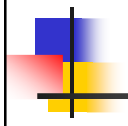


# BM 307 Dosya Organizasyonu (File Organization)

Hazırlayan: M.Ali Akcayol
Gazi Üniversitesi
Bilgisayar Mühendisliği Bölümü

## Konular

- Direct File Organization
  - Progressive Overflow
  - Buckets
  - Linear Quotient
  - Brent's Method
  - Binary Tree

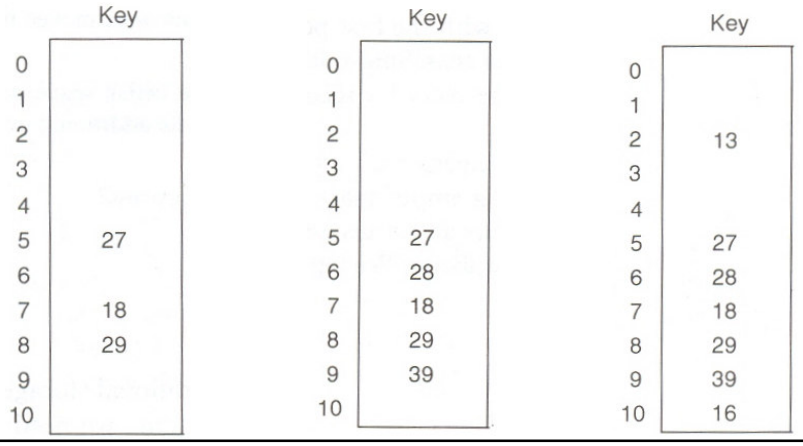
## Direct File Organization
### Progressive overflow

- Coalesced hashing temel dezavantajı linkler için ek yer gerektirmesidir
- Progressive overflow (linear probing) bir sonraki adres için link kullanmaz
- Progressive overflow yer doluysa bir sonraki adrese bakar
- Tablo dairesel olarak düşünülür ve sondaki adresten sonra başa dönülür
- Başarısız aramada ilk boşluğa kadar bakıldığından performans düşüktür

## Direct File Organization
### Progressive overflow

Örnek

- 27, 18, 29, 28, 39, 13, 16
- Hash(key) = key **mod** 11



## Direct File Organization
### Progressive overflow

Değerlendirme

- 16 için 6 probe gerekmektedir. Coalesced hashing 2 probe gerektirmekteydi.
- **Secondary clustering** oluşmaktadır. Farklı home adreslere sahip iki veya daha fazla kaydın aynı probe adresin sırasında olmasıdır.
- En büyük dezavantajı secondary clustering'den dolayı anahtar bilgisine ulaşmak için fazla probe gerektirmesidir.
- Secondary storage access çok yavaş olduğundan çakışma çözümü için pratik bir çözüm değildir.
- Her bir kayıt için ortalama 2.3 probe gerektirir. Bu sayı coalesced hashing'te 1.4 ' tür.

## Direct File Organization
### Progressive overflow

Değerlendirme

- Sequential search metoduna göre daha iyidir. Bir başlangıç noktası belirler ve başarısız arama için dosya sonuna kadar gitmeye gerek kalmaz ilk boşlukta arama sonlandırılır.
- Silme işleminde silinen elemanın yerine işaretçi **(tombstone)** konulur. 18 anahtarının silindikten sonraki durumu şekilde görülmektedir.
- Insert işlemini hızlandırmak için dosyada bulunan yerler için hafızada bit dizileri oluşturulur.
- Boş bir adres önce primary memory'den bulunur sonra dosyaya ulaşılır.

| | Key |
|---|---|
| 0 | |
| 1 | |
| 2 | 13 |
| 3 | |
| 4 | |
| 5 | 27 |
| 6 | 28 |
| 7 | ✦ |
| 8 | 29 |
| 9 | 39 |
| 10 | 16 |

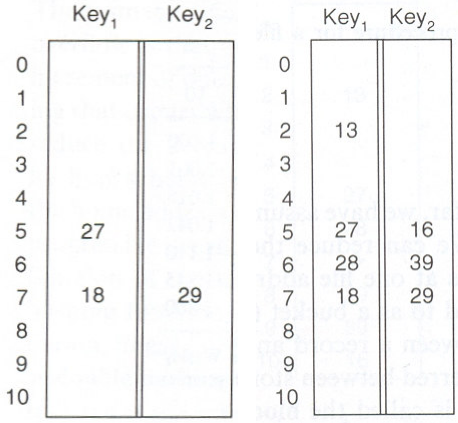
# Direct File Organization
## Use of Buckets

- Şimdiye kadar görülen çakışma çözümlerinde bir adrese sadece bir kayıt yazılmaktadır.
- Bir dosya adresine birden fazla kayıt yazılırsa auxilary memory'ye erişim sayısı azaltılır.
- Bir kayıt alanına birden çok kaydın yazılması **bucket** (page, block) olarak adlandırılır.
- Bir bucket içine kayıt edilebilen kayıt sayısı **blocking factor** olarak adlandırlır.
- Blocking factor arttıkça auxilary storage erişim sayısı azalmaktadır ihtiyaç duyulan yer artmaktadır.

# Direct File Organization
## Use of Buckets

Örnek

- 27, 18, 29, 28, 39, 13, 16
- Hash(key) = key **mod** 11




- Ortalama probe sayısı 1.0' dir.
- Progressive overflow'da probe sayısı 2.3'tür.
- Packing factor 7/22= 33% tür, progressive overflow'da 7/11 = 64% idi.

# Direct File Organization
## Use of Buckets

Karşılaştırma

- Bir bucket birden fazla kayıt için bölünür.
- Her bir kaydın fixed-length olması veya variable-length ise delimiter kullanılması gerekir.
- Bir bucket içindeki kaydı bulmak için geçen süre auxilary memory erişim süresinin yanında çok küçüktür.

**TABLE 3.2** MEAN NUMBER OF PROBES FOR SUCCESSFUL LOOKUP, PROGRESSIVE OVERFLOW, VARIOUS BUCKET SIZES

| $\alpha$ (in percent) | Blocking factor 1 | 2 | 5 | 10 |
|---|---|---|---|---|
| 20 | 1.125 | 1.024 | 1.007 | 1.000 |
| 40 | 1.333 | 1.103 | 1.012 | 1.001 |
| 60 | 1.750 | 1.293 | 1.066 | 1.015 |
| 70 | 2.167 | 1.494 | 1.136 | 1.042 |
| 80 | 3.000 | 1.903 | 1.289 | 1.110 |
| 90 | 5.500 | 3.147 | 1.777 | 1.345 |
| 95 | 10.500 | 5.600 | 2.700 | 1.800 |

*Source*: Knuth, Donald. E., *Sorting and Searching*, © 1973, Addison-Wesley Publishing Company, Inc., Reading, MA, p. 536, Table 4. Reprinted with permission.

# Direct File Organization
## Linear Quotient

- Çakışma durumunda değişken adres artırımı kullanılır.
- Secondary clustering önlenmesi amaçlanmaktadır.
- Linear quotient metodunda adres artırımı insert edilen anahtar değerine bağlıdır.
- İkinci bir hash fonksiyonu anahtar değerini artırım değerine dönüştürür.
- **Double hashing** metodlarındandır ve $H_1$ ve $H_2$ olarak iki hash fonksiyonu kullanılır.

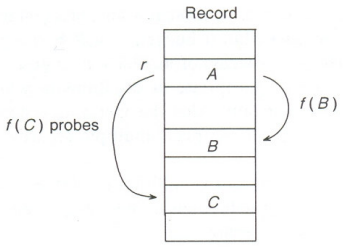
# Direct File Organization
## Linear Quotient

- Artırım için çok farklı fonksiyonlar kullanılabilir.

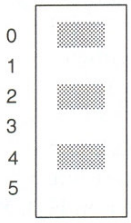
  $H_2$ = Quotient (Key / P) **mod** P

  $H_2$'= (Key **mod** (P-2)) + 1

  P asal sayı olarak tablo boyutunu göstermektedir.




- Linear quotient metodunda synonym'ler aynı probe zincirinde bulunmazlar.
- Dosya boyutunun asal sayı olması döngüyü engeller. (Örnek: 6 boyutundaki bir tabloda, 0, 2, 4 doluysa 0' a gelen yeni kayıt için yer bulunamayabilir.)




# Direct File Organization
## Linear Quotient

Örnek

- 27, 18, 29, 28, 39, 13, 16
- $H_1$(key) = key **mod** 11
- $H_2$ = Quotient (Key / 11) **mod** 11

| | Key | Key | Key |
|---|---|---|---|
| 0 | | | |
| 1 | | 39 | 39 |
| 2 | | | 13 |
| 3 | | | |
| 4 | | | |
| 5 | 27 | 27 | 27 |
| 6 | | 28 | 28 |
| 7 | 18 | 18 | 18 |
| 8 | | | 16 |
| 9 | 29 | 29 | 29 |
| 10 | | | |

# Direct File Organization

## Linear Quotient

### Değerlendirme

- Örnekte 16 anahtarı için 4 probe gerekmektedir. Progressive overflow'da 6 probe gerekmekteydi.
- Erişim için gereken ortalama probe sayısı 13/7 = 1.9, progressive overflow'da bu oran 2.3 ve LISCH coalesced hashing'te ise 1.4 olmuştur.
- Değişken artırım oranı secondary clustering'i önler ve böylece başarılı ve başarısız aramada performans artar.
- Progressive overflow metodunda olduğu gibi linear quotient metodundada daha önce eklenen kayıtların yeri değişmez. Static metottur.

# Direct File Organization

## Linear Quotient

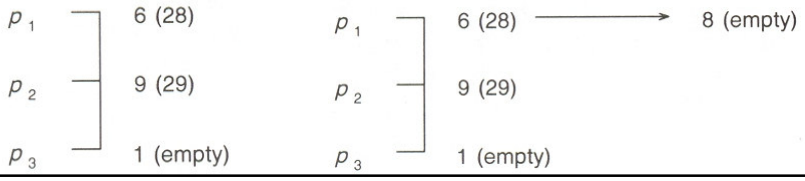
### Değerlendirme (Devam)

- Tabloya 67 anahtarını eklemek istersek home adresi 1 olur ve 39 bulunmaktadır.
- 67 için artırım değeri 6 olur ve sırasıyla 7, 2, 8, ve 3' e bakılır. 3'e yazıldığında toplam 5 probe gerekir.
- 39 olmasaydı sadece 1 probe gerekecekti.
- 39 anahtarı bir sonraki artırım adresine (4) taşınırsa 39 için probe 1 artar ancak 67 için 4 azalır.
- 39 ve 67 için toplam probe sayısı 8 iken yer değiştirmeyle bu sayı 5 olur. Toplam probe sayısı 3 azalır.

| | Key |
|---|---|
| 0 | |
| 1 | 39 |
| 2 | 13 |
| 3 | |
| 4 | |
| 5 | 27 |
| 6 | 28 |
| 7 | 18 |
| 8 | 16 |
| 9 | 29 |
| 10 | |

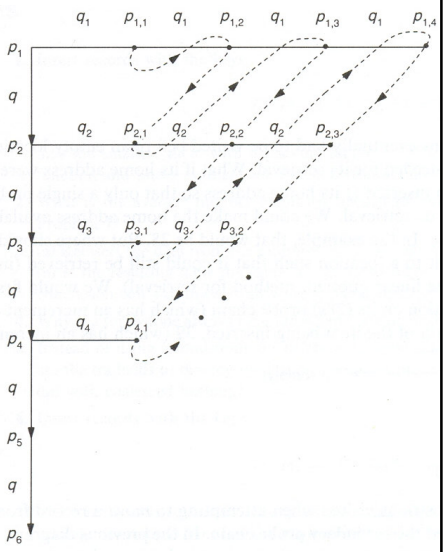
# Direct File Organization
## Brent's Method

- Dinamik bir metottur ve daha önce eklenen kayıtların yerini değiştirir.
- Bu metod insert için daha fazla işlem gerektirir. Ancak bir kayıt tabloya bir kez insert edilir çok kez okunur.
- **Primary probe chain,** bir kaydın **insert** veya **retrieval** edilmesi için gereken probe sayısıdır.
- **Secondary probe chain,** bir kaydın kendi primary probe chain'i içinde **move** edilmesi için gereken probe sayısıdır.
- 39 için gereken primary probe chain'deki probe sayısı 3'tür. Home adresinde (6) 28 vardir.
- Amaç toplam probe sayısını minimize etmektir.




# Direct File Organization
## Brent's Method

- **Dikey çizgi primary probe chain adreslerini gösterir.**
- **Yatay çizgiler insert edilmek istenen kayın primary chain'inde bulunan kayıtların kendi primary chain'lerini gösterir.**
- **q değerleri insert edilmek istenen kaydın primary probe chain'i içerisindeki artırım oranını göstermektedir.**
- **$q_i$ değerleri ise secondary probe chain'deki artırım oranlarını göstermektedir. $q_i$ değerleri farklı olabilir.**
- **i indisleri primary probe chain'deki probe sayısını ifade eder.**
- **j indisleri taşınacak kaydın ek probe sayısını ifade eder.**
- **Retrieval probe sayısını minimize etmek için (i+j) değerini minimize eden boş adresleri bulmamız gerekmektedir.**
- **s değeri hiç taşıma yapmadan gereken probe sayısı ise (i+j) <s olmalıdır.**
- **Eşitlik durumunda taşıma yapılmadan algoritma sonlandırılır.**



# Direct File Organization
## Brent's Method

Örnek

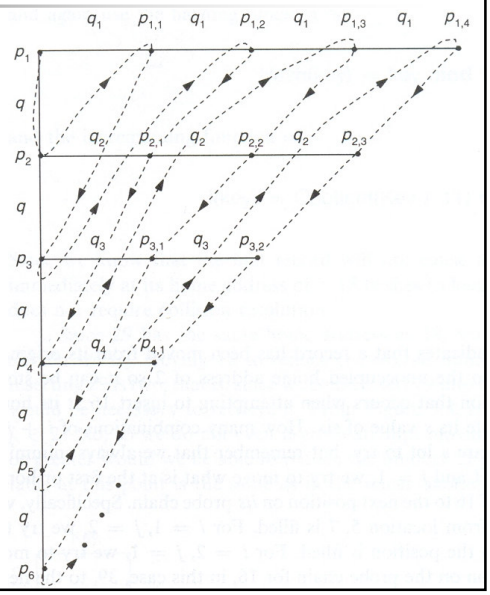
- 27, 18, 29, 28, 39, 13, 16
- $H_1$(key) = key **mod** 11
- $H_2$ = Quotient (Key / 11) **mod** 11

- Koyu yazılanlar taşındığını göstermektedir.
- 27 taşınarak 16 insert edilmiştir.
- 27 ve 16 için toplam probe sayısı 7'den 5'e inmiştir.
- Ortalama probe sayısı 1.7'dir. Bu değer linear quotient için 1.9, progressive overflow için 2.3 ve LISCH coalesced hashing için 1.4'tür.

| | Key | Key | Key |
|---|---|---|---|
| 0 | | | **27** |
| 1 | | | |
| 2 | | | 13 |
| 3 | | | |
| 4 | | | |
| 5 | 27 | 27 | 16 |
| 6 | | 39 | 39 |
| 7 | 18 | 18 | 18 |
| 8 | | **28** | **28** |
| 9 | 29 | 29 | 29 |
| 10 | | | |

# Direct File Organization
## Brent's Method

- Önceki şemada önce s değeri hesaplanmaktaydı ve daha sonra (i+j) < s şartına bakılmaktaydı.
- Yandaki şekilde ise s değeri hesaplanmadan doğrudan $p_1$'den başlayan yol takip edilmektedir. İlk bulunan boş yerde algoritma sonlandırılmaktadır.
- Primary chain üzerinde taşıma yapılmadan doğrudan insert işlemi yapılır

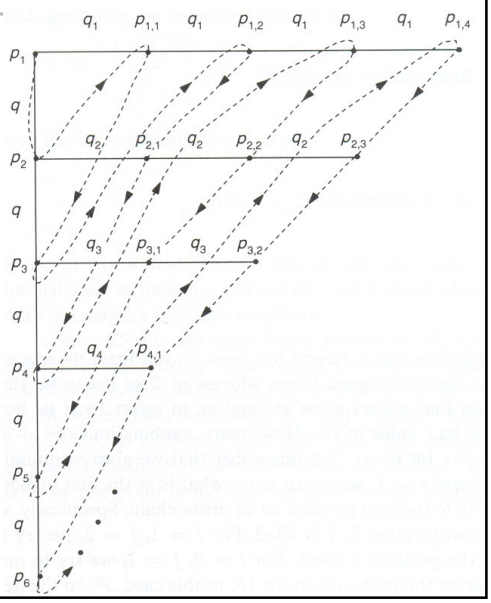


# Direct File Organization

## Brent's Method

- 16 anahtarının insert edilmesi.
- 27 yer değiştiriyor.

| | Key | Key |
|---|---|---|
| 0 | | **27** |
| 1 | | |
| 2 | 13 | 13 |
| 3 | | |
| 4 | | |
| 5 | 27 | 16 |
| 6 | 39 | 39 |
| 7 | 18 | 18 |
| 8 | **28** | **28** |
| 9 | 29 | 29 |
| 10 | | |




# Direct File Organization

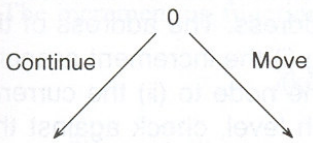
## Brent's Method

Değerlendirme

- Brent's metodu sadece insert için kullanılır. Retrieval için linear quotient kullanılır.
- Bir kaydın primary probe chain'e insert edilmesi ve diğer kaydın secondary probe chain'de taşınması arasında sürekli işlem yapılır.
- Bir kaydın silinmesi için tombstone kullanılır.

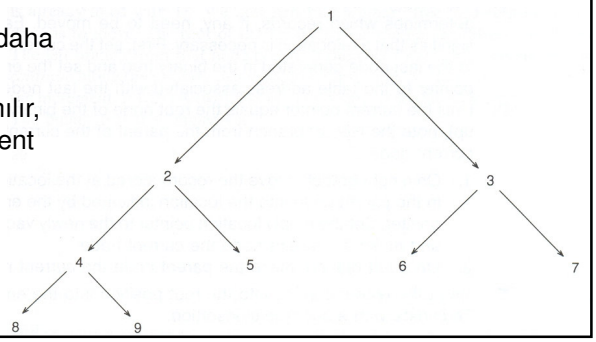
# Direct File Organization
## Binary Tree

- Alt probe chain'lerdeki birden fazla anahtarın aynı anda taşınması toplam probe sayısını azaltır.
- Brent's metoduna göre daha az retrieval probe gerektirir.
- Bir problemin çözümü için uygun veri yapısının seçilmesinin önemini göstermektedir.
- Bir anahtarın ne zaman taşınması gerektiği ve nereye taşınacağı binary tree kullanılarak belirlenir.
- Bir anahtar için iki farklı işlem yapılabilir. Primary probe chain'de bir sonraki adrese gidilir (**continue**) veya primary probe chain'deki kayıt secondary probe chain'de bir sonraki adrese taşınır (**move**).
- Binary tree'de her node için iki seçim vardır. Sol dallanma continue sağ dallanma ise move işlemi için kullanılır.




# Direct File Organization
## Binary Tree

- Binary tree breadth first yaklaşımıyla top-down ve left-to-right şeklinde oluşturulur.
- Binary decision tree sadece insert için kullanılır ve hangi adresin uygun olduğuna karar verir.
- Herbir insert için yeni bir binary tree oluşturulur.
- Boş bir yaprak veya tablonun dolu olması durumlarında algoritma sonlandırılır.
- Brent's metoduna göre daha fazla ön işlem gerektirir.
- Sadece insert için kullanılır, retrieval için linear quotient metodu kullanılır.

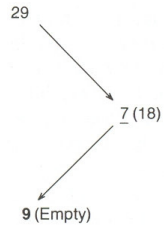


# Direct File Organization
## Binary Tree

Örnek

- 27, 18, 29, 28, 39, 13, 16 ve ek olarak 41, 17, 19
- Hash(key) = key **mod** 11
- i(key) = Quotient (Key / 11) **mod** 11

- 27 home adresi 5'e insert edilir.
- 18 home adresi 7'ye insert edilir.
- 29 home adresi 7'ye gelir ancak 18 ile çakışır. Yandaki şekildeki binary tree oluşturulur.
- Binary tree için root 29'un home adresidir (7).
- 7'den sonraki her sola dallanma 29 için primary probe chain'i gösterir. Soldaki node değeri i(29) = 2 olarak elde edilir. Yeni adres 7+2=9'dur ve boş olduğu için 29 insert edilir.
- Şekilde altı çizgili node root node, koyu yazılanlar adresleri ve parantez içindekiler ise anahtarları göstermektedir.

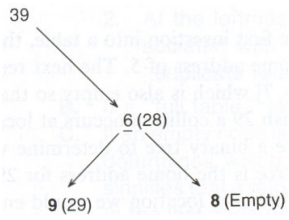



| | Key |
|---|---|
| 0 | |
| 1 | |
| 2 | |
| 3 | |
| 4 | |
| 5 | 27 |
| 6 | |
| 7 | 18 |
| 8 | |
| 9 | 29 |
| 10 | |

# Direct File Organization
## Binary Tree

Örnek (Devam)

- 28 home adresi 6'ya insert edilir.
- 39 home adresi 6 dolu olduğu için binary tree oluşturulur. 6'dan sonraki adres 9 doludur. 6 adresindeki 28 için sonraki adres 8 boştur.
- 6'daki 28, 8'e taşınır ve 39 kendi home adresi olan 6'ya insert edilir.
- Koyu yazılanlar taşınan kayıtları göstermektedir.
- 13 home adresi 2'ye insert edilir.



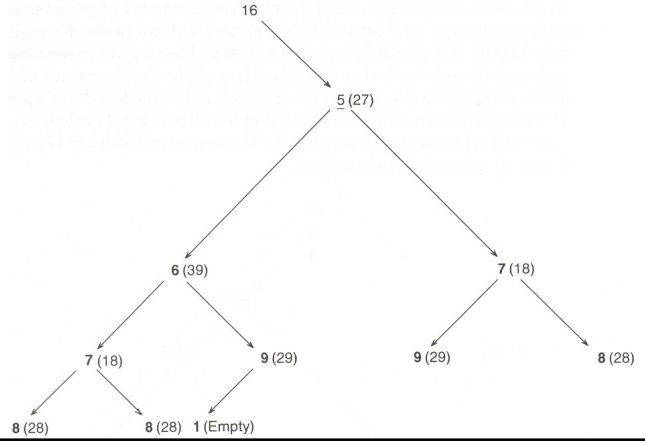

| | Key |
|---|---|
| 0 | |
| 1 | |
| 2 | |
| 3 | |
| 4 | |
| 5 | 27 |
| 6 | 39 |
| 7 | 18 |
| 8 | **28** |
| 9 | 29 |
| 10 | |

# Direct File Organization

## Binary Tree

Örnek (Devam)

- 16 home adresi 5 dolu olduğu için binary tree oluşturulur.
- Oluşturulan ağaçta 39 bir sonraki primary chain adresine taşınır ve 16 primary chain'deki 6 adresine insert edilir.



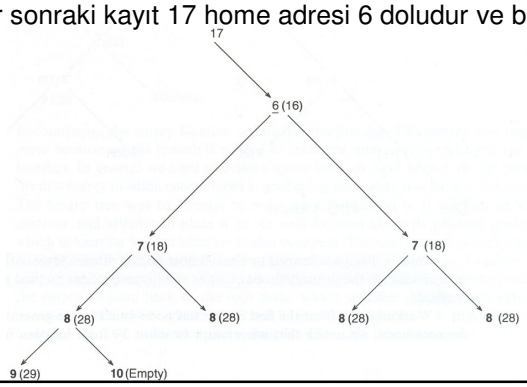

| | Key |
|---|---|
| 0 | |
| 1 | **39** |
| 2 | 13 |
| 3 | |
| 4 | |
| 5 | 27 |
| 6 | 16 |
| 7 | 18 |
| 8 | **28** |
| 9 | 29 |
| 10 | |

# Direct File Organization

## Binary Tree

Örnek (Devam)

- Şu ana kadar standart kayıt seti insert edildi.
- Ortalama retrival için probe sayısı 1.7'dir. Bu oran LISCH coalesced hashing için 1.4 ve Brent's metodu için 1.7'dir.
- Ek kayıtları insert edelim. 41 home adresi 8 doludur ve bir sonraki primary chain adresi olan 0' insert edilir.
- Bir sonraki kayıt 17 home adresi 6 doludur ve binary tree oluşturulur.



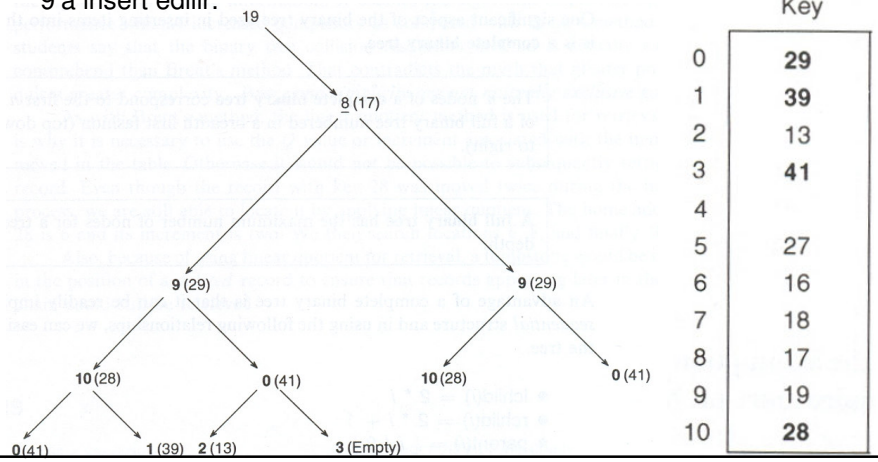

| | Key |
|---|---|
| 0 | 41 |
| 1 | **39** |
| 2 | 13 |
| 3 | |
| 4 | |
| 5 | 27 |
| 6 | 16 |
| 7 | 18 |
| 8 | 17 |
| 9 | 29 |
| 10 | **28** |

# Direct File Organization

## Binary Tree

Örnek (Devam)

- Son kayıt 19 home adresi 8 doludur ve binary tree oluşturulur.
- 41 anahtarı 0'dan 3'taşınır, 29 anahtarı 9'dan 0'a taşınır ve 19 anahtarı 9'a insert edilir.




| | Key |
|---|---|
| 0 | **29** |
| 1 | **39** |
| 2 | 13 |
| 3 | **41** |
| 4 | |
| 5 | 27 |
| 6 | 16 |
| 7 | 18 |
| 8 | 17 |
| 9 | 19 |
| 10 | **28** |

# Direct File Organization

## Binary Tree

Implementation

- Oluşturulan ağaç **Complete Binary Tree**'dir. İlk n node **Full Binary Tree**'deki ilk n node'dur.
- Full Binary Tree yüksekliğine göre maksimum node sahiptir.
- Complete binary tree'deki her node için
  - lchild (i) = 2 * i
  - rchild (i) = 2 * i + 1
  - parent (i) = floor (i / 2)
- Oluşturulan ağacın derinliği $O(\log_2 n + c)$ ile ifade edilir. n tablodaki giriş sayısını gösterir c ise 1 veya 2 olarak alınır.
- Tablonun dolu olduğu oluşturulan ağacın derinliğinin $\log_2 n + 2$ değerinden büyük olmasıyla anlaşılır.

## Direct File Organization

### Binary Tree

Değerlendirme

- Brent's metoduna göre anlaşılması daha kolaydır.
- Brent's metodunda olduğu gibi sadece insert için kullanılır ve retrieval için liner quotient kullanılır.
- Kayıt silmede diğer metodlarda olduğu gibi tombstone kullanılır.
- Birden fazla kaydı aynı anda taşıyabilir.

## Direct File Organization

### Haftalık Ödev

1000 tane rastgele değere sahip anahtar için Binary Tree ile insert işlemini gerçekleştiren bir programı C#.NET ile gerçekleştiriniz. Ortalama retrieval probe sayılarını hesaplayınız.