BMB2006
VERİ YAPILARI

Doç. Dr. Murtaza CİCİOĞLU

Bursa Uludağ Üniversitesi

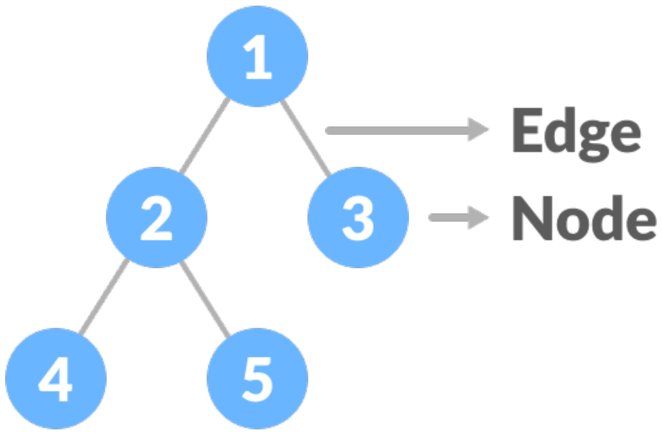
Bilgisayar Mühendisliği Bölümü

Bu ders notları Olcay Taner Yıldız ile Erkan TANYILDIZI hocaların notları kullanılarak hazırlanmıştır.

Hafta 8: Ağaçlar (Trees)

Amaç:

- Ağaç çalışma yapısı
- Ağaç kullanım alanları



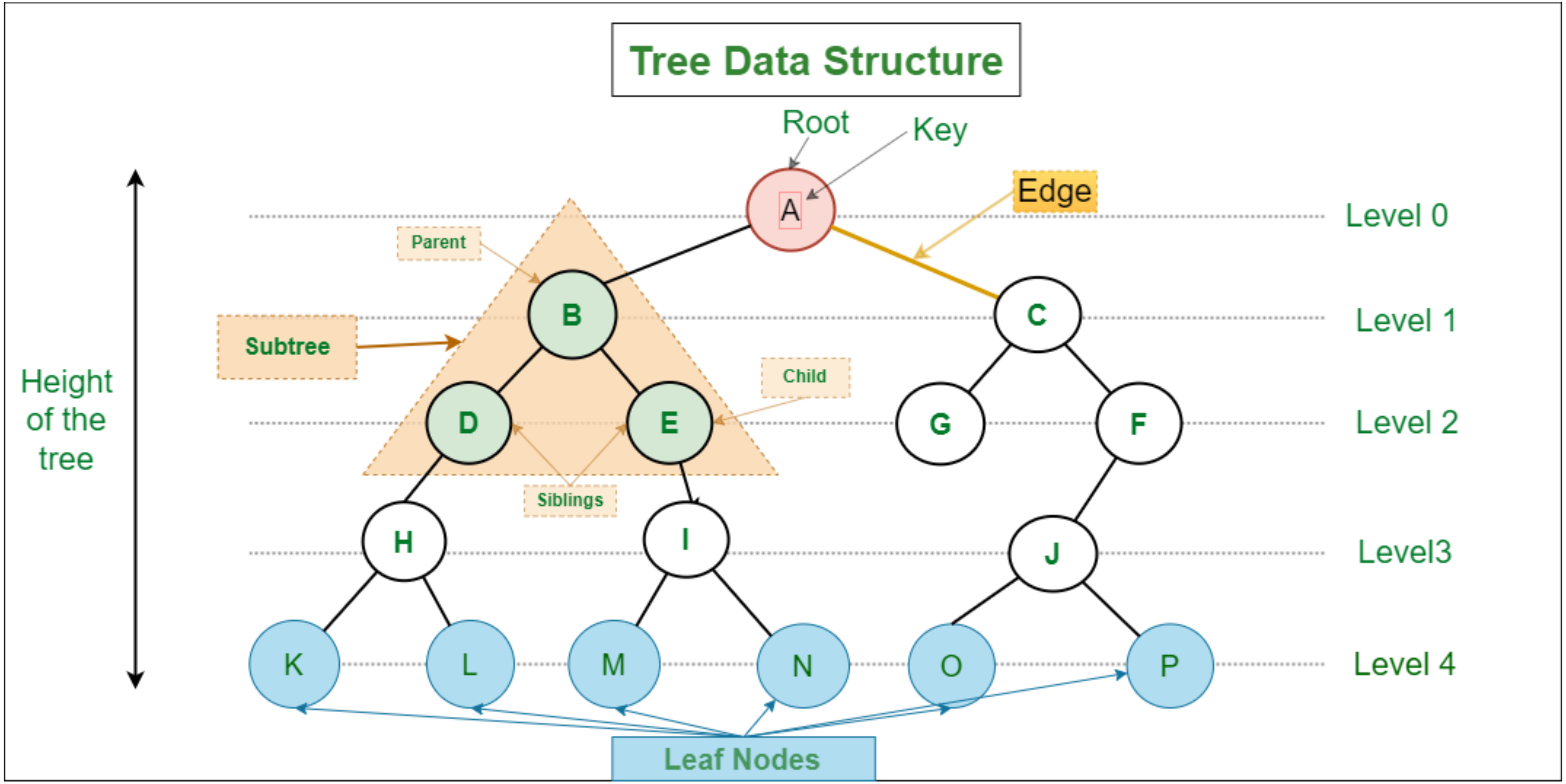
Yol haritası:

- Giriş
- İkili Arama Ağaç Tanımı
- Temel İkili Arama Ağacı İşlemleri
- Gezintiler
- AVL Ağacı
- B+ Ağacı

Giriş

- Birbirine hiyerarjik yapıda bağlı ve çevrim oluşturmayan sonlu düğümler kümesidir.
- (DAG) yönlü çevrimsiz graf, rekürsif yapıya uygun, O(logN)
- Ağaç veri yapısı doğrusal olmayan veri yapılarındandır.
- Ağaç, bir kök işaretçisi, sonlu sayıda düğümleri ve onları birbirine bağlayan dalları olan bir veri modelidir.
- Ağaçlar hiyerarşik ilişkileri göstermek için kullanılır.
- Her biri değişik bir uygulamaya doğal çözüm olan *ikili arama ağacı, kodlama ağacı, sözlük ağacı, kümeleme ağacı* gibi çeşitli ağaç şekilleri vardır; üstelik uygulamaya yönelik özel ağaç şekilleri de çıkarılabilir.

Giriş



Tree Data Structure
Root
Key
Edge
A
Parent
B
C
Level 0
Level 1
Subtree
Child
Height
of the
tree
D
E
G
F
Level 2
Siblings
H
I
J
Level3
K
L
M
N
O
P
Level 4
Leaf Nodes

Giriş

- **Düğüm (Node)** : Ağacın her bir elemanına düğüm adı verilir.
- **Kök Düğüm (Root)**: Ağacın başlangıç düğümüdür.
- **Çocuk (Child)**: Bir düğüme doğrudan bağlı olan düğümlere çocuklar denilir.
- **Kardeş Düğüm (Sibling)** : Aynı düğüme bağlı düğümlere kardeş düğüm veya kısaca kardeş denir.
- **Aile (Parent)** : Düğümlerin doğrudan bağlı oldukları düğüm aile olarak adlandırılır; diğer bir deyişle aile, kardeşlerin bağlı olduğu düğümdür.
- **Ata (Ancestor) ve Torun (Dedscendant)** : Aile düğümünün daha üstünde kalan düğümlere ata denilir; torun, bir düğümün çocuğuna bağlı olan düğümlere denir.

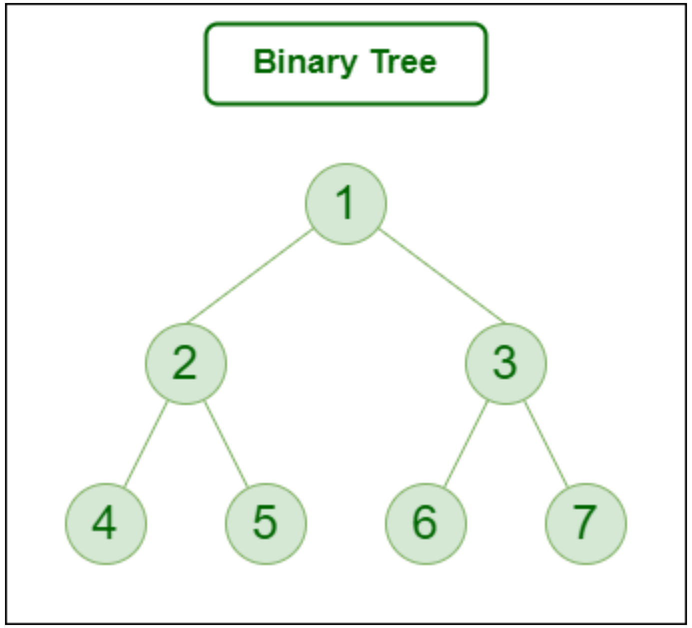
Giriş

- **Derece (Degree)** : Bir düğümden alt hiyerarşiye yapılan bağlantıların sayısıdır; yani çocuk veya alt ağaç sayısıdır.
- **Düzey (Level) ve Derinlik (Depth)** : Düzey, iki düğüm arasındaki yolun üzerinde bulunan düğümlerin sayısıdır. Kök düğümün düzeyi 1, doğrudan köke bağlı düğümlerin düzeyi 2'dir. Bir düğümün köke olan uzaklığı ise derinliktir. Kök düğümün derinliği 1 dir.
- **Yaprak (Leaf)** : Ağacın en altında bulunan ve çocukları olmayan düğümlerdir.
- **Yükseklik (Height)** : Bir düğümün kendi silsilesinden en uzak yaprak düğüme olan uzaklığıdır.
- **Yol(Path)** : Bir düğümün aşağıya doğru (çocukları üzerinden) bir başka düğüme gidebilmek için üzerinden geçilmesi gereken düğümlerin listesidir.

Giriş

- Ağaç yapısını gerçekleştirmek için 2 yol vardır.
  - İndis yöntemi
  - Düğüm bağlantısı

- Ağaç İşlemlerinden bazıları;
  - add()
  - remove()
  - inorder()
  - preorder()
  - postorder()
  - isEmpty()



Kullanım alanları

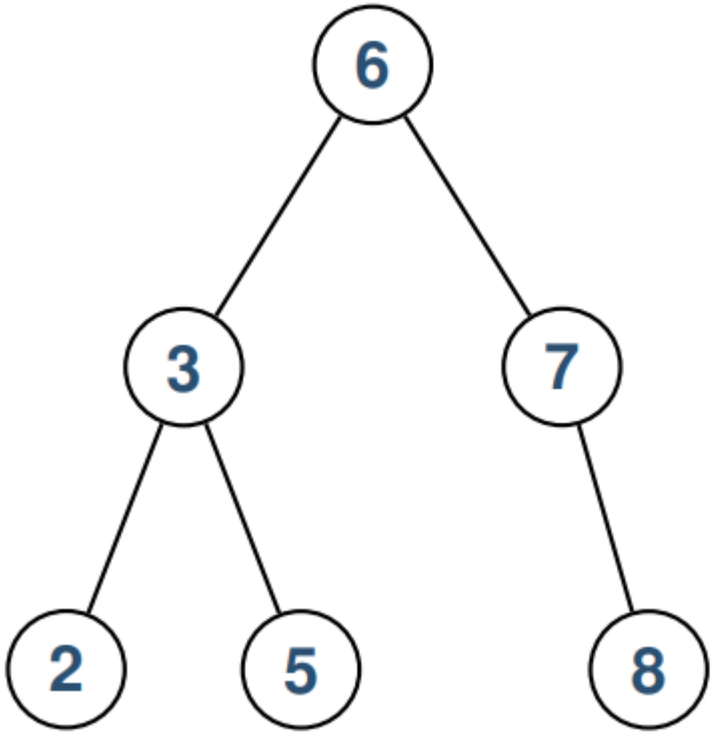
- Organizasyon şeması,
- Dosya sistemleri,
- Programlama ortamları
- Verimli arama, ekleme ve silme işlemleri
- DNS
- Makine öğrenmesi (Decision Trees)
- XML parser
- Veritabanı indeksleme
- ……

İkili Arama Ağacı

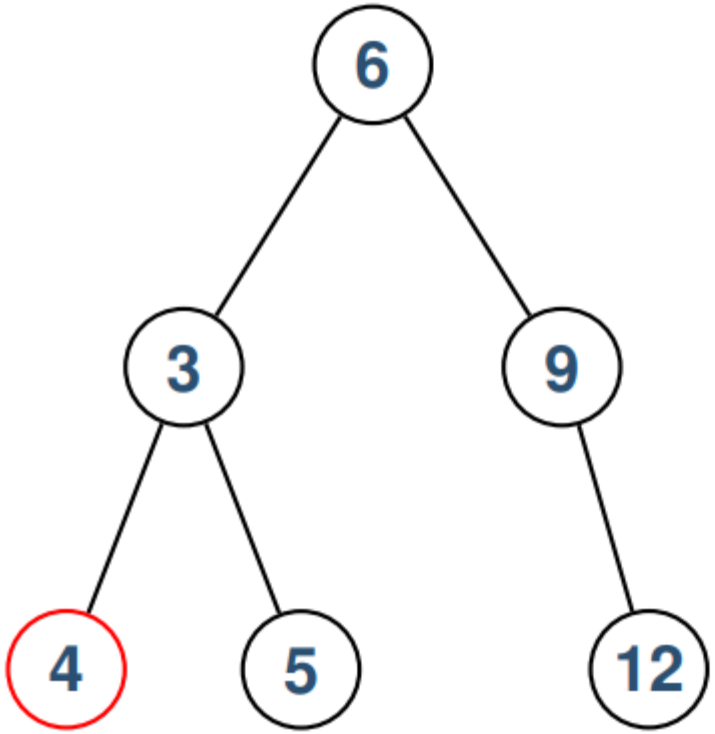
- En çok bilinen ağaç türleri ikili arama ağacı olup kodlama ağacı, sözlük ağacı, kümeleme ağacı gibi birçok ağaç uygulaması vardır.
- İkili arama ağacında bir düğüm en fazla iki tane çocuğa sahip olabilir ve alt/çocuk bağlantıları belirli bir sırada yapılır.
- Sonlu düğümler kümesidir. Bu küme boş bir küme olabilir (empty tree). Boş değilse şu kurallara uyar.
  - Kök olarak adlandırılan özel bir düğüm vardır.
  - Her düğüm en fazla iki düğüme bağlıdır.
  - Left child : Bir node'un sol işaretçisine bağlıdır.
  - Right child : Bir node'un sağ işaretçisine bağlıdır.
  - Kök hariç her düğüm bir daldan gelmektedir.
  - Tüm düğümlerden yukarı doğru çıkıldıkça sonuçta köke ulaşılır.
  - Düğümün sol tarafı kökten küçük sağ tarafı kökten büyük

İkili Arama Ağacı

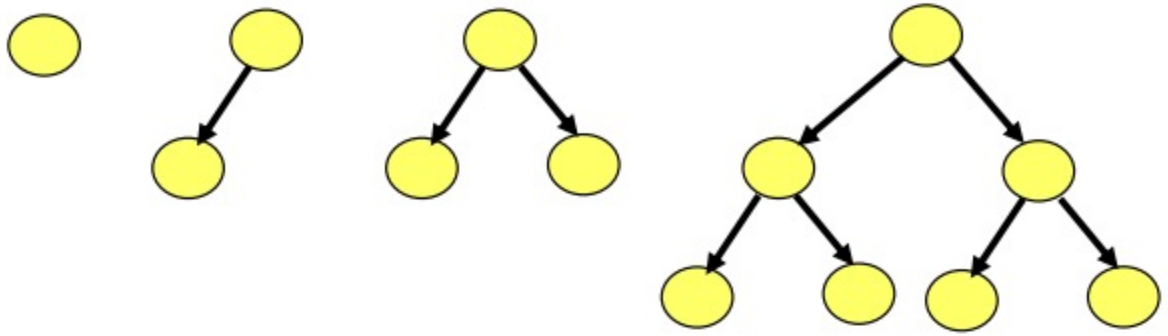
- Maksimum düğüm sayısı: 2^n-1 (n derinlik)



4 nereye eklenmeli?



İkili Arama Ağacı



Derinlik 1: N = 1, 1 düğüm 2^1 -1

Derinlik 2: N = 2, 3 düğüm, 2^2 -1 düğüm

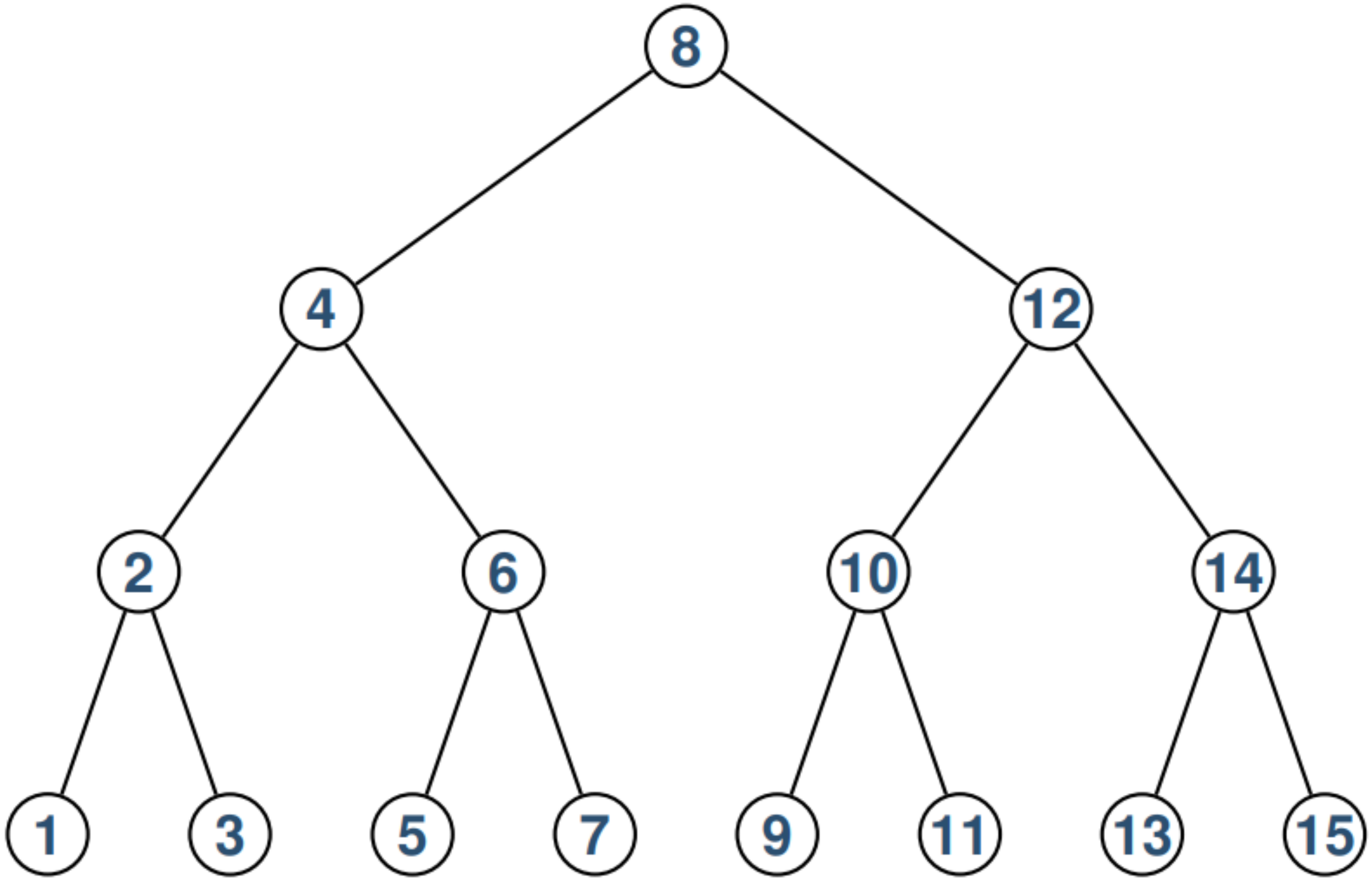
Herhangi bir d derinliğinde, N = ?

Derinlik d: N = 2^d-1 düğüm (tam bir ikili ağaç)

En küçük derinlik: Θ(log N)

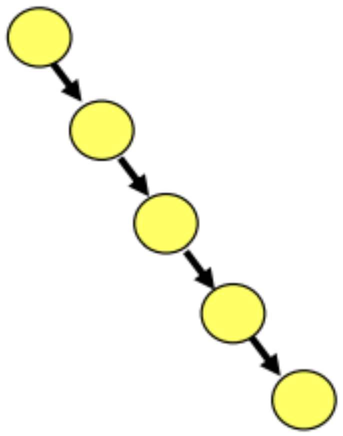
İkili Arama Ağacı

- 15 elemandan oluşan 4 derinliğindeki dengeli bir ikili ağaç



İkili Arama Ağacı

- N düğümlü ikili ağacın minimum derinliği: Θ(log N)
- İkili ağacın maksimum derinliği ne kadardır?
  - Dengesiz ağaç: Ağaç bir bağlantılı liste olursa!
  - Maksimum derinlik = N
  - Amaç: Arama gibi operasyonlarda bağlantılı listeden daha iyi performans sağlamak için derinliğin log N de tutulması gerekmektedir.



- Bağlantılı liste
- Derinlik = N

İkili Arama Ağacı Düğüm Bağlantısı

- Tam sayılar içeren düğüm tanımı

```
struct node{
        int data;
        struct node* left;
        struct node* right;};

typedef struct node Node;
typedef Node* Nodeptr;
```

```
Nodeptr newNode(int data){
        Nodeptr node=(Nodeptr)malloc(sizeof(Node));
        node->data=data;
        node->left=NULL;
        node->right=NULL;
        return node;
}
```

İkili Arama Ağacı Düğüm Bağlantısı

Tam sayılar içeren ikili arama ağacı tanımı

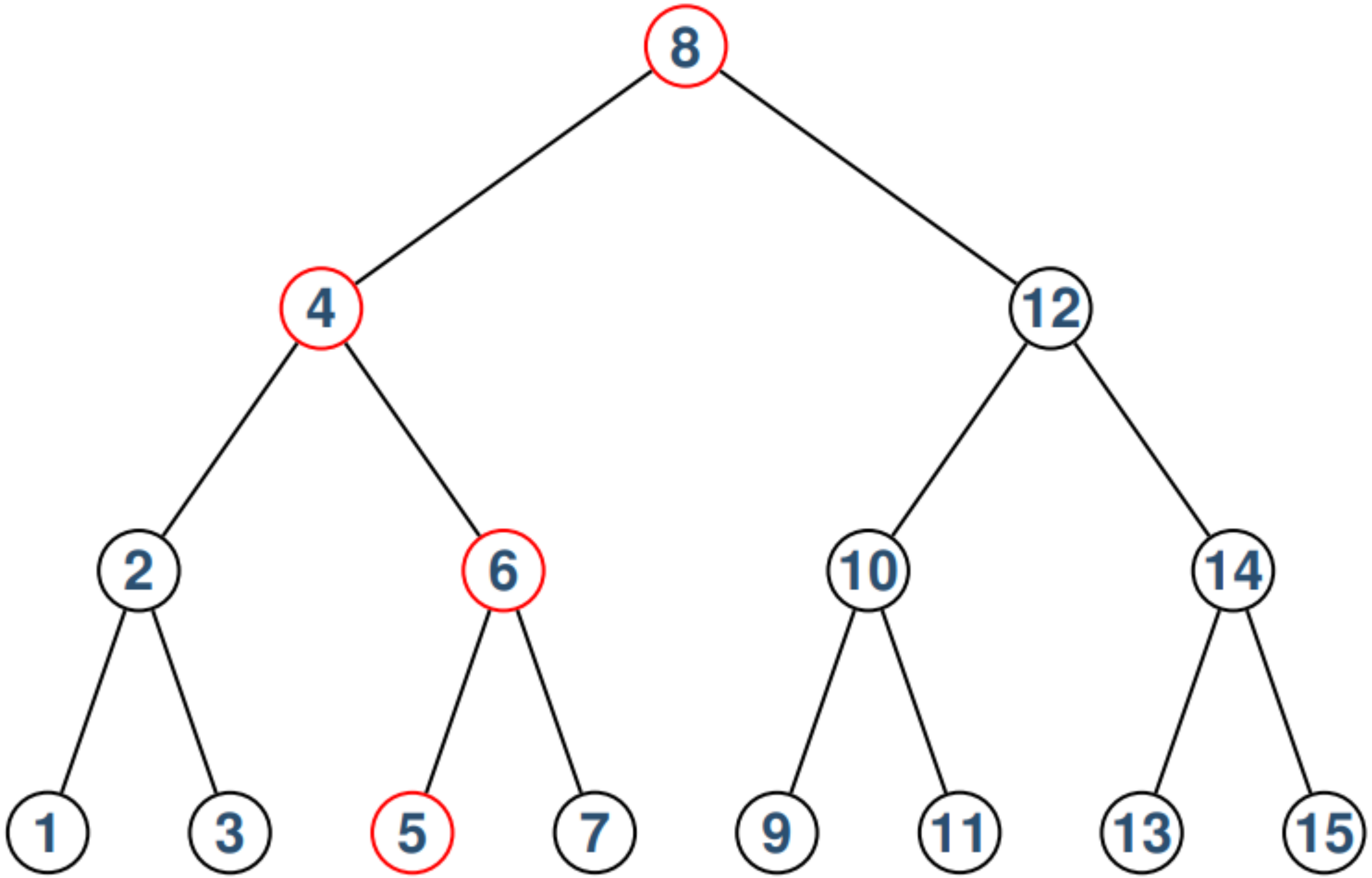
```
struct tree{
        Nodeptr root; };

typedef struct tree Tree;
typedef Tree* Treeptr;

Treeptr newTree(){
        Treeptr tree=(Treeptr)malloc(sizeof(Tree));
        tree->root=NULL;
        return tree;
}
```

Temel İkili Arama Ağacı İşlemleri

Örnek bir ikili arama agacında 5'i arama



Temel İkili Arama Ağacı İşlemleri

Verilen bir degeri ikili arama agacında arayan özyinelemeli algoritma

```
Nodeptr search(Nodeptr node, int x){
        if(!node)
                return NULL;
        if(node->data==x)
                return node;
        else
                if(node->data>x)
                        return search(node->left,x);
                else
                        return search(node->right,x);
}
```

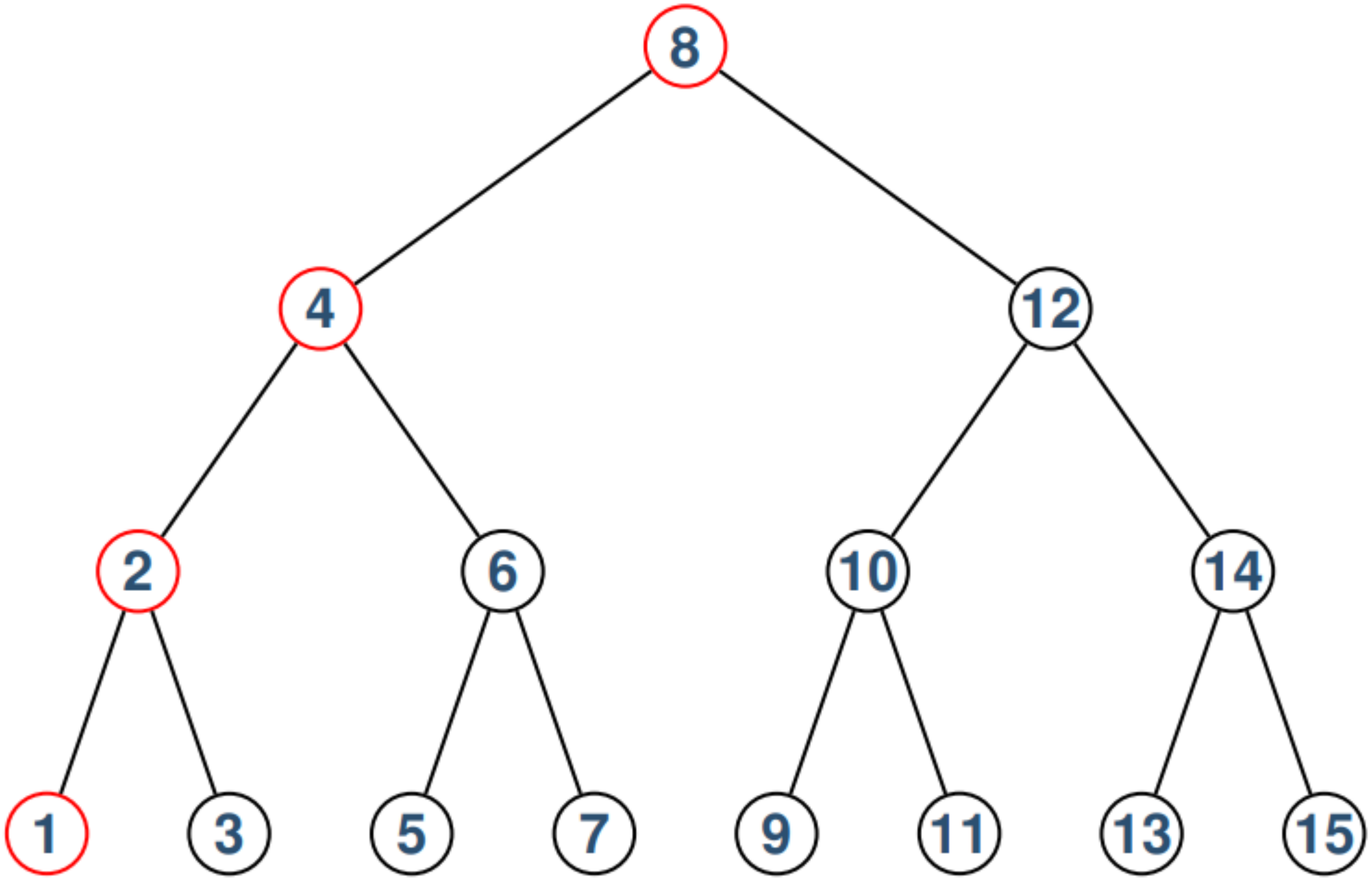
Temel İkili Arama Ağacı İşlemleri

Verilen bir degeri ikili arama agacında arayan özyinelemesiz algoritma

```
Nodeptr search(Treeptr tree, int x){
        Nodeptr temp=tree->root;
        while(temp!=NULL){
                if(temp->data== x)
                        return temp;
                else
                        if(temp->data>x)
                                temp=temp->left;
                        else
                                temp=temp->right;}}
```

Temel İkili Arama Ağacı İşlemleri

Ikili arama agacındaki en küçük elemanı arama



Temel İkili Arama Ağacı İşlemleri

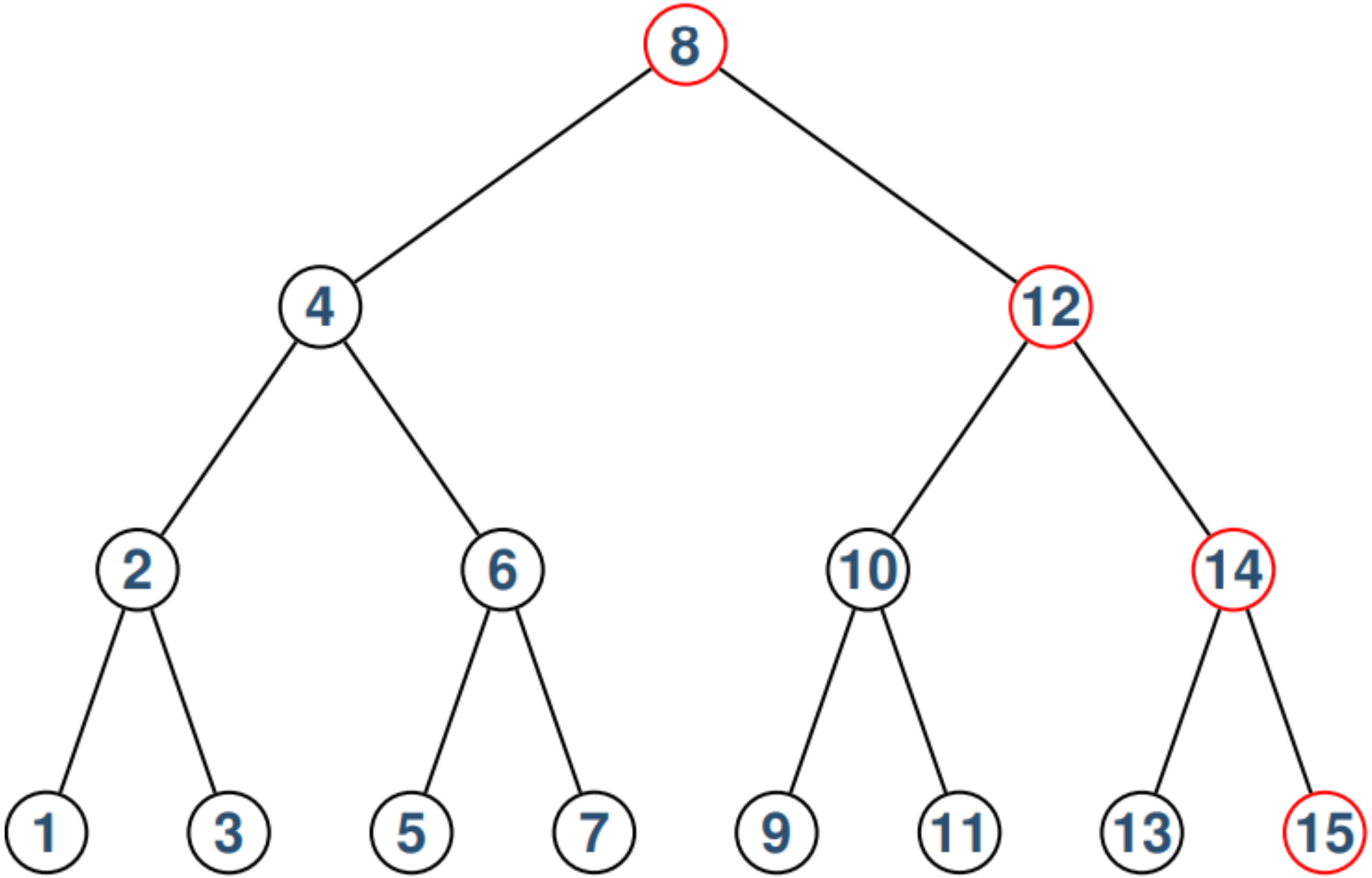
Ikili arama agacındaki en küçük elemanı arama

```
Nodeptr minNode(Nodeptr node){
        Nodeptr temp=node;
        while(temp->left)
                temp=temp->left;
        return temp;
}
```

```
Nodeptr minNode(Nodeptr node){
        if(node->left==NULL)
                return node;
        else
                return minNode(node->left);
}
```

Temel İkili Arama Ağacı İşlemleri

Ikili arama agacındaki en büyük elemanı arama



Temel İkili Arama Ağacı İşlemleri

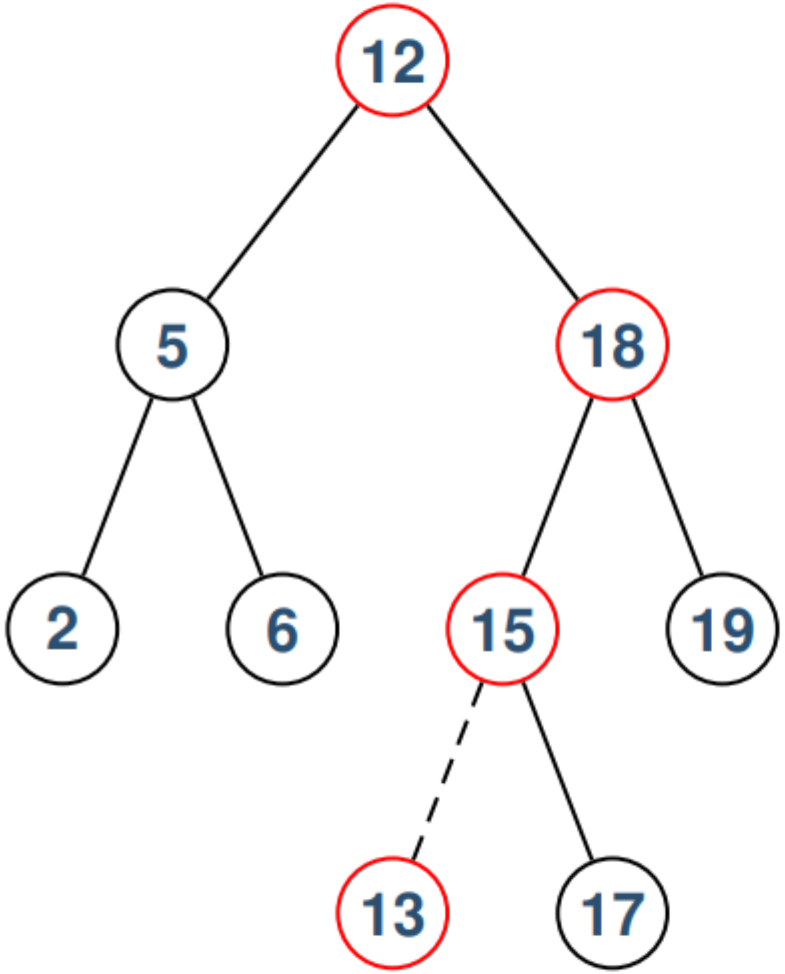
Ikili arama agacındaki en büyük elemanı arama

```
Nodeptr maxNode(Nodeptr node){
	Nodeptr temp=node;
	while(temp->right)
		temp=temp->right;
	return temp;
}
```

```
Nodeptr maxNode(Nodeptr node){
	if(node->right==NULL)
		return node;
	else
		return maxNode(node->right);
}
```

Temel İkili Arama Ağacı İşlemleri

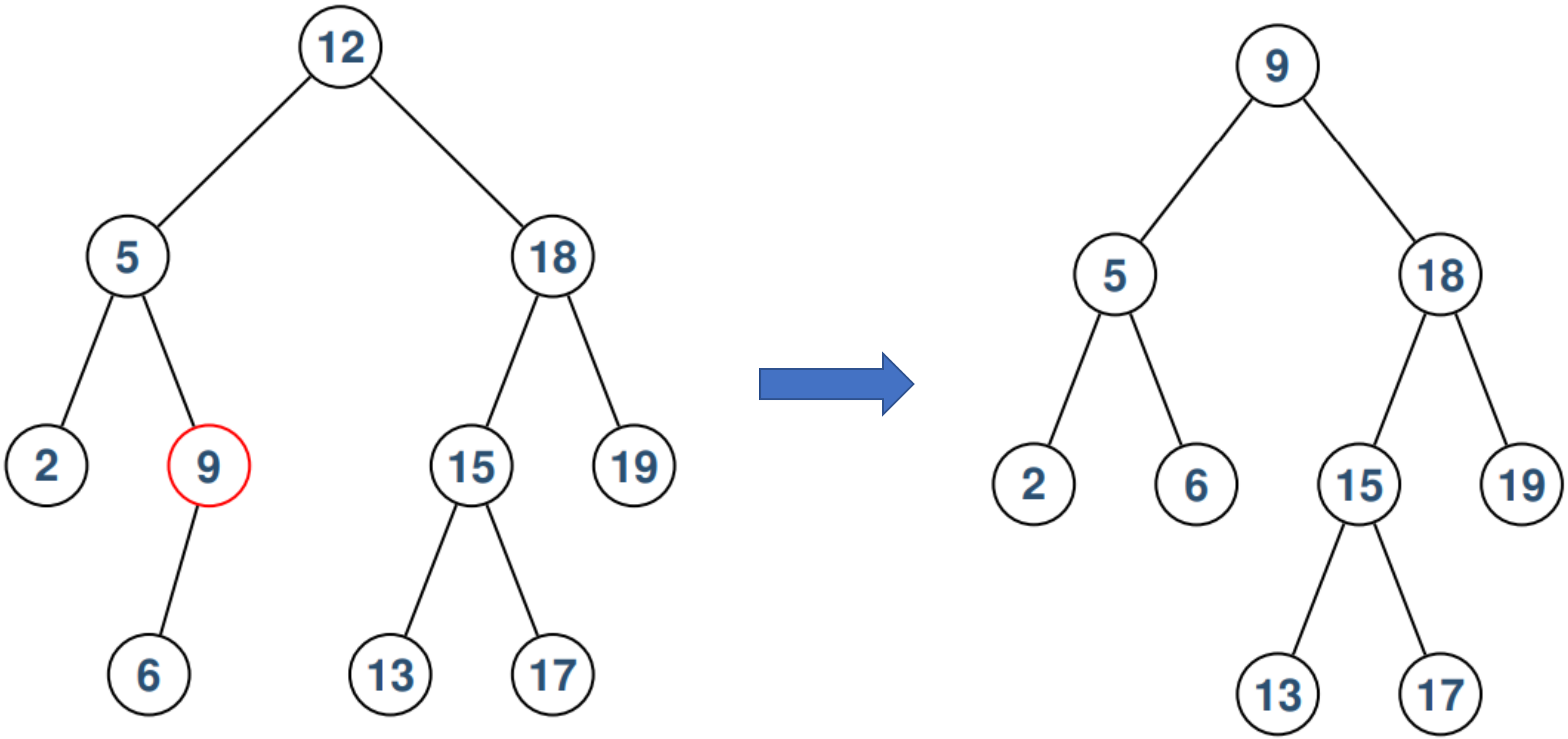
Örnek bir ikili arama agacına 13 elemanının eklenmesi



```
void addTree(Treeptr tree, Nodeptr node){
	Nodeptr n=tree->root;
	Nodeptr temp=NULL;
	while(n!=NULL){
		temp=n;
		if(node->data < n->data)
			n=n->left;
		else
			n=n->right;}
	if(temp==NULL)
		tree->root=node;
	else
		if(node->data < temp->data)
			temp->left=node;
		else
			temp->right=node;}
```

Temel İkili Arama Ağacı İşlemleri

Örnek bir ikili arama agacın kök elemanının silinmesi (1)



```
void removeTree(Treeptr tree, int data){
        Nodeptr temp, n=tree->root;
        while(n->data != data){
                if(n->data > data)
                        n=n->left;
                else
                        n=n->right;}
        while(1){
                temp=maxNode(n->left);
                if(temp==NULL)
                        temp=minNode(n->right);
                if(temp==NULL)
                        break;
                n->data=temp->data;
                n=temp;}}
```

Temel İkili Arama Ağacı İşlemleri

- Arama, Ekleme ve Silme işlemleri O(log n) kadar zaman alır.
- Dizilerde O(n) ve bağlı listede arama O(n) zaman almaktadır.
- Bağlı Listede ekleme veya silme noktası belli ise O(1) zaman almaktadır. Arama gerekirse yine O(n)'lik bir zaman
- İkili arama ağaçları dengeli tutulabilirse, bir anahtar değerini aramada oldukça hızlıdırlar. Böyle olduğunda n elemanlı bir ağaç en fazla O(log n) düzeyden oluşur. Bir değerin bulunması veya ağaçta olmadığının belirlenmesi için en fazla O(log n) karşılaştırma yapılır.

Gezintiler

```
//preorder NLR, NRL
void show(Nodeptr node){
        printf("%d - ", node->data);
        if(node->left)
                show(node->left);
        if(node->right)
                show(node->right);
}
```

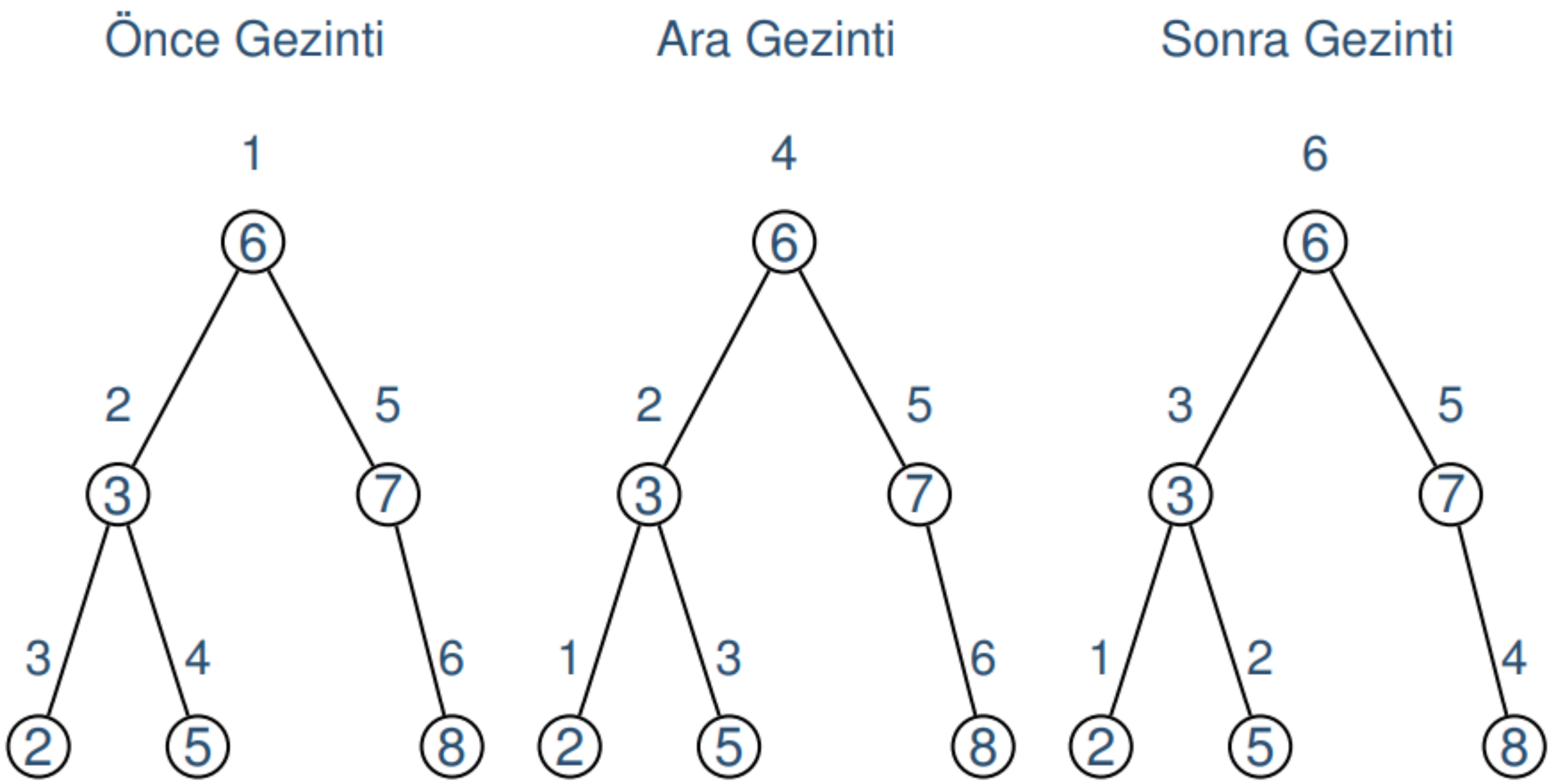
Gezintiler

```
//inorder LNR, RNL
void show(Nodeptr node){
		if(node->left)
			show(node->left);
		printf("%d - ", node->data);
		if(node->right)
			show(node->right);
}
```

Gezintiler

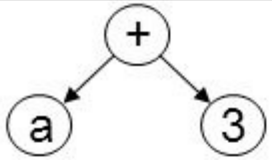
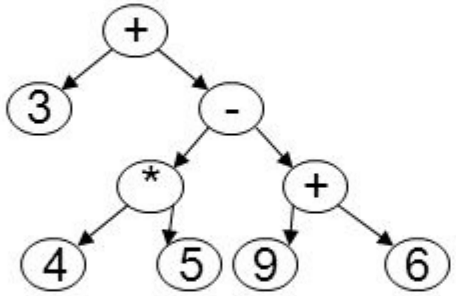
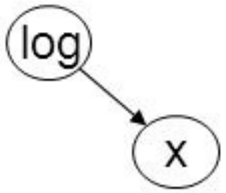
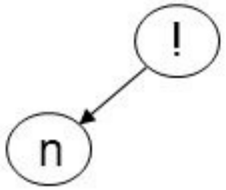
```
//postorder LRN, RLN
void show(Nodeptr node){
		if(node->left)
			show(node->left);
		if(node->right)
			show(node->right);
		printf("%d - ", node->data);
}
```

Gezintiler



Önce Gezinti
Ara Gezinti
Sonra Gezinti
1
4
6
6
6
6
2
5
2
5
3
5
3
7
3
7
3
7
3
4
6
1
3
6
1
2
4
2
5
8
2
5
8
2
5
8

Gezintiler

Expression	Expression Tree	Inorder Traversal Result
(a+3)		a + 3
3+(4*5-(9+6))		3+4*5-9+6
log(x)		log x
n!		n !

Ağacı Özyinelemesiz Gezme

- İçeriği bir ağaç düğümü (alt ağaç) olan eleman yapısı

```
public class Dugum{
    int icerik;
    Dugum sol;
    Dugum sag;
    public Dugum(int icerik){
        this.icerik =  icerik ;
        sol  = null;
        sag = null;
    }
}
```

```
public class Eleman{
    Dugum dugum;
    Eleman ileri ;
    public Eleman(Dugum dugum){
        this.dugum = dugum;
         ileri   = null;
    }
}
```

Ağacı Özyinelemesiz Gezme

- Bir ikili arama ağacındaki dügüm sayısını bulan algoritma (stack)

```
int dugumSayisi(){
    Dugum d;
    Eleman e;
    Cikin c;
    int sayi = 0;
    c = new Cikin();
    d = kok;
    if (d != null){
        e = new Eleman(d);
        c.cikinEkle(e);
    }
```

```
    while (!c.cikinBos()){
        e = c. cikinSil ();
        d = e.dugum;
        sayi++;
        if (d.sol != null){
            e = new Eleman(d.sol);
            c.cikinEkle(e);
        }
        if (d.sag != null){
            e = new Eleman(d.sag);
            c.cikinEkle(e);
        }
    }
    return sayi;
}
```

Ağacı Özyinelemesiz Gezme

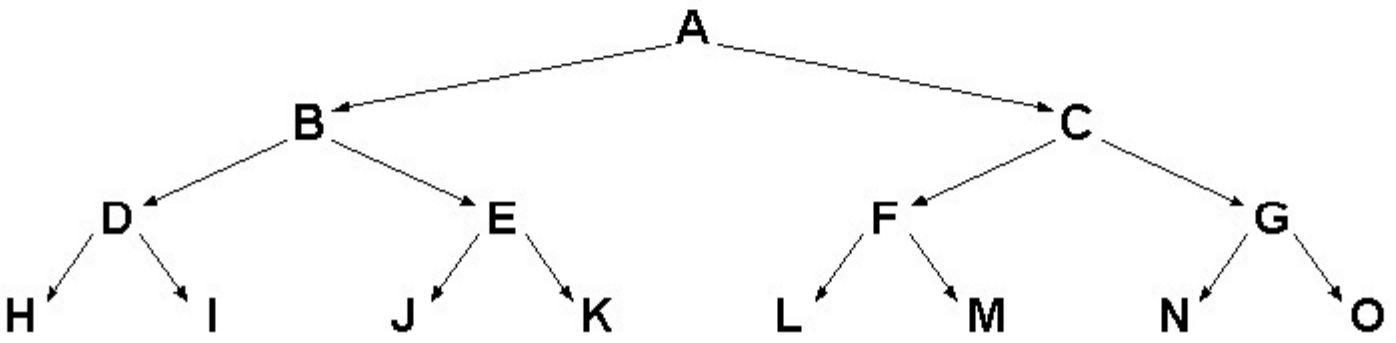
- Bir ikili arama ağacındaki dügüm sayısını bulan algoritma (queue)

```
int dugumSayisi(){
    Dugum d;
    Eleman e;
    Kuyruk k;
    int sayi = 0;
    k = new Kuyruk();
    d = kok;
    if (d != null){
        e = new Eleman(d);
        k.kuyrugaEkle(e);
    }
```

```
    while (!k.kuyrukBos()){
        e = k.kuyrukSil ();
        d = e.dugum;
        sayi++;
        if (d.sol  != null){
            e = new Eleman(d.sol);
            k.kuyrugaEkle(e);
        }
        if (d.sag != null){
            e = new Eleman(d.sag);
            k.kuyrugaEkle(e);
        }
    }
    return sayi;
}
```

Dizi tabanlı

- İkili arama ağacı gerçeklemenin bir diğer yolu dizi kullanmaktır.

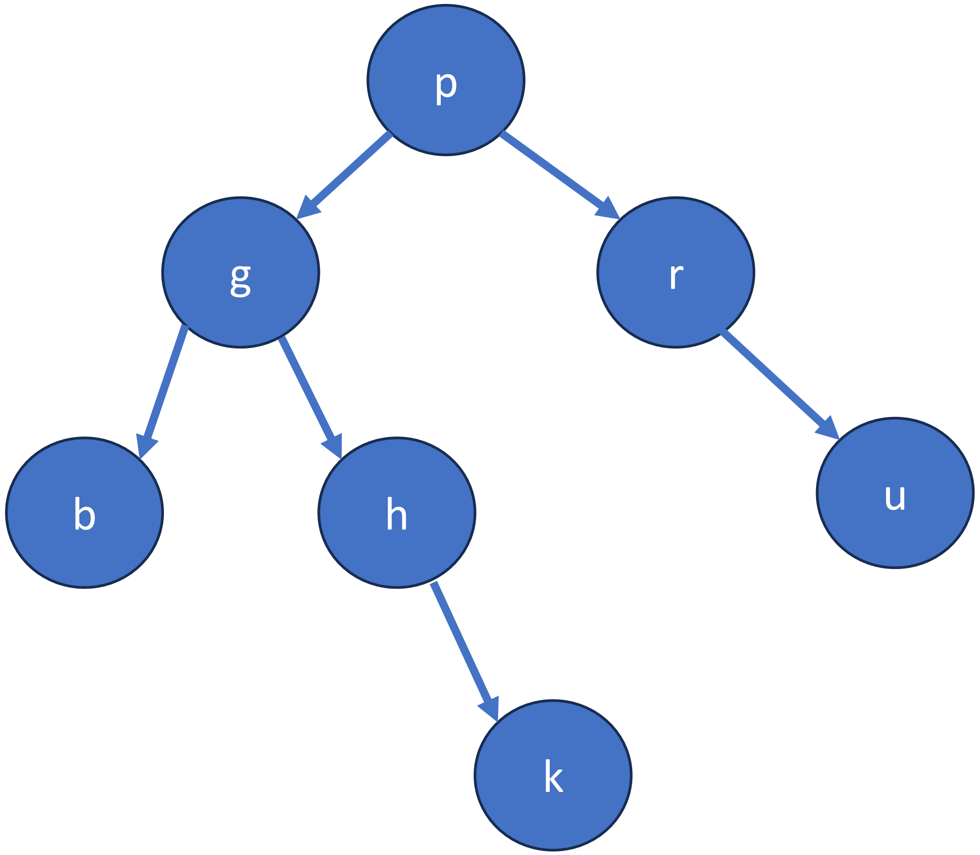


0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	B	C	D	E	F	G	H	I	J	K	L	M	N	O

```
#define MAX_NODES( depth )  ((int)pow( 2, (depth) ))
#define LEFT( inx )         (2 * (inx) + 1)
#define RIGHT( inx )        (2 * (inx) + 2)
#define PARENT( inx )       ((inx - 1) / 2)
```

Dizi tabanlı

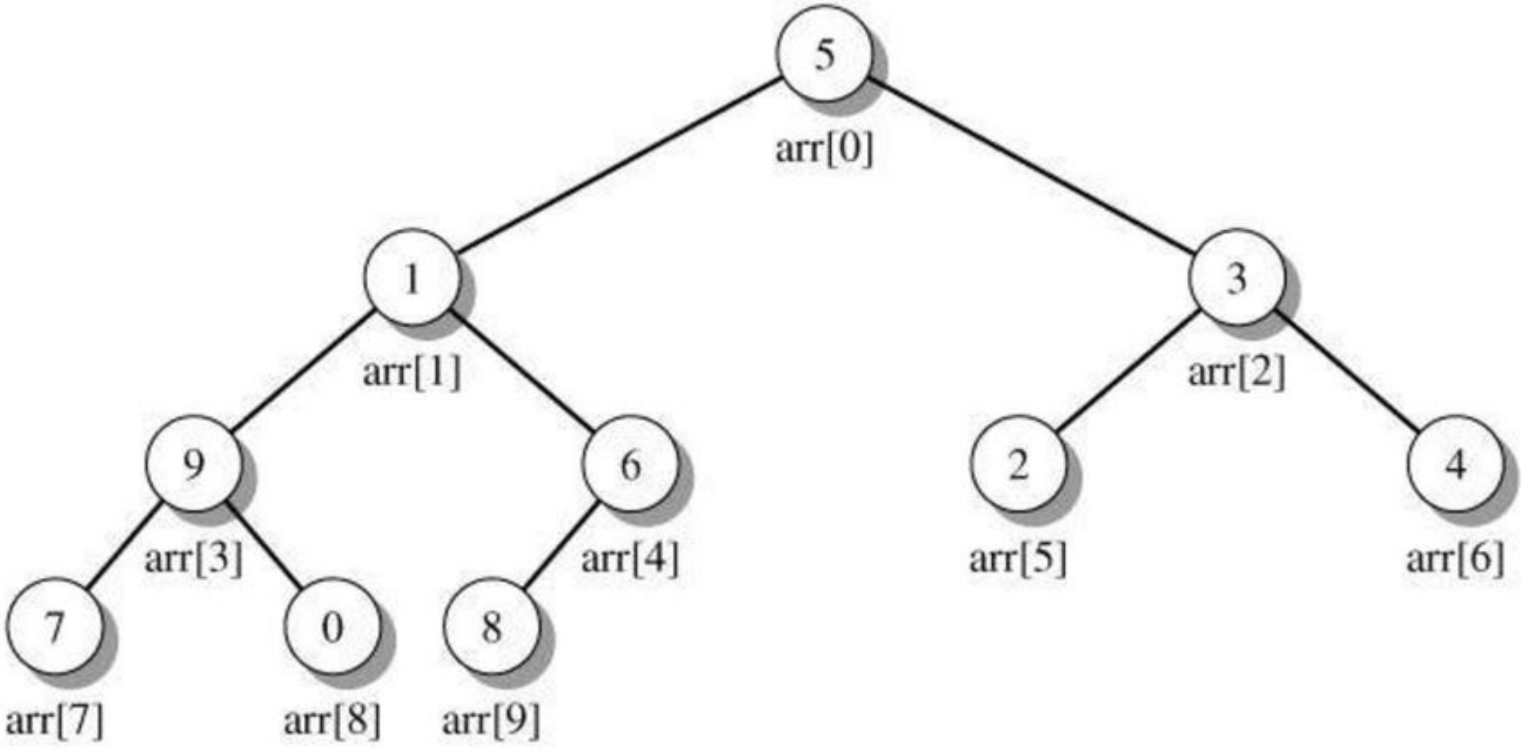
- İkili arama ağacını diziye yerleştiriniz



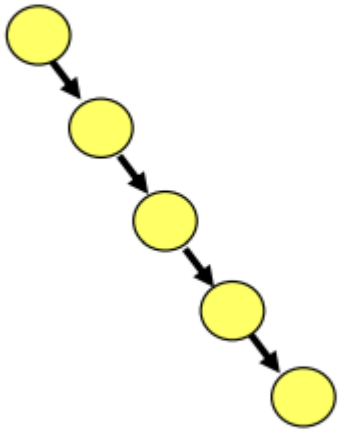
Dizi tabanlı

- İkili arama ağacı gerçeklemenin bir diğer yolu dizi kullanmaktır.

```
Integer[] arr = {5, 1, 3, 9, 6, 2, 4, 7, 0, 8};
```



Complete binary tree for an array arr with 10 elements.



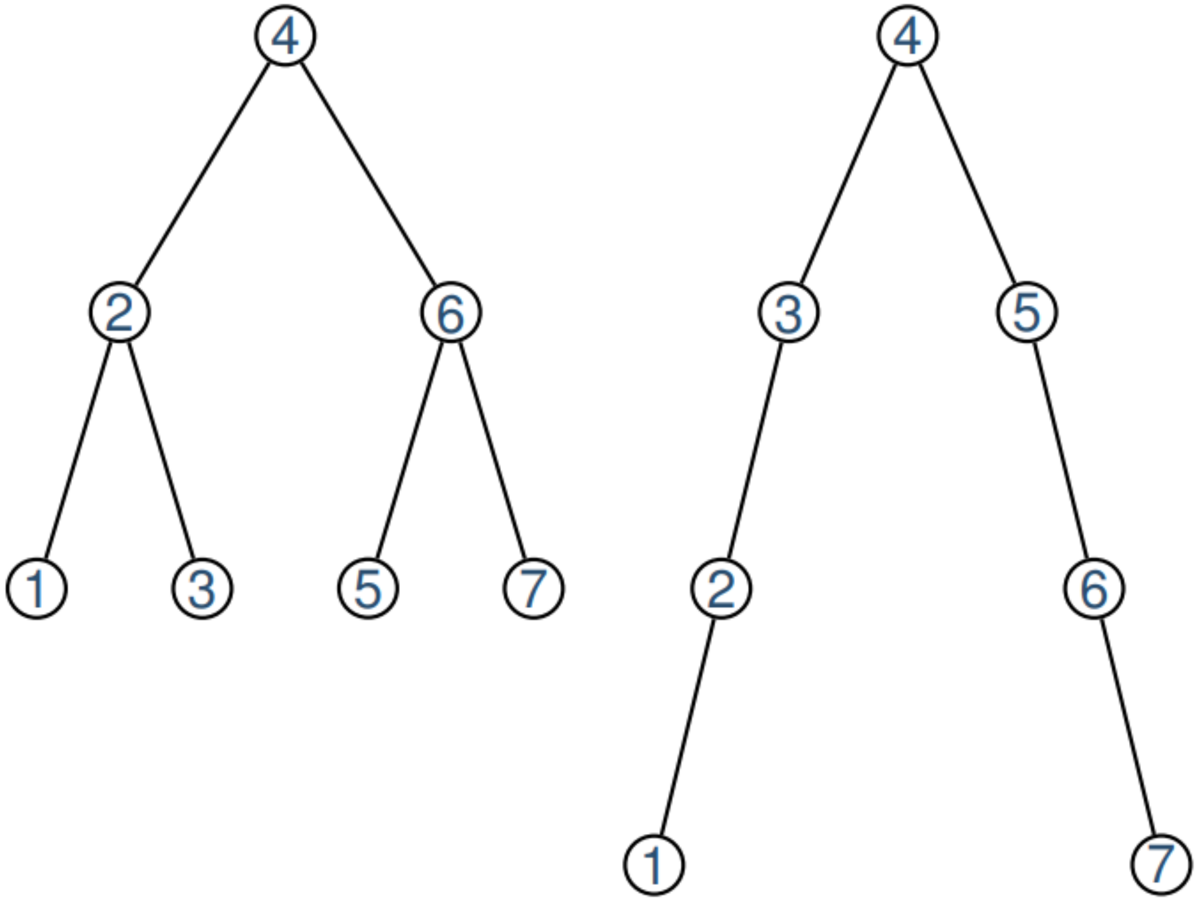
?????

AVL Ağacı (Dengeli)

- (Adelson-Velskii ve Landis)
- İkili arama ağacı dengeli olduğu sürece ağaçta belirli bir elemanı aramak, eleman eklemek, eleman silmek gibi işlemler O(log N) zamanda yapılabilmektedir.
- Ağaç dengesiz olduğunda bu süre hatta O($\sqrt{N}$) zamana kadar çıkabilir.
- T'deki her iç düğüm v için, v'nin çocuklarının yükseklikleri en fazla 1 farklılık gösterebilir. Yani, T'deki bir v düğümünün x ve y çocukları varsa, o zaman
- $|r(x) - r(y)| \leq 1$

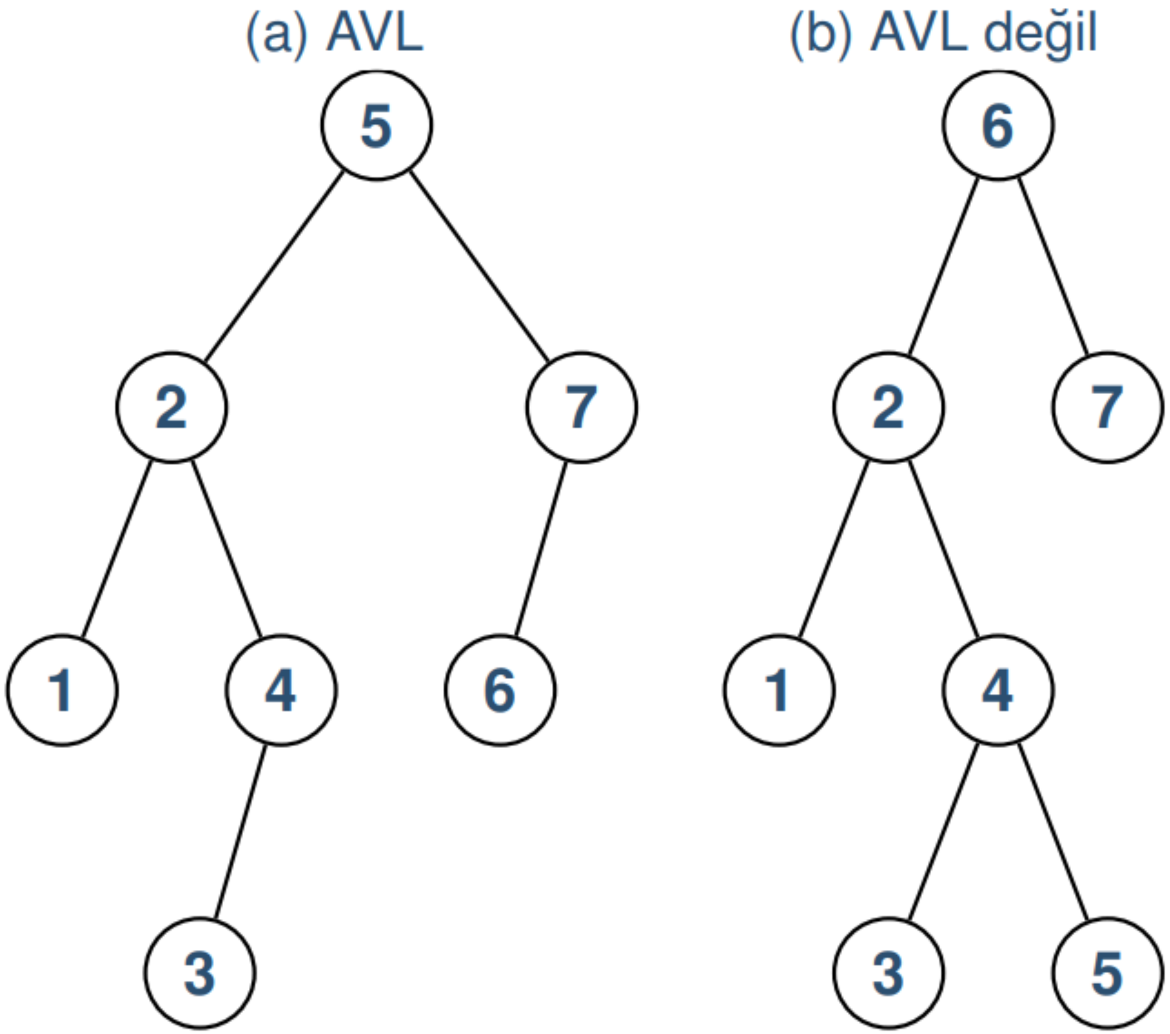
AVL Ağacı (Dengeli)

- Aşağıda 1-7 arası sayıların farklı sırada ikili arama ağacına eklenmesiyle oluşan değişik ağaçlar gösterilmiştir.
- Sayıların hangi sırada geleceği bilinemediğinden uygulamada bu iki ağacın biri ile karşılaşılabilir.
- Bu durumda dengeli ağaçlar tercih edilir.
- Dolayısıyla ağacın dengesi bozulduğunda ağacı yeniden dengelemek gerekir.



AVL Ağacı (Dengeli)

- AVL (Adelson-Velskii-Landis) ağacı denge koşullu bir ikili arama ağacıdır.
- Denge koşulu oldukça basittir ve ağacın derinliğinin O(log N) kalmasını sağlar.
- AVL ağacında her düğümün sol ve sağ alt ağaçlarının boy farkı en fazla 1 olabilir.
- Tek bir düğümün bile sağ ve sol alt ağaçlarının boy farkı 1'den büyükse o ikili arama ağacı AVL ağacı değildir.

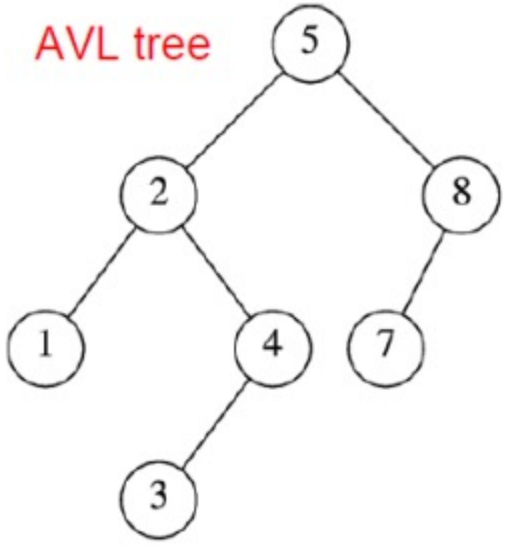


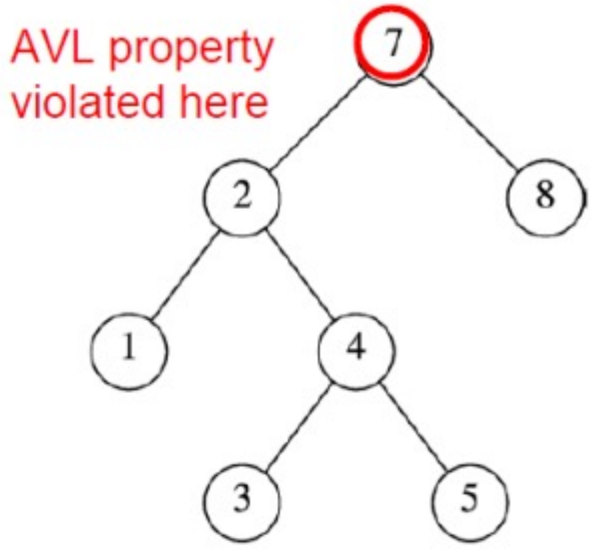
AVL Ağacı (Dengeli)

- Kök düğümün sağ ve sol alt ağaçları aynı boyda olmalıdır.
- Her bir düğümün sağ ve sol alt ağaçları aynı boyda olmalıdır.
- AVL ağacında bilinmesi gereken bir kavram denge faktörüdür. (Balance Factor – Denge Faktörü)
  - Balance Factor = Sağ Yükseklik – Sol Yükseklik
  - Veya Balance Factor = Sağ Düzey – Sol Düzey
- Denge faktörünün -1,0,1 arasında değerler alabilir. (Mutlak değer 1'den büyük olması söz konusu değildir).

AVL Ağacı (Dengeli)

- Balance Factor Örneği





Balance Factor = 2-3 = |-1| =1

Balance Factor = 1-3 = |-2| =2

AVL Ağacı (Dengeli)

- Yandaki AVL ağacının herhangi bir düğümünün tanımı olan algoritma verilmiştir.
- AVL veri yapısı normal düğüm veri yapısı gibi tanımlanmaktadır.
- Her düğümün içinde bir tane içerik alanı, sol ve sağ çocuk düğümlerini gösteren bir alan bulunur.
- Bu iki işaretçinin kendisi de bir AVL düğümü olduğundan AVL düğüm tanımı özyinelemelidir.

```
public class AvlDugum{
    int icerik;
    int boy;
    AvlDugum sol;
    AvlDugum sag;
    public AvlDugum(int icerik){
        this.icerik = icerik ;
        sol = null;
        sag = null;
        boy = 1;
    }
}
```

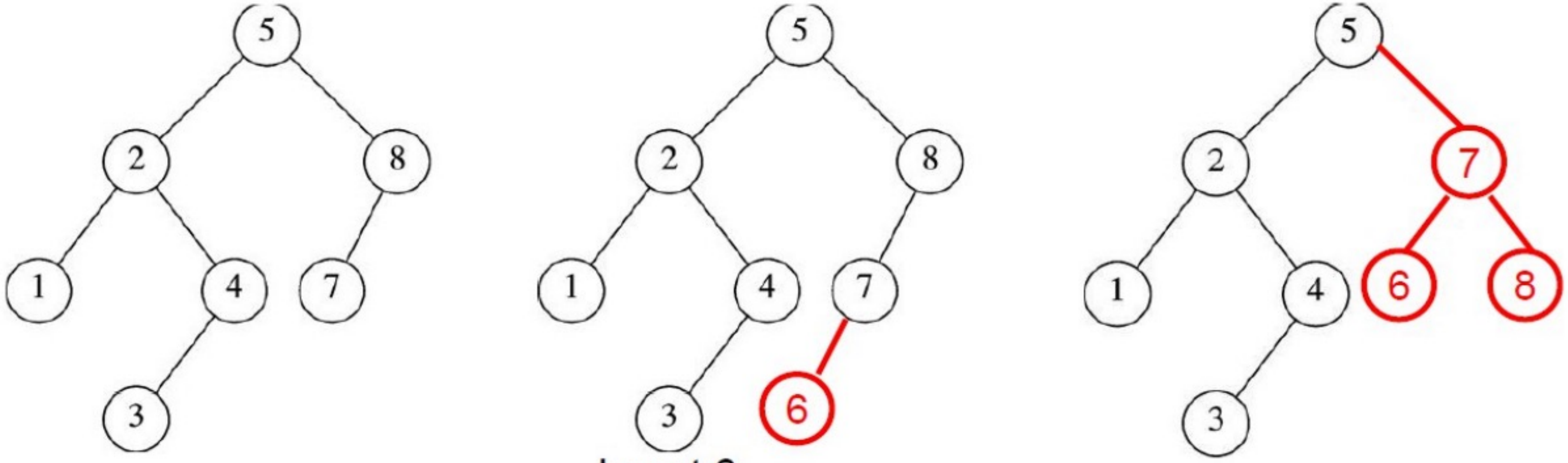
AVL Ağacı (Dengeli)

- Yandaki AVL ikili arama ağacının tanımı olan algoritma verilmiştir.
- AVL düğümü veri yapısı özyinelemeli olduğundan tek başına bir ağacı tanımlamak için yeterlidir.
- Bağlı liste yapısındaki gibi AVL ağacının kök düğümünü gösterecek işaretçi yeterlidir.

```
public class AvlAgac{
    AvlDugum kok;
    public AvlAgac(){
        kok = null;
    }
}
int boy(AvlDugum d){
    if (d == null)
        return 0;
    else
        return d.boy;
}
```

AVL Ağacı (Dengeli)

- 6 eklendiğinde 8'e göre denge bozulmuş olur.
  - Denge faktörü için sadece kök'ün sağına soluna bakılmaz. Alt dallarında dengesine bakılır.
  - 8'e göre bakacak olursak |2-0| = 2'dir. Bu durumda rotasyon işlemine gidilir.



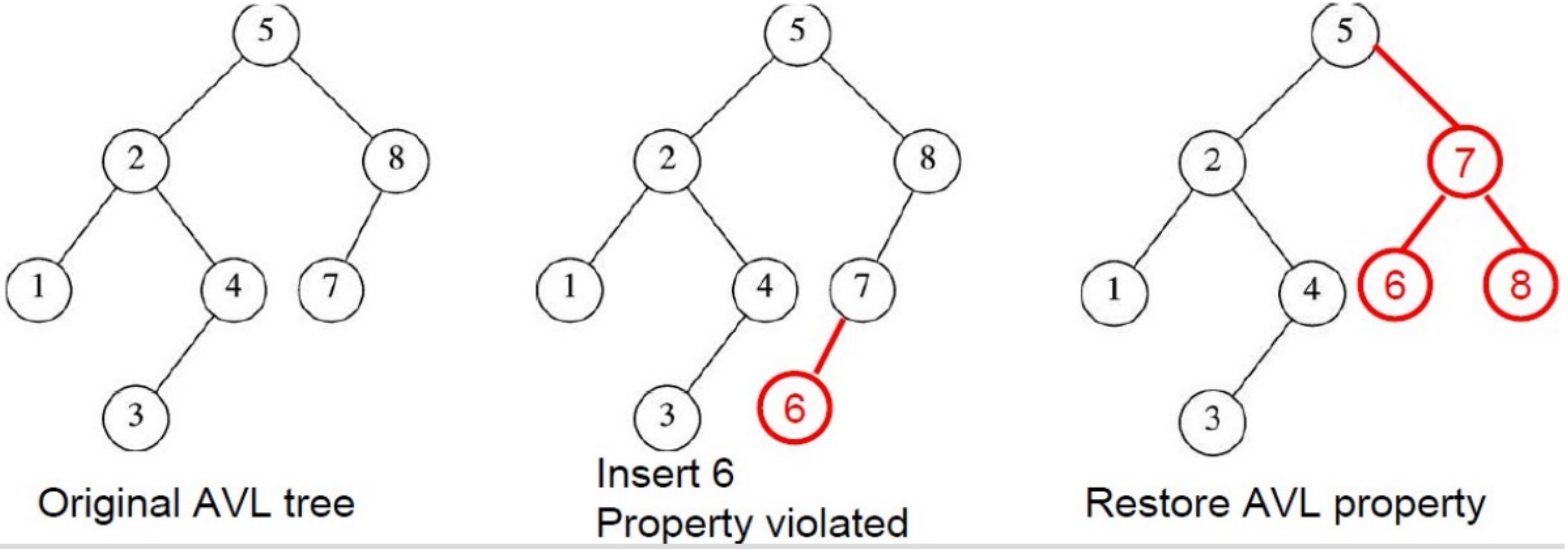
Original AVL tree

Insert 6
Property violated

Restore AVL property

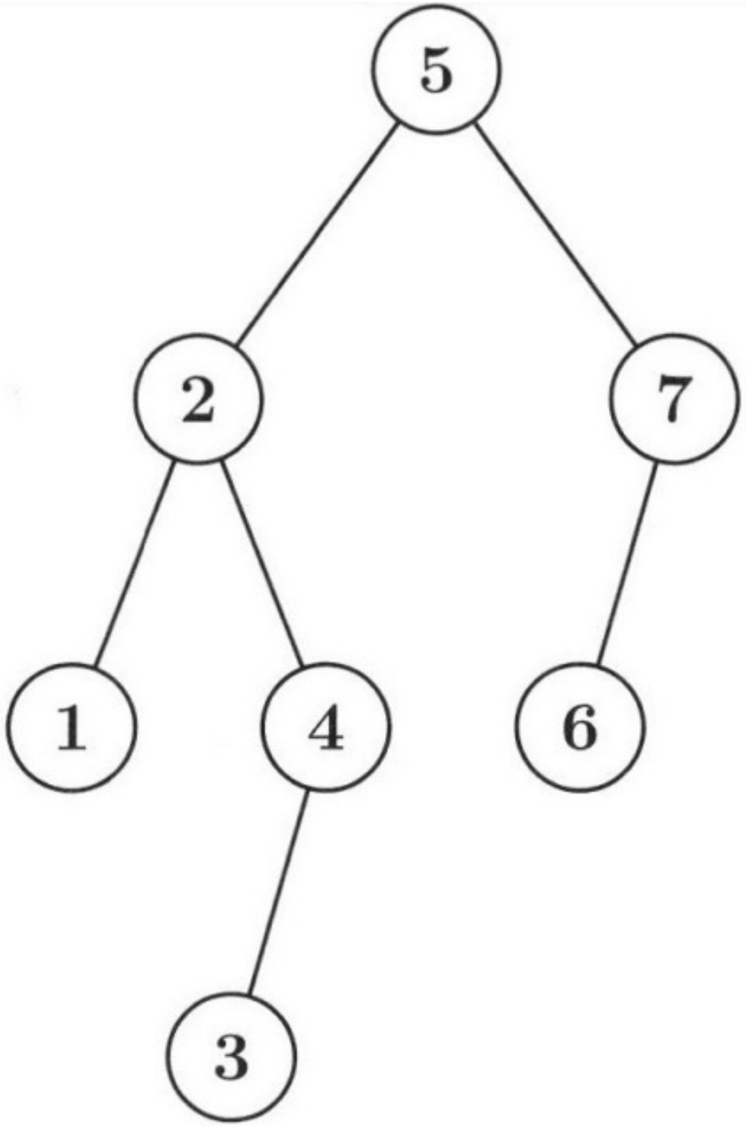
AVL Ağacı (Dengeli)

- Bu durumda rotasyon işlemine karar verilir.
- Şekilde rotasyonu göstermektedir.
- İki türlü rotasyon olabilir: Tek – Çift



AVL Ağacı (Dengeli)

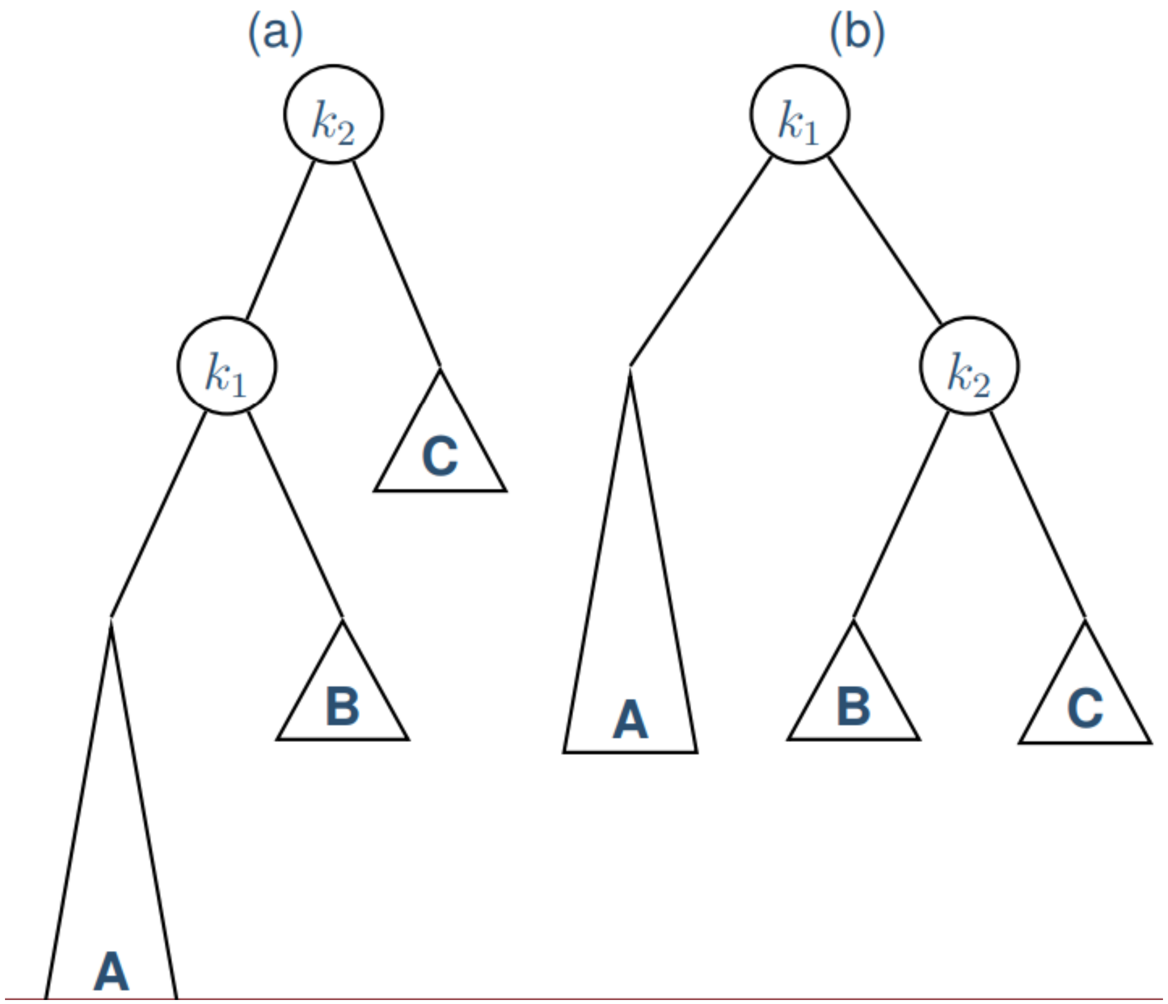
- AVL ağacına yeni bir eleman eklendiğinde
  - Yeni düğümden ağacın köküne giden tüm düğümlerin boy özelliğini değiştirmek
  - Ağacın AVL özelliğinin yeniden getirilmesi gerekir.
- Örneğin yandaki şekil (a)'da AVL ağacında 6'nın, 3'ün veya 1'in soluna veya sağına yeni bir eleman eklenirse AVL ağacı özelliği bozulur.
- AVL ağacı özelliğini geri getirmek için basit bir rotasyon yapmak yeterlidir.
- AVL özelliği bozulmuş düğüm d olsun. İkili arama ağacında bir düğümün en fazla iki çocuğu olduğundan, dengesizlik ancak d'nin iki alt ağacının boylarının farkının iki olmasıyla mümkündür.



AVL Ağacı (Dengeli)

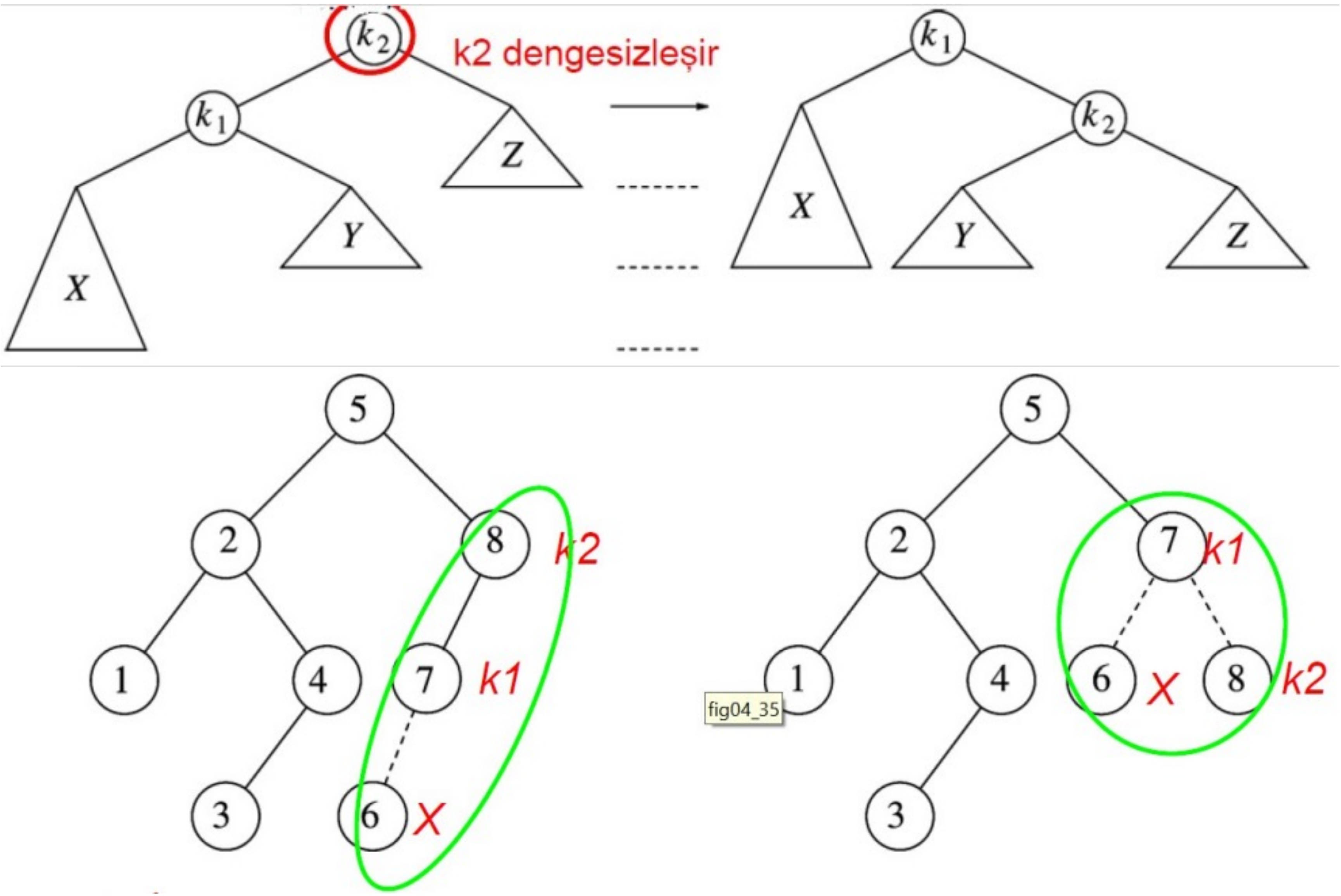
- Burada dört durum söz konusu olabilir:
  1. d'nin sol çocuğunun sol alt ağacına bir eleman eklendiğinde
  2. d'nin sol çocuğunun sağ alt ağacına bir eleman eklendiğinde
  3. d'nin sağ çocuğunun sol alt ağacına bir eleman eklendiğinde
  4. d'nin sağ çocuğunun sağ alt ağacına bir eleman eklendiğinde
- 1. ve 4. durumlar ile 2. ve 3. durumlar birbirlerinin simetriği olup temelde iki değişik durum oluştururlar.
- 1. ve 4. durumları tek rotasyon, 2. ve 3. durumları da çift rotasyon

Durum 1’i çözmek için tek rotasyon

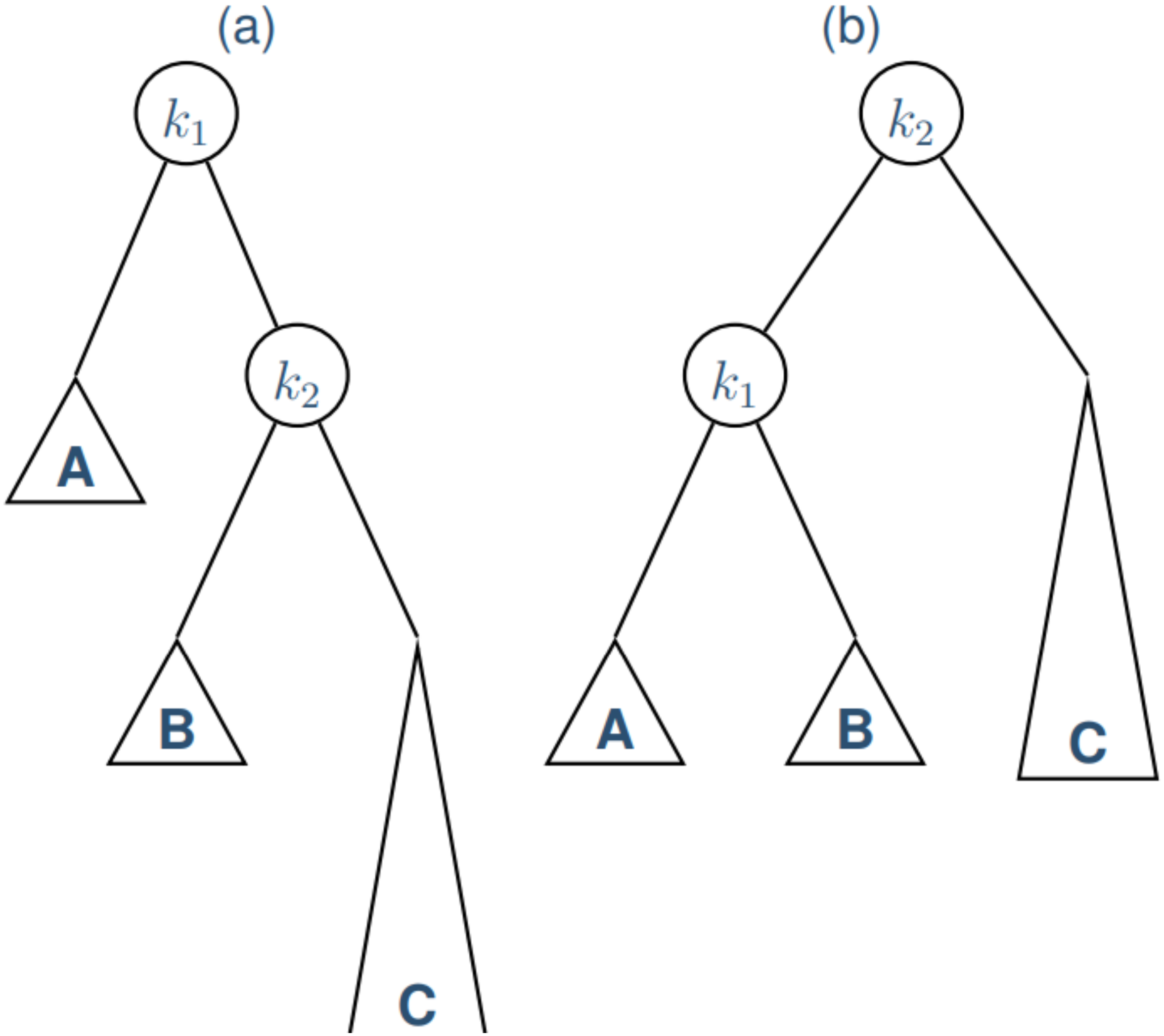



```
AvlDugum solTekRotasyon(AvlDugum k2){
    AvlDugum k1 = k2.sol;
    k2.sol  = k1.sag;
    k1.sag = k2;
    k2.boy = azami(boy(k2.sol), boy(k2.sag)) + 1;
    k1.boy = azami(boy(k1.sol), k1.sag.boy) + 1;
    return k1;
}
```

Durum 1'i çözmek için tek rotasyon

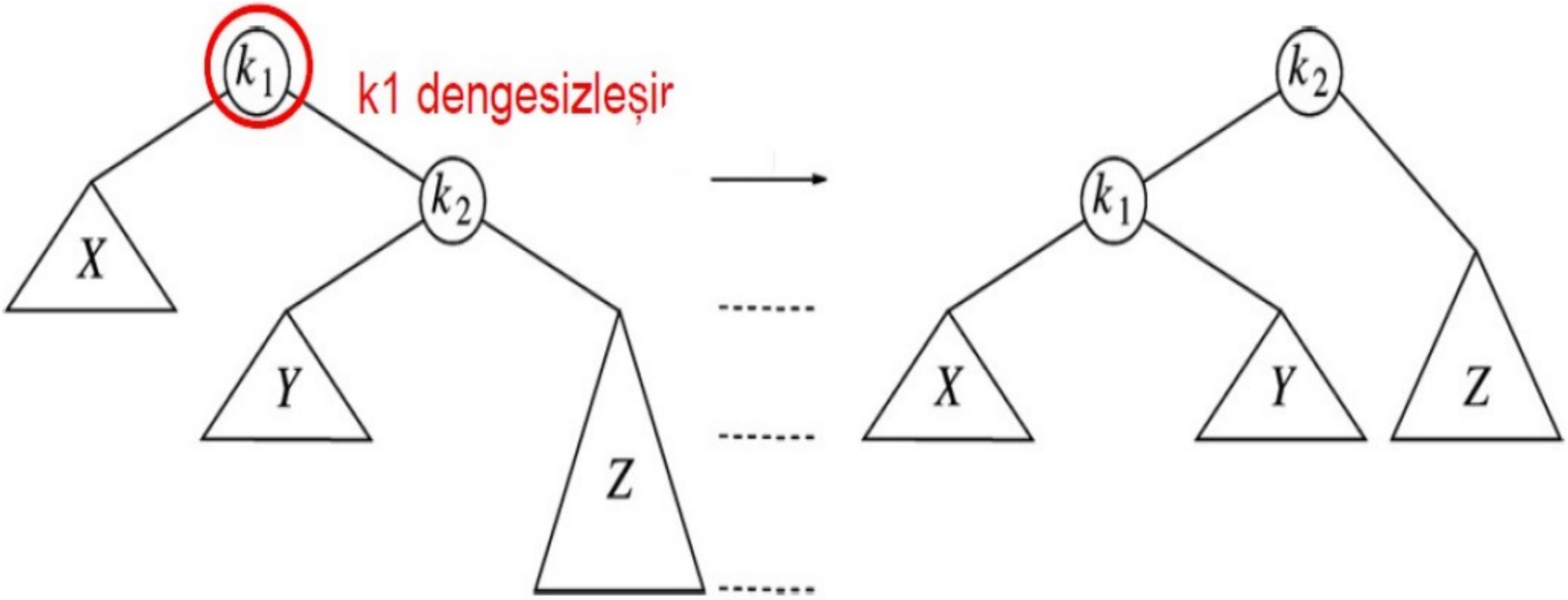


Durum 4'i çözmek için tek rotasyon

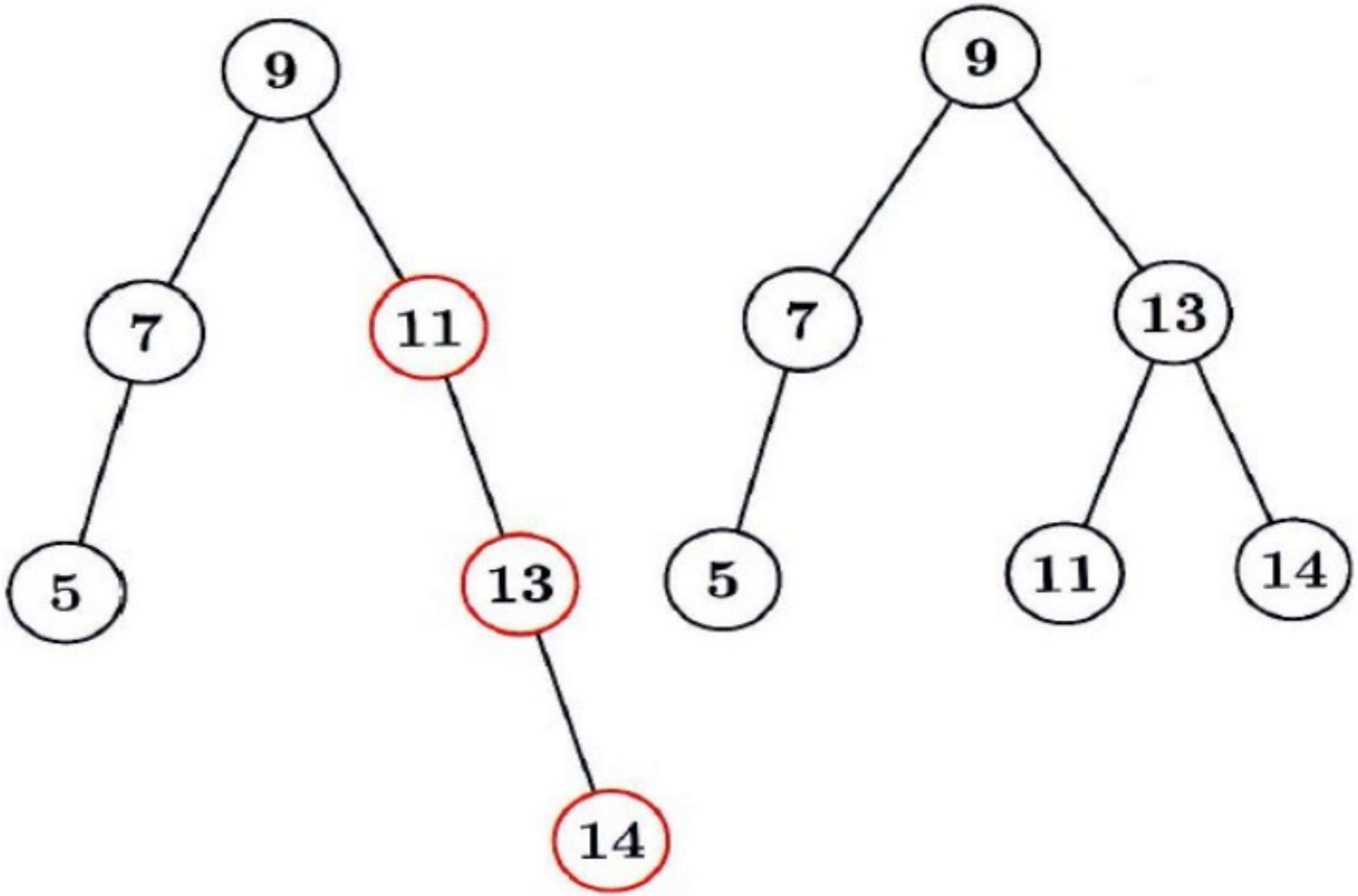


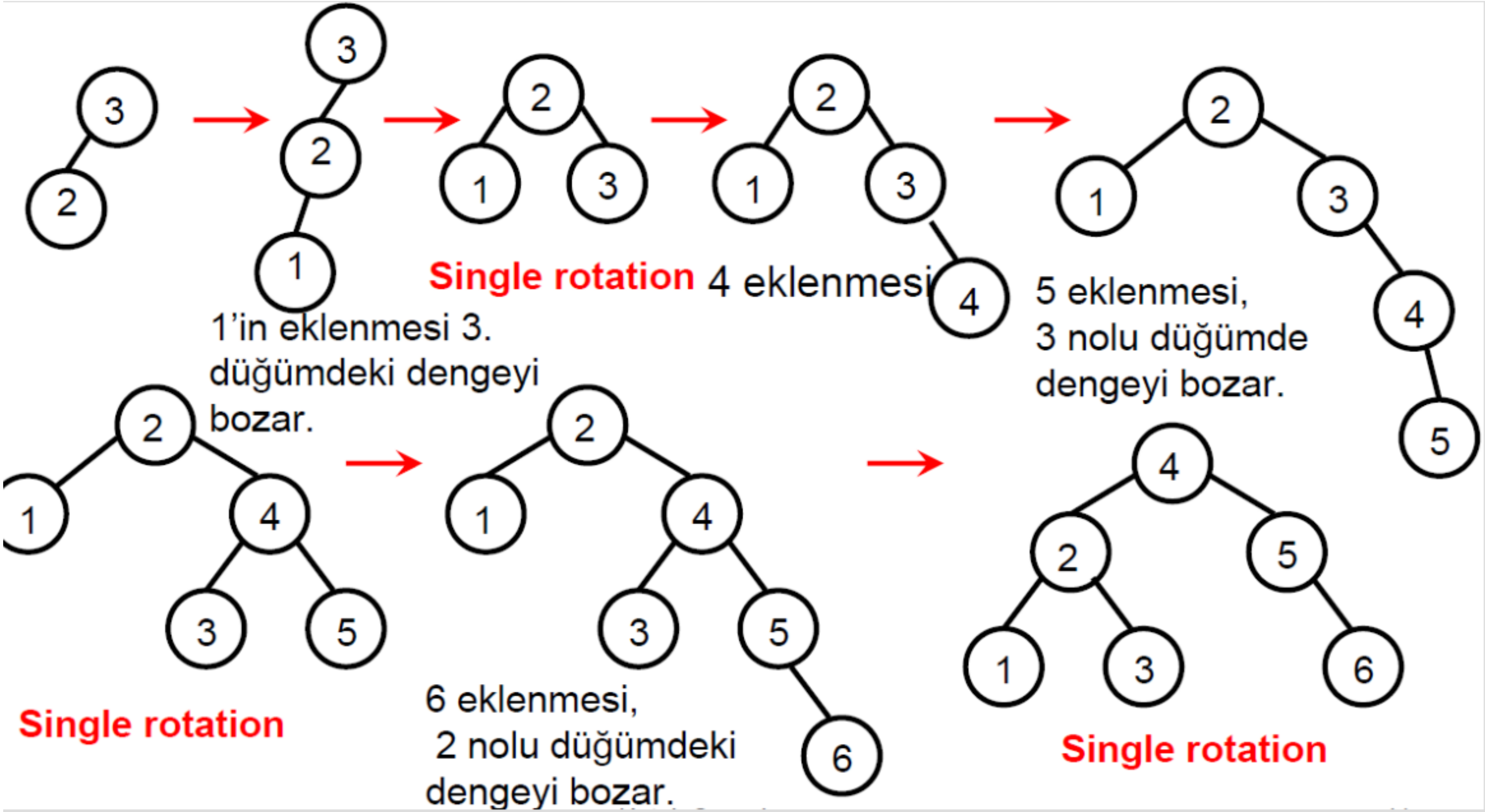

```
AvlDugum sagTekRotasyon(AvlDugum k1){
    AvlDugum k2 = k1.sag;
    k1.sag = k2.sol;
    k2.sol  = k1;
    k2.boy = azami(k2.sol.boy, boy(k2.sag)) + 1;
    k1.boy = azami(boy(k1.sol), boy(k1.sag)) + 1;
    return k2;
}
```

Durum 4'i çözmek için tek rotasyon

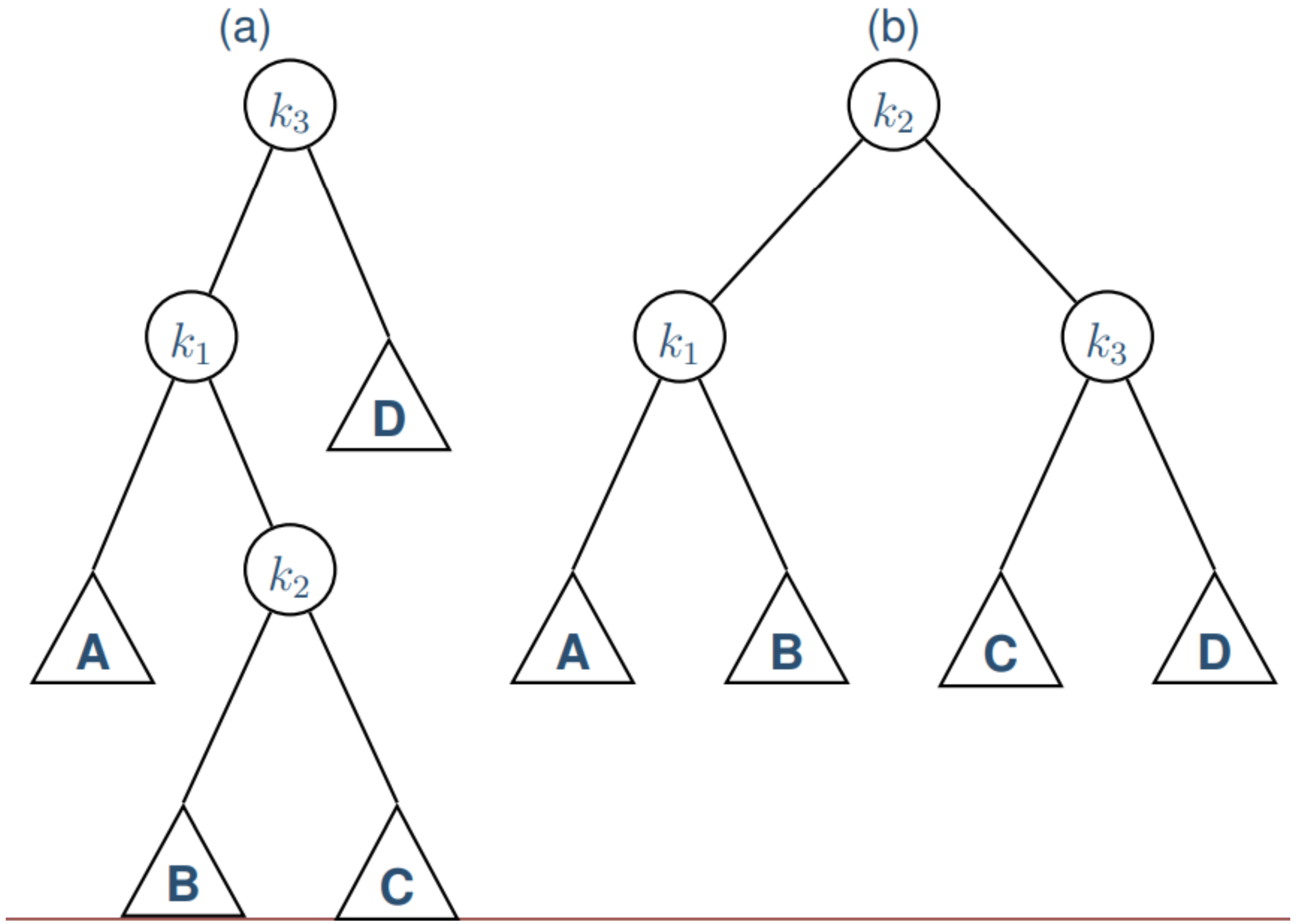


Durum 4’i çözmek için tek rotasyon



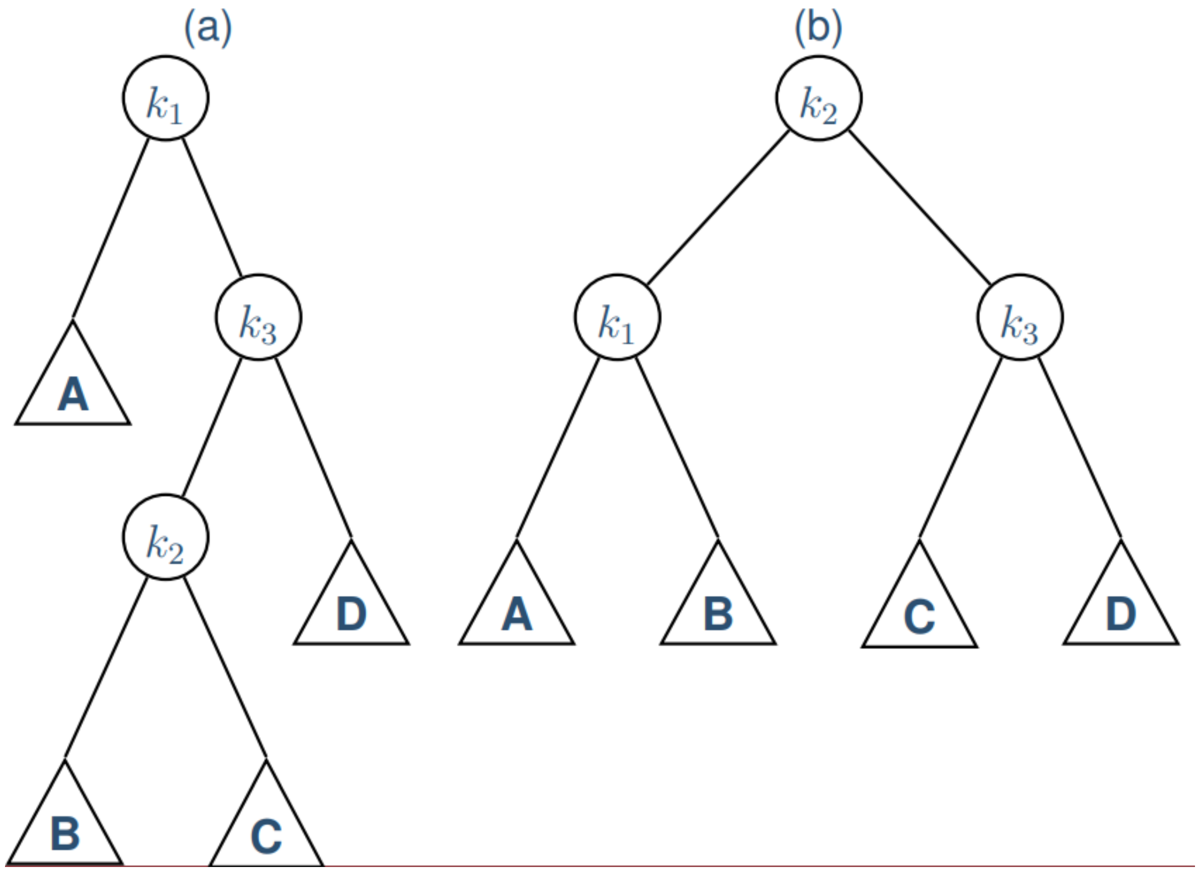

1'in eklenmesi 3. düğümdeki dengeyi bozar.
Single rotation
4 eklenmesi
5 eklenmesi, 3 nolu düğümde dengeyi bozar.
Single rotation
6 eklenmesi, 2 nolu düğümdeki dengeyi bozar.
Single rotation

Durum 2'i çözmek için çift rotasyon




```
AvlDugum solCiftRotasyon(AvlDugum k3){
    k3.sol  = sagTekRotasyon(k3.sol);
    return solTekRotasyon(k3);
}
```

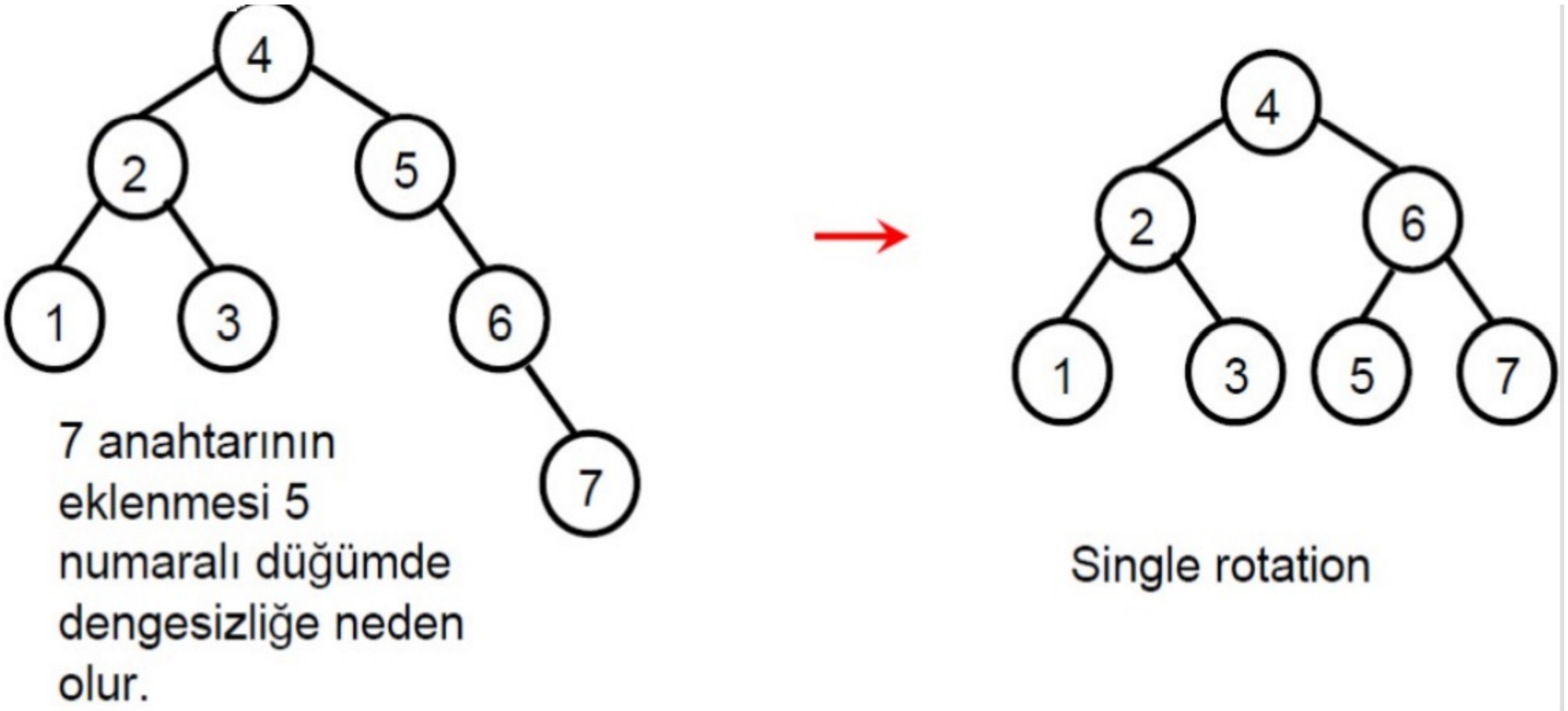
Durum 3'ü çözmek için çift rotasyon



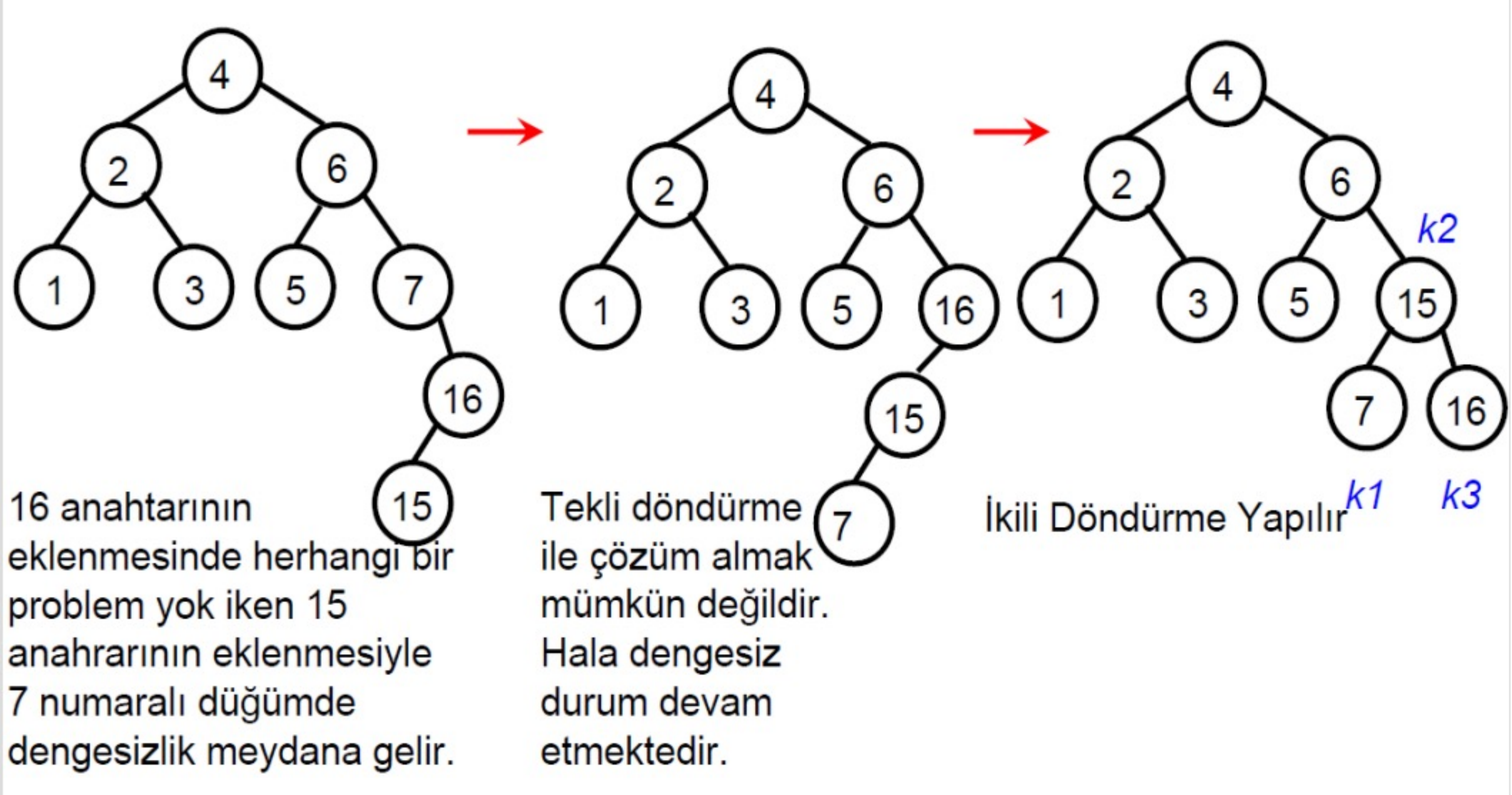

```
AvlDugum sagCiftRotasyon(AvlDugum k1){
    k1.sag = solTekRotasyon(k1.sag);
    return sagTekRotasyon(k1);
}
```

Örnek

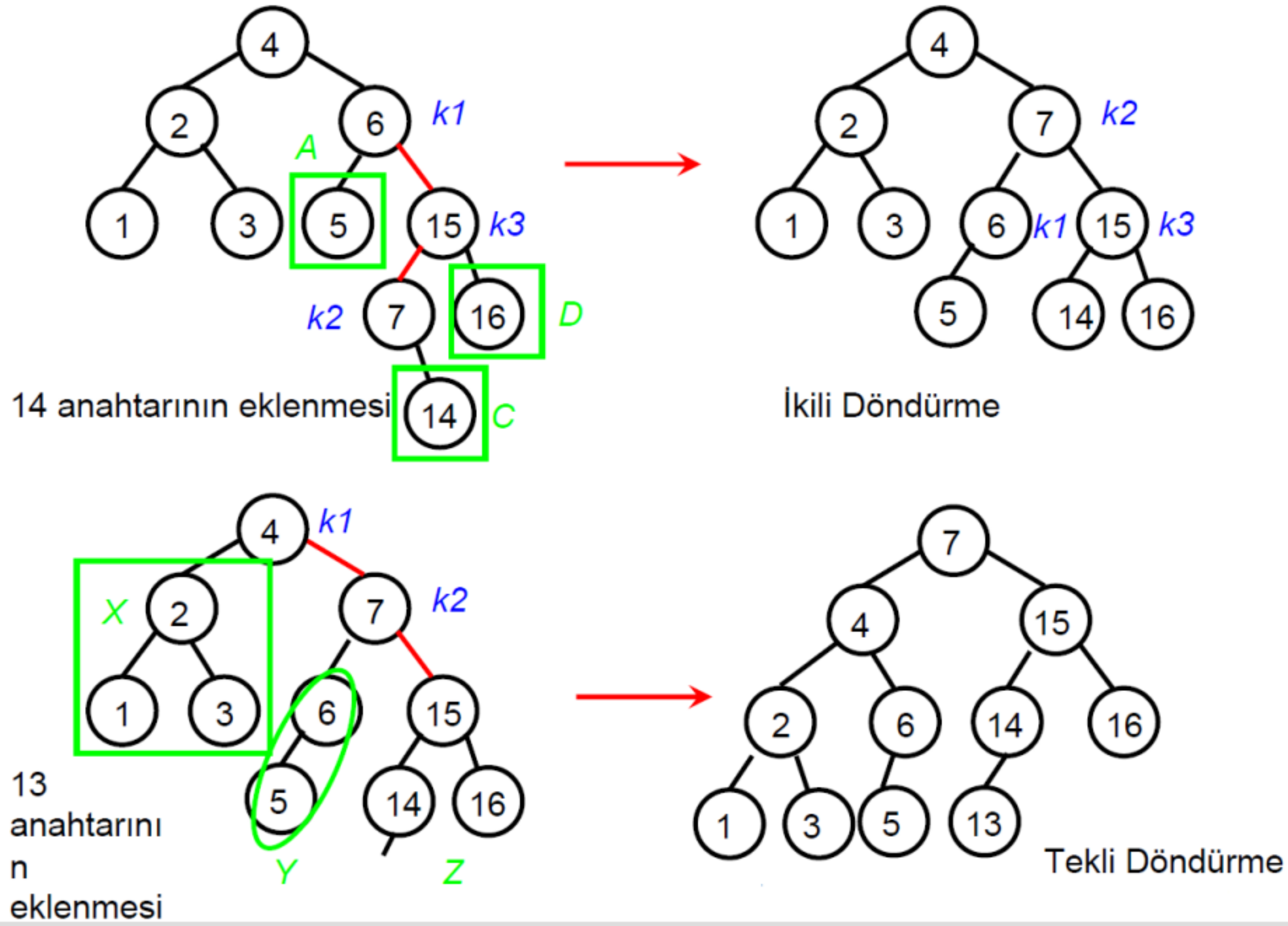
- Sırayla 7, 16, 15, 14, 13, 12, 11, 10, 8, 9 anahtarlarını ağacımıza ekleyelim.



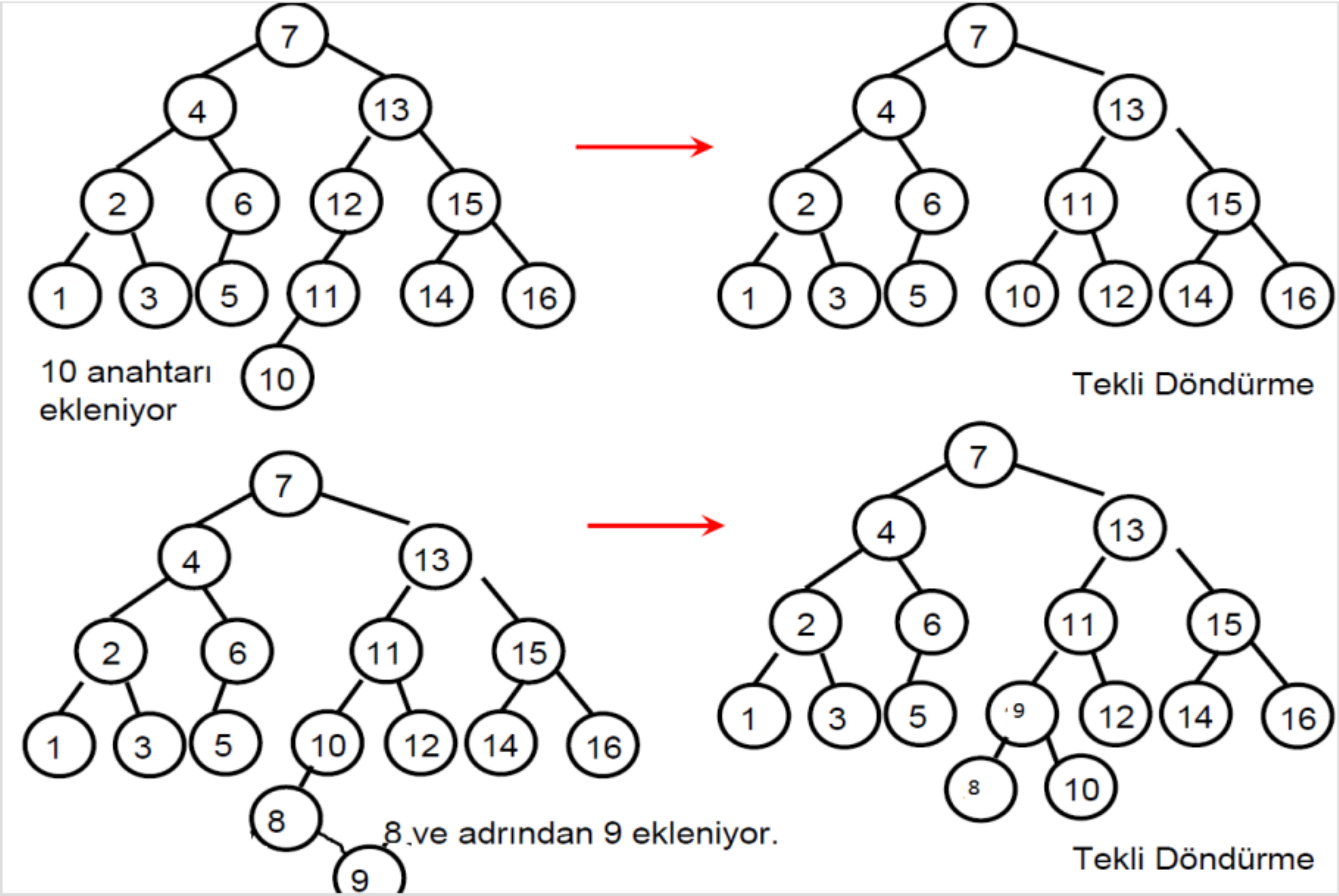
Örnek


16 anahtarının eklenmesinde herhangi bir problem yok iken 15 anahrarının eklenmesiyle 7 numaralı düğümde dengesizlik meydana gelir.
Tekli döndürme ile çözüm almak mümkün değildir. Hala dengesiz durum devam etmektedir.
İkili Döndürme Yapılır
k2
k1
k3

Örnek


k1
k2
k3
A
C
D
14 anahtarının eklenmesi
İkili Döndürme
X
Y
Z
13 anahtarını n eklenmesi
Tekli Döndürme

Örnek


7
4
13
2
6
12
15
1
3
5
11
14
16
10
10 anahtarı
ekleniyor
7
4
13
2
6
11
15
1
3
5
10
12
14
16
Tekli Döndürme
7
4
13
2
6
11
15
1
3
5
10
12
14
16
8
9
8 ve adrından 9 ekleniyor.
7
4
13
2
6
11
15
1
3
5
9
12
14
16
8
10
Tekli Döndürme

AVL

- Tüm kayıtların eklenmesi beklenmeden her eklemeden sonra bir veya iki döndürme yapılır.
- Dengesiz bir ağacın kök düğümünün denge faktörü-/+ 2 olur.
- Ekleme işlemi sırasında izlenen yol üzerinde -/+1 denge faktörü değerine sahip olan düğüm yoksa ağaç hala dengededir.
- Ekleme eklenen son yaprak kendi parent'ının ilk child'ı ise yol üzerindeki tüm düğümlerin denge faktör değeri yeniden düzenlenir.
- Eğer eklenen son yaprak kendi parent'ının ikinci child'ı ise sadece parent denge faktör değeri yeniden düzenlenir.

AVL ağacına yeni bir eleman ekleyen algoritma

```
void agacaEkle(Avldugum yeniEleman){
    Avldugum y = null, x = kok, t;
    Eleman e;
    int yon1 = 0, yon2 = 0;
    Cikin c = new Cikin(100);
    while (x != null){
        y = x;
        e = new Eleman(y);
        c.cikinEkle(e);
        yon1 = yon2;
        if (yeniEleman.icerik < x. icerik ){
            x = x.sol;
            yon2 = SOL;
        }else{
            x = x.sag;
            yon2 = SAG;
        }
    }
    cocukYerlestir(y, yeniEleman);
```

```
    while (!c.cikinBos()){
        e = c. cikinSil ();
        x = e.dugum;
        x.boy = azami(boy(x.sol), boy(x.sag)) + 1;
        if (Math.abs(boy(x.sol) - boy(x.sag)) == 2){
            if (yon1 == SOL && yon2 == SOL)
                t = solTekRotasyon(x);
            if (yon1 == SOL && yon2 == SAG)
                t = solCiftRotasyon(x);
            if (yon1 == SAG && yon2 == SOL)
                t = sagCiftRotasyon(x);
            if (yon1 == SAG && yon2 == SAG)
                t = sagTekRotasyon(x);
            e = c. cikinSil ();
            y = e.dugum;
            cocukYerlestir(y, t);
            break;
        }
    }
}
```

Ödev

- Verilen bir modelin ağaç olup olmadığını kontrol eden fonksiyonu yazınız.
- (x+y/z) – (A^2+4*B) ifadesine ilişkin ikili ağacı çizdiren fonksiyonu yazınız.
- Rastgele sayı üreterek 5 derinliğinde bir ikili ağaç oluşturan, preorder, inorder ve postorder gezinti sonuçlarını veren fonksiyonu yazınız.