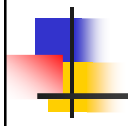


BM 307 Dosya Organizasyonu (File Organization)

Hazırlayan: M.Ali Akcayol
Gazi Üniversitesi
Bilgisayar Mühendisliği Bölümü

Konular

- Sequential File Organization
  - Binary Search
  - Interpolation Search
  - Self-Organizing Sequential Search
- Direct File Organization
  - Locating Information
  - Hashing Functions
  - Collision Resolution
  - Coalesced Hashing

Sequential File Organization
Background

- Fields
- Records (Fixed-length, Variable-length)
- Files
- Primary Key
- Secondary Keys

Employee name	Number	Title	Department	Manager	Salary	
Scott Matthews	0123	Programmer	Accounting	Melissa Jones	0026000	. . .
Tammy Boger	0240	Analyst	Data proc.	Morris Lancaster	0032000	. . .
Morris Lancaster	0067	Manager	Data proc.	Charles Hall	0045000	. . .
Larry Cookman	0189	Programmer	Security	John Bittle	0025000	. . .
David Caudle	0423	Analyst	Accounting	Melissa Jones	0047000	. . .
Richard Lehner	0847	Instructor	Training	Tom Nelson	0031000	. . .
Robert Blom	1002	Programmer	Accounting	George Steel	0027000	. . .
Nancy White	0417	Programmer	Operations	Curt Alexander	0028000	. . .
Randy Wheeler	0293	Analyst	Data proc.	William Crocker	0031500	. . .
John Bittle	0367	Manager	Security	Robert Wilson	0043000	. . .

Sequential File Organization

- Sequential Search
  - Ortalama n/2 probe gerektirir
  - Kayıt sırasında sıralama arama için gereken probe sayısını azaltır
  - Computational complexity *O*(n)
- Binary Search
  - Aranan alanın ortasından aramaya başlanır
  - Her defasında kayıtların yarısı elenir
  - Computational complexity $O(\log_2 n)$

```
[13,  16,  18,  27,  28,  29,  38,  39,  53]
[13,  16,  18,  27],  28,  29,  38,  39,  53
13,   16,  [18,  27],  28,  29,  38,  39,  53
13,   16],  [18,  27,  28,  29,  38,  39,  53
```

Sequential File Organization
Binary Search (Algoritma)

Algorithm 2.1
BINARY SEARCH

```
     proc binary_search
          /*  The n records of the file are ordered in in-
              creasing order of the keys.  */
1         LOWER := 1
2         UPPER := n
3         while LOWER ≤ UPPER do
4              MIDDLE := ⌊ (LOWER + UPPER) / 2 ⌋
5              if key[sought] = key[MIDDLE]
6                   then terminate successfully.
7              else if key[sought] > key[MIDDLE]
8                   then LOWER := MIDDLE + 1
9                   else UPPER := MIDDLE − 1
10        end
11        terminate unsuccessfully.
     end binary_search
```

Sequential File Organization
Interpolation Search

- Approximate relative position
- Telefon rehberinde bir kişinin aranması
- Aranan kaydın pozisyonun tahmin edilmesiyle bir sonraki pozisyon belirlenir

$$NEXT := \left\lceil LOWER + \frac{key[aranan] - key(LOWER)}{key[UPPER] - key(LOWER)} (UPPER\text{-}LOWER) \right\rceil$$

- Worst case computational complexity O(n)
- Average case computational complexity $O(\log_2 \log_2 n)$
- Anahtarların düzgün dağılımında performans artar
- Data primary memory'deyse binary search tercih edilmelidir, auxilary memory'deyse interpolation search tercih edilmelidir

Sequential File Organization

Self-Organizing Sequential Search

- Kayıtların yerlerini değiştirir
- Sık kullanılan kayıtlar dosyanın başına kaydırılır

En yaygın aşağıdaki 3 algoritma kullanılır

- Move_to_front
- Transpose
- Count

Sequential File Organization

- Move_to_front
  - Bir kayda ulaşıldığında dosyanın başına alınır
  - Bir kayıt ulaşıldıktan sonra çok az kullanılırsa diğer kayıtların arama süresini artırır
  - Bağlı yapılarla oluşturulur
  - Yer kısıtı olamadığı ve hızlı erişimin önemli olduğu durumlarda kullanılmalıdır
  - İşletim sistemlerinde kullanılan LRU (least recently used) algoritmasıyla aynıdır

Kayıtlara “fileorganization“ sırasında erişime örnek

	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
time ↓	l	i	f	a	b	c	d	e	g	h	j	k	m	n	o	p	q	r	s	t	u	v	w	x	y	z
	a	g	r	o	e	l	i	f	b	c	d	h	j	k	m	n	p	q	s	t	u	v	w	x	y	z
	n	o	i	t	a	z	g	r	e	l	f	b	c	d	h	j	k	m	p	q	s	u	v	w	x	y

Sequential File Organization

- Transpose
  - Bir kayda ulaşıldığında önündeki kayıtla yer değiştirilir
  - Move_to_front algoritmasına göre daha kararlıdır
  - Bir kayıt dosyanın başına gelene kadar bir çok kez ulaşılmalıdır
  - Kolay oluşturulabilir fazladan yer gereksinimi yoktur
  - Yer kısıtlı olduğunda kullanılmalıdır
- Count
  - Kayıtlar erişilme sayısında göre büyükten küçüğe doğru sıralanırlar
  - Ekstra kayıt alanı gerektirir
  - Eğer erişim sayıları başka bir amaç için kullanılacaksa bu yapı kullanılabilir

Direct File Organization

- İdeal olarak aranan anahtarla ilişkili kayda doğrudan bir probe ile ulaşmak istenir
- Anahtar tek bir adres (unique) olabilir
  - Eğer anahtar adres olursa bir seferde ulaşılabilir

  - Kullanılan alandan çok daha fazlası ayrılır

Direct File Organization

- Anahtar tek bir adrese (unique) dönüştürülebilir
  - Bir algoritmayla primary key bir adrese dönüştürülür
- Örnek : Havayolları için rezervasyon sistemi
  - Hergün 1-999 arası uçuş olsun
  - Yıl boyunca günler 1-366 ile numaralandırılır
  - Uçuş numarası ve gün ardarda eklenerek uçuşa ait adres bilgisi elde edilebilir

Location = uçuş numarası || gun numarası , adres aralığı 001001-999366 olur
(**Son adres hariç diğer adreslerde ???367 - ???999 arası hiç kullanılmaz)**

Location = gun numarası || uçuş numarası , adres aralığı 001001-366999 olur

Direct File Organization

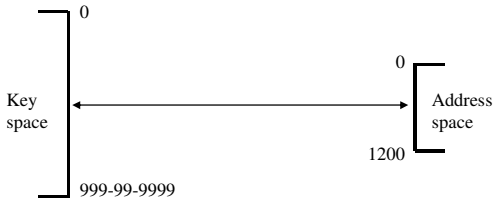
- Anahtar muhtemel bir adrese (probable) dönüştürülebilir
  - Adres alanında kullanılmayan alanlar silindiğinde 1-1 ilişki kaybolur ve anahtar alanından daha küçük adres alanı elde edilir
  - Daha geniş aralıktaki anahtarları daha küçük alandaki adreslere dönüştürmek için **hashing functions** kullanılır

Hash (key) ⟶ probable address

  - Başlangıçta elde edilen adres **home address** olarak adlandırılır
  - Hashing fonksiyonunun
    - adres aralığında anahtarları düzgün dağıtması istenir
    - hızlı çalışması istenir

Direct File Organization

- İki farklı anahtar aynı adrese eşleştirildiğinde **collision** (çakışma) olur



- Hash fonksiyonu aşağıdaki iki kısımdan oluşur;
  - eşleştirme fonksiyonu
  - collision çözme metodu

Direct File Organization
Hashing Functions

- Key mod N f(key) = key **mod** N
  - Örnek 27 mod 8 = 3
- Key mod P f(key) = key **mod** P
  - P ≥ N olmak üzere en küçük asal sayı
- Truncation
  - Anahtar içerisindeki en sağdaki veya en soldaki bir parçası alınır
  - Alınan kısım uygulamaya göre seçilir
- Folding
  - Folding by boundary

1	2	3	4	5	6	7	8	9

```
  3  2  1
  4  5  6
+ 9  8  7
```

  - Folding by shifting

```
  1  2  3
  4  5  6
+ 7  8  9
```

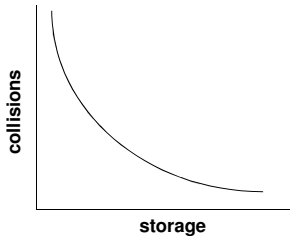
Direct File Organization
Hashing Functions

- Squaring
  - Bir anahtarın önce karesi alınır daha sonra truncation yapılabilir
- Radix conversion
  - 10 tabanından farklı bir tabandaymış gibi sayı 10 tabanına çevrilir
    $1234 = 1 * 11^3 + 2 * 11^2 + 3 * 11^1 + 4 * 11^0 = 1331 + 242 + 33 + 4 = 1610$
  - Ardından truncation kullanılabilir
- Polinomial hashing
  - f(information area) ⟶ cyclic check bytes
- Alphabetic keys
  - Alfanümerik anahtarların nümerik değerleri elde edilir
  - Karakter bazında veya tamamı birden sayısal değere dönüştürülür
  - Pascal'daki **variant records** yapısı bir değişkeni birden fazla farklı tür ve isimle kullanabilir

Direct File Organization
Collisions

- Bir data kümesi için bazı hashing fonksiyonları diğerlerine göre daha düzgün dağıtım yapabilir
- Bir hashing fonksiyonu aynı adrese çok sayıda kaydı atıyorsa (çok sayıda collision oluyorsa) **primary clustering** oluşur
- Axuliary memory'de erişim maliyeti çok fazla olduğundan daha karmaşık bir hashing fonksiyonuyla çakışmaları azaltmak gerekir
- Çakışmaları önlemek için diğer bir yöntem ise packing factor değerini azaltmaktır

$$\text{Packing factor} = \frac{\text{number of records stored}}{\text{total number of storage locations}}$$



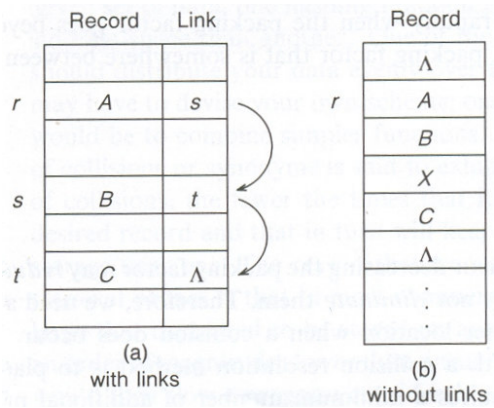
Direct File Organization
Collision Resolution

- Collision resolution with links
- Collision resolution without links
  - Static positioning of records
  - Dynamic positioning of records
- Collision resolution with pseudolinks

Direct File Organization

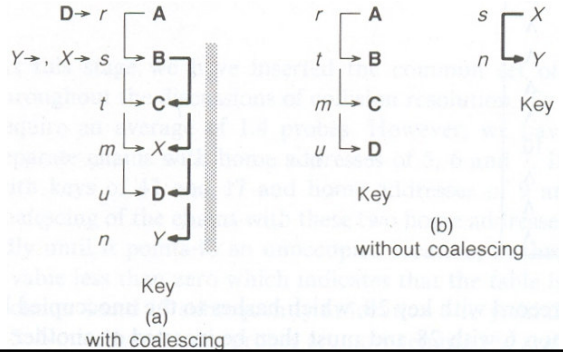
Collision resolution with links

- Aynı adrese sahip olan kayıtlar (synonym chain) arasında link oluşturulur
- Dezavantajı link için yer gerektirir
- Implied link kullanılabilir. Bir sonraki adres hesaplanabilir veya belirli bir kurala bağlı olabilir
- Aynı adrese sahip olan olan kayıtlar ait oldukları adresten sonraki ilk boş adrese yazılırlar
- Avantajı ayrıca yer gerektirmez



Direct File Organization
Coalesced Hashing

- Synonym chain elemanları arasında bağlantı için link kullanır
- Bir kayda ait home adresi başka bir synonym chain'de bir elemana ayrılmış olabilir
- Yeni eleman kendi home adresinden başlayarak zinciri takip eder ve zincirin sonuna yazılır
- İki farklı home adresine sahip iki farklı zincir birlikte büyür veya küçülür



Direct File Organization
Coalesced Hashing

(Örnek)

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

42 ve 17 eklendi

	Record	Link
0		Λ
1		Λ
2		Λ
3		Λ
4		Λ
5	27	Λ
6		Λ
7	18	10
8		Λ
9		Λ
R→ 10	29	Λ

	Record	Link
0		Λ
1		Λ
2		Λ
3		Λ
4		Λ
5	27	Λ
6	28	9
7	18	10
8		Λ
R→ 9	39	Λ
10	29	Λ

	Record	Link
0		Λ
1		Λ
2	13	Λ
3		Λ
4		Λ
5	27	8
6	28	9
7	18	10
R→ 8	16	Λ
9	39	Λ
10	29	Λ

	Record	Link
0		Λ
1		Λ
2	13	Λ
R→ 3	17	Λ
4	42	3
5	27	8
6	28	9
7	18	10
8	16	Λ
9	39	4
10	29	Λ

Direct File Organization
Coalesced Hashing

Değerlendirme

- **Örnekteki son tabloda packing factor = 9/11 (82%)**
- **Packing factor azaltılırsa coalescing azalır**
- **Coalescing azalırsa arama ve ekleme süresi azalır**
- **Erken eklenen kayıt listenin başında yer alır. Çok kullanılan kayıtlar önce eklenirse ortalama erişim süresi azalır**
- **Silme işlemi karmaşıktır**
- **Listenin sonundaki eleman silinen elemanın yerine getirilir**
- **Taşınan elemanın silinenle aynı home adrese sahip olması gerekir**
- **Coalescing olmadığı bilinirse bir kayıt bağlı listedeki gibi basit bir şekilde silinebilir**

Yandaki şekilde 39 silinirse elde edilen son durum görülmektedir

		Record	Link
	0		Λ
	1		Λ
	2	13	Λ
R→	3	17	Λ
	4	42	3
	5	27	8
	6	28	9
	7	18	10
	8	16	Λ
	9	39	4
	10	29	Λ

		Record	Link
	0		Λ
	1		Λ
	2	13	Λ
	3	17	Λ
	4		Λ
R→	5	27	8
	6	28	9
	7	18	10
	8	16	Λ
	9	42	3
	10	29	Λ

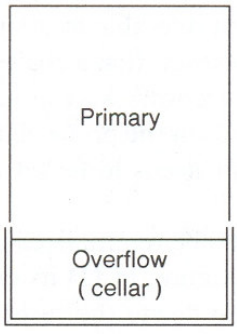
Direct File Organization
Coalesced Hashing

Variants

- Table organization (overflow area kullanımı)
- Çakışma olan kaydın zincire bağlanma şekli
- Çakışma olan kayıt için kullanılacak olan boş yerin seçimi

Table Organization

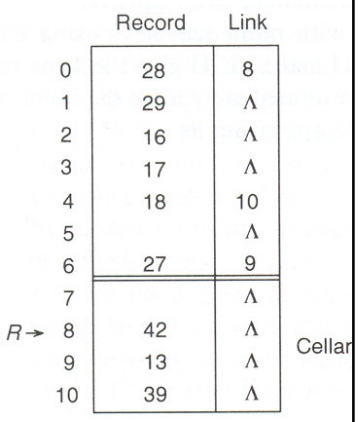
- Tablo primary area ve overflow area olarak iki kısımdan oluşur
- Adres factor = (primary area) / (total table size)
- Çalışmalar adres faktörünün 0.86 olmasının en iyi performansı verdiğini göstermiştir



Direct File Organization
Coalesced Hashing

Variants

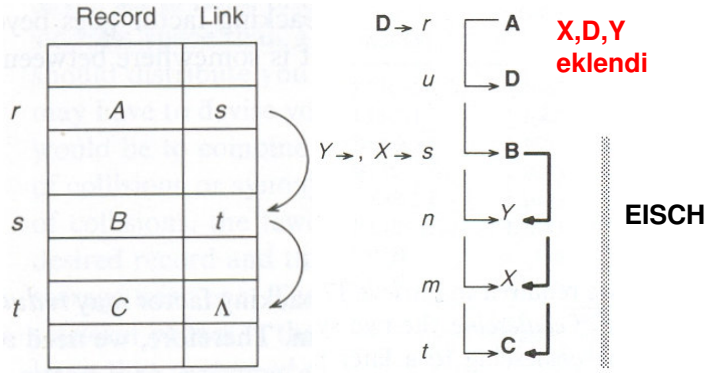
- Late Insertion Standart Colesced Hashing (**LISCH**) overflow olmadan kullanılan algoritmadır ve zincirin sonuna ekleme yapar
- Bu algoritmanın diğer bir türü Late Insertion Coalesced Hashing (**LICH**) overflow area kullanır
- 27, 18, 29, 28, 39, 13, 16, 42, 17 değerleri key **mod** 7 hashing fonksiyonuyla yandaki gibi yerleştirilir
- Bir kayıt için ortalama probe sayısı 1.3 tür. LISCH ile bu oran 1.8 dir.
- Genel olarak 90% packing factor ile 0.86 overflow area kullanımı probe sayısını LISCH'ye göre 6% azaltmaktadır



Direct File Organization
Coalesced Hashing

Variants

- Early Insertion Standart Colesced Hashing (**EISCH**) overflow olmadan kullanılan algoritmadır ve zincirin başındaki elemandan hemen sonra ekleme yapar



- LISCH algoritmasında tüm kayıtların alınması için 19 probe gerekir, EISCH algoritmasında ise 17 probe gerekir

Direct File Organization
Coalesced Hashing

Variants

- EISCH algoritmasıyla 17 sayısı aşağıdaki şekildeki gibi eklenir
- Hash (key) = key **mod** 11

		Record	Link
	0		Λ
	1		Λ
	2	13	Λ
	3		Λ
R→	4	42	Λ
	5	27	8
	6	28	9
	7	18	10
	8	16	Λ
	9	39	4
	10	29	Λ

		Record	Link
	0		Λ
	1		Λ
	2	13	Λ
R→	3	17	9
	4	42	Λ
	5	27	8
	6	28	3
	7	18	10
	8	16	Λ
	9	39	4
	10	29	Λ

Direct File Organization
Coalesced Hashing

Variants

- Random Early Insertion Standart Colesced Hashing (**REISCH**) overflow olmadan kullanılan algoritmadır ve zincirin başındaki elemandan hemen sonra ekleme yapar. Eklenecek boş alanı rastgele seçer.
- REISCH algoritması EISCH algoritmasına göre 1% iyileştirme sağlar
- Random Late Insertion Standart Colesced Hashing (**RLISCH**) overflow olmadan kullanılan algoritmadır ve zincirin sonuna ekleme yapar. Eklenecek boş alanı rastgele seçer.
- Bidirectional Late Insertion Standart Colesced Hashing (**BLISCH**) boş alanı tablonun bir altından birde üstünden seçer. Boş alana alınan bilgi listenin sonuna eklenir.
- Bidirectional Early Insertion Standart Colesced Hashing (**BEISCH**) boş alanı tablonun bir altından birde üstünden seçer. Boş alana alınan bilgi listenin başındaki elemandan sonra eklenir.

Direct File Organization

Coalesced Hashing

Karşılaştırma

TABLE 3.1 MEAN NUMBER OF PROBES FOR SUCCESSFUL LOOKUP ($n = 997$) FOR VARIANTS OF COALESCED HASHING

α method	0.2	0.4	0.6	0.8	0.9	0.95	0.99
EISCH	1.1065	1.2277	1.3684	1.5290	1.6182	1.6653	1.7033
LISCH	1.1063	1.2316	1.3789	1.5657	1.6737	1.7337	1.7827
BEISCH	1.1055	1.2286	1.3721	1.5336	1.6236	1.6728	1.7107
BLISCH	1.1055	1.2341	1.3836	1.5703	1.6818	1.7423	1.7898
REISCH	1.1063	1.2322	1.3693	1.5257	1.6124	1.6614	1.7014
RLISCH	1.1085	1.2384	1.3876	1.5653	1.6723	1.7296	1.7790
EICH	1.1116	1.2256	1.3408	1.4942	1.5867	1.6347	1.6762
LICH	1.1116	1.2256	1.3406	1.4888	1.5801	1.6281	1.6695

Source: Hsiao, Yeong-Shiou, and Alan L. Tharp, "Analysis of Other New Variants of Coalesced Hashing," Technical Report TR-87-2, Computer Science Department, North Carolina State University, 1987.

Direct File Organization

Haftalık Ödev

1000 tane rastgele değere sahip anahtar için LISCH, EISCH, LICH, EICH, RLISCH, REISCH, BLISCH, BEISCH coalesced hashing algoritmalarını C#.NET ile gerçekleştiriniz. Performanslarını karşılaştıran bir arayüz programı hazırlayınız.