Max - Min Heap Tree

(Max ve Min Yığıt Ağaçları)

Max - Min Heap

- Öncelikli kuyruk konusunu hatırlayın. Kuyruğa sonradan eklenmesine rağmen öncelik seviyesine göre önce çıkabiliyordu.
- Öncelik kuyruğu oluşturmada farklı veri yapıları benimsenebilir. Yığınlar bunlardan sadece biridir.
- Tam ikili ağaç, yığıt kurmak amacıyla kullanılabilir. Yığınlar ise bir dizide gerçekleştirilebilir.
- Yığını dizide tam ikili ağaç yapısını kullanarak gerçekleştirdiğimizde yığın, dizinin 1. elemanından başlar, 0. indis kullanılmaz. Dizi sıralı değildir, yalnız 1. indisteki elemanın en öncelikli (ilk alınacak) eleman olduğu garanti edilir.

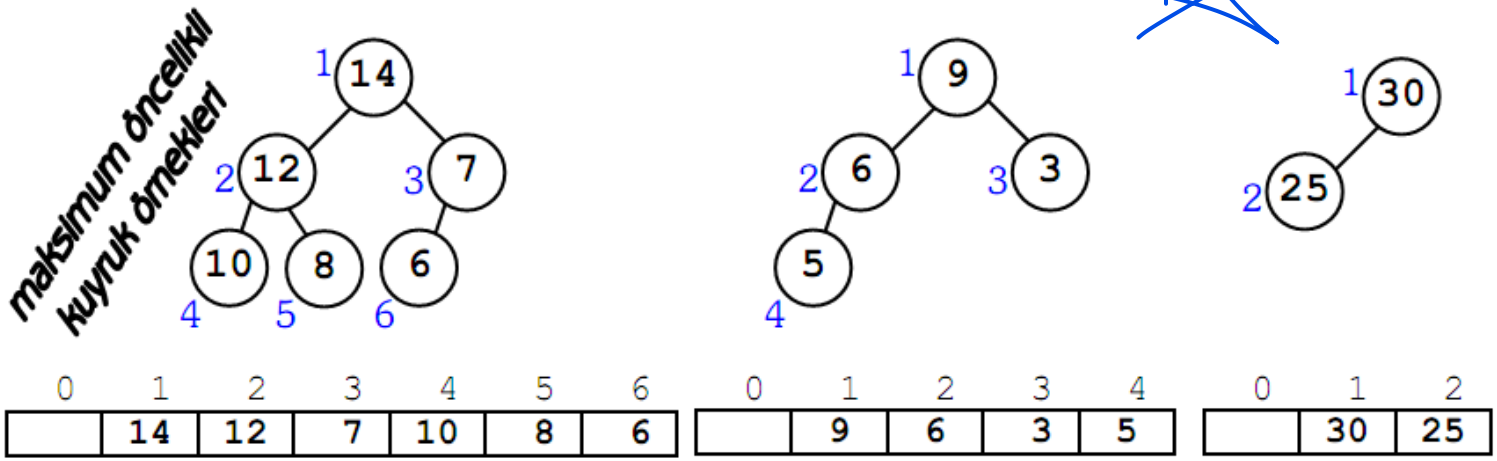
Max - Min Heap

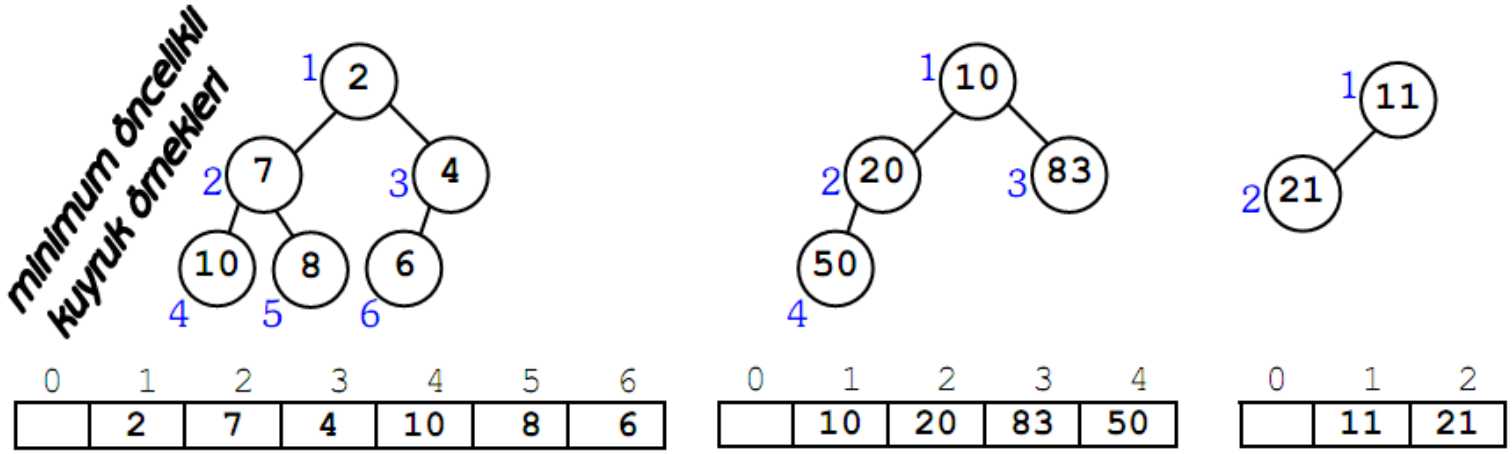
- Yığın denince aklımıza complete binary tree gelecek, search tree değil. Hatırlayalım; tam ikili ağacın tüm düzeyleri dolu, son düzeyi ise soldan sağa doğru doludur. İkili arama ağacı ile yığın arasında ise bir bağlantı yoktur.

- **Tanım**:
- Tam ikili ağaçtaki her düğümün değeri çocuklarından küçük değilse söz konusu ikili ağaç **maksimum yığın (max heap)** olarak isimlendirilir.

- Tam ikili ağaçtaki her düğümün değeri, çocuklarından büyük değilse söz konusu ikili ağaç **minimum yığın (min heap)** olarak isimlendirilir.

Max - Min Heap

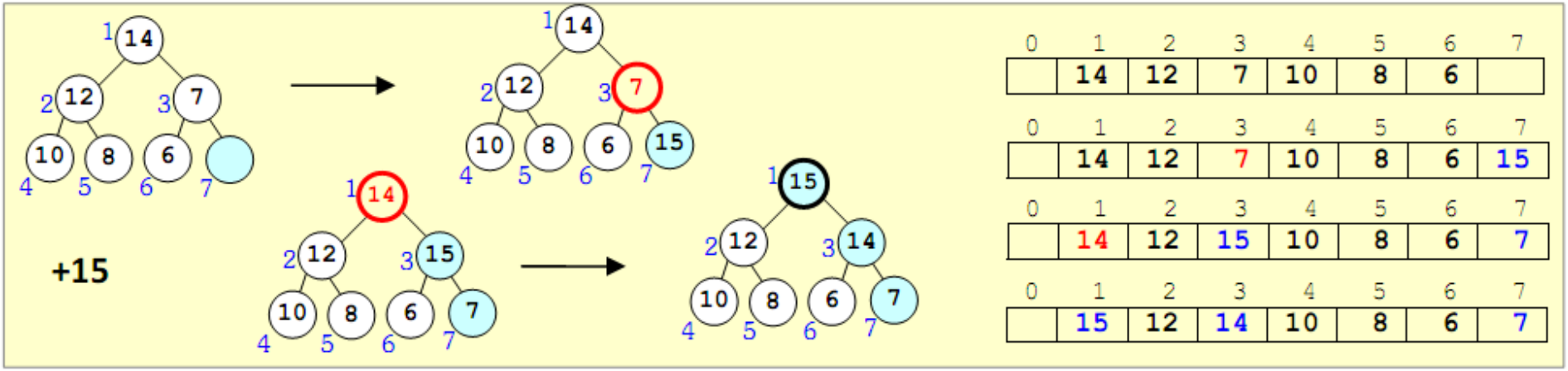

maksimum öncelikli kuyruk örnekleri
1 14
2 12
3 7
4 10
5 8
6 6
0 1 2 3 4 5 6
14 12 7 10 8 6
1 9
2 6
3 3
4 5
0 1 2 3 4
9 6 3 5
1 30
2 25
0 1 2
30 25


minimum öncelikli kuyruk örnekleri
1 2
2 7
3 4
4 10
5 8
6 6
0 1 2 3 4 5 6
2 7 4 10 8 6
1 10
2 20
3 83
4 50
0 1 2 3 4
10 20 83 50
1 11
2 21
0 1 2
11 21

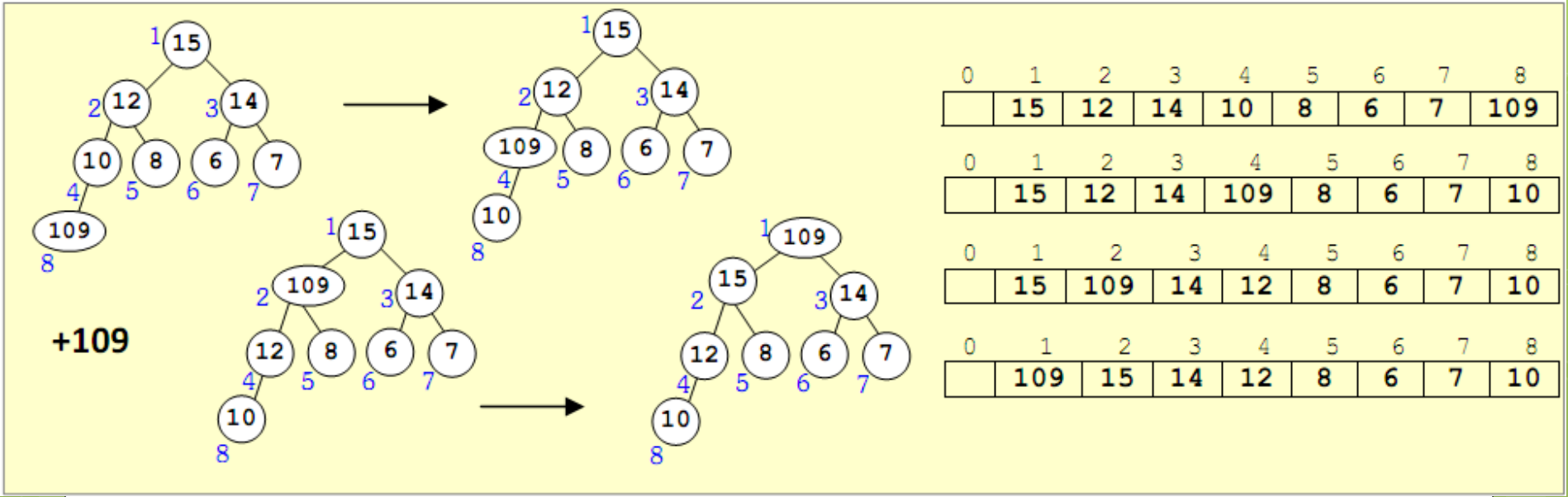
Maksimum Öncelikli Kuyruğa Öğe Ekleme

- A[1..n]
- Sırasıyla 15, 109, 107, 3, 15 değerleri eklenecektir. En iyi durumda Ω(1)'de de ekleme yapılır (3'ün eklemesi).
- Root at A[1]
- Parent of A[i] = A[⌊i/2⌋]
- Left child of A[i] = A[2i]
- Right child of A[i] = A[2i + 1]
- Mevcut tüm elemanlardan daha büyük bir öğe eklendiğinde ise yeni öğenin köke kadar kaydırılması gerekeceği için O(lgn)'de ekleme yapıldığını görüyoruz (109'un eklenmesi). Yani karmaşıklığın üst sınırı **lgn**, alt sınırı ise **1** olur. Gerçekte karmaşıklık bu ikisi arasında değişebilir. Bu durumda zaman karmaşıklığı **O(lgn)** 'dir.

Maksimum Öncelikli Kuyruğa Öğe Ekleme

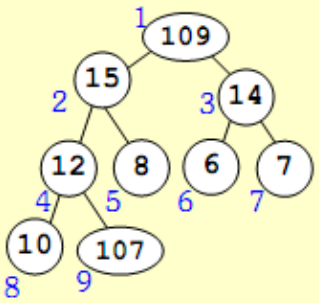


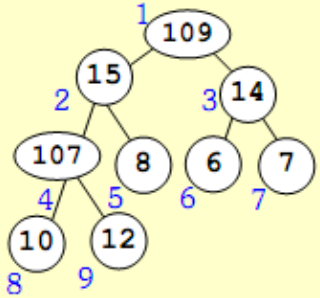
0	1	2	3	4	5	6	7
	14	12	7	10	8	6	
	14	12	7	10	8	6	15
	14	12	15	10	8	6	7
	15	12	14	10	8	6	7

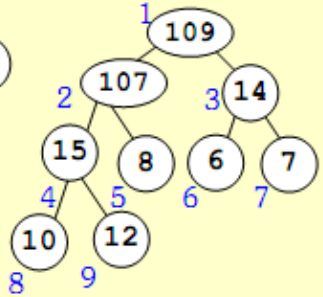


0	1	2	3	4	5	6	7	8
	15	12	14	10	8	6	7	109
	15	12	14	109	8	6	7	10
	15	109	14	12	8	6	7	10
	109	15	14	12	8	6	7	10

Maksimum Öncelikli Kuyruğa Öğe Ekleme





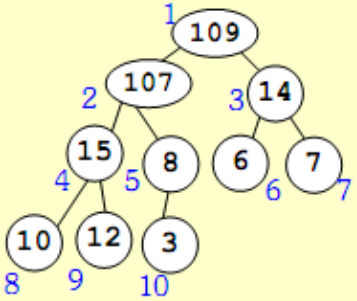


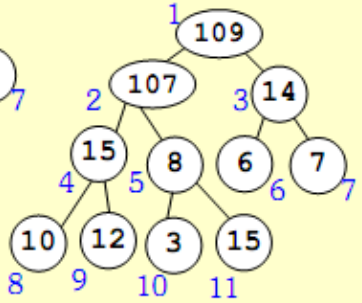
+107

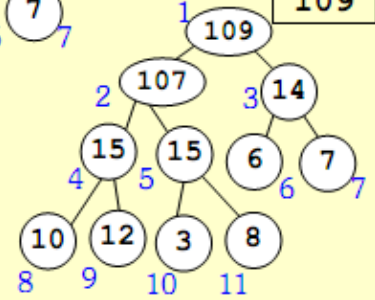
0	1	2	3	4	5	6	7	8	9
	109	15	14	12	8	6	7	10	107

0	1	2	3	4	5	6	7	8	9
	109	15	14	107	8	6	7	10	12

0	1	2	3	4	5	6	7	8	9
	109	107	14	15	8	6	7	10	12







+3 +15

0	1	2	3	4	5	6	7	8	9	10
	109	107	14	15	8	6	7	10	12	3

1	2	3	4	5	6	7	8	9	10	11
109	107	14	15	8	6	7	10	12	3	15

1	2	3	4	5	6	7	8	9	10	11
109	107	14	15	15	6	7	10	12	3	8

Maksimum Öncelikli Kuyruk fonksiyonları

```
HeapControl(dizi) //yığının boş olup olmadığı
{
  if (heapsize != dizi.Length)
     return true;
  return false;
}

Left(i)  { return 2 * (i + 1) - 1; }
Right(i) { return 2*(i+1); }
Parent(i){ return (i-1)/2;}
```

Maksimum Öncelikli Kuyruğa Öğe Ekleme

```
HeapInsert(dizi, key)
{ if (HeapControl(dizi))
  {i = heapsize;
     heapsize++;
    while (i > 0 && dizi[Parent(i)] < key)
    { dizi[i] = dizi[Parent(i)];
      i = Parent(i);
    }
    dizi[i] = key;
  }
}
```

Maksimum Öncelikli Kuyruğu İnşa Etme

```
BuildHeap(dizi)
{
   heapsize = dizi.Length - 1;

   for ( i = (dizi.Length-1)/2;i>=0; i--)

        heapify(dizi, i);
}
```

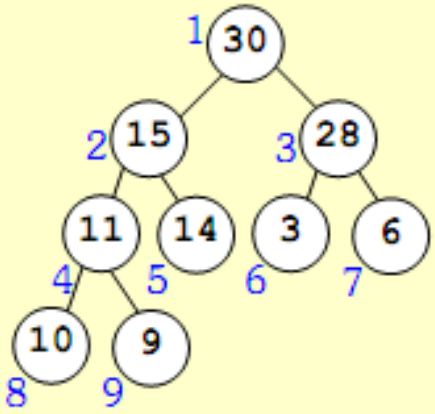
Maksimum Öncelikli Kuyruğu oluşturma

```
heapify(dizi, i)
{ L = Left(i); R = Right(i);
 if (L <= heapsize && dizi[L] > dizi[i])
    largest = L;
 else  largest = i;
 if (R <= heapsize && dizi[R] > dizi[largest])
    largest = R;
 if (largest != i)
  {temp = dizi[largest]; dizi[largest] = dizi[i];
   dizi[i] = temp;  heapify(dizi, largest);
  }
}
```

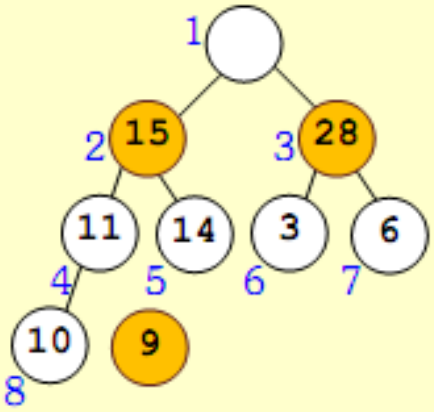
Maksimum Öncelikli Kuyruktan Öğe Alma/Silme

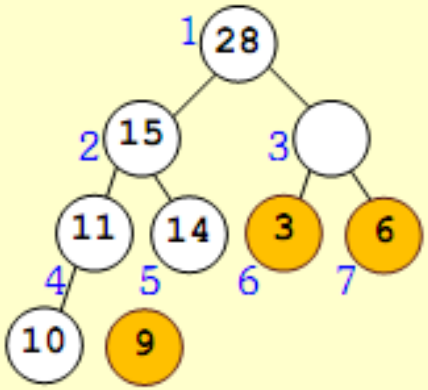
- Yığını diziyle gerçekleştirmiştik. Yığının tanımı gereği ilk alınacak eleman, dizideki ilk elemandır. Bunun yanında “tam ağaç” yapısını bozmamak için en sondaki elemanı da uygun bir konuma kaydırmalıyız.
- Bu kaydırmayı yaparken maksimum yığın yapısını korumaya dikkat etmeliyiz.
- Her adımda (iki çocuk ve sondaki eleman olmak üzere) dikkat edilecek 3 değer var. İki gösterge gibi; i ile, en sondaki elemanı kaydıracağım konumu ararken; j ile de üzerinde bulunduğum düğümün çocuklarını kontrol ediyorum. Bu üç değerden (iki çocuk ve sondaki eleman) en büyük olanı, i’nin gösterdiği yere taşıyorum. Sonra da taşınan değerin eski konumuna ilerleyip aynı kıyaslamaları yapıyorum. En sonunda, sondaki düğüm bir yere yerleşene kadar. En sondaki öğe bazı durumlarda arada bir yere girebilir; en kötü durumda ise arada bir yere yerleşemez, en sona konması gerekir. O(lgn).

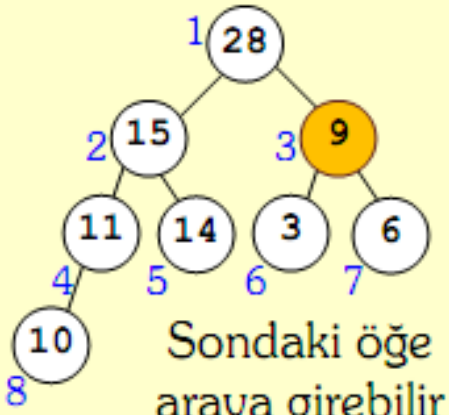
Maksimum Öncelikli Kuyruktan Öğe Alma/Silme



30, 15, 28, 11, 14, 3, 6, 10, 9
verisi üzerinde silme işlemleri



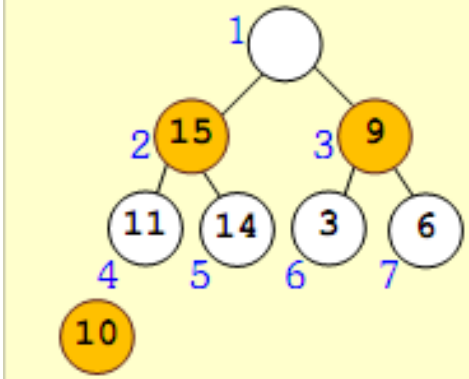


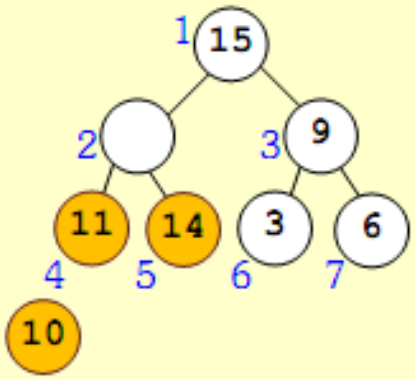


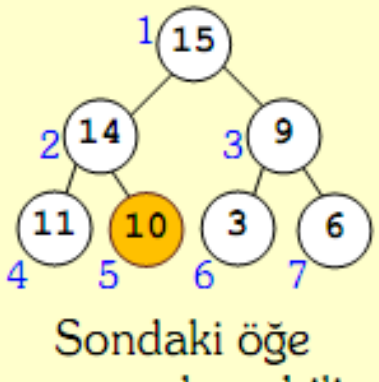
Sondaki öğe
araya girebilir

Maksimum Öncelikli Kuyruktan Öğe Alma/Silme

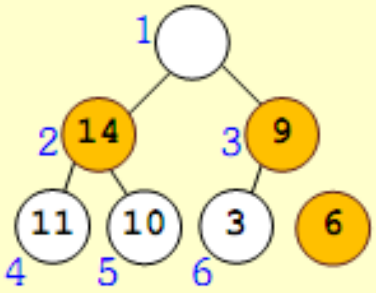
30, 15, 28, 11, 14, 3, 6, 10, 9
verisi üzerinde silme işlemleri

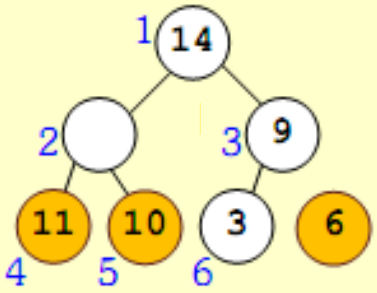


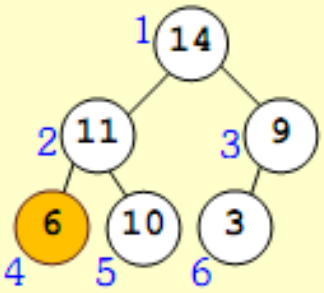




Sondaki öğe
en sona konabilir

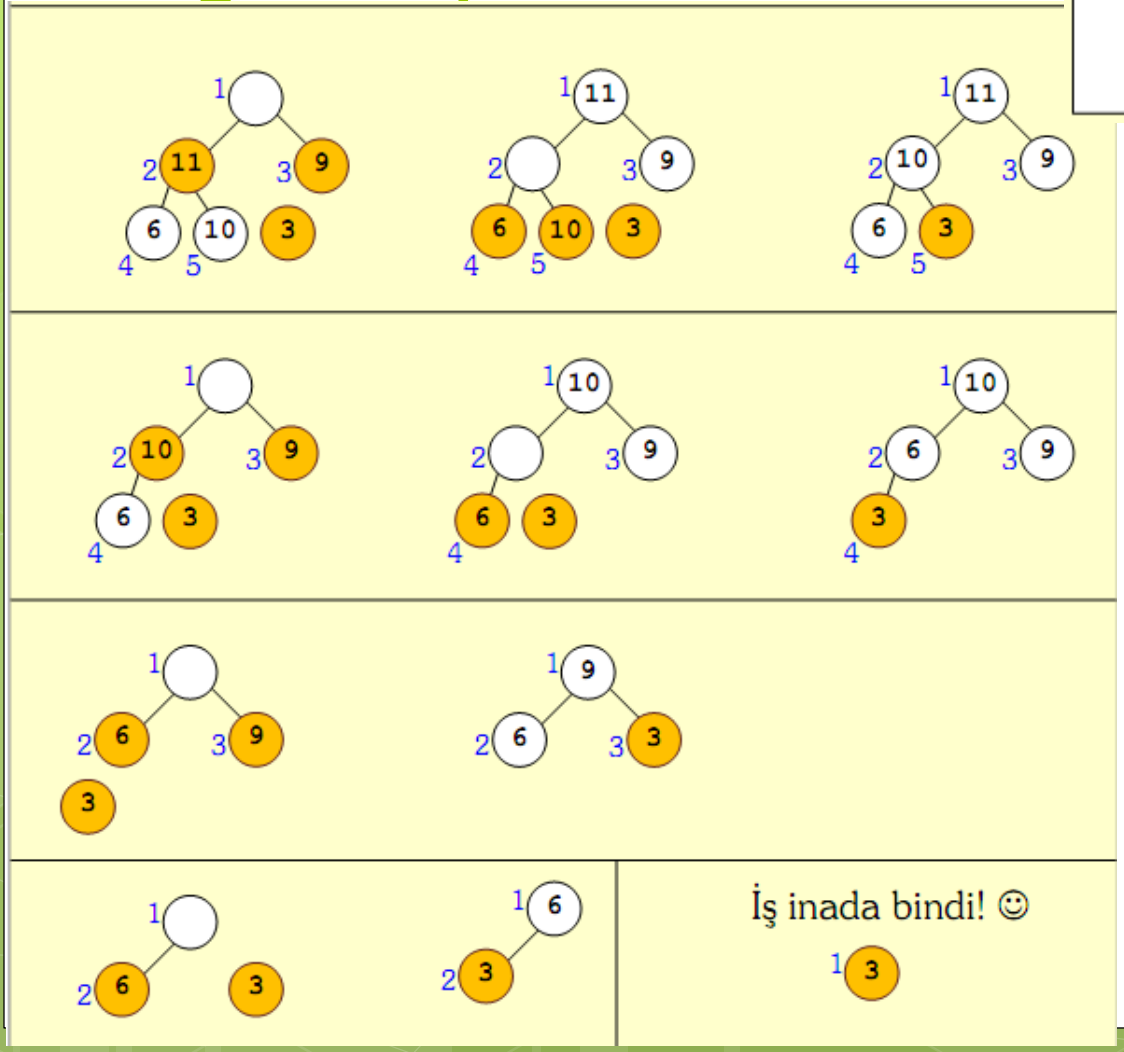






Maksimum Öncelikli Kuyruktan Öğe Alma/Silme

30, 15, 28, 11, 14, 3, 6, 10, 9
verisi üzerinde silme işlemleri



Maksimum Öncelikli Kuyruk Java

```
HeapExtract_or_Sort(dizi)
{ BuildHeap(dizi);
    for (i = heapsize; i >=0; i--)
      {       temp = dizi[0];
               dizi[0] = dizi[i];
               dizi[i] = temp;
               heapsize--;
               heapify(dizi, 0);
          }
}
```

Yazan:Şadi Evren ŞEKER

Çok Yollu Ağaçlar

(Multi-Way Trees)

Çok Yollu Ağaçlar (Multi-Way Trees)

- B -Trees
- B* -Trees
- B+ -Trees
- B# -Trees

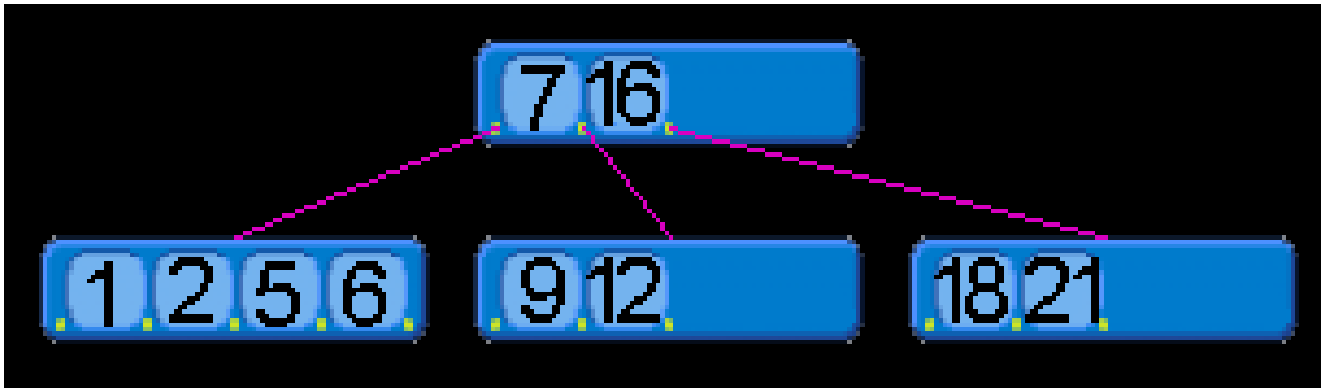
Çok Yollu Ağaçlar (Multi-Way Trees)

- Disk üzerindeki bilgilerin elde edilmesinde kullanılır.
- 3600 rpm ile dönen bir disk için bir tur 16.7ms'dir.
- Ortalama olarak 8 ms'de (gecikme zamanı) istediğimiz noktaya konumlanırız.
- Saniyede yaklaşık 125 kez diskte konumlanabiliriz.
- Bir saniyede 25 milyon komut gerçekleştirebiliriz.
- Bir disk erişimi yaklaşık 200.000 komut zamanı almaktadır.
- (**7200 Rpm** disk **bir turunu** 8.3 **ms**'de tamamlar, ortalama gecikme süresi 4 ms dir.

- **Multi-Way ağaçlar disk erişim sayısını azaltmayı amaçlamaktadır.**

Çok Yollu Ağaçlar (Multi-Way Trees)

- Bir multi-way ağaç sıralı bir ağaçtır ve aşağıdaki özelliklere sahiptir.
  - Bir m-way arama ağacındaki her node, m-1 tane anahtar (key) ve m tane çocuğa sahiptir.
  - Bir node'daki anahtar, sol alt ağaçtaki tüm anahtarlardan büyüktür ve sağ alt ağaçtaki tüm anahtarlardan küçüktür.



Çok Yollu Ağaçlar -B-Trees

- Root (kök) node en az iki tane yaprak olmayan node'a sahiptir.
- Yaprak ve kök olmayan her node k-1 tane anahtara ve k adet alt ağaç referansına sahiptir. ($m/2 \leq k \leq m$)
- Her yaprak node'u k -1 anahtara sahiptir. ($m/2 \leq k \leq m$)
- Bütün yapraklar aynı seviyededir.
- Herhangi bir döndürmeye gerek kalmadan (AVL tree deki gibi) otomatik olarak balance edilirler.

Çok Yollu Ağaçlar -B-Trees

- **Bir anahtar ekleme;**

1. Eğer boş alanı olan bir yaprağa yerleştirilecekse doğrudan yaprağın ilgili alanına yerleştirilir.

2. Eğer ilgili yaprak doluysa, yaprak ikiye bölünür ve anahtarların yarısı yeni bir yaprak oluşturur. Eski yapraktaki en son anahtar bir üst seviyedeki node' aktarılır ve yeni yaprağı referans olarak gösterir.

3. Eğer root ve tüm yapraklar doluysa, önce ilgili yaprak ikiye bölünür ve eski yapraktaki en son anahtar root'a aktarılır. Root node'da dolu olduğu için ikiye bölünür ve eski node'daki en son anahtar root yapılır.

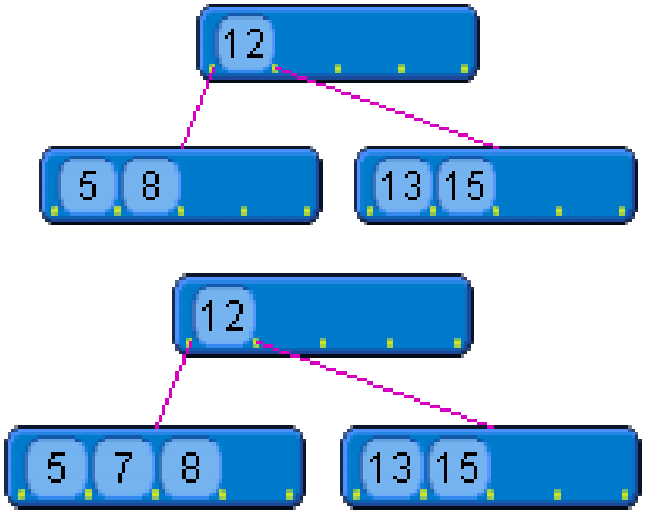
Çok Yollu Ağaçlar -B-Trees

- B -Tree Oluşturulması
- class BtreeNodeC
- {
- public int m = 4;
- public bool yaprak = true;
- public int[] keys = new int[m-1];
- BtreeNodeC[] referanslar = new BtreeNodeC[m];
- public BtreeNodeC(int key)
- {
- this.keys[0] = key;
- for (int i=0; i<m; i++)
- referanslar[i] = null;
- }
- }

GENEL ÖRNEK:
http://www.jbixbe.com/doc/tutorial/BTree.html

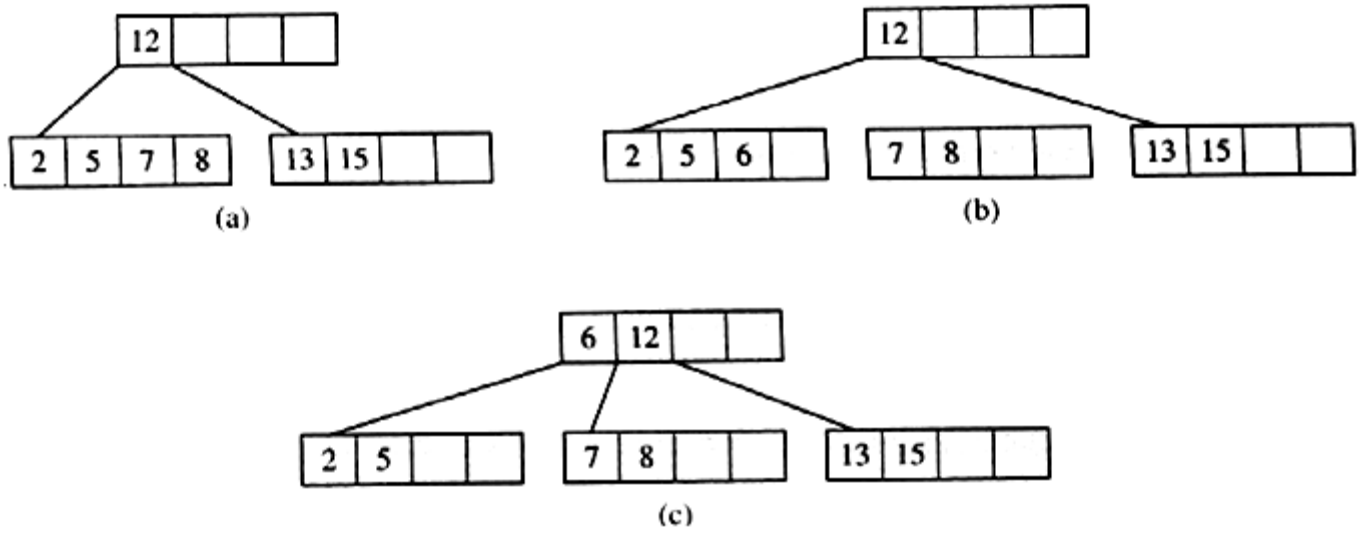
Çok Yollu Ağaçlar -B-Trees

- **1- Yerleştirilecek yaprak boş ise,**
- Örnek olarak 7'nin eklenmesi



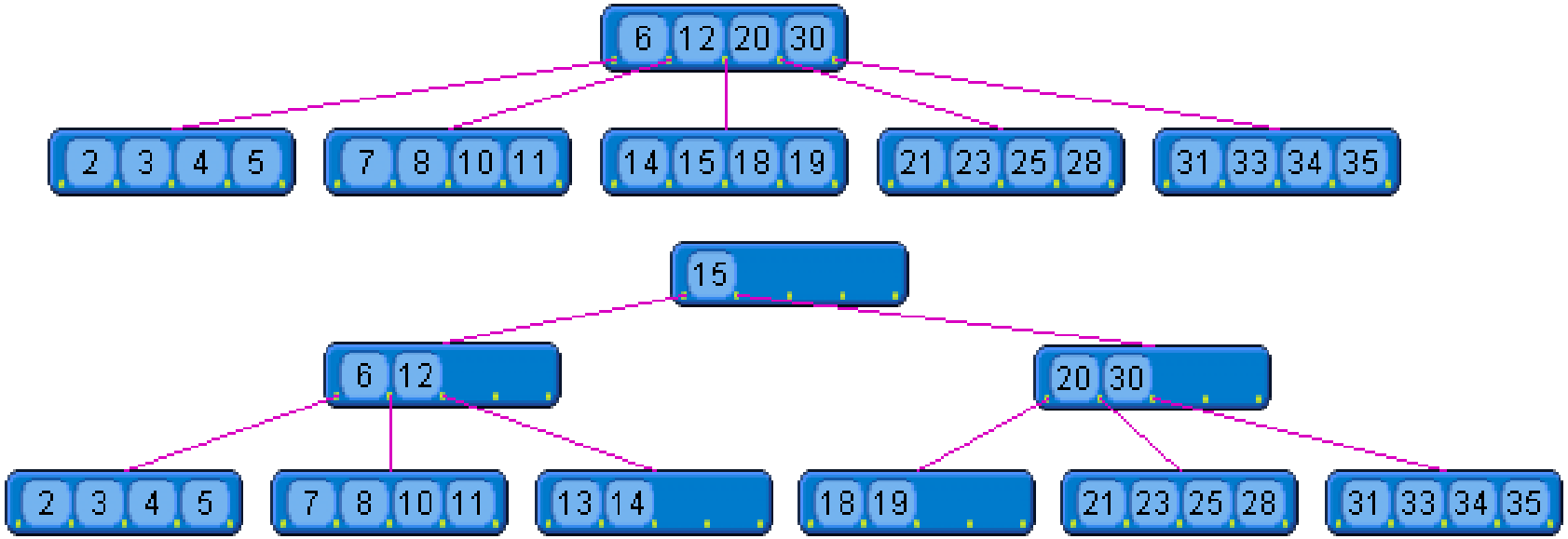
Çok Yollu Ağaçlar -B-Trees

- **2- Yerleştirilecek yaprak dolu ise,**
- Örnek: Anahtar olarak 6 eklenmesi



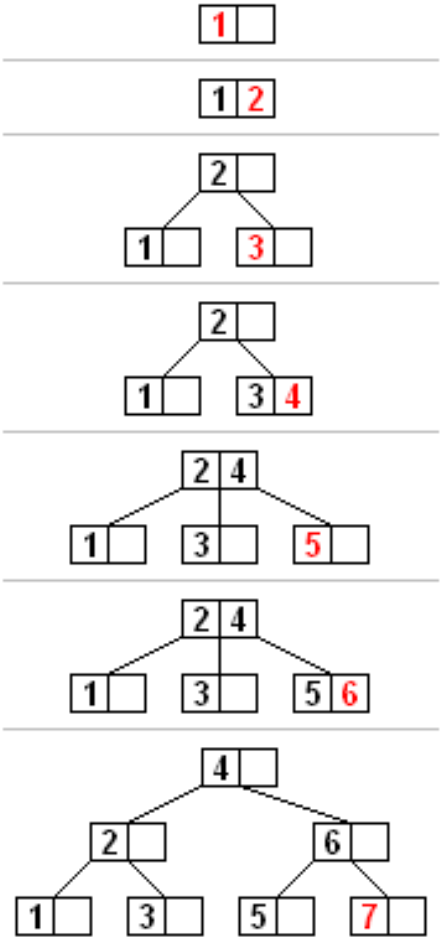
Çok Yollu Ağaçlar -B-Trees

- **3-Yerleştirilecek yaprak dolu ve root node'da dolu ise,**
- Örnek: Anahtar olarak 13 eklenmesi



Çok Yollu Ağaçlar -B-Trees

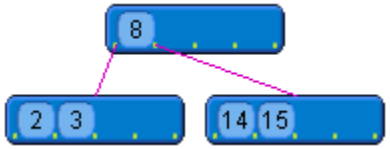
- Örnek



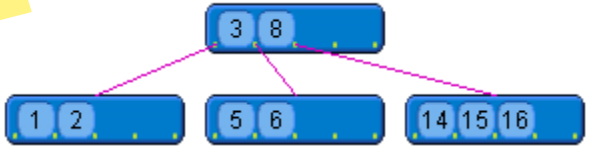
Çok Yollu Ağaçlar -B-Trees

- Örnek:5.Derece bir M-Way ağaca anahtar ekleme
- Ekle: 2,8,14,15

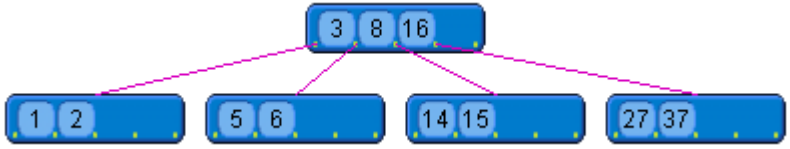
- Ekle:3



- Ekle:1,16,6,5



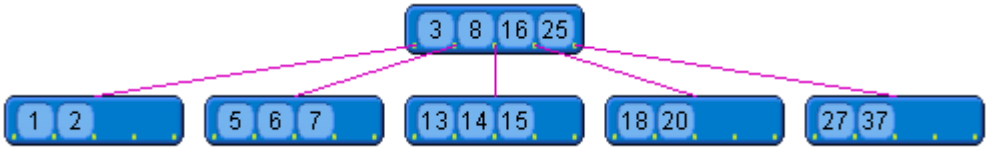
- Ekle:27,37



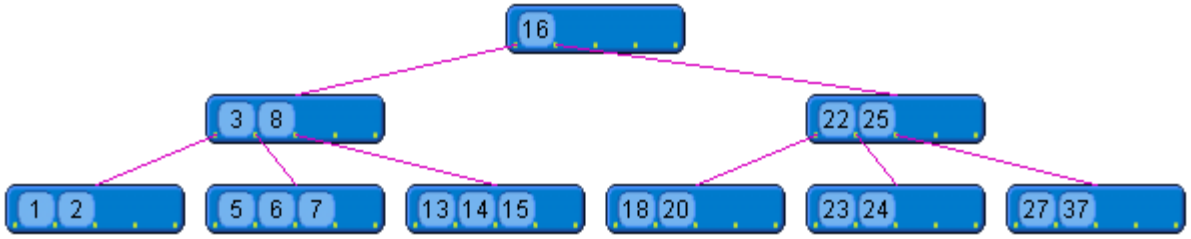
Çok Yollu Ağaçlar -B-Trees

- Örnek:5.Derece bir M-Way ağaca anahtar ekleme

- Ekle: 18,25,7,13,20

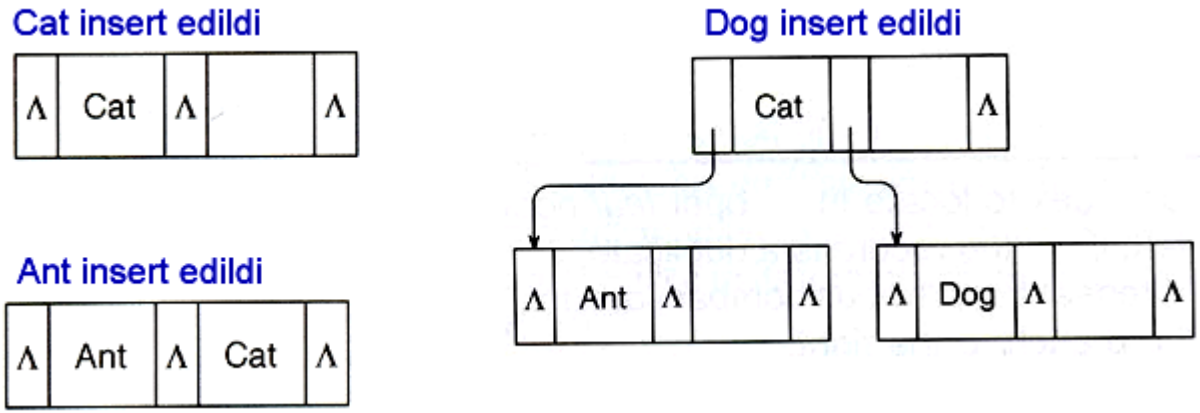


- Ekle: 22,23,24



Çok Yollu Ağaçlar -B-Trees

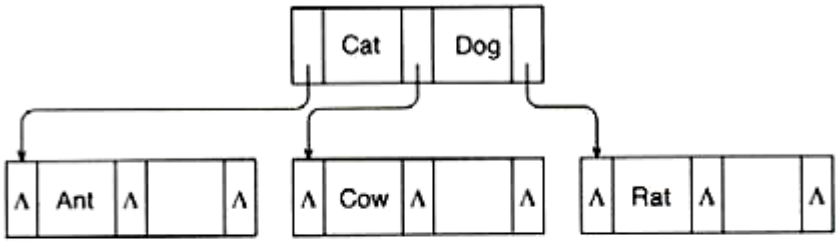
- Örnek
- *d = 1* (capacity order)
- *cat, ant, dog, cow, rat, pig, gnu*



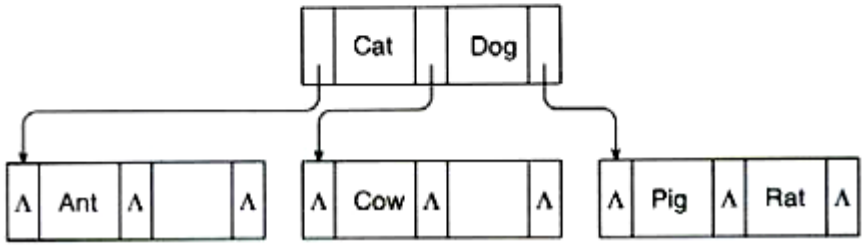
Çok Yollu Ağaçlar -B-Trees

- Örnek
- *d = 1* (capacity order)
- *cat, ant, dog, cow, rat, pig, gnu*

Cow ve Rat insert edildi

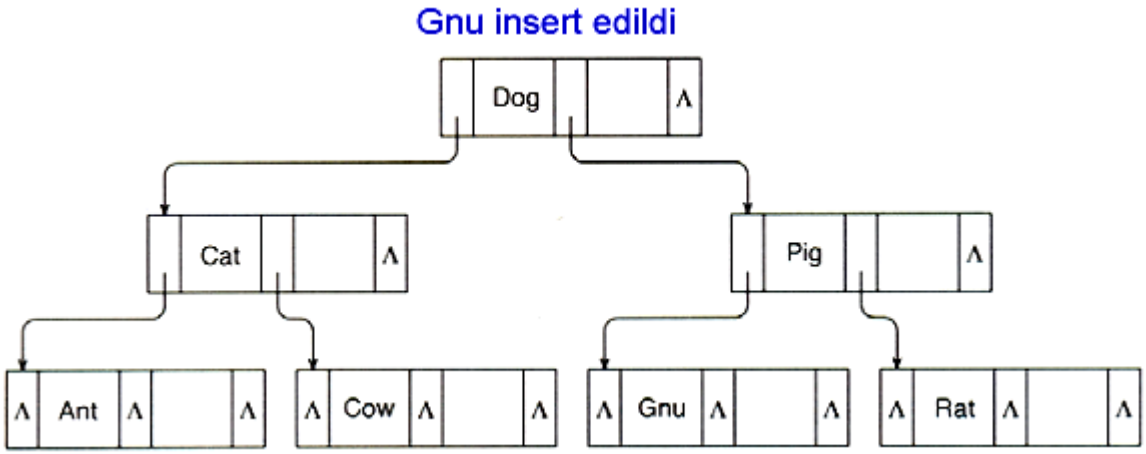


Pig insert edildi



Çok Yollu Ağaçlar -B-Trees

- Örnek
- *d = 1* (capacity order)
- *cat, ant, dog, cow, rat, pig, gnu*

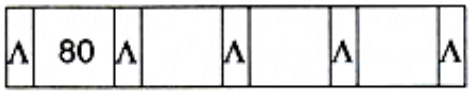


- Doldurma faktörü = depolanan kayıt sayısı / kullanılan yer sayısı
- = 7 / 14 = 50 %

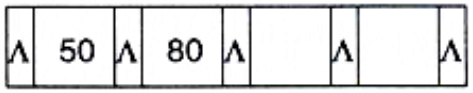
Çok Yollu Ağaçlar -B-Trees

- Örnek
- *d = 2* (capacity order)
- *80,50,100,90,60,65,70,75,55,64,51,76,77,78,200,300,150*

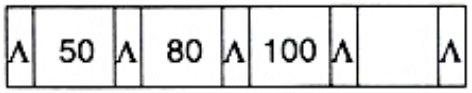
1.Insert 80.



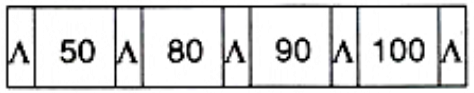
2. Insert 50.



3. Insert 100.

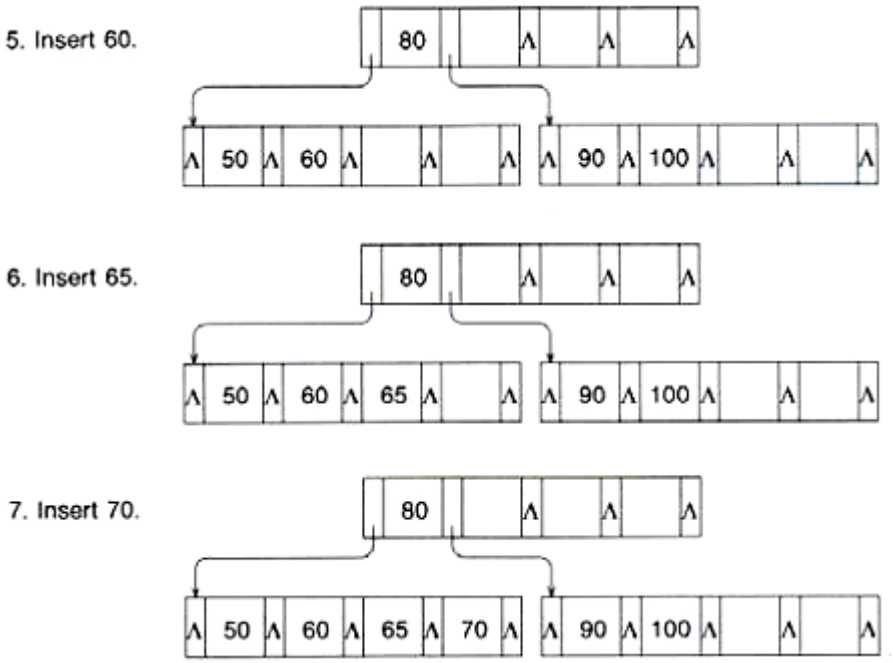


4. Insert 90



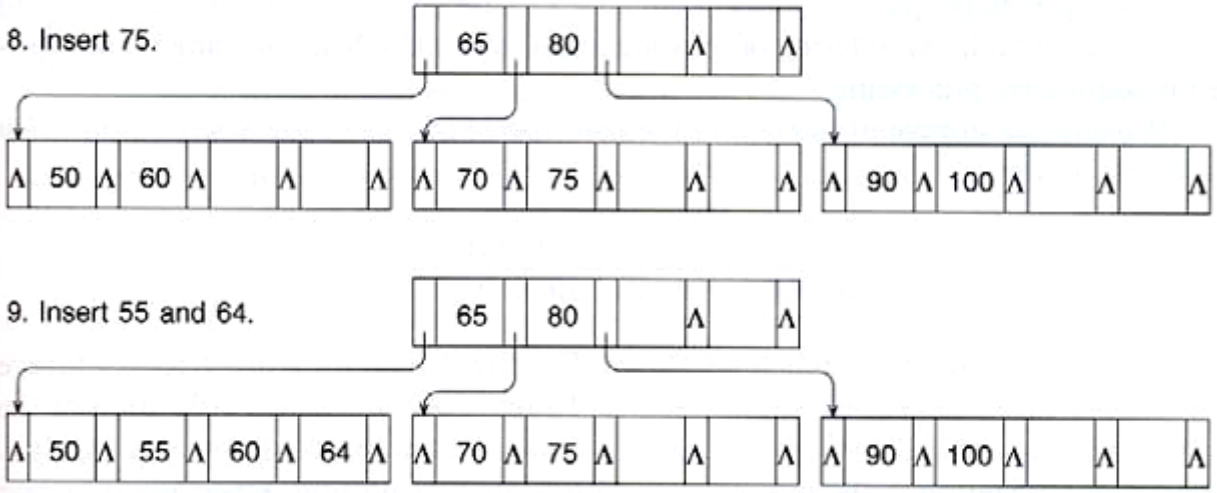
Çok Yollu Ağaçlar -B-Trees

- Örnek
- *d = 2* (capacity order)
- *80,50,100,90,60,65,70,75,55,64,51,76,77,78,200,300,150*



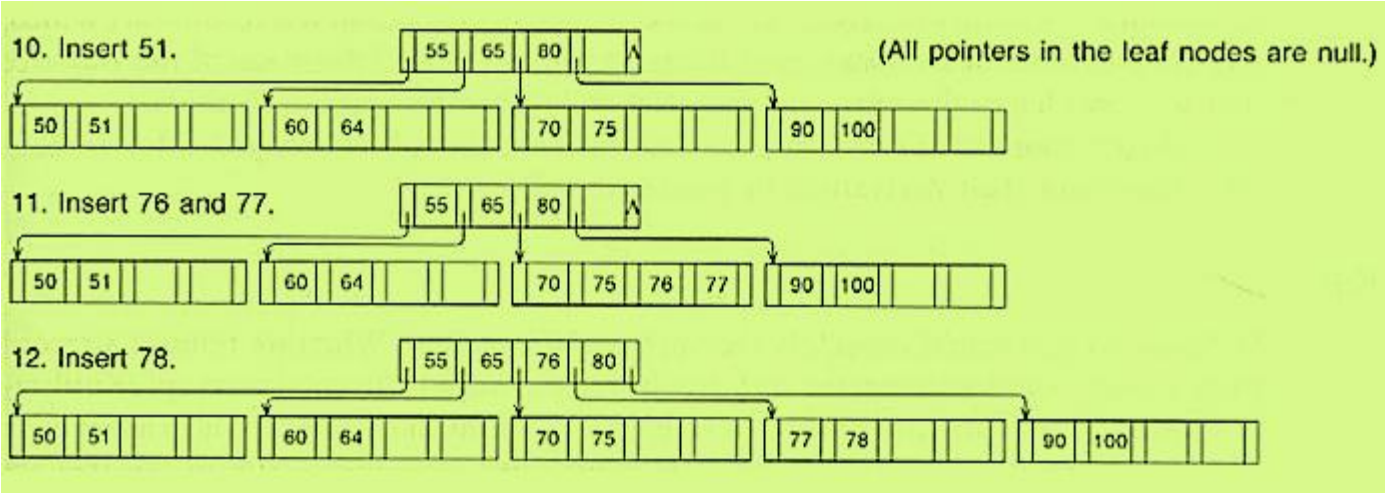
Çok Yollu Ağaçlar -B-Trees

- Örnek
- *d = 2* (capacity order)
- *80,50,100,90,60,65,70,75,55,64,51,76,77,78,200,300,150*



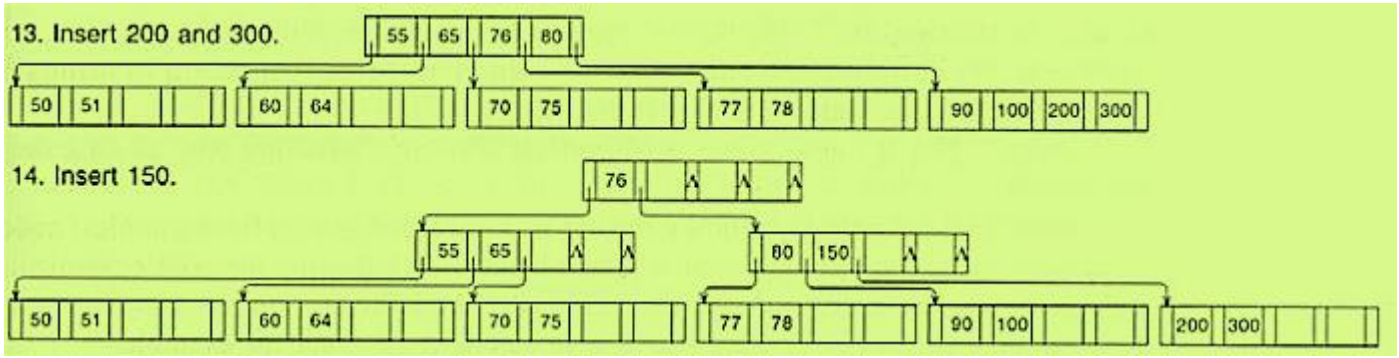
Çok Yollu Ağaçlar -B-Trees

- Örnek
- *d = 2* (capacity order)
- *80,50,100,90,60,65,70,75,55,64,51,76,77,78,200,300,150*



Çok Yollu Ağaçlar -B-Trees

- Örnek
- *d = 2* (capacity order)
- *80,50,100,90,60,65,70,75,55,64,51,76,77,78,200,300,150*

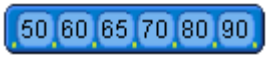


- *Packing factor = depolanan kayıt sayısı / kullanılan yer sayısı*
- *= 17 / 36 = 47 %*

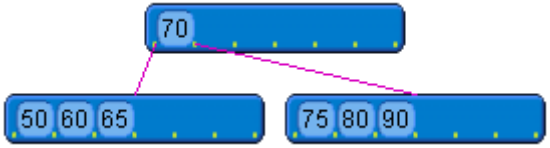
Çok Yollu Ağaçlar -B-Trees

- Örnek
- *d = 3* (capacity order)
- *80,50,90,60,65,70,75,55,64,51,76,77,78,10,13,15,1,3,5,6,20,32*

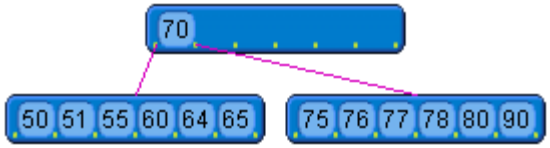
- *80,50,90,60,65,70*



- *75*



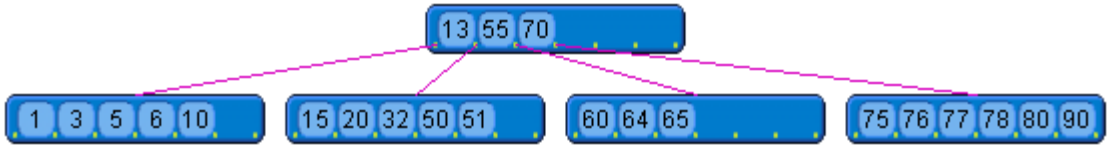
- *55,64,51,76,77,78*



Çok Yollu Ağaçlar -B-Trees

- *10*

- *13,15,1,3,5,6,20,32*



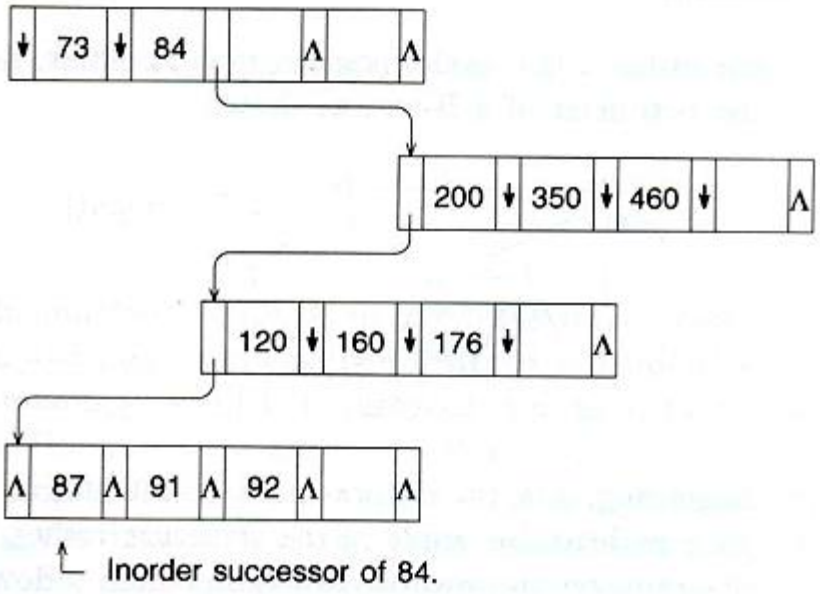
- *Packing factor = depolanan kayıt sayısı / kullanılan yer sayısı*
- *= 22 / 30 = 73 %*

B-Trees

Silme

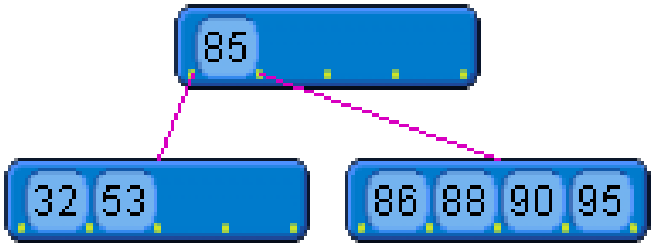
Çok Yollu Ağaçlar -B-Trees

- **Kural 1:** Minumum kapasitenin üzerindeki yapraklardan kayıt rahatlıkla silinebilir.
- **Kural 2:** Bir yaprak olmayan node üzerinden kayıt silindiğinde inorder takipçisi yerine yazılır. (Not: Eklerken soldaki en büyük düğüm, silerken sağdaki en küçük düğüm alınıyor)

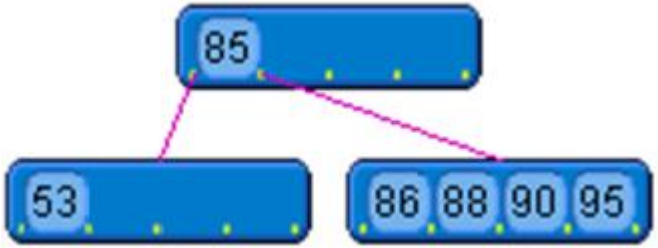


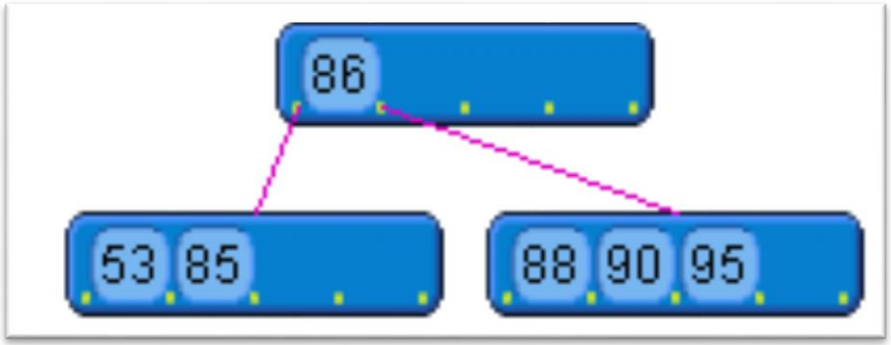
Çok Yollu Ağaçlar -B-Trees

- **Kural 3:** Bir node'daki kayıt sayısı minimum kapasite'den aşağı düşerse ve kardeş node'u fazla kayda sahipse, parent ve kardeş node ile yeniden düzenleme yapılır. Bir anlamda sola döndürme yapılır.



32 silindi.

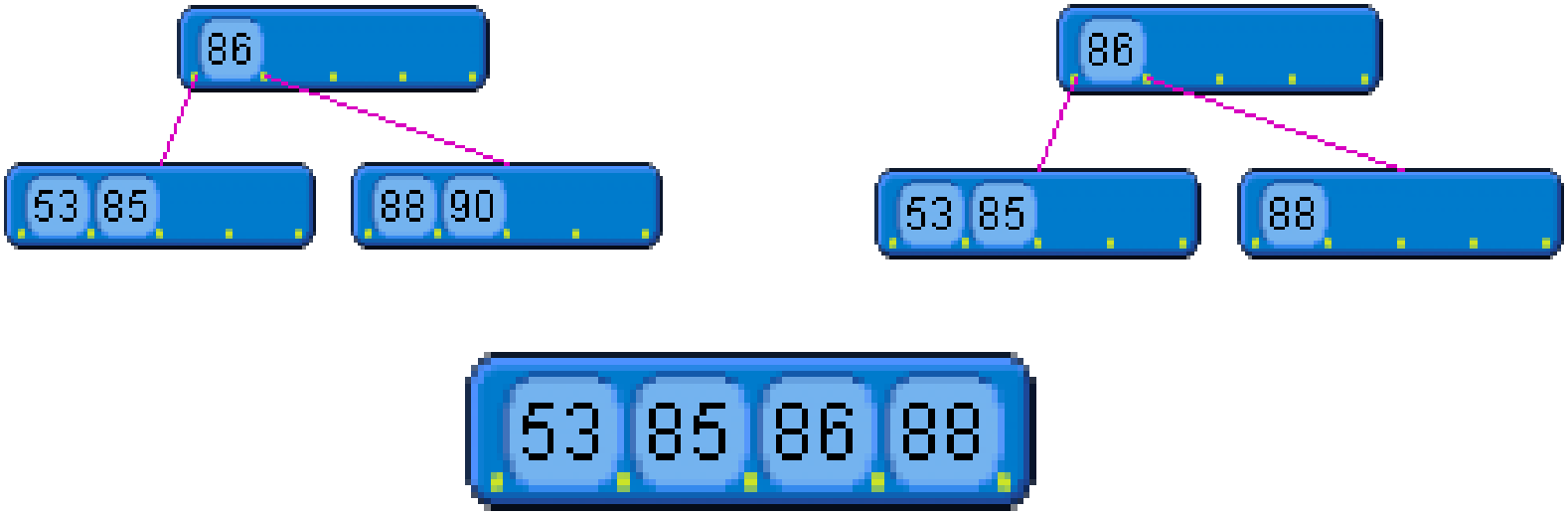




Çok Yollu Ağaçlar -B-Trees

- **Kural 4:** İki kardeş node minimum kapasitenin altına düşerse ikisi ve parent node'daki kayıt birleştirilir.

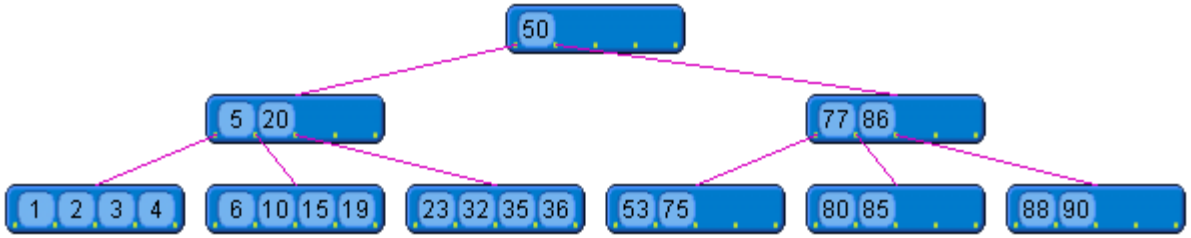
90 silindi



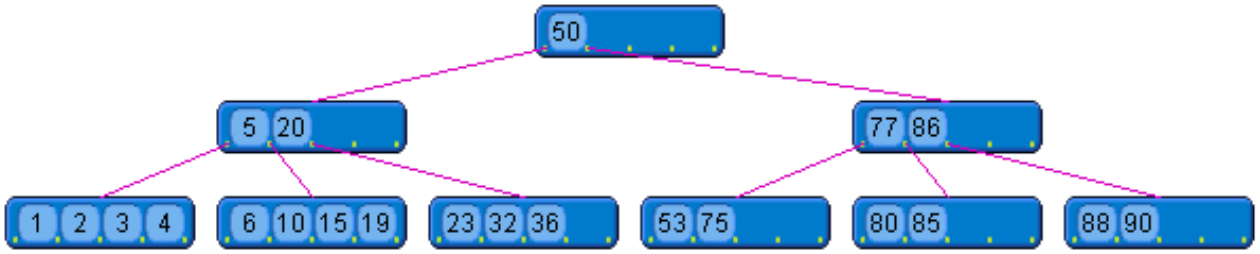
- *85 sağa döndürülürse soldaki minimum kapasitenin altına düşer*

Çok Yollu Ağaçlar -B-Trees

- **Örnek :1. Kural>** Minimum kapasitenin üstündeki yapraktan kayıt silinmesi

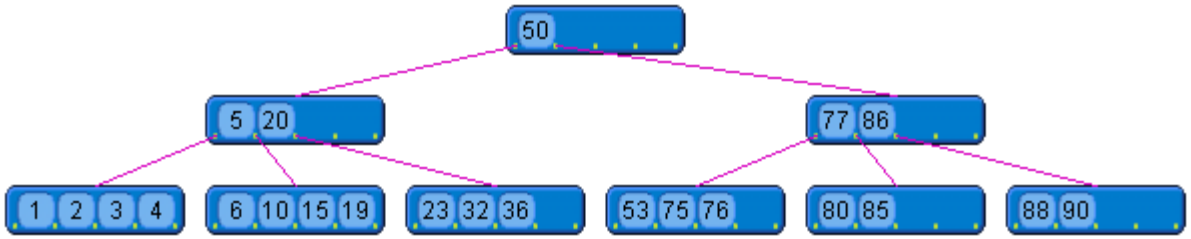


- *35 silindi.*

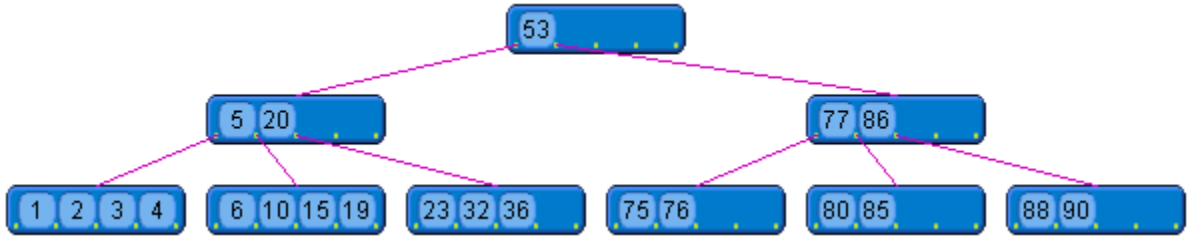


Çok Yollu Ağaçlar -B-Trees

- **Örnek : 2. Kural>** Bir nonleaf node'da kayıt silinmesi ve minimum kapasitenin üzerindeki bir node'dan kayıt aktarılması.

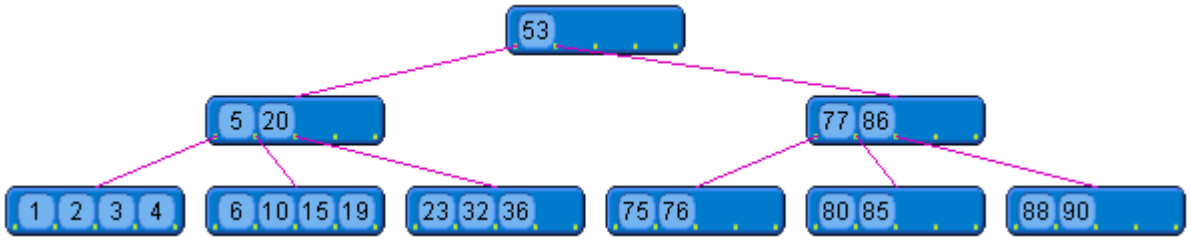


- 50 silindi

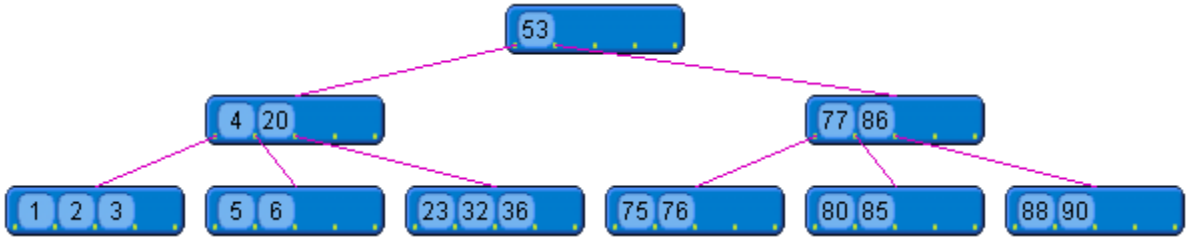


Çok Yollu Ağaçlar -B-Trees

- **Örnek :3.Kural>** Bir leaf node'da kayıt silinmesi ve minimum kapasitenin altına düşülmesi.

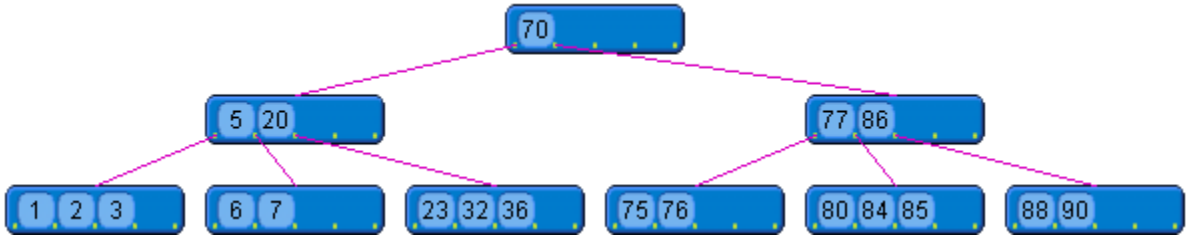


- 10,15,19 silinmesi (en fazla çocuğa sahip düğümden al)
- Sağa döndürme

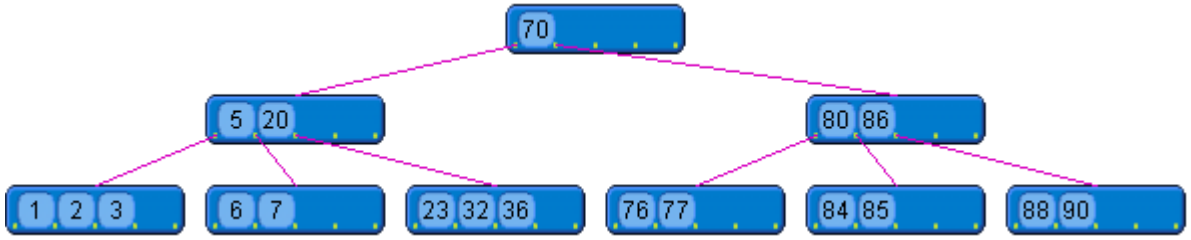


Çok Yollu Ağaçlar -B-Trees

- **Örnek :3.Kural>** Bir leaf node'da kayıt silinmesi ve minimum kapasitenin altına düşülmesi.

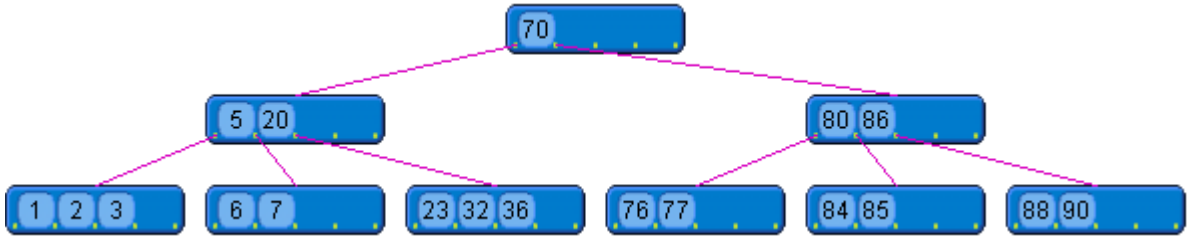


- 75 silindi. (silinen değer 77 solunda -Sola döndürme)

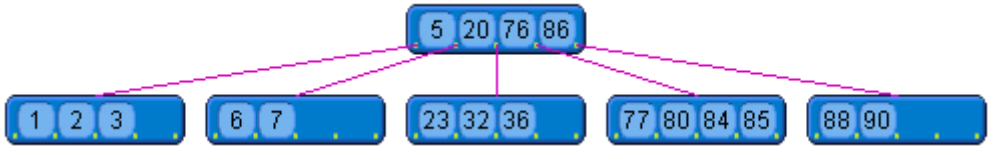


Çok Yollu Ağaçlar -B-Trees

- **Örnek:4.Kural >** Bir leaf node'da kayıt silinmesi ve minimum kapasitenin altına düşülmesi ve node'ların birleştirilmesi. (Kök dahil sağ taraftaki hangi düğüm silinirse silinsin minumum kapasitenin altına düşülecektir. Birleştirme işlemi gerekir.)

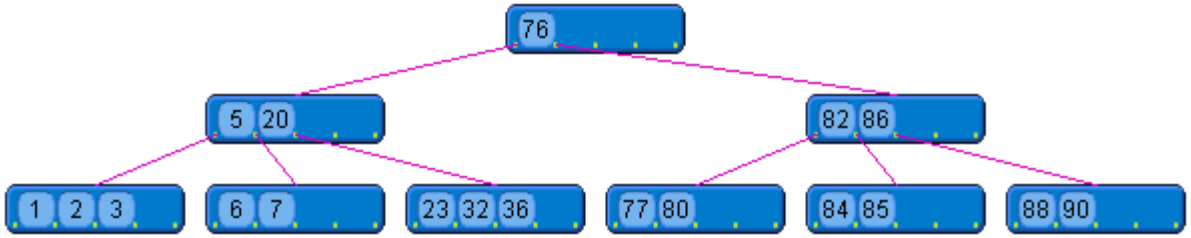


- 70 silindi

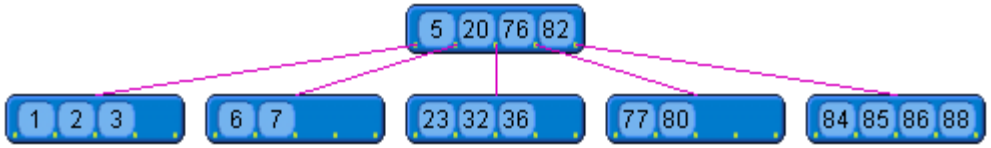


Çok Yollu Ağaçlar -B-Trees

- **Örnek:4.Kural >** Bir leaf node'da kayıt silinmesi ve minimum kapasitenin altına düşülmesi ve node'ların birleştirilmesi. (Kök dahil sağ taraftaki hangi düğüm silinirse silinsin minumum kapasitenin altına düşülecektir. Birleştirme işlemi gerekir.)

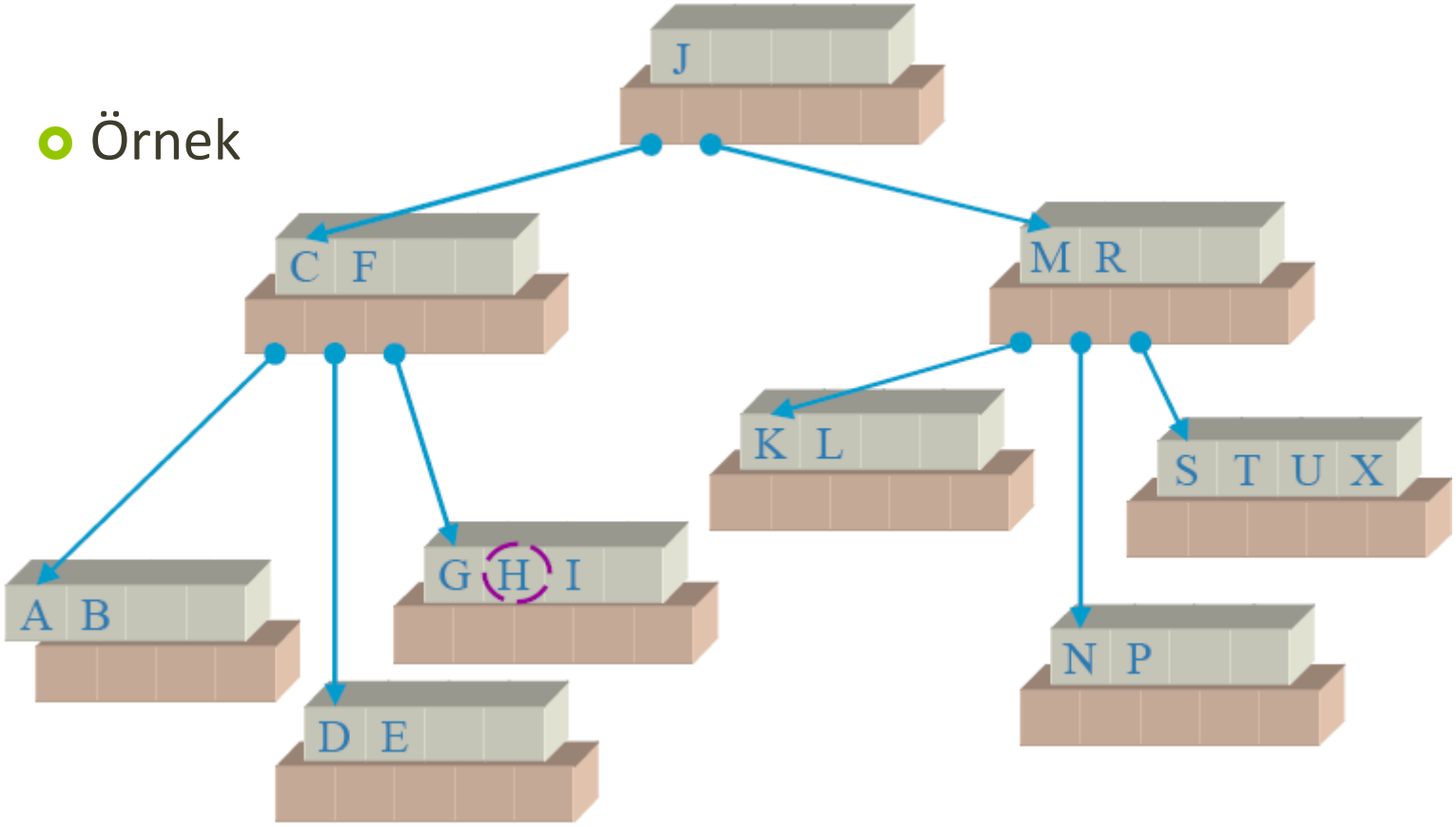


- 90 silindi

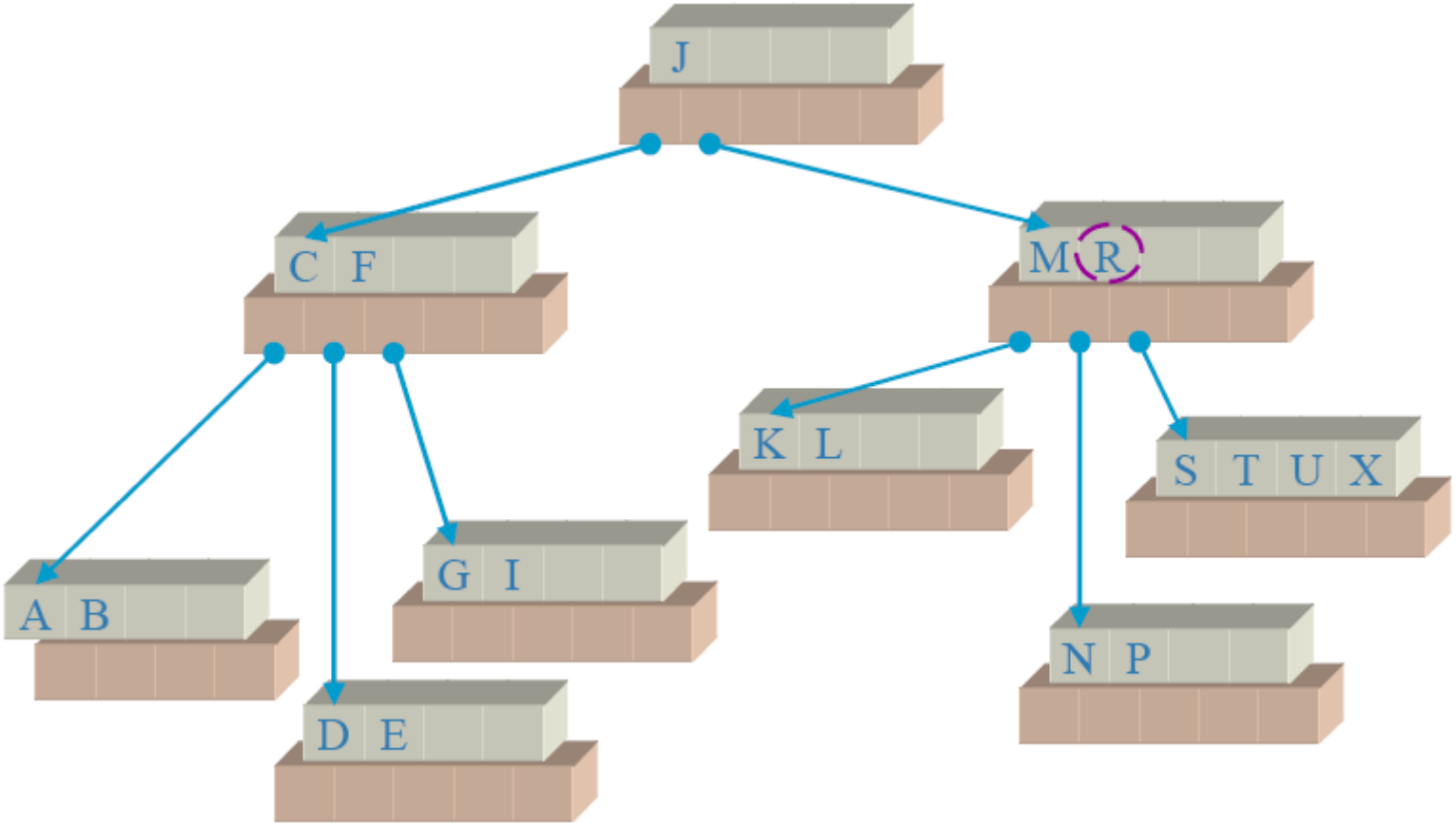


Çok Yollu Ağaçlar -B-Trees

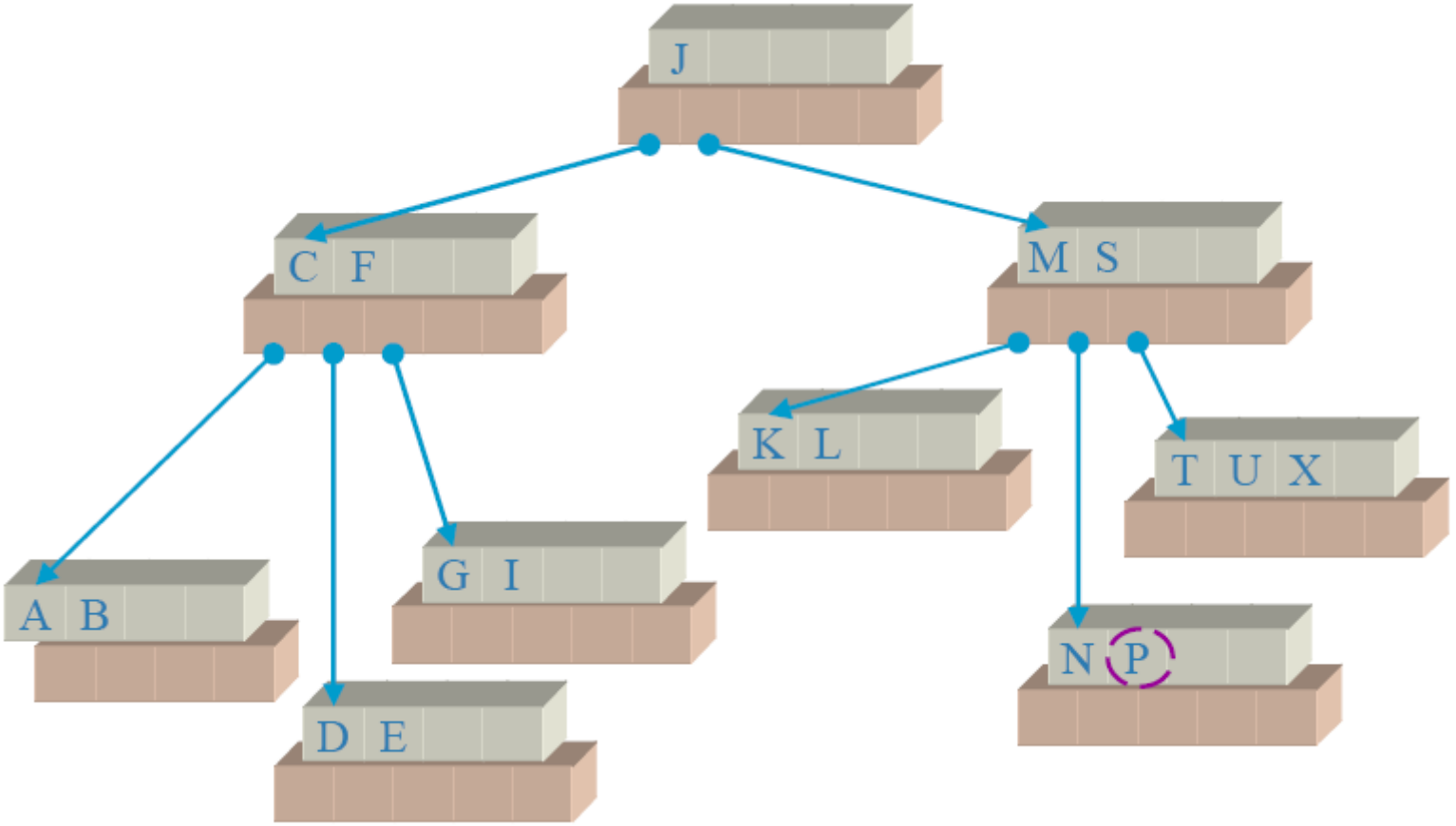
- Örnek



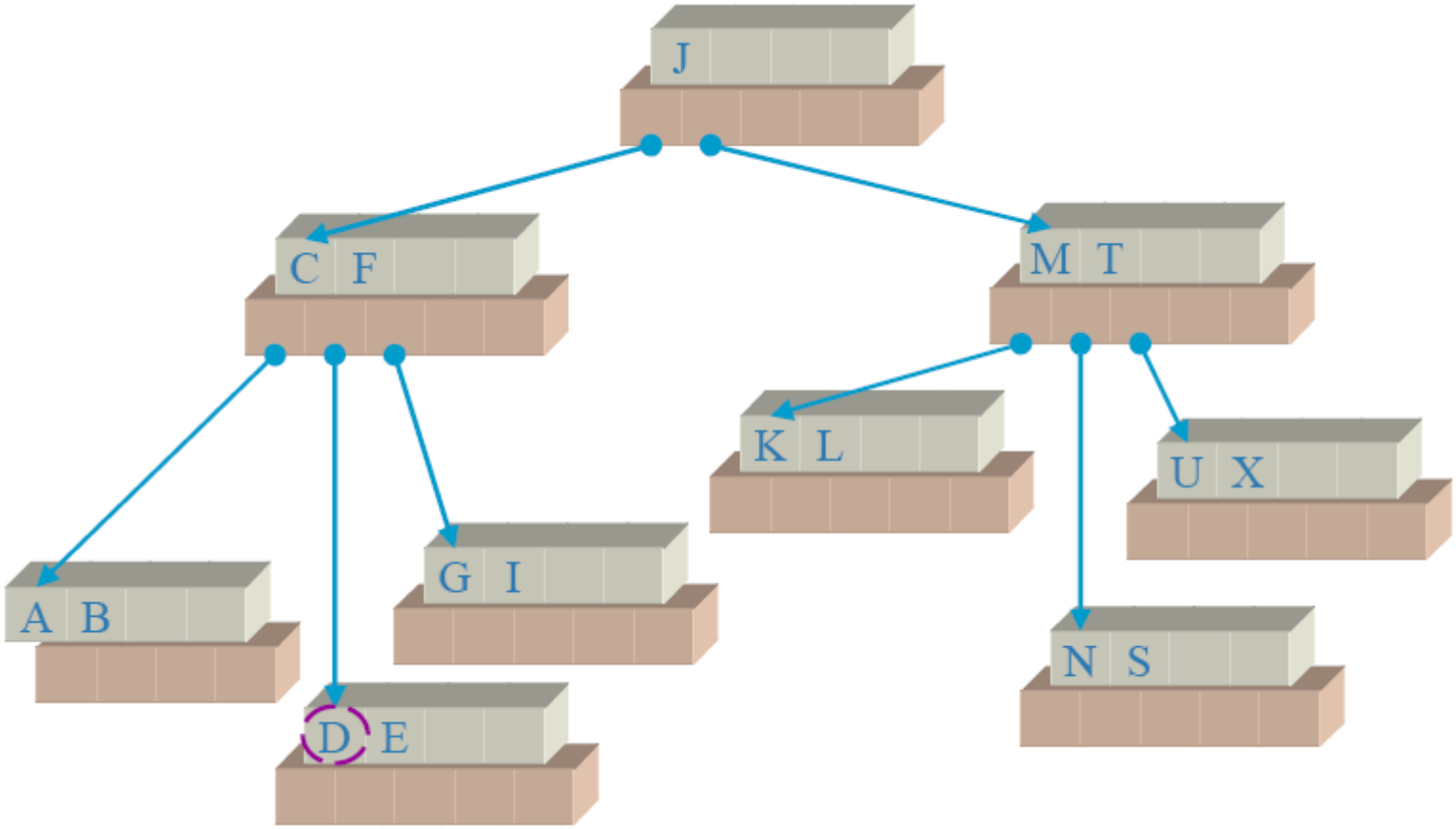
Çok Yollu Ağaçlar -B-Trees



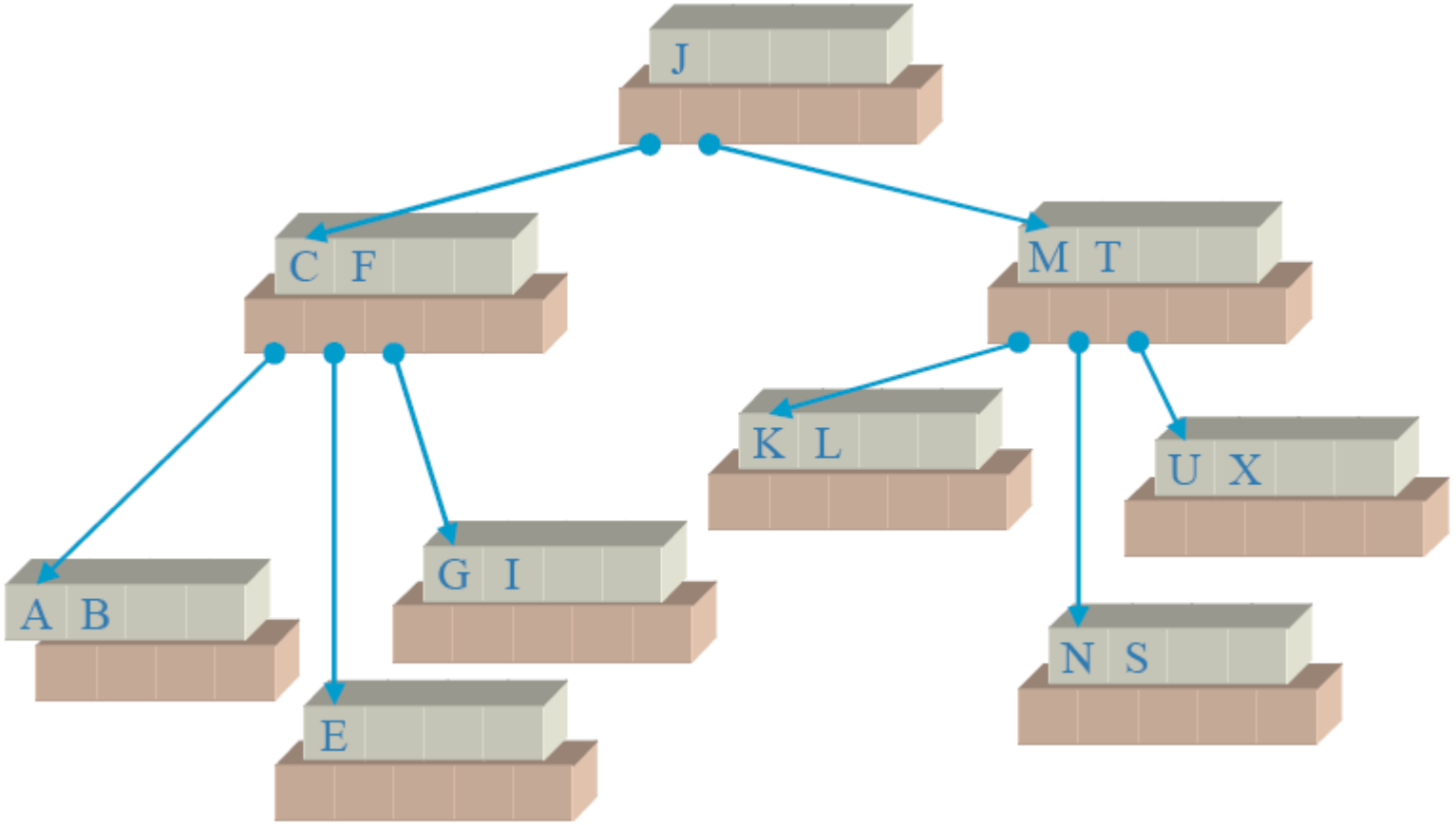
Çok Yollu Ağaçlar -B-Trees



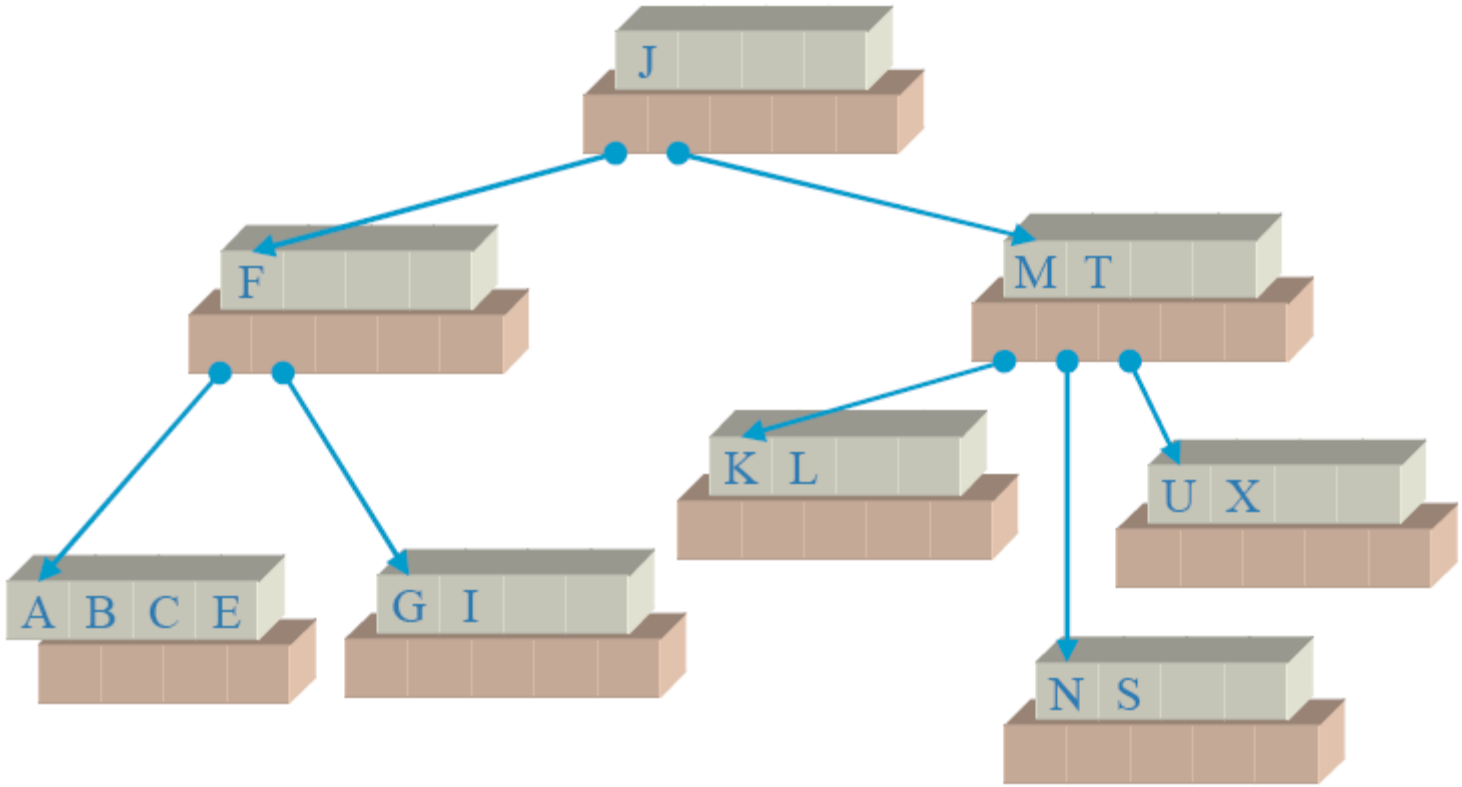
Çok Yollu Ağaçlar -B-Trees



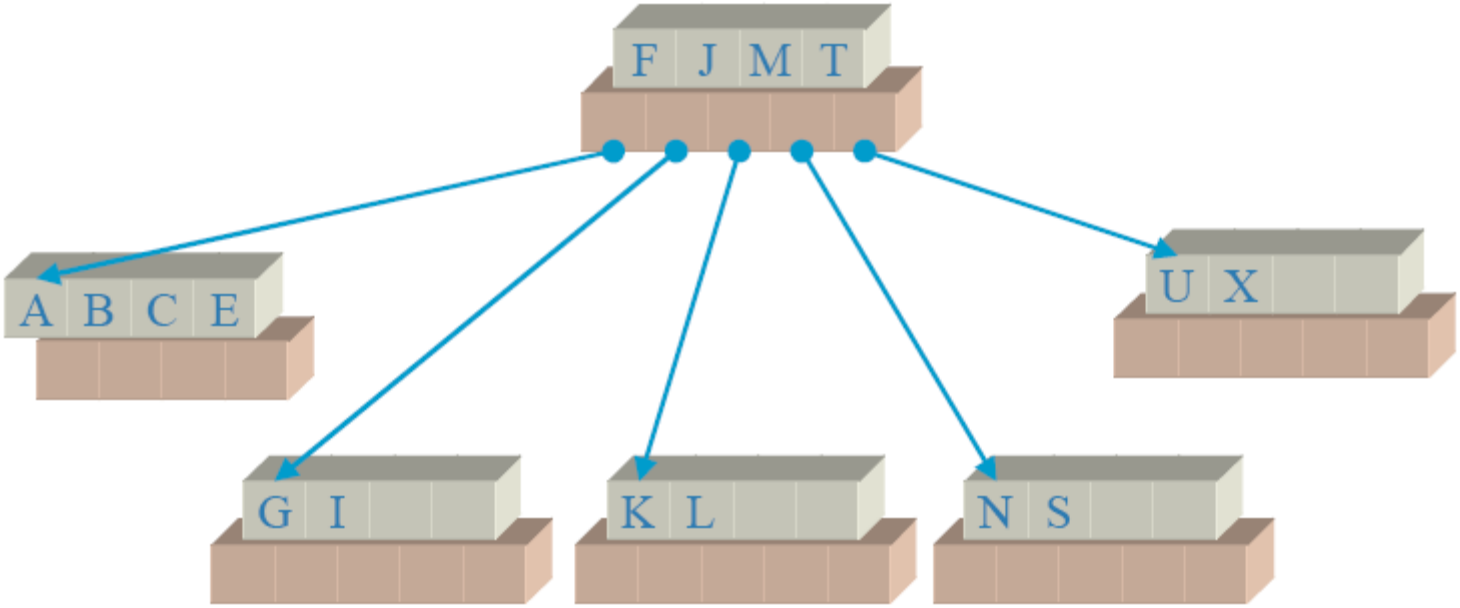
Çok Yollu Ağaçlar -B-Trees



Cok Yollu Ağaclar -B-Trees

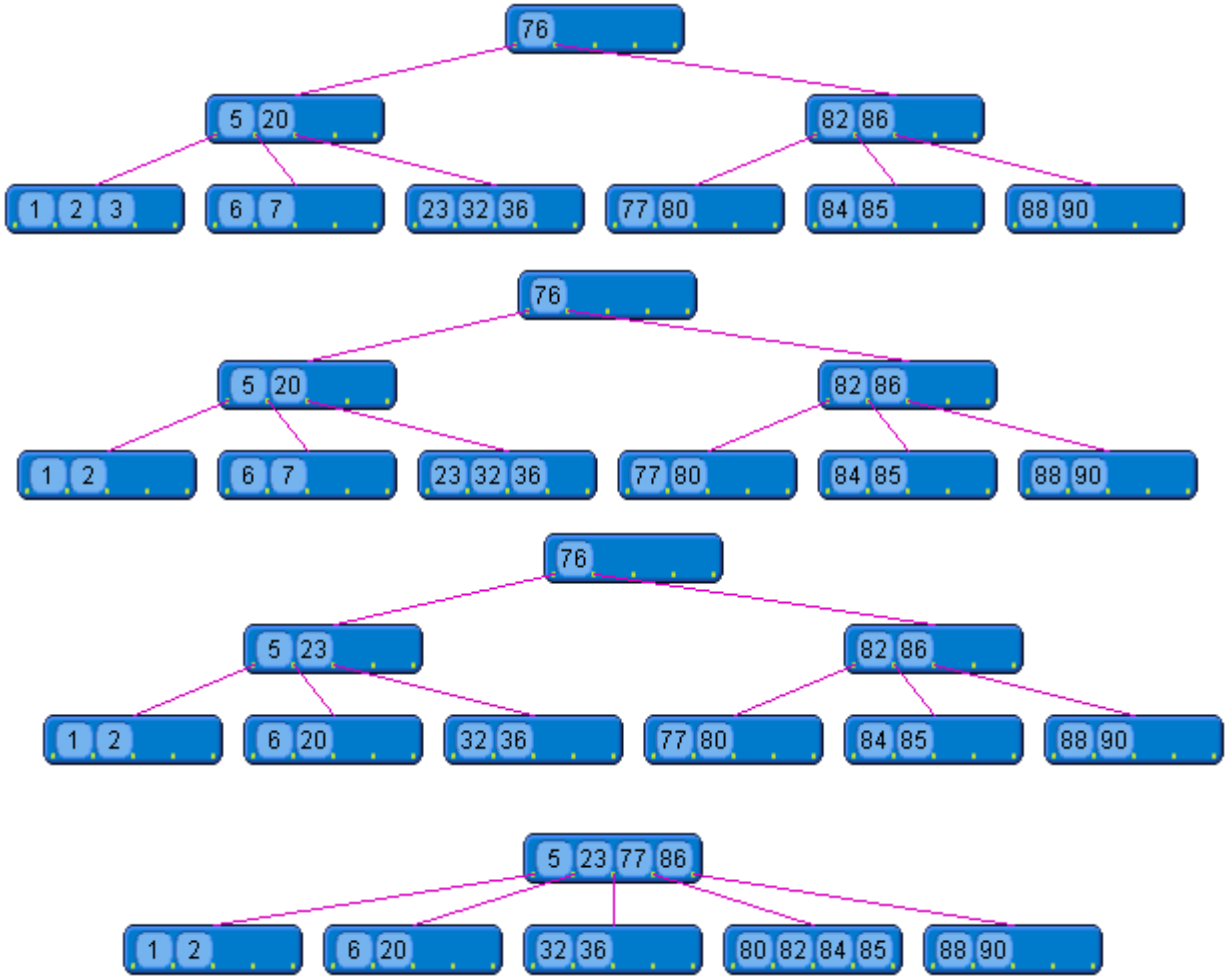


Çok Yollu Ağaçlar -B-Trees



Çok Yollu Ağaçlar -B-Trees

- **Örnek: Verilen ağaçtan sırasıyla 3,7,76 değerlerini siliniz**

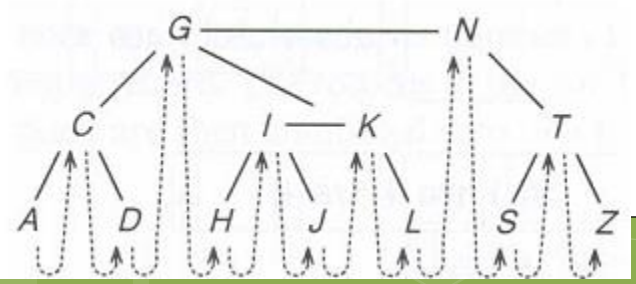


Çok Yollu Ağaçlar -B-Trees

- **Değerlendirme**
- B-Tree'lerde erişim araştırması kayda ulaşma değil node'a ulaşmadır
- Ağacın yüksekliği aşağıdaki gibi ifade edilir

$$\text{Height} \leq \log_{d+1} \frac{n+1}{2} + 1$$

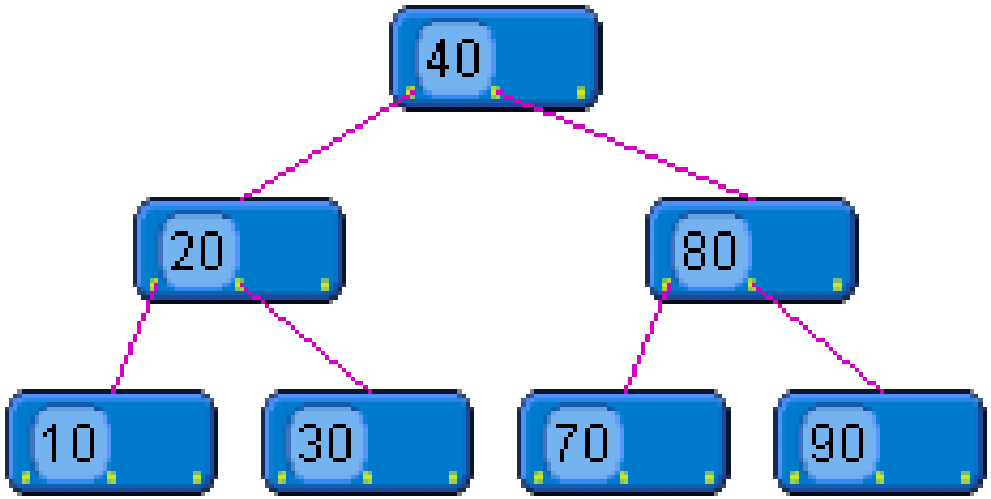
- Worst case erişim performansı $O(\log_d n)$ 'dir. n, B-tree'deki kayıt sayısıdır, d, kapasitedir.
- Kapasite sırası 50 olan ve 1 milyon kayıt bulunduran B-tree'de bir kayda ulaşmak için en fazla 4 erişim araştırması gerekir.
- B-tree'deki kayıtlara sıralı ulaşmak için inorder dolaşma yapılır.



Çok Yollu Ağaçlar -B-Trees

- Uygulama 1: d=1 olan ağaç için aşağıdaki değerleri girip B-tree ağacını oluşturunuz

- 20,10,90,40,30,80,70



Çok Yollu Ağaçlar -B-Trees

- Uygulama 2: Ağaçtan sırasıyla 30,80,40 düğümlerini siliniz