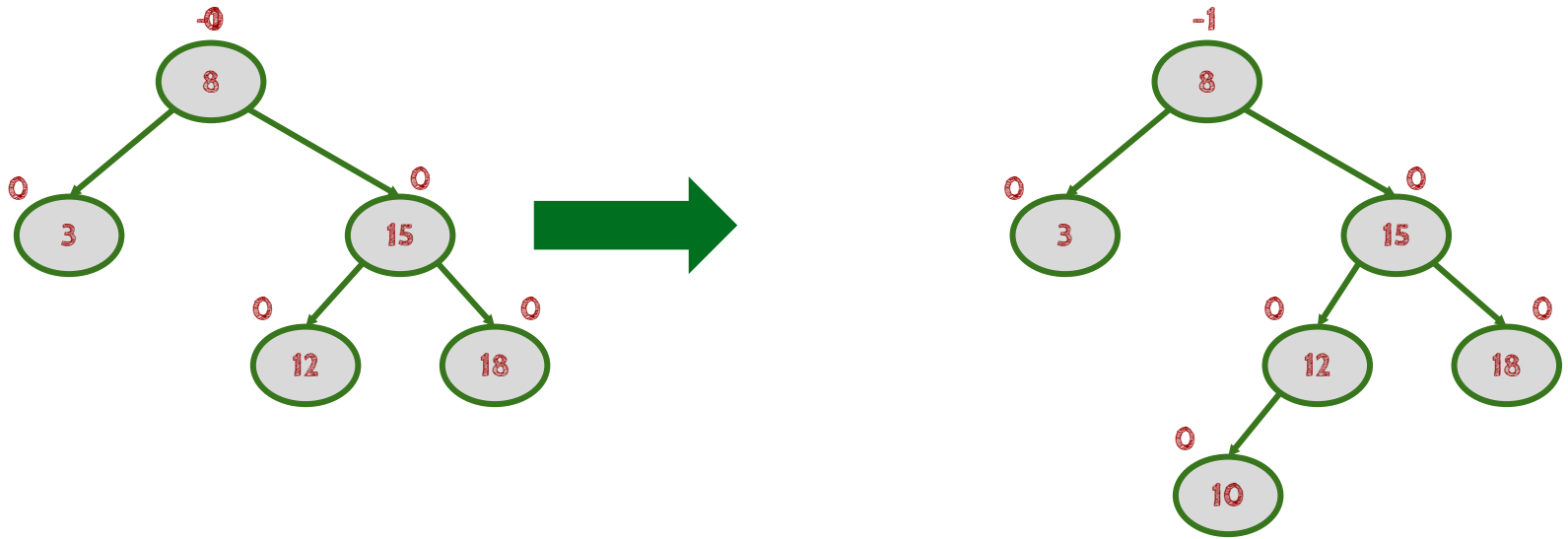
Dengeleme islemi : sagın sagı



Dengeleme islemi : solun sagı

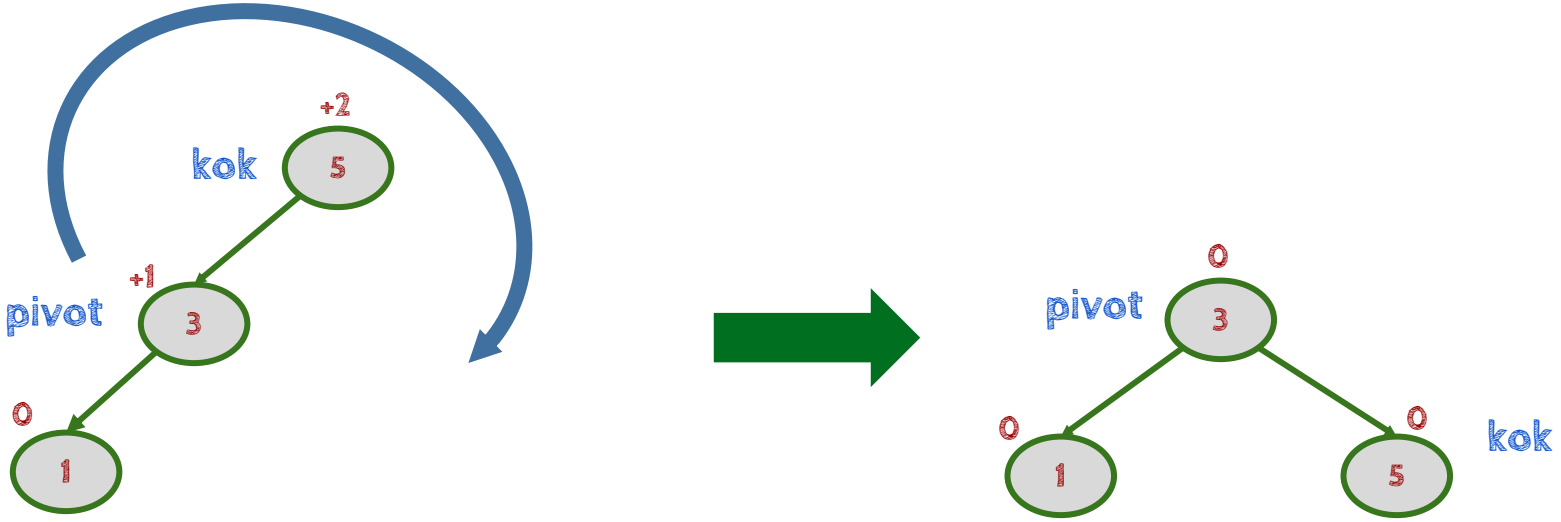


Dengeleme islemi : sagın solu

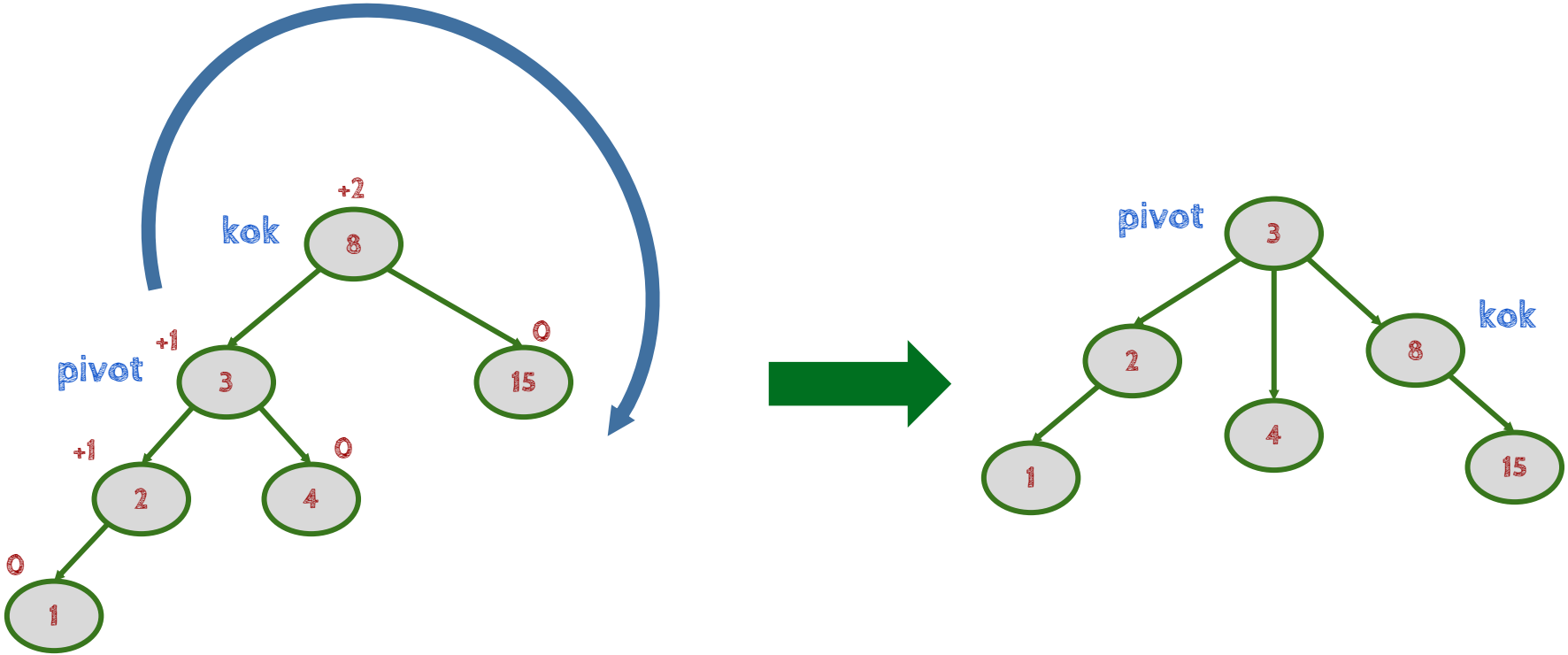


Döndürme : solun solu

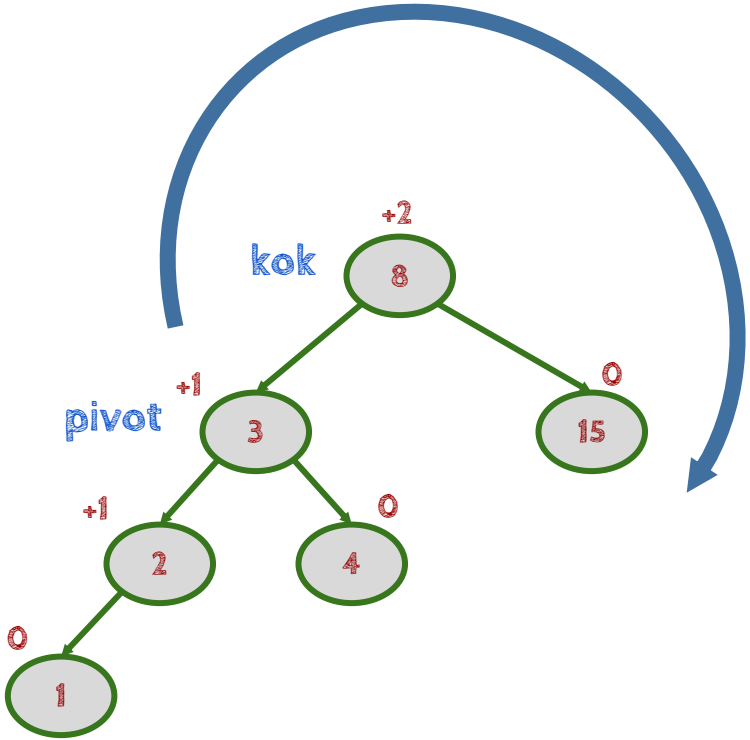
saga döndürme

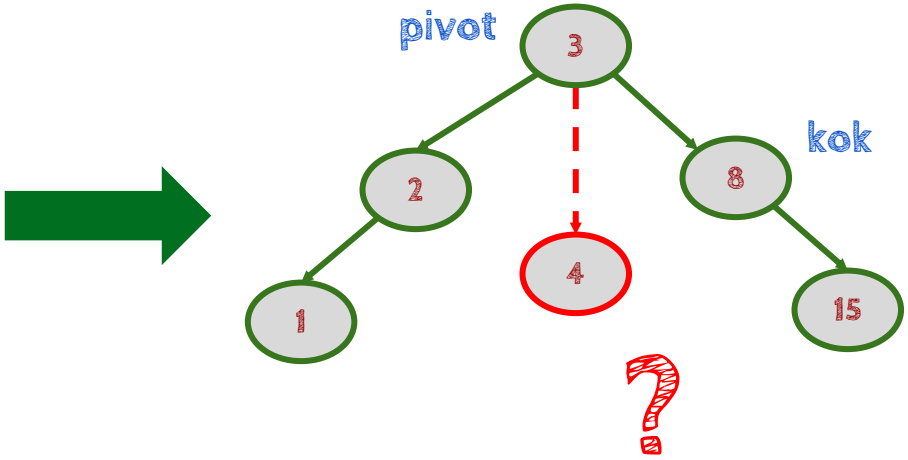


Döndürme : solun solu

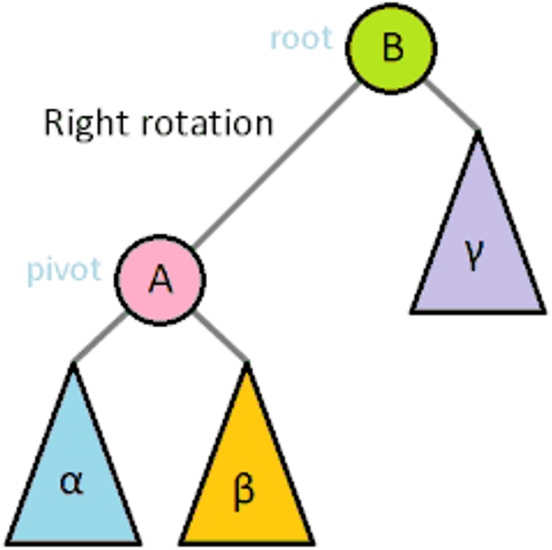


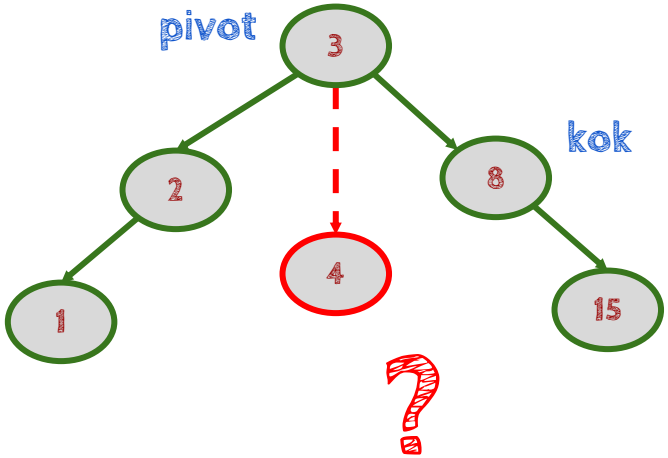
Dengeleme : solun solu



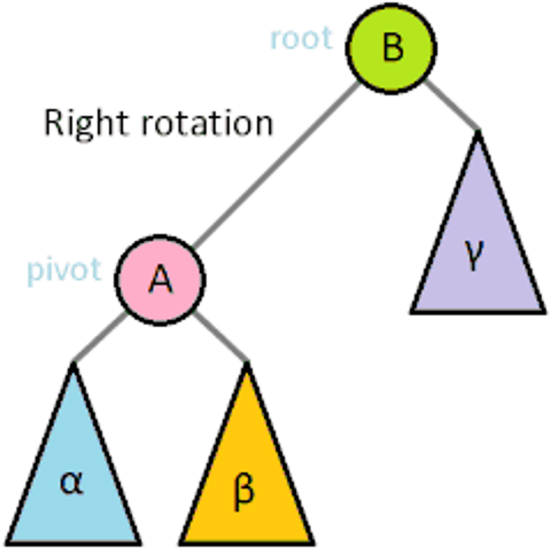


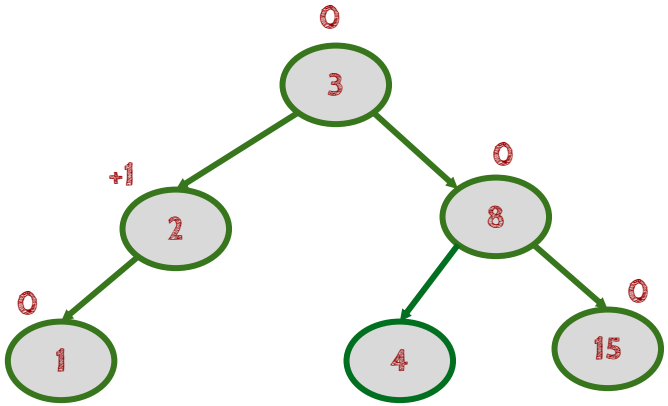
Dengeleme : solun solu





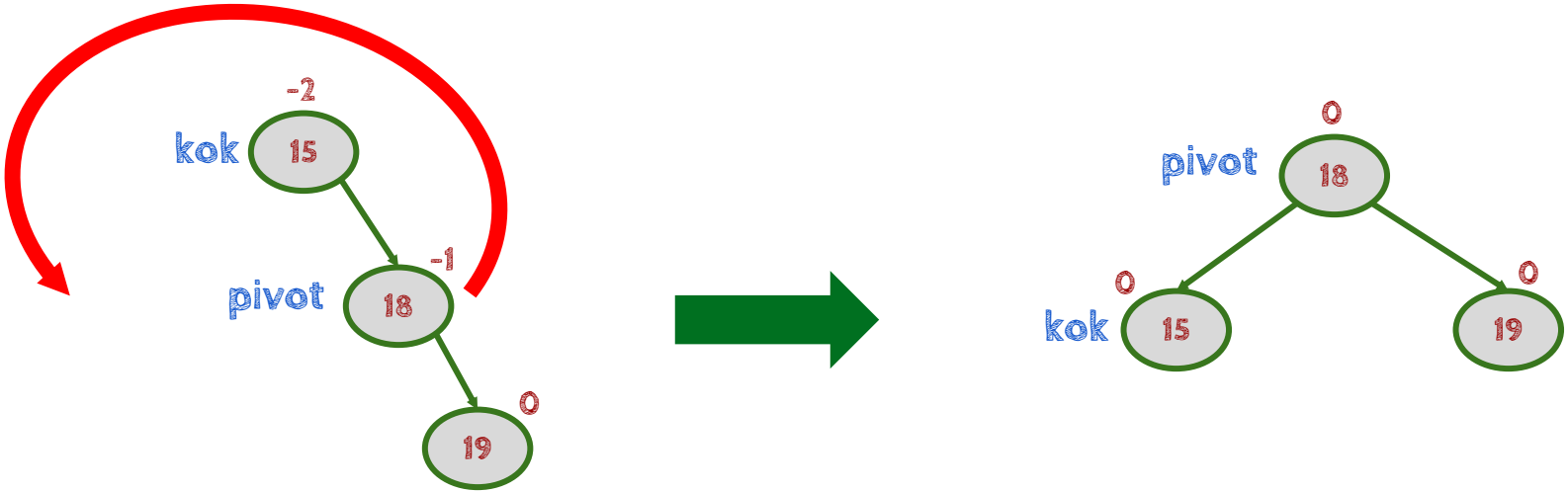
Dengeleme : solun solu



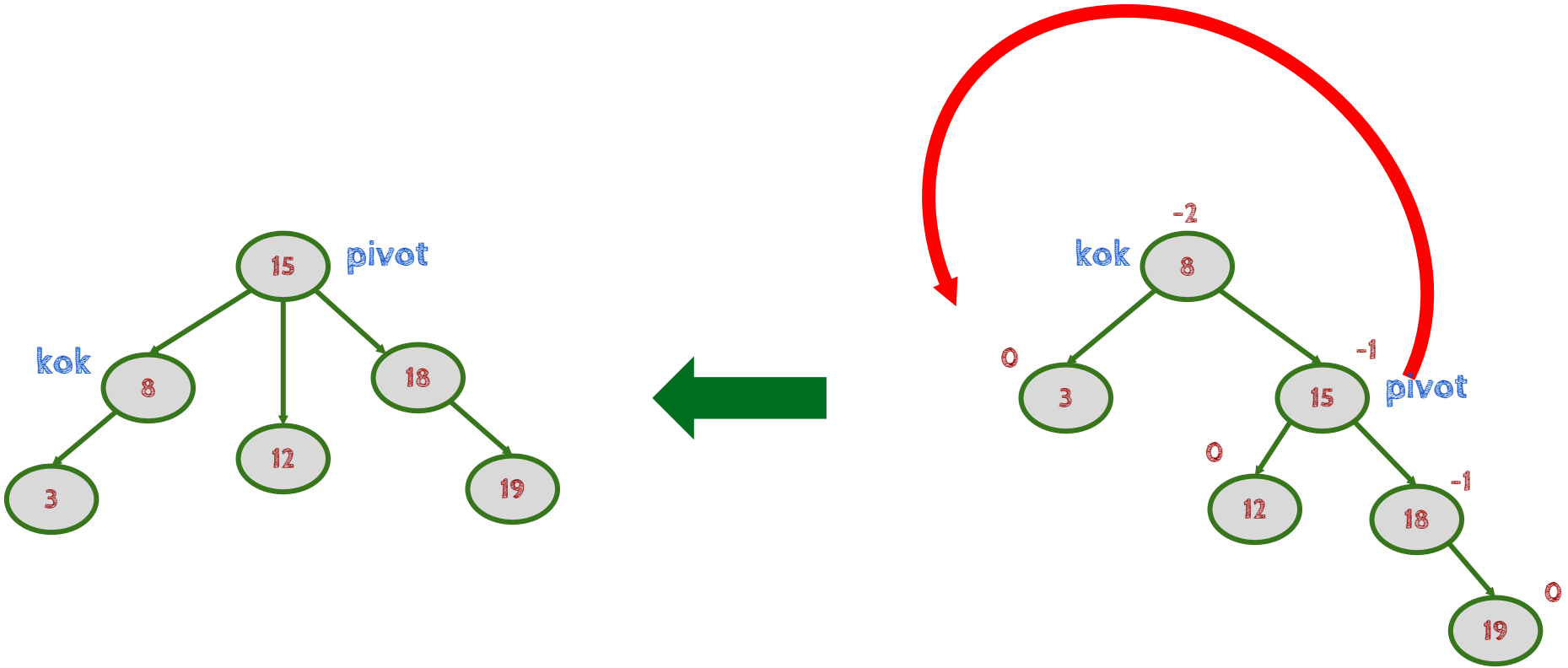


Döndürme : sağın sağı

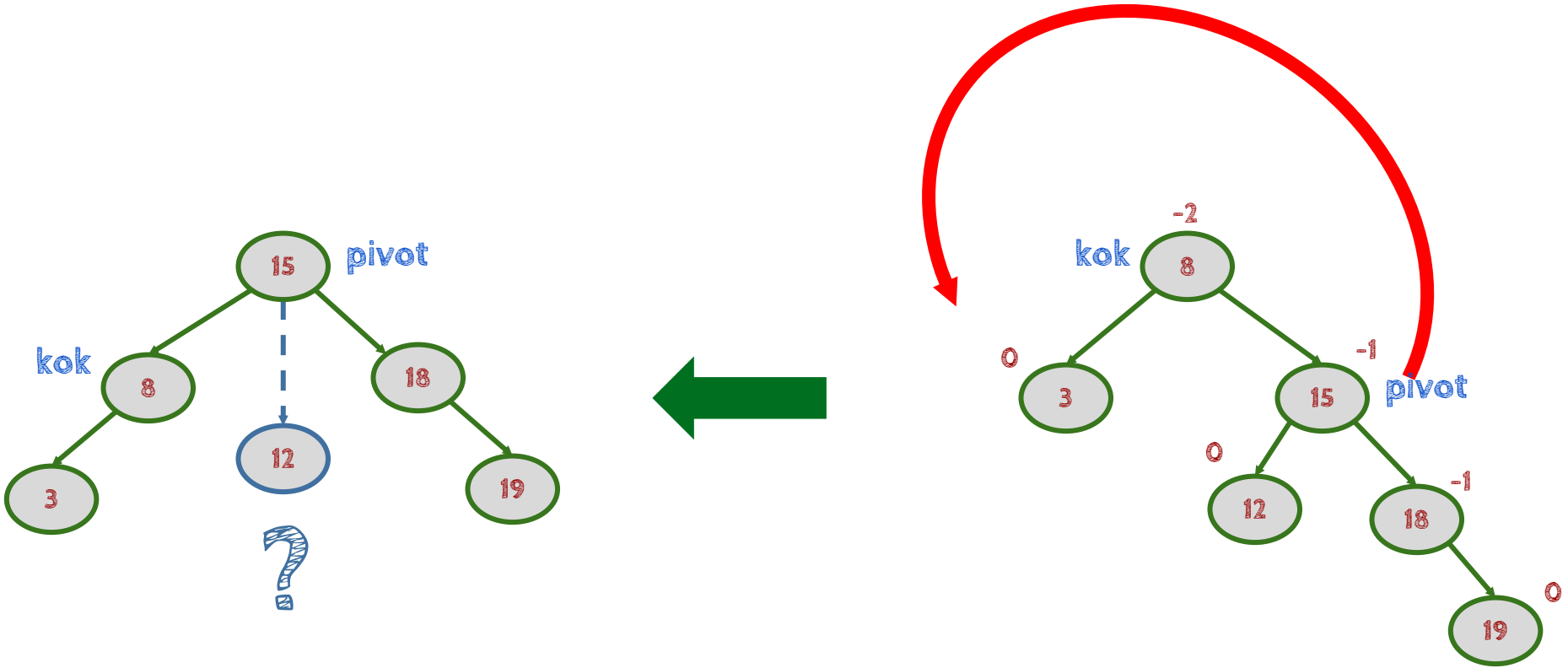
sola döndürme



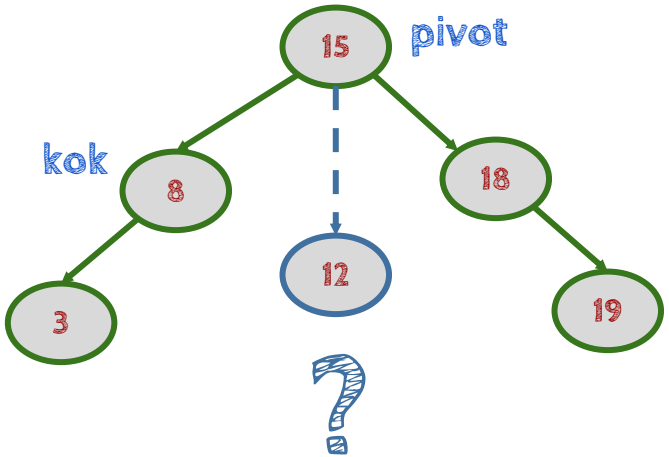
Döndürme : sağın sağı

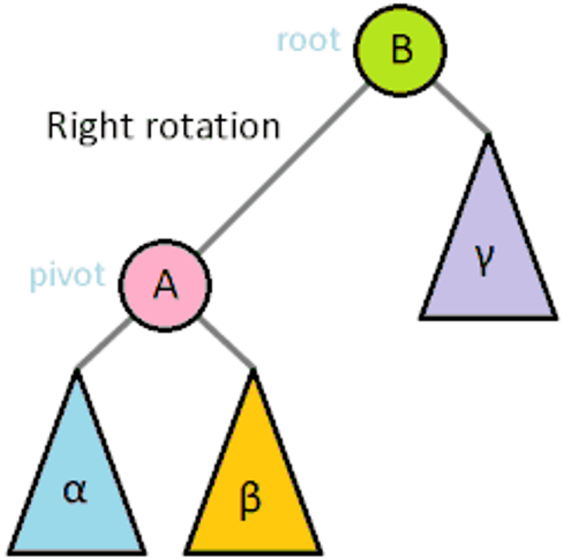


Dengeleme : sağın sağı

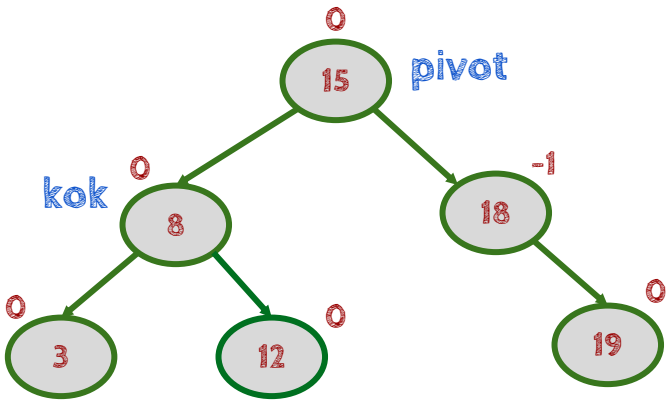


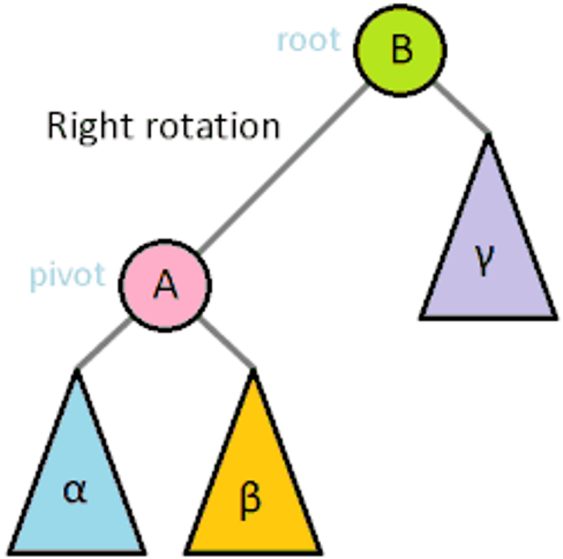
Dengeleme : sağın sağı





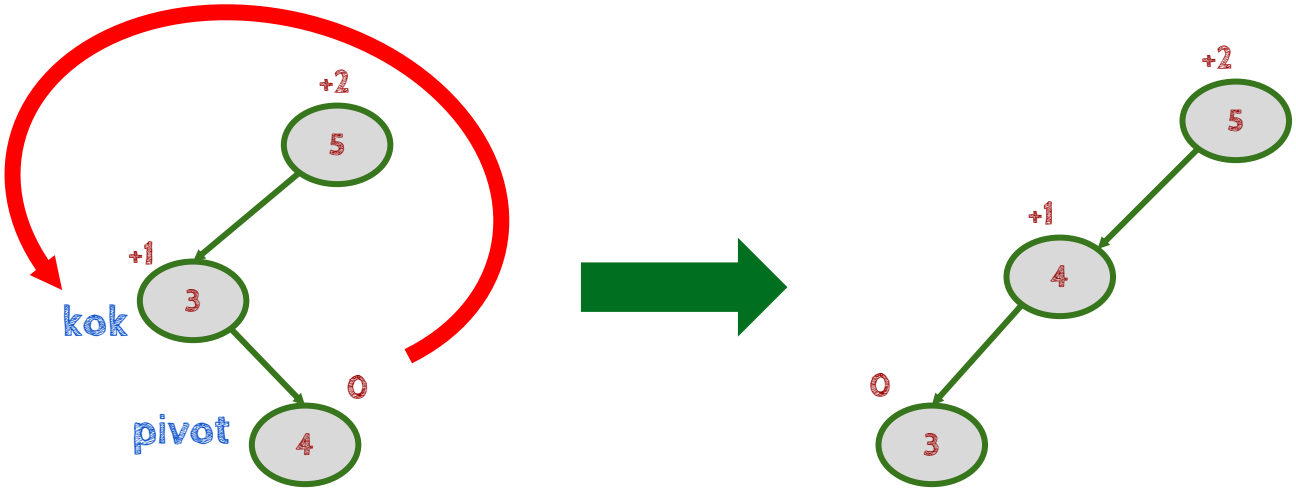
Dengeleme : sağın sağı





Döndürme : solun sağı

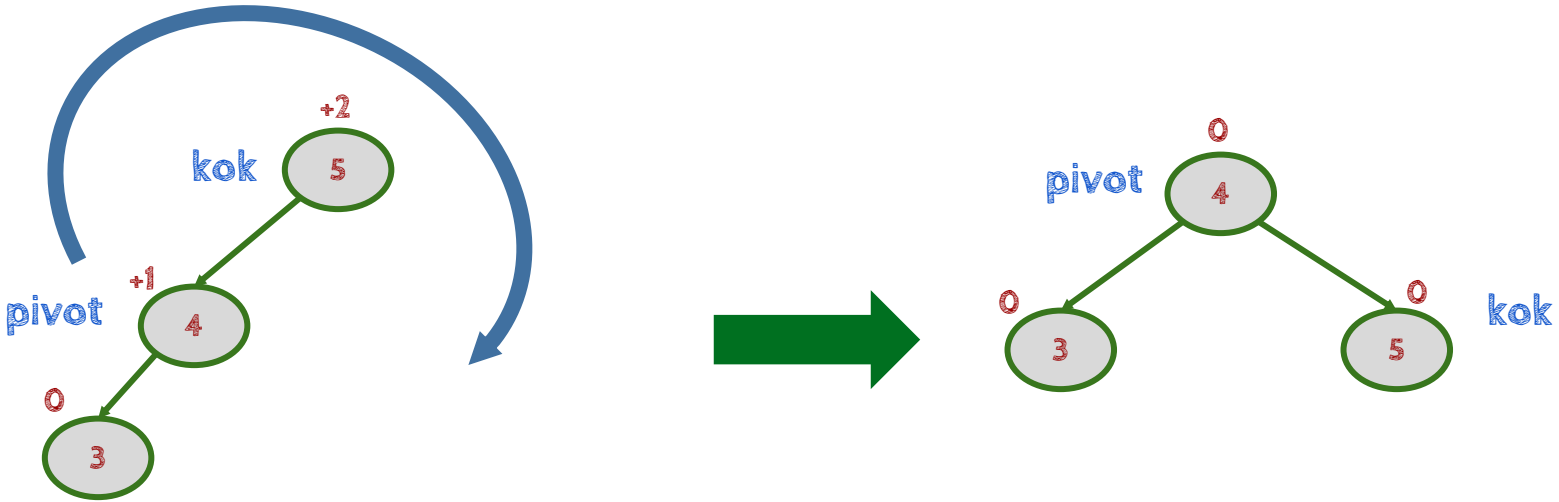
1. Adım: sola döndürme



Döndürme : solun sağı

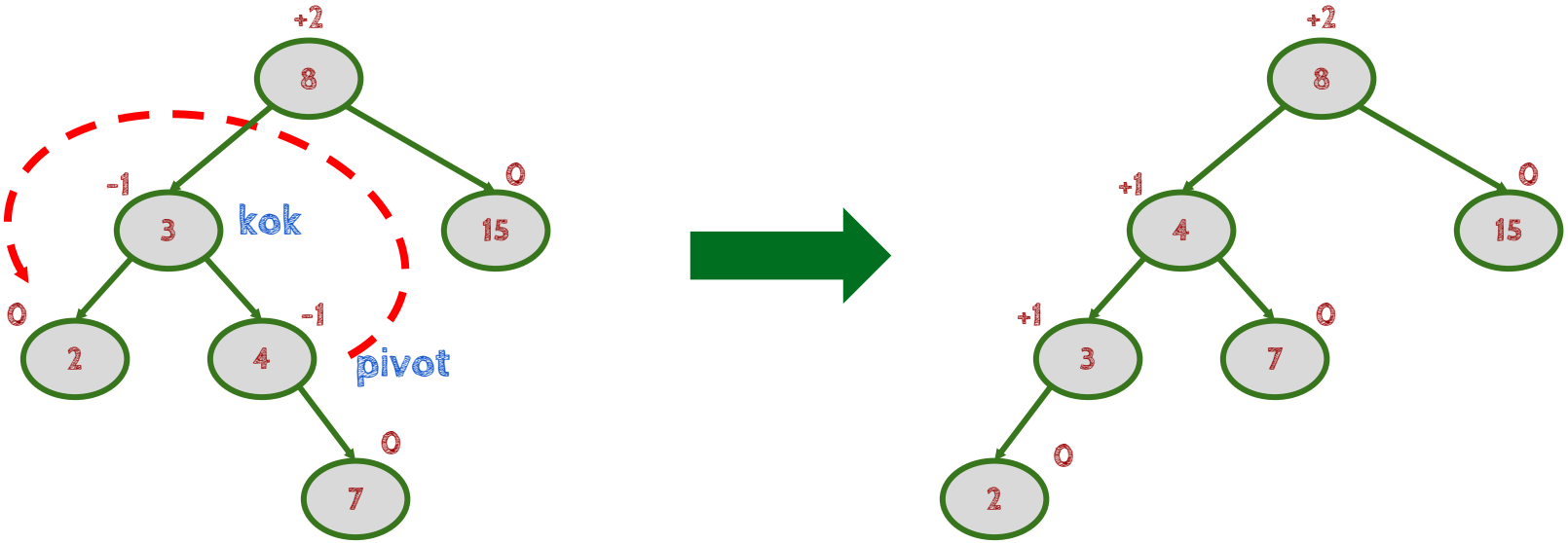
2. Adım: solun solu problemi

saga döndürme



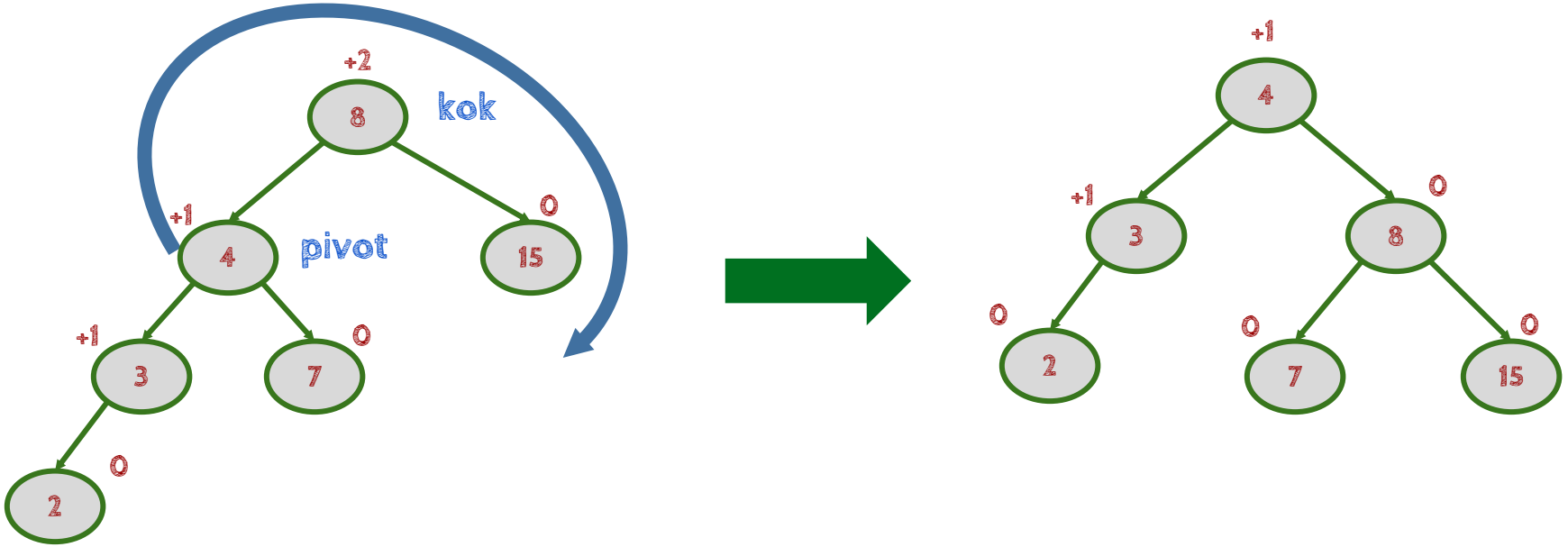
Dengeleme : solun sağı

1. Adım: sola döndürme



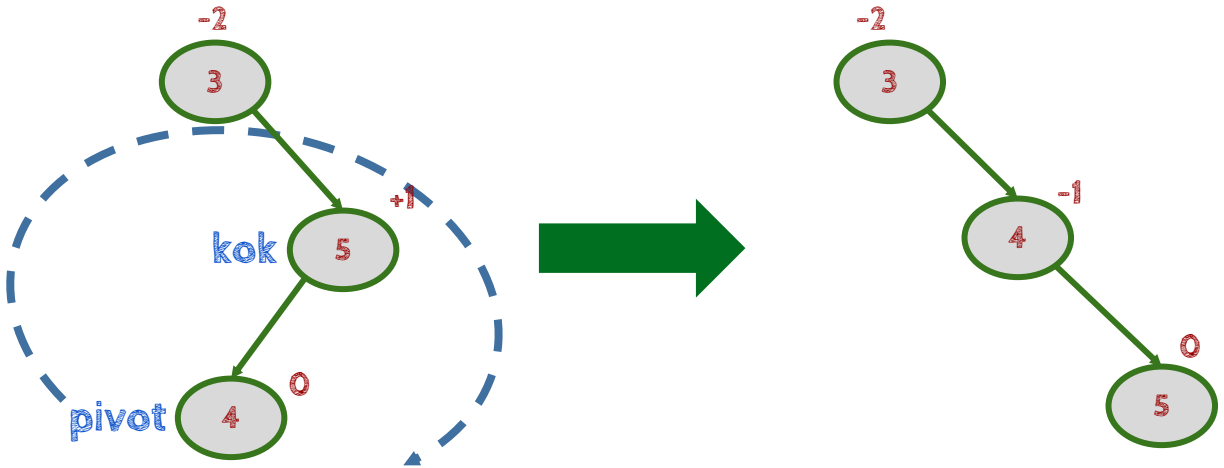
Dengeleme : solun sağı

2. Adım: saga döndürme



Döndürme : sağın solu

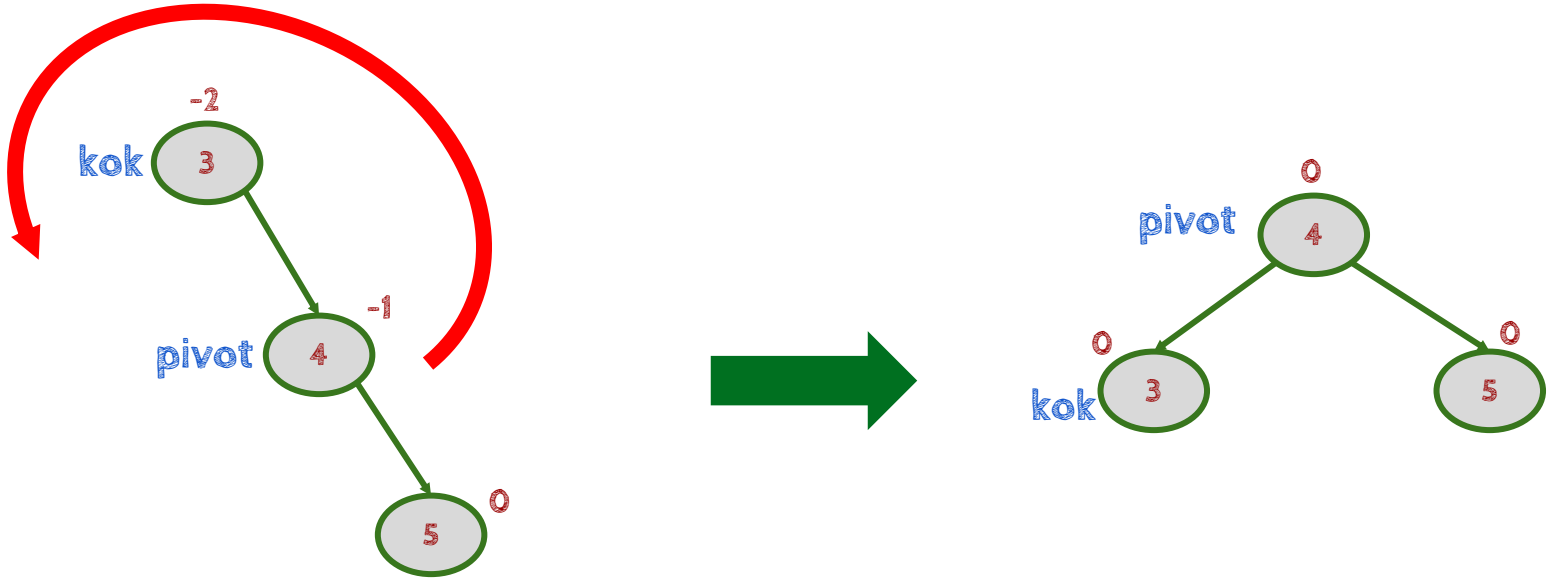
1. Adım: saga döndürme



Döndürme : sağın solu

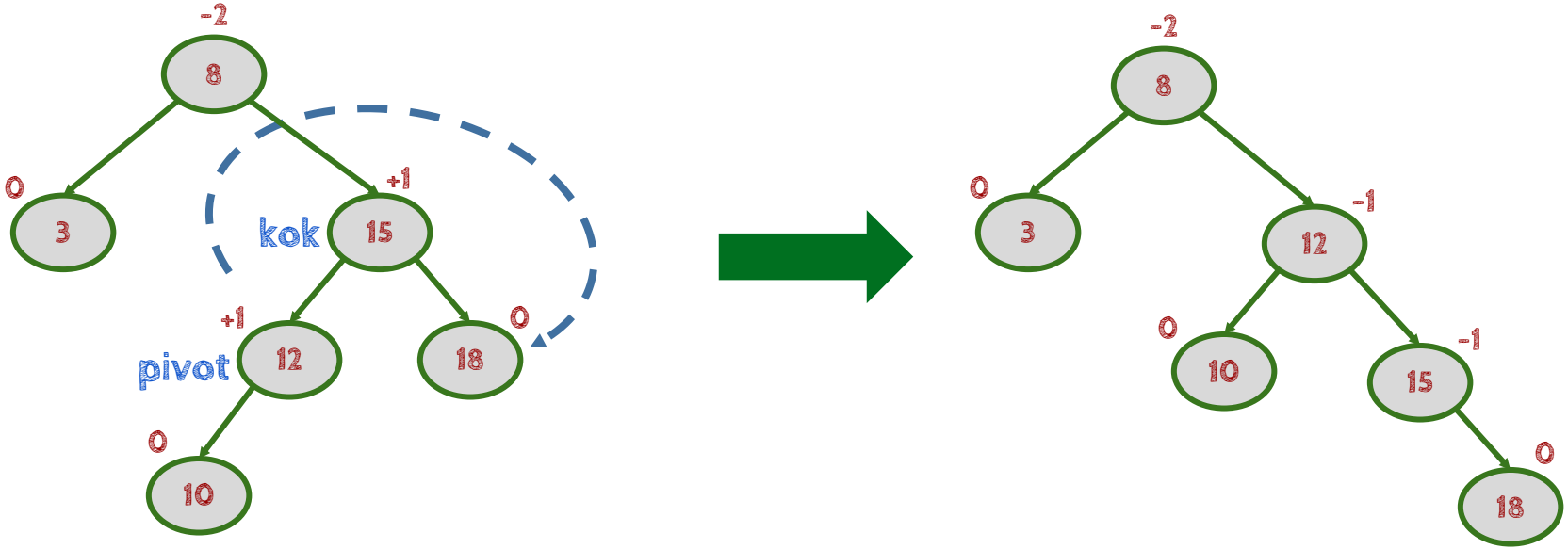
2. Adım: sağın sağı problemi

sola döndürme



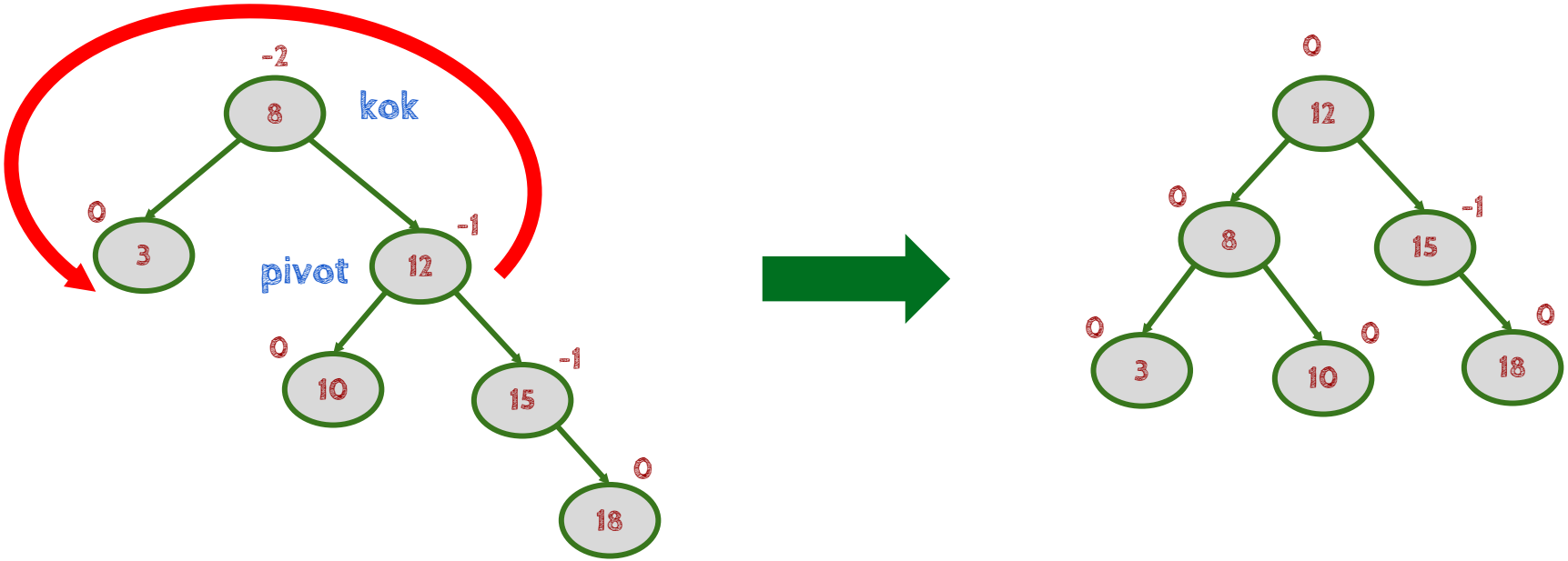
Döndürme : sağın solu

1. Adım: saga döndürme



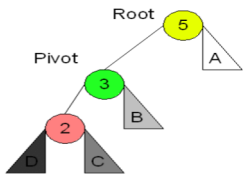
Döndürme : sağın solu

2. Adım: sola döndürme

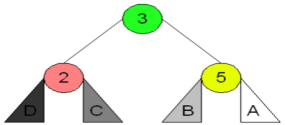


AVL Ağacı Döndürme

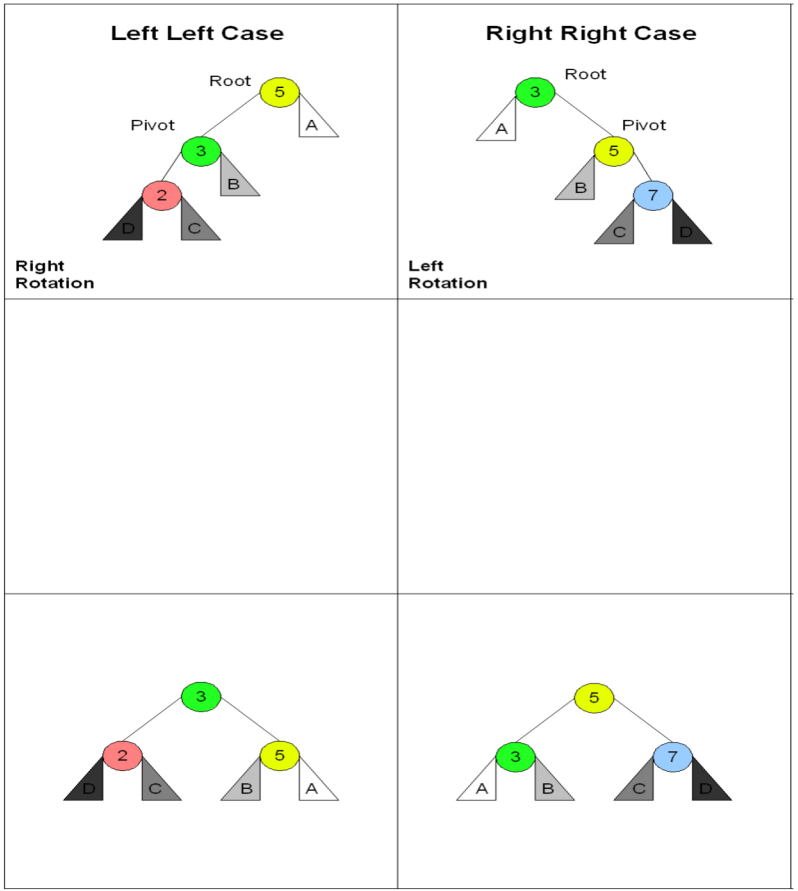
Left Left Case



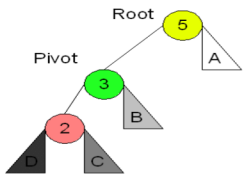
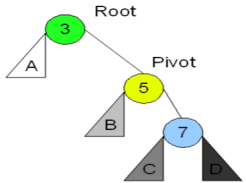
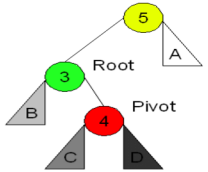
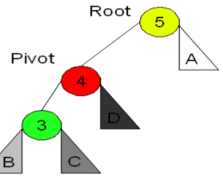
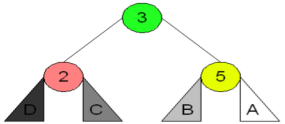
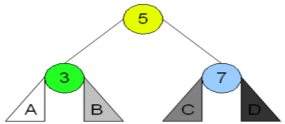
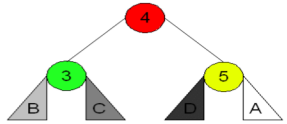
Right Rotation



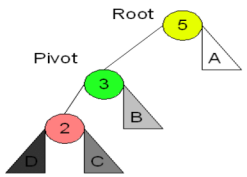
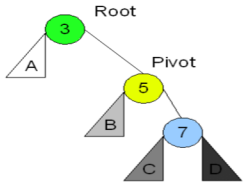
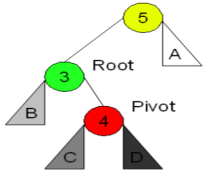
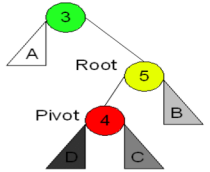
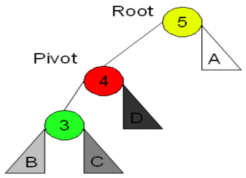
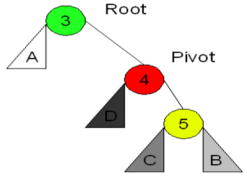
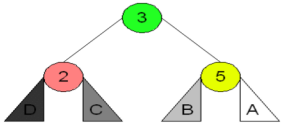
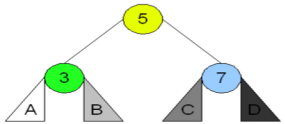
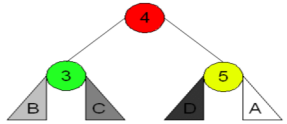
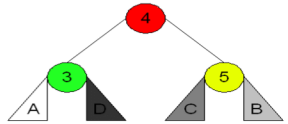
AVL Agacı Döndürme



AVL Ağacı Döndürme

Left Left Case	Right Right Case	Left Right Case
 Right Rotation	 Left Rotation	 Left Rotation
		 Right Rotation
		

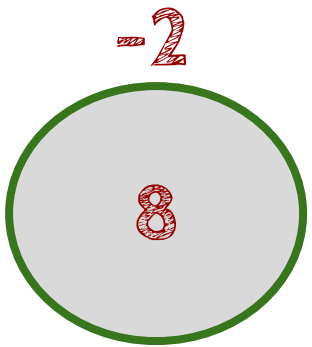
AVL Ağacı Döndürme

Left Left Case	Right Right Case	Left Right Case	Right Left Case
			
Right Rotation	**Left Rotation**	**Left Rotation**	**Right Rotation**
			
		Right Rotation	**Left Rotation**
			

AVL Ağacı Gerçeklestirim

```
#include<stdio.h>
#include<stdlib.h>

// AVL ağacı için düğüm
struct Node
{
        int key;
        struct Node *left;
        struct Node *right;
        int height;
};
```



AVL Ağacı Gerçeklestirim

```
// Ağaca ait yüksekliği dönen fonksiyon
int height(struct Node *N)
{
        if (N == NULL)
                return 0;
        return N->height;
}

int max(int a, int b)
{
        return (a > b)? a : b;
}
```

AVL Ağacı Gerçeklestirim

```
/* Ağaca yeni düğüm ekleme */
struct Node* newNode(int key)
{
        struct Node* node = (struct Node*)malloc(sizeof(struct Node));
        node->key   = key;
        node->left   = NULL;
        node->right  = NULL;
        node->height = 1;  // Yeni düğüm ilk olarak yaprakta eklenir
        return(node);
}
```

AVL Ağacı Gerçeklestirim

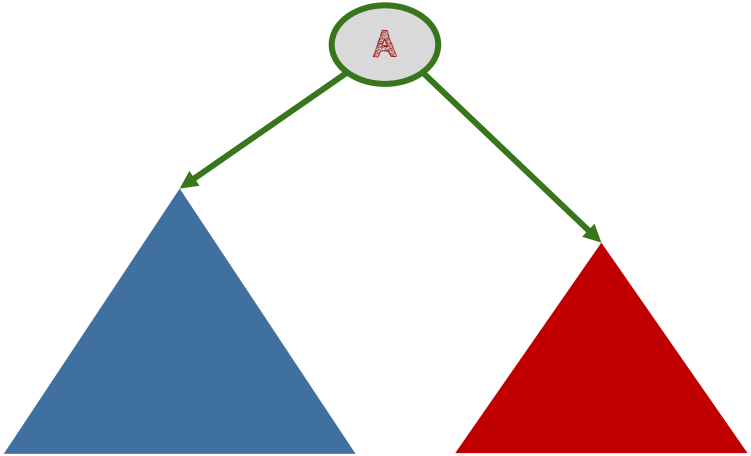
```
/*
 y (sol taraf) veya x (sağ taraf) pivotları ile döndürülecek
 alt ağaçlar T1, T2 ve T3'dür.
              y                                      x
             / \        Sağa  Döndürme            /   \
            x   T3      – – – – – – – >          T1    y
           / \          < - - - - - - -              / \
         T1  T2        Sola Döndürme              T2  T3
*/
struct Node *rightRotate(struct Node *y)
{
                struct Node *x = y->left;
                struct Node *T2 = x->right;
                // Döndürme
                x->right = y;
                y->left = T2;
                // Yükseklikler güncelleniyor
                y->height = max(height(y->left), height(y->right))+1;
                x->height = max(height(x->left), height(x->right))+1;
                // yeni kok
                return x;
}
```

AVL Ağacı Gerçeklestirim

```
/*
 y (sol taraf) veya x (sağ taraf) pivotları ile döndürülecek
 alt ağaçlar T1, T2 ve T3'dür.
              y                                                  x
             / \          Sağa  Döndürme          /   \
            x   T3        – – – – – – – >         T1     y
           / \            < - - - - - - -                / \
         T1  T2          Sola Döndürme                 T2 T3
*/
struct Node *leftRotate(struct Node *x)
{
                    struct Node *y = x->right;
                    struct Node *T2 = y->left;
                    // döndürme
                    y->left = x;
                    x->right = T2;
                    //  Yükseklikler güncelleniyor
                    x->height = max(height(x->left), height(x->right))+1;
                    y->height = max(height(y->left), height(y->right))+1;
                    // yeni kok
                    return y;
}
```

AVL Agacı Gerçeklestirim

```
// N. düğüm için denge faktörü
int getBalance(struct Node *N)
{
        if (N == NULL)
                return 0;
        return height(N->left) - height(N->right);
}
```



AVL Ağacı Gerçeklestirim

```
struct Node* insert(struct Node* node, int key){
                /* BST ağacına ekleme */
                if (node == NULL)  return(newNode(key));
                if (key < node->key)  node->left  = insert(node->left, key);
                else if (key > node->key)  node->right = insert(node->right, key);
                else  return node;
                /* 2. Yükseklikler güncelleniyor */
                node->height = 1 + max(height(node->left), height(node->right));
                /* 3. Yeni ekleme işlemi ile denge hesaplanıyor */
                int balance = getBalance(node);
                // Eklenen düğüm dengesiz ise 4 durum vardır
                // Solun solu
                if (balance > 1 && key < node->left->key)          return rightRotate(node);
                // Sağın sağı
                if (balance < -1 && key > node->right->key)  return leftRotate(node);
                // Solun sağı
                if (balance > 1 && key > node->left->key){
                                node->left =  leftRotate(node->left);  return rightRotate(node);
                }
                // Sağın solu
                if (balance < -1 && key < node->right->key) {
                                node->right = rightRotate(node->right);   return leftRotate(node);
                }
                return node;
}
```

AVL Agacı Gerçeklestirim

```
// Ağaç içinde dolaşma
void preOrder(struct Node *root)
{
        if(root != NULL){
                printf("%d ", root->key);
                preOrder(root->left);
                preOrder(root->right);
        }
}
```

Köke ugra
Sol alt agacı preorder olarak dolas
Sag alt agacı preorder olarak dolas

AVL Ağacı Gerçeklestirim

```
int main()
{
        struct Node *root = NULL;
        root = insert(root, 10);
        root = insert(root, 20);
        root = insert(root, 30);
        root = insert(root, 40);
        root = insert(root, 50);
        root = insert(root, 25);
        /* AVL ağacı
          30
                        /   \
                  20   40
                 /  \    \
            10  25   50
        */
        printf("AVL ağacında Preorder dolaşma:\n");
        preOrder(root);
        return 0;
}
```

Köke ugra
Sol alt agacı preorder olarak dolas
Sag alt agacı preorder olarak dolas

30 20 10 25
40 50

Sorular