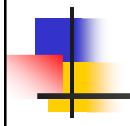


BM 307 Dosya Organizasyonu (File Organization)

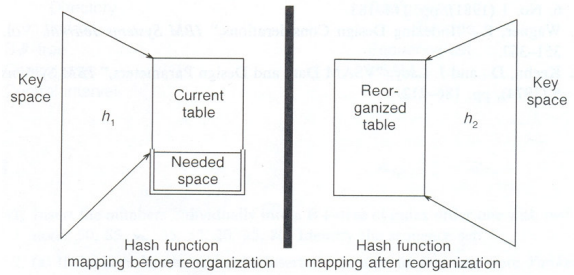
Hazırlayan: M.Ali Akcayol
Gazi Üniversitesi
Bilgisayar Mühendisliği Bölümü

Konular

- Hashing Techniques and Expandable Files
  - Background
  - Extendible Hashing
  - Dynamic Hashing
  - Linear Hashing
  - Değerlendirme

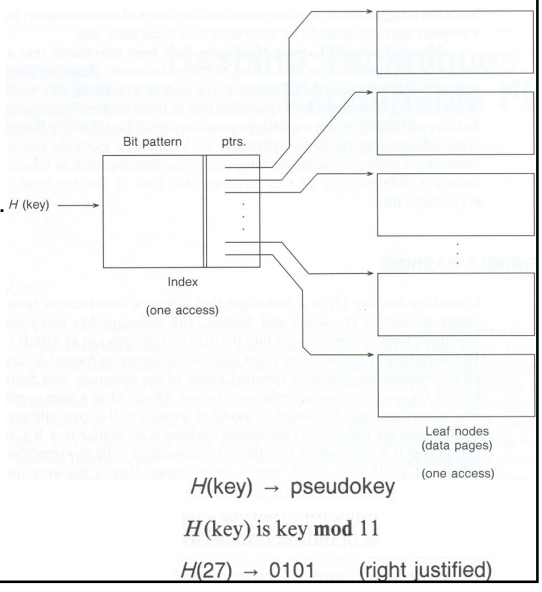
Hashing Techniques and Expandable Files
Background

- Hashing ve collision resolution metodları statik dosya boyutunu esas alırlar.
- Doğrudan erişimli dosyalarda %85 packing factor ile ortalama 1.5 probe kullanılarak bir kayda erişilir.
- En büyük dezavantajları ise yeni kayıt eklendikçe veya mevcut kayıtlar silindikçe dosya boyutunun değişmemesidir.
- Bu metodlarda dosya boyutunu değiştirdikten sonra bütün kayıtların yeni bir hash fonksiyonu ile tekrar insert edilmesi gerekmektedir.



Hashing Techniques and Expandable Files
Extendible Hashing

- Kayıtlar eklendikçe veya silindikçe tablonun boyutunu sürekli ayarlar.
- B+ - ağaçlarındaki parçalanma ve birleştirme işlemine benzer yeniden yapılanma gerçekleştirir.
- Ancak B+ - ağaçlarında index derinliği 1'den fazla olabilirken Extendible hashing metodunda index derinliği (dept) 1'e eşittir.
- Doğrudan erişimde B+ - ağaçlarından daha az probe gerektirir.
- Herbir anahtar için hash fonksiyonuyla bir **pseudokey** elde edilirek index girişi sağlanır.



H(key) → pseudokey

H(key) is key **mod** 11

H(27) → 0101 (right justified)

Hashing Techniques and Expandable Files
Extendible Hashing

- Index kısmının içerdiği bit kadar en önemli bit (most significant bits - leftmost) pseudokey üzerinde karşılaştırılır.
- Bir sayfada overflow olduğunda iki sayfaya bölünür ve kayıtlar soldan itibaren farklı ilk k (page depth) bit gözönüne alınarak iki sayfaya dağıtılır.
- Eğer iki sayfa mevcut k değerine göre ayrılamıyorsa k değeri bir artırılır. k değeri farklılık olan bite kadar artırılır.
- Eğer page depth değeri index depth değerinden büyük olursa index depth değeri artırılır.
- Index depth (n) en büyük page depth değerine eşittir.
- Her index depth değerini artırmadan sonra tablo boyutu 2 ile çarpılarak büyütülür.
- Tablo boyutu index derinliği 1 için 2, 2 için 4 ve 3 için 8 olur. (2^n)
- Bir kayda ulaşmak için, bir erişim index ve bir erişimde page için olmak üzere toplam iki probe gerekir.
- Eğer index primary memory'de saklanırsa auxiliary memory'e bir erişim yeterlidir.

Hashing Techniques and Expandable Files
Extendible Hashing

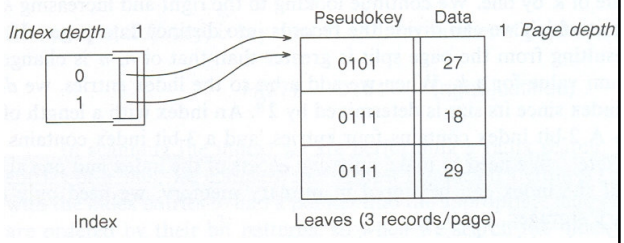
- Örnek

Her sayfada 3 kayıt olacak şekilde 27, 18, 29, 28, 42, 13 ve 16 değerlerini insert edelim. Hash fonksiyonuyla 4-bit pseudokey (right-justified) değerleri üretilsin.

$H(\text{key}) = \text{key } \textbf{mod } 11$

- Başlangıçta index derinliği 1 olur.
- Index kısmının üstünde index derinliği ve sayfa kısmının üstünde ise sayfa derinliği bilgileri yer almaktadır.

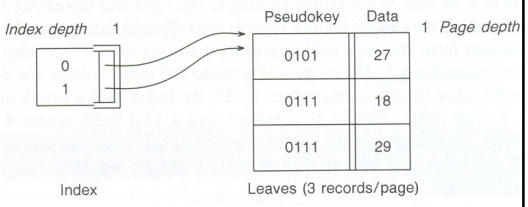
$f(27) = 5 = 0101$
$f(18) = 7 = 0111$
$f(29) = 7 = 0111$
$f(28) = 6 = 0110$
$f(42) = 9 = 1001$
$f(13) = 2 = 0010$
$f(16) = 5 = 0101$



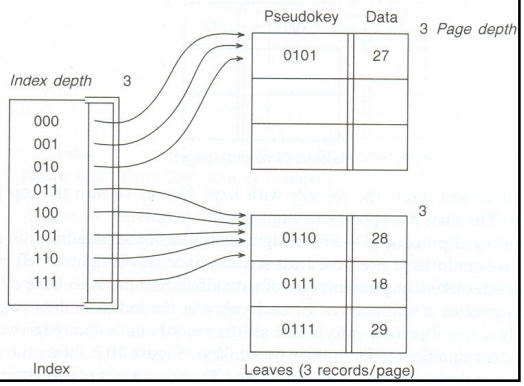
Hashing Techniques and Expandable Files
Extendible Hashing

- Örnek

$f(27) = 5 = 0101$
$f(18) = 7 = 0111$
$f(29) = 7 = 0111$
$f(28) = 6 = 0110$
$f(42) = 9 = 1001$
$f(13) = 2 = 0010$
$f(16) = 5 = 0101$



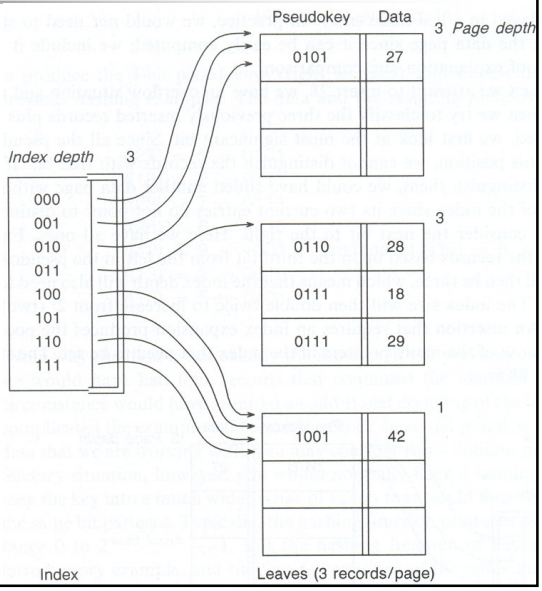
- 28 eklendi.
- Soldan sağa doğru ilk farklı bit bulunur ve ona göre iki parçaya ayrılır.
- Sayfa ve index derinlikleri yeniden düzenlenir.



Hashing Techniques and Expandable Files
Extendible Hashing

- Örnek

$f(27) = 5 = 0101$
$f(18) = 7 = 0111$
$f(29) = 7 = 0111$
$f(28) = 6 = 0110$
$f(42) = 9 = 1001$
$f(13) = 2 = 0010$
$f(16) = 5 = 0101$



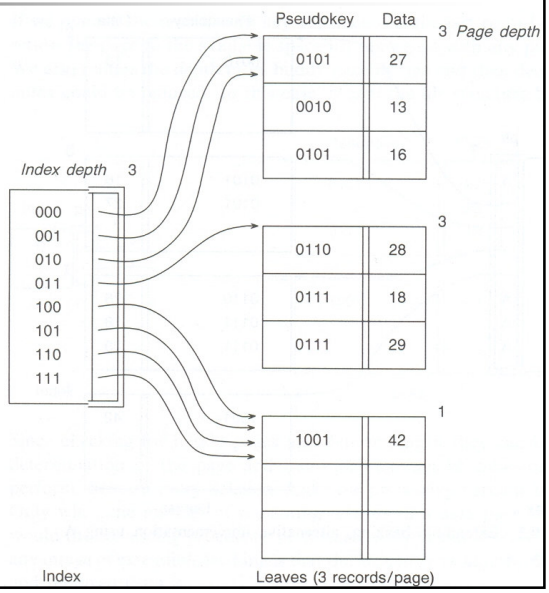
- 42 eklendi.
- 2. sayfada overflow olur.
- Yeni bir sayfa eklendi ve ilk bite göre ayrıldı.

Hashing Techniques and Expandable Files
Extendible Hashing

- Örnek

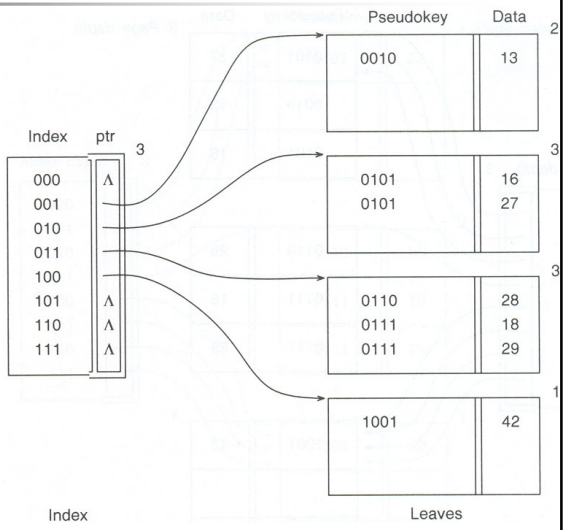
$f(27) = 5 = 0101$
$f(18) = 7 = 0111$
$f(29) = 7 = 0111$
$f(28) = 6 = 0110$
$f(42) = 9 = 1001$
$f(13) = 2 = 0010$
$f(16) = 5 = 0101$

- 13 ve 16 eklendi.



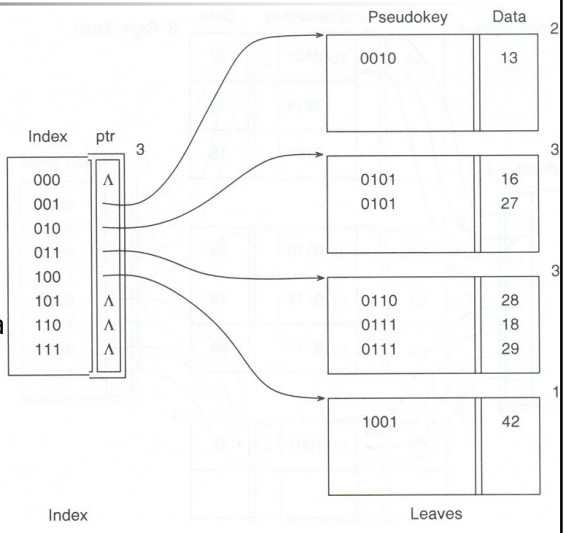
Hashing Techniques and Expandable Files
Extendible Hashing

- Pseudokey işlemlerinde soldan sağa doğru yerine sağdan sola doğru işlemde yapılabilir.
- Diğer bir extendible hashing, index sayfasında **null pointer** kullanabilir.
- Böylece bir sayfadaki tüm kayıtların derinlik değerine sahip bitleri aynıdır.
- Avantajı, bir sayfa için null değeri varsa gereksiz arama yapılmaz.
- Dezavantajı, her bölünmeden sonra (farklılıktan sonraki bölünme) yeni bir sayfa oluşturulur.



Hashing Techniques and Expandable Files
Extendible Hashing

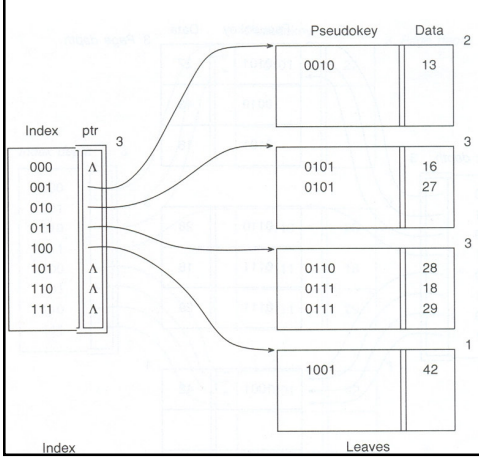
- **Silme**
  - Silme, eklemenin tersi şeklinde yapılır.
  - Bir kayıt silindiğinde buddy page'i ile birleştirilmeye çalışılır.
  - Buddy page (k-1) adet sol bitin aynı olduğu ve k. bitin farklı olduğu sayfadır. (k sayfa derinliğidir)
  - Silmede sadece sayfa sayısı azaltılmaz, index derinliğide azaltılır.



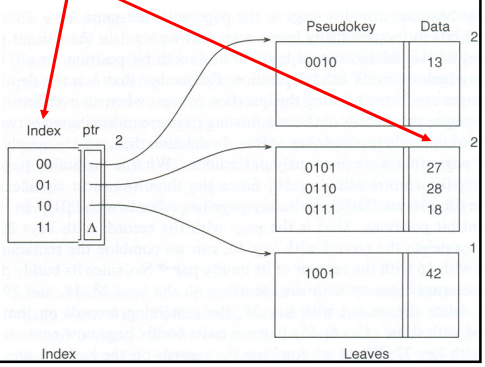
Hashing Techniques and Expandable Files
Extendible Hashing

- **Silme**

16 ve ardından 29 anahtarının silinmesi.



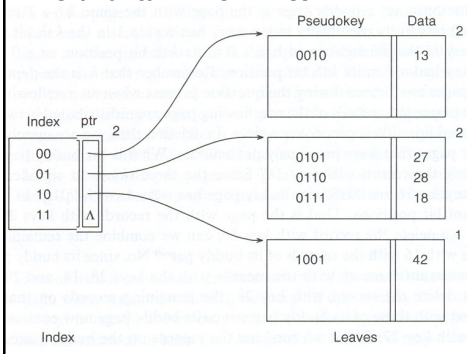
En yüksek sayfa derinlik değeri 2 olduğu için index derinliğide 2 yapılır



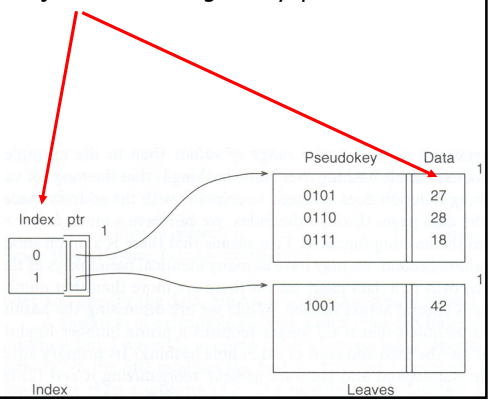
Hashing Techniques and Expandable Files
Extendible Hashing

- **Silme**

13 anahtarının silinmesi.



En yüksek sayfa derinlik değeri 1 olduğu için index derinliğide 1 yapılır

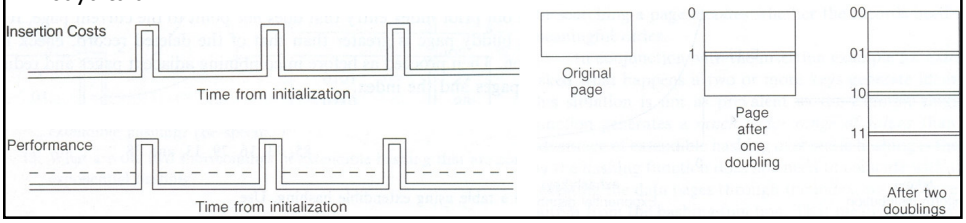


Hashing Techniques and Expandable Files
Extendible Hashing

- **Değerlendirme**
  - Her silmede sayfaların birleştirilmelerinin kontrol edilmesi ve birleştirme işlemi zaman alır ve performansı düşürür.
  - Bir sayfadaki kayıt sayısı her sayfa için belirlenen bir threshold (eşik) değerinin altına düştüğünde birleştirme işlemi yapılabilir.
  - Bir kayda erişmek için hash fonksiyonuyla pseudokey değeri elde edilir. Ardından index derinlik değerine göre aynı olan bitlerle ilgili sayfaya ulaşılır.
  - Sayfaya ulaşıldıktan sonra sıralı kayıt yapılmışsa arama istenen bir metodla yapılabilir (sequential, binary, hashing, ...).
  - Uygulamada aynı pseudokey değerine sahip çok fazla kayıt olma ihtimali çok düşüktür. (Seçilen hash fonksiyonu anahtarları çok geniş bir aralıkta düzgün dağıtır).

Hashing Techniques and Expandable Files
Extendible Hashing

- **Değerlendirme**
  - Extendible hashing potansiyel dezavantajlara sahiptir.
  - Hash fonksiyonu düzgün dağılım yaparak insert işlemi yapar. Belirli bir süre hiç bölünme olmadan insert devam eder ancak yaklaşık aynı süre içinde çok fazla bölünme olur ve insert performansı çok düşer.
  - Index boyutu primary memory'ye sığmayacak kadar büyürse auxiliary memory'ye yazılır ve probe sayısı 1'den 2'ye yükselir.
  - İki dezavantajda **bounded index exponential hashing** metoduyla çözülür.
  - Anahtarların dağılımı yeni bir hash fonksiyonuyla yeniden düzenlenir. exhash(x) = $2^{\text{hash(key)}}$-1
  - Index hafızaya sığmayacak kadar büyürse data page boyutu ikiye çarpılarak büyültülür.



Hashing Techniques and Expandable Files
Dynamic Hashing

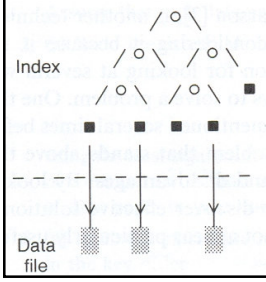
- Binary tree internal ve external node'lardan oluşur. External node'lar gerçek data sayfasını gösterirler.
- Internal node'lar doğru external node'a ulaşmayı sağlarlar.
- Internal node'larda pseudokod içindeki 0'lar için sola 1'ler için sağa gidilir.
- Hash fonksiyonu 0 ve 1'leri rastgele oluşturduğu için ağaç stokastik olarak dengelidir.

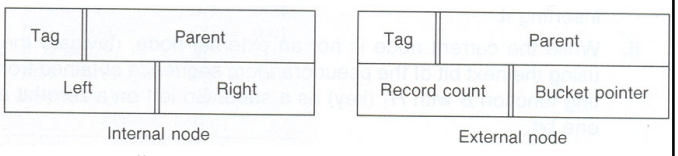
$H_1(\text{Key}_i)$

$B(H_1(\text{Key}_i)) = (b_{i0}, b_{i1}, b_{i2}, \ldots)$ $\quad b_{ij} \in \{0,1\}$ for all j

Hashing Techniques and Expandable Files
Dynamic Hashing

- Extendible hashing metodunun bir dezavantajı bölme işleminde index büyütmenin zaman almasıdır ve düzgün dağılımlı olmamasıdır.
- Bunu ortadan kaldırmak için bounded index exponantial hashing kullanılabilir veya dynamic hashing kullanılabilir.
- Dynamic hashing metodunda index sürekli büyür.
- Kayda ulaşmak için gerekli doğru sayfayı elde etmek için bir binary tree index kullanılır.
- Extendible hashing metodunda olduğu sabit uzunlukta bir pseudokey kullanmak yerine değişken pseudokey kullanılır.
- Bit serisi şeklinde pseudokey oluşturulur.



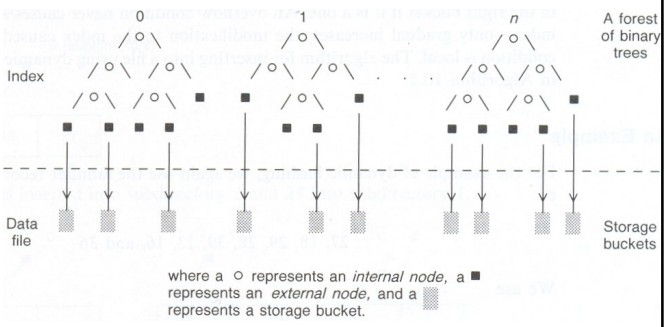


Parent link ağacın üst kısmına ulaşmak için kullanılır
Record count data sayfasında veya bucket'ta bulunan bilgi sayısını gösterir
Bucket pointer bilgilerin bulunduğu sayfayı gösterir

Hashing Techniques and Expandable Files
Dynamic Hashing

- Dynamic hashing ağaç yüksekliğinin artmasını önlemek için birden fazla ağaç yapısını aynı anda kullanır.
- Bilginin hangi ağaçta olduğu başka bir hash fonksiyonuyla belirlenir.

$H_0(\text{Key}_i) \rightarrow \{0, 1, \ldots, n\}$



- H_0 ile önce doğru ağaç seçilir, daha sonra H_1 ve B ile ağaçta ilerlenir.
- Overflow olduğunda external node, internal node olur ve iki tane yeni external node oluşturulur.
- Kayıtlardan sıradaki bit değeri 0 olan soldaki yeni external node'a, 1 olan ise sağdaki external node'a aktarılır.

Hashing Techniques and Expandable Files
Dynamic Hashing

- Örnek

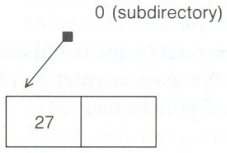
27, 18, 29, 28, 39, 13, 16, 36

Her data sayfası iki tane kayıt saklasın.

H_0 = key **mod** 3 H_1 = key **mod** 11

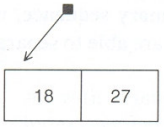
TABLE 10.1 PSEUDORANDOM BIT SEQUENCES FOR SEEDS 0–10

$B(0) = 1011$	$B(5) = 0101$
$B(1) = 0000$	$B(6) = 0001$
$B(2) = 0100$	$B(7) = 1110$
$B(3) = 0110$	$B(8) = 0011$
$B(4) = 1111$	$B(9) = 0111$
$B(10) = 1001$	



27 insert edildi.

$H_0(27) = 0$
$H_1(27) = 5$



18 insert edildi.

$H_0(18) = 0$
$H_1(18) = 7$

Hashing Techniques and Expandable Files
Dynamic Hashing

- Örnek

27, 18, 29, 28, 39, 13, 16, 36

H_0 = key **mod** 3

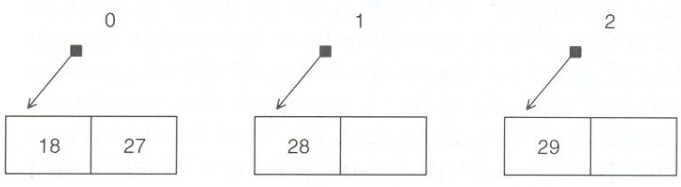
H_1 = key **mod** 11

TABLE 10.1 PSEUDORANDOM BIT SEQUENCES FOR SEEDS 0–10

$B(0) = 1011$	$B(5) = 0101$
$B(1) = 0000$	$B(6) = 0001$
$B(2) = 0100$	$B(7) = 1110$
$B(3) = 0110$	$B(8) = 0011$
$B(4) = 1111$	$B(9) = 0111$
$B(10) = 1001$	

29 insert edildi.
$H_0(29) = 2$
$H_1(29) = 7$

28 insert edildi.
$H_0(28) = 1$
$H_1(28) = 6$



Hashing Techniques and Expandable Files
Dynamic Hashing

- Örnek

27, 18, 29, 28, 39, 13, 16, 36

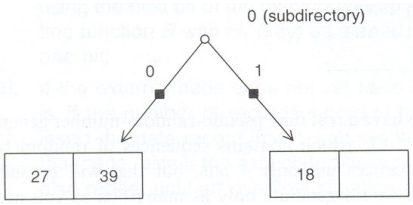
H_0 = key **mod** 3

H_1 = key **mod** 11

TABLE 10.1 PSEUDORANDOM BIT SEQUENCES FOR SEEDS 0–10

$B(0) = 1011$	$B(5) = 0101$
$B(1) = 0000$	$B(6) = 0001$
$B(2) = 0100$	$B(7) = 1110$
$B(3) = 0110$	$B(8) = 0011$
$B(4) = 1111$	$B(9) = 0111$
$B(10) = 1001$	

39 insert edildi. Overflow oldu. Yeni bir external node oluşturuldu. Tüm bilgiler iki node arasında dağıtıldı.
$H_1(27)$ = 5, B(5)'nın ilk biti (0) kullanıldı.
$H_1(39)$ = 6, B(6)'in ilk biti (0) kullanıldı.
$H_1(18)$ = 7, B(7)'in ilk biti (1) kullanıldı.



Hashing Techniques and Expandable Files
Dynamic Hashing

- Örnek

27, 18, 29, 28, 39, 13, 16, 36

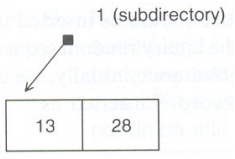
H_0 = key **mod** 3

H_1 = key **mod** 11

TABLE 10.1 PSEUDORANDOM BIT SEQUENCES FOR SEEDS 0–10

$B(0) = 1011$	$B(5) = 0101$
$B(1) = 0000$	$B(6) = 0001$
$B(2) = 0100$	$B(7) = 1110$
$B(3) = 0110$	$B(8) = 0011$
$B(4) = 1111$	$B(9) = 0111$
$B(10) = 1001$	

13 insert edildi.
$H_0(13)$ = 1



Hashing Techniques and Expandable Files
Dynamic Hashing

- Örnek

27, 18, 29, 28, 39, 13, 16, 36

H_0 = key **mod** 3

H_1 = key **mod** 11

TABLE 10.1 PSEUDORANDOM BIT SEQUENCES FOR SEEDS 0–10

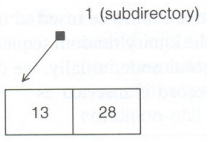
$B(0) = 1011$	$B(5) = 0101$
$B(1) = 0000$	$B(6) = 0001$
$B(2) = 0100$	$B(7) = 1110$
$B(3) = 0110$	$B(8) = 0011$
$B(4) = 1111$	$B(9) = 0111$
$B(10) = 1001$	

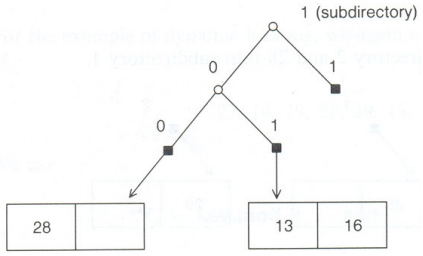
16 insert edildi. Overflow oldu yeni external node oluşturuldu ve tüm kayıtlar yeniden dağıtıldı.

$H_1(16) = 5$, B(5) kullanıldı. (0101)

$H_1(13) = 2$, B(2) kullanıldı. (0100)

$H_1(28) = 6$, B(6) kullanıldı. (0001)





Hashing Techniques and Expandable Files
Dynamic Hashing

- Örnek

27, 18, 29, 28, 39, 13, 16, 36

H_0 = key **mod** 3 H_1 = key **mod** 11

36 insert edildi. Overflow oldu. Yeni bir external node oluşturuldu. Tüm bilgiler iki node arasında dağıtıldı.

$H_0(36) = 0$

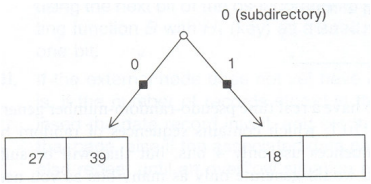
$H_1(36) = 3$, B(3) (**01**10)

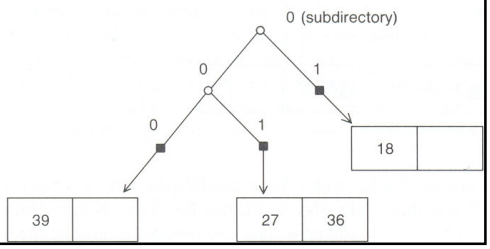
$H_1(27) = 5$, B(5) (**01**01)

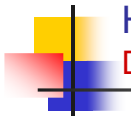
$H_1(39) = 6$, B(6) (**00**01)

TABLE 10.1 PSEUDORANDOM BIT SEQUENCES FOR SEEDS 0–10

$B(0) = 1011$	$B(5) = 0101$
$B(1) = 0000$	$B(6) = 0001$
$B(2) = 0100$	$B(7) = 1110$
$B(3) = 0110$	$B(8) = 0011$
$B(4) = 1111$	$B(9) = 0111$
$B(10) = 1001$	







Hashing Techniques and Expandable Files
Dynamic Hashing

- **Değerlendirme**
  - Bir anahtara erişmek için H_0 ile bulunduğu ağaç, H_1 ile izlenecek yol için gerekli pseudocode belirlenir.
  - External node'a ulaşıldığında auxiliary memory'de bilginin olduğu sayfaya ulaşılır.
  - Bütün retrieval süreleri yaklaşık eşittir çünkü stokastik olarak ağaç dengelenmiştir.
  - Tüm ağaçları primary memory'e alabilirsek auxiliary memory'e sadece birkez ulaşmamız yeterlidir.
  - Ağaçların primary memory'ye sığmadığı durumlarda, external node'larda bağlı listerle yeni bucket'lar oluşturulur.

Hashing Techniques and Expandable Files
Linear Hashing

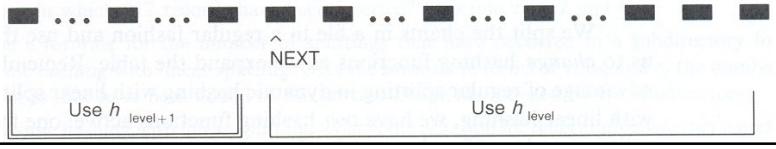
- Extendible hashing ve dynamic hashing metodlarının en önemli dezavantajı link kullanmalarıdır.
- Linear hashing metodunda link kullanılmaz ve ek hafızaya ihtiyaç duyulmaz.
- Linear hashing metodunda birden fazla hash fonksiyonu kullanılır ve sürekli değiştirilir.
- Hem bilgi eklemede hemde bilgi silmede, bir bulunulan seviye ve birde sonraki seviye için iki hash fonksiyonu kullanılır.
- Her seviye artışında dosya boyutu iki ile çarpılır, her seviye azalışında dosya boyutu ikiye bölünür.
- Dosyadaki kayıt sayısı için bir upper bound ve birde lower bound belirlenir.
- Bilgi eklemede upper bound'un üzerinde kayıt oluşmuşsa yeni bir sayfa eklenir ve aktif sayfadaki bilgiler bu sayfaya dağıtılır.
- Bilgi silmede lower bound'un altına düşülmüşse aktif sayfanın önündeki sayfaya, aktif sayfadan sonraki kayıtlar aktarılır.

Hashing Techniques and Expandable Files
Linear Hashing

- Upper bound aşıldığı zaman bölünecek sayfa için bir NEXT pointer'ı bulunur.

$$h_{level}(key) = key \bmod [N\,(2^{level})]$$

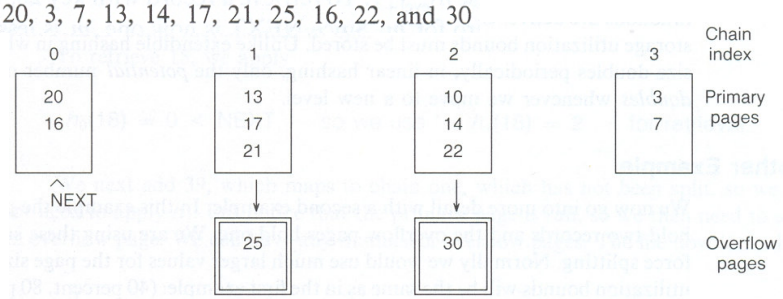
- Bulunulan seviye için hash fonksiyonuyla ilgili sayfaya ulaşılır.
- Level bir artırıldığında mod değeri iki kat artar.
- Mod değeri her arttığında adreslenebilecek adres alanı iki katına çıkar.
- Bir hash fonksiyonu bulunulan kısım için adresleme yapar, diğeri ise expand edilen kısım için adresleme yapar.
- Her level için bölme işlemi 0. sayfadan başlar ve N.(2^{level}) değerine kadar devam eder.
- Bulunulan level için N.(2^{level}) değerinden sonra tekrar başa dönülür ve seviye bir artırılır. (N başlangıçtaki sayfa sayısını gösterir.)
- Bir anahtara ulaşmak için önce hash değeri elde edilir.
- Eğer hash değeri NEXT pointer'ından daha önceki bir node'u belirlerse, $h_{level+1}$ fonksiyonuyla gerçek sayfaya ulaşılır.



Hashing Techniques and Expandable Files
Linear Hashing

- **Örnek 1:** Başlangıçta 4 sayfa oluşturulsun. Ana sayfa 3 ve overflow sayfası iki kayıt saklasın. Lower ve Upper Bound (40% - 80%).

10, 20, 3, 7, 13, 14, 17, 21, 25, 16, 22, and 30



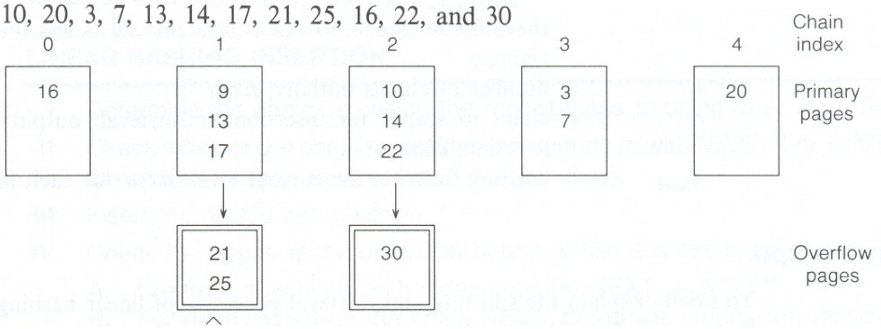
- Packing factor değeri 12/16 = 75%
- Yeni bir kayıt olarak 9 eklensin. 1. sayfaya gider ve overflow olur. (13/16) = 81%
- NEXT 0.sayfayı gösterdiği için 0.sayfa bölünür ve anahtarlar bir sonraki seviyenin hash fonksiyonuyla dağıtılır.

$$h_1(key) = key \bmod 8$$

Hashing Techniques and Expandable Files
Linear Hashing

- **Örnek 1:**

10, 20, 3, 7, 13, 14, 17, 21, 25, 16, 22, and 30



- 16 mod 8 = 0, 20 mod 8 = 4
- 20 değerine erişmek istenirse önce bulunulan seviyenin hesh fonksiyonu kullanılır.
- $h_0(20) = 0$ değeri NEXT pointer'ından küçük olduğu için 1.level hash fonksiyonu kullanılır ve $h_1(20) = 4$ değeri bulunur.
- Her sayfaya ulaşım bir probe olarak alınır. Sayfadaki arama primary memory'de olduğu için önemsenmez.

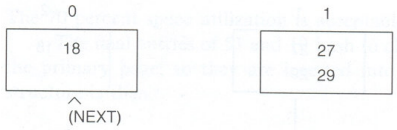
Hashing Techniques and Expandable Files
Linear Hashing

- **Örnek 2:** Başlangıçta 2 sayfa (N) oluşturulsun. Ana sayfa 2 ve overflow sayfası 1 kayıt saklasın. Lower ve Upper Bound (40% - 80%). Level = 0, NEXT = 0

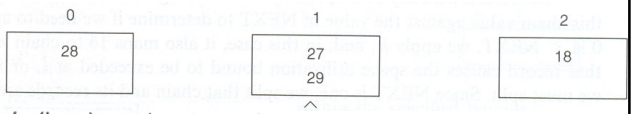
27, 18, 29, 28, 39, 13, 16, 51, and 19

$h_0(\text{key}) = \text{key mod } 2$

27, 18, 29 insert edildi.
Packing factor = 75%



28 insert edildi.
Packing factor = 100% oldu.



Yeni bir node eklendi ve 0.node h_1'e göre dağıtıldı.
$h_1(28) = 0$
$h_1(18) = 2$

NEXT artırıldı.

$h_1(\text{key}) = \text{key mod } 4$

18'e erişirken önce $h_0(18) = 0 <$ NEXT olduğu görülür ve $h_1(18) = 2$ ile elde edilir.

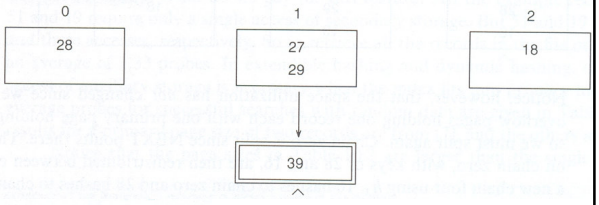
Hashing Techniques and Expandable Files
Linear Hashing

- **Örnek 2:** Başlangıçta 2 sayfa (N) oluşturulsun. Ana sayfa 2 ve overflow sayfası 1 kayıt saklasın. Lower ve Upper Bound (40% - 80%). Level = 0, NEXT = 0

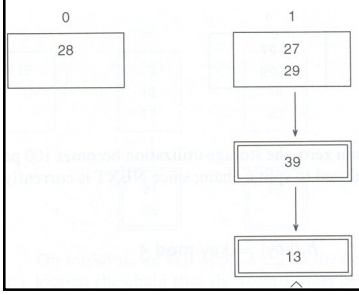
27, 18, 29, 28, 39, 13, 16, 51, and 19

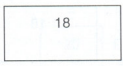
h_0(key) = key **mod** 2

h_1(key) = key **mod** 4



39 insert edildi.
Packing factor = 5/7 = 71%





13 insert edildi.
Packing factor = 6/8 = 75%

Hashing Techniques and Expandable Files
Linear Hashing

- **Örnek 2:** Başlangıçta 2 sayfa (N) oluşturulsun. Ana sayfa 2 ve overflow sayfası 1 kayıt saklasın. Lower ve Upper Bound (40% - 80%). Level = 0, NEXT = 0

27, 18, 29, 28, 39, 13, 16, 51, and 19

h_0(key) = key **mod** 2

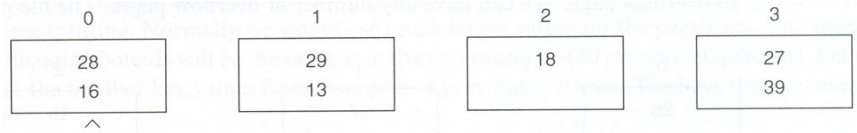
h_1(key) = key **mod** 4

16 insert edildi.
$h_o(16) = 0 <$ NEXT olduğundan
$h_1(16) = 0$ olur.
Packing factor = 7/8 = 87%
Yeni sayfa eklenir, 1.Sayfa bölünür ve h_1 ile kayıtlar dağıtılır.

NEXT artırıldığında **NEXT + 1 ≥ N * 2^{level}** olur
NEXT = 0, level = level + 1 yapılır.

Sonraki seviye için hash fonksiyonu. Her bölünmede yeni sayfa eklenince kullanılacak.

h_2(key) = key **mod** 8



Hashing Techniques and Expandable Files
Linear Hashing

- **Örnek 2:** Başlangıçta 2 sayfa (N) oluşturulsun. Ana sayfa 2 ve overflow sayfası 1 kayıt saklasın. Lower ve Upper Bound (40% - 80%). Level = 0, NEXT = 0

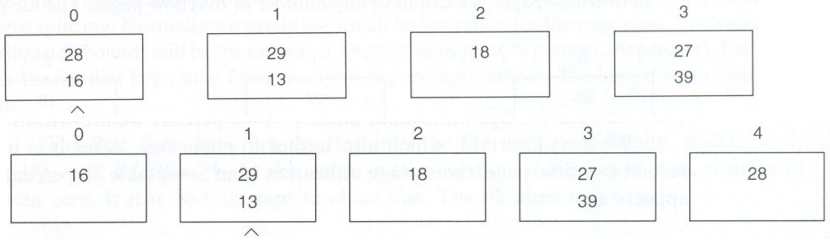
27, 18, 29, 28, 39, 13, 16, 51, and 19

h_0(key) = key **mod** 2

h_1(key) = key **mod** 4

h_2(key) = key **mod** 8

Packing factor = 7/8 = 87% olduğundan 0.sayfa bölünür, yeni sayfa eklenir ve h_2 ile kayıtlar dağıtılır. NEXT artırılır. Yeni packing factor = 7/10 = 70%



Hashing Techniques and Expandable Files
Linear Hashing

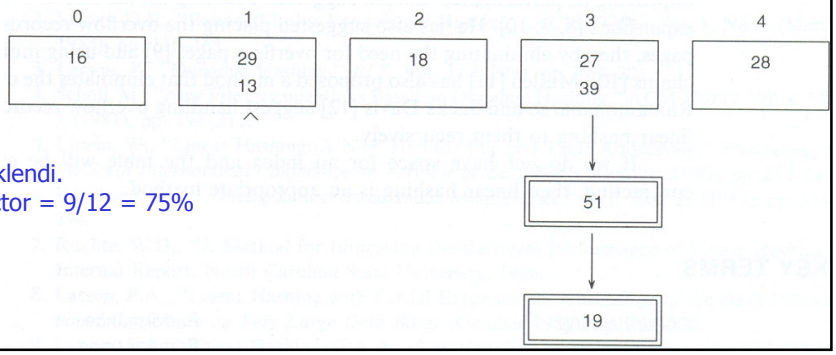
- **Örnek 2:** Başlangıçta 2 sayfa (N) oluşturulsun. Ana sayfa 2 ve overflow sayfası 1 kayıt saklasın. Lower ve Upper Bound (40% - 80%). Level = 0, NEXT = 0

27, 18, 29, 28, 39, 13, 16, 51, and 19

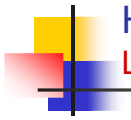
h_0(key) = key **mod** 2

h_1(key) = key **mod** 4

h_2(key) = key **mod** 8



51 ve 19 eklendi.
Packing factor = 9/12 = 75%



Hashing Techniques and Expandable Files

Linear Hashing

- **Değerlendirme**
  - 28'e ulaşılmak istenirse $h_1(28) = 0 <$ NEXT olduğundan $h_2(28) = 4$ bulunur. Sadece bir probe ile ulaşılır.
  - 19'a ulaşılmak istenirse $h_1(19) = 3 >$ NEXT 3.sayfaya gidilir. Sayfadaki overflow alanlarda aranır ve toplam probe sayısı 3 olur.

TABLE 10.2 AVERAGE RETRIEVAL PROBES FOR LINEAR HASHING

Primary page size	2 records			10 records			20 records			50 records		
Overflow page size	1	1	1	4	3	3	7	6	5	15	14	12
Packing factor	75%	85%	95%	75%	85%	90%	75%	85%	90%	75%	85%	90%
Average probes	1.43	1.68	2.22	1.14	1.33	1.57	1.08	1.24	1.44	1.05	1.20	1.35

Hashing Techniques and Expandable Files

Haftalık Ödev

Dynamic hashing ve Linear hashing metodlarında silme işleminin nasıl yapıldığını araştırınız ve bir rapor hazırlayınız.