Collision

- Collision merupakan kondisi dimana terdapat lebih dari satu key yang menempati slot address yang sama
- Collision dapat diminimalisir dengan cara :

  - Mengganti fungsi hash

  - Mengurangi packing factor
- Packing Factor / packing density / load factor adalah perbandingan antara jumlah data yang tersimpan terhadap jumlah slot address yang tersedia

$$PackingFactor = \frac{number\_of\_records\_stored}{total\_number\_of\_storage\_locations}$$

Collision Resolution

- Mengganti fungsi hash atau mengurangi packing factor hanyalah suatu teknik untuk mengurangi terjadinya collision, tetapi tidak mengeliminasinya

- Karenanya, diperlukan Collision Resolution, yaitu prosedur untuk menempatkan data yang memiliki home address yang sama, sedemikian hingga banyaknya akses dari home address seminimum mungkin

Collision Resolution

- Misalnya kita memiliki record A, B, dan C yang sinonim dengan home addressnya ada di r, maka kita membuat link untuk memindahkan record B ke posisi s dari posisi r, kemudian record C akan menempati lokasi t, sedangkan linknya diisi dengan ^ atau null.

Dengan Link

	Record	Link
r	A	S
s	B	T
t	C	^

Tanpa Link

	Record
	^
R	A
	B
	X
	C
	^
	.
	.
	.
	.
	.

Metoda Collision Resolution

- Metoda collision resolution diklasifikasikan sebagai berikut :
  - Collision Resolution with Link

    → Coalesced Hashing
  - Collision Resolution without Link

    → Progresive Overflow, Linear Qoutient, Binary Tree, Brent's method
  - Collision Resolution with pseudolinks

    → Computed Chaining

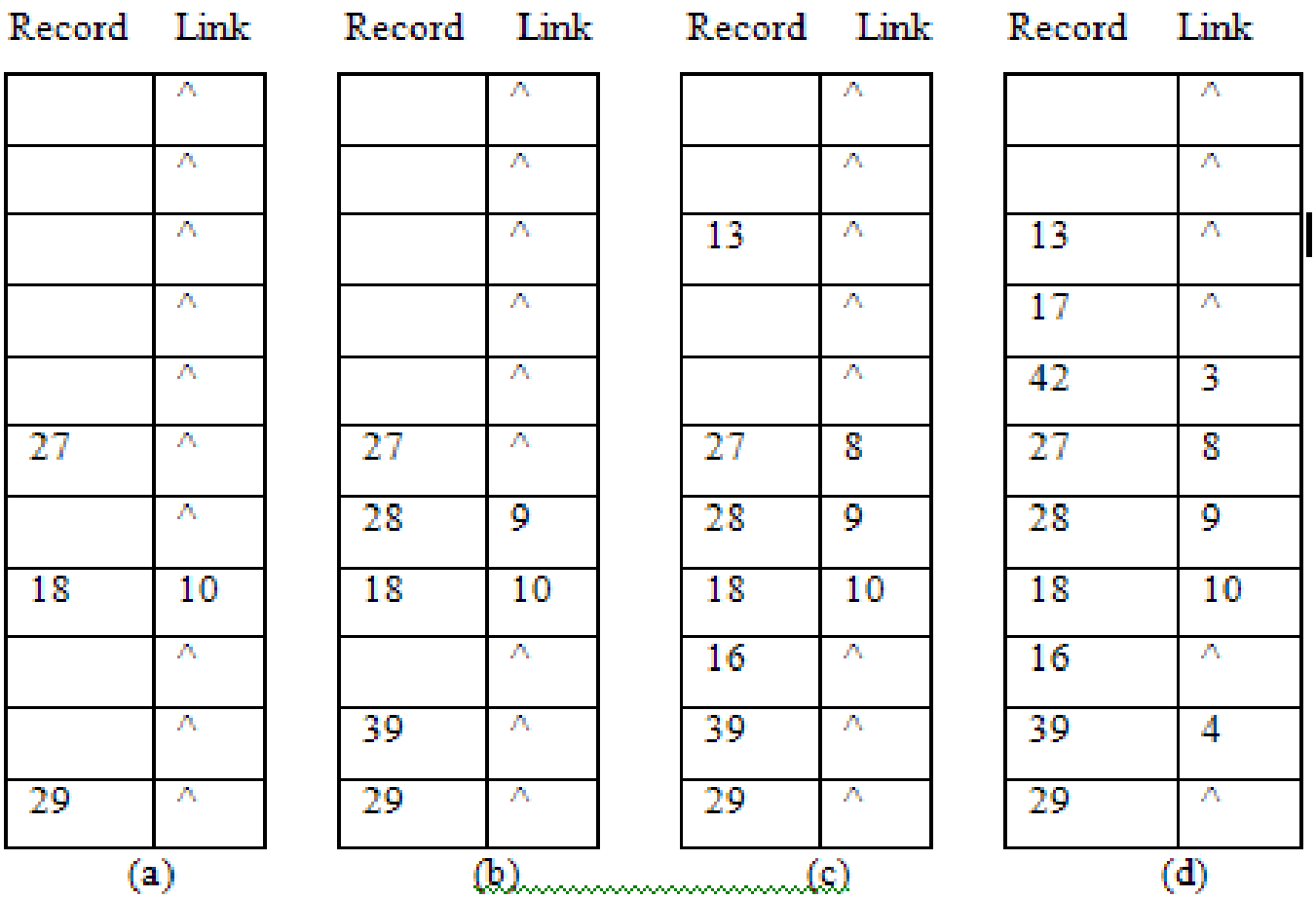
Collision Resolution with Link→Coalesced Hashing

1. LISCH (Late Insertion Standart Coalesced Hashing)

- Suatu metode dimana record baru yang disisipkan akan menempati pada posisi terakhir dari probe chain.
- Contoh : dengan menggunakan metode collision dan fungsi hash : Hash (key) = key mod 11, dimana 11 adalah ukuran table.
- Record yang akan dimasukkan dengan key : ***27, 18, 29, 28, 39, 13 dan 16***.
- Hasil address dari 0 sampai 10 dari record tersebut yaitu : Hash(27)=5, Hash(18)=7, Hash(29)=7, Hash(28)=6, Hash(39)=6, Hash(13)=2, Hash(16)=5.

Collision Resolution with Link→Coalesced Hashing

Dalam bentuk gambaran penyisipan setiap langkah LISCH adalah



Collision Resolution with Link→Coalesced Hashing

2. EISCH (Early Insertion Standart Coalesced Hashing)

- Suatu metode dimana record baru yang disisipkan diletakkan pada posisi tengah setelah record sinonimnya dengan merubah nilai link dari record sinonimnya.
- Contoh : dengan menggunakan metode collision dan fungsi hash : Hash (key) = key mod 11, dimana 11 adalah ukuran table.
- Record yang akan dimasukkan dengan key : ***27, 18, 29, 28, 39, 13 dan 16***.
- Hasil address dari 0 sampai 10 dari record tersebut yaitu : Hash(27)=5, Hash(18)=7, Hash(29)=7, Hash(28)=6, Hash(39)=6, Hash(13)=2, Hash(16)=5, Hash(42)=9 dan Hash(17)=6 .

Collision Resolution with Link→Coalesced Hashing

Dalam bentuk gambaran penyisipan setiap langkah EISCH adalah

Record	Link
	^
	^
13	^
	^
42	^
27	8
28	9
18	10
16	^
39	4
29	^

(a)

Record	Link
	^
	^
13	^
17	9
42	^
27	8
28	3
18	10
16	^
39	4
29	^

(b)

Collision Resolution without Link

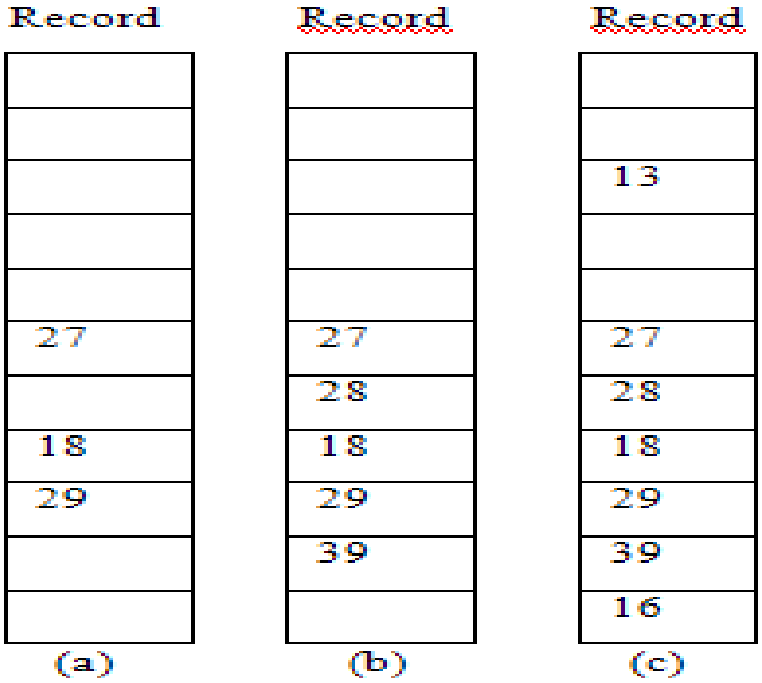
1. Progresive Overflow

- Kerugian dari coalesced hashing adalah diperlukannya storage tambahan untuk menyimpan field link.
- Untuk mengatasinya dapat menggunakan bentuk sederhana dalam menyimpan record dengan metode progressive overflow
- Progressive overflow :

  - Record yang akan disimpan diletakkan pada lokasi berikutnya dari record pada posisi home addressnya.
  - Apabila lokasi yang dipergunakan untuk menyisipkan lebih besar dari ukuran table maka lokasinya dikembalikan keposisi pertama dari tabel,
  - Dan seterusnya sampai ditemukan lokasi kosong untuk menyimpan record tersebut.

Collision Resolution without Link

- Contoh : dengan menggunakan fungsi hash : Hash (key) = key mod 11, dimana 11 adalah ukuran table.
- Misal kita memiliki record dengan key : ***27, 18, 29, 28, 39, 13 dan 16***.
- Dapat dicari nilai masing-masing address dari 0 sampai 10 dari record tersebut yaitu : Hash(27)=5, Hash(18)=7, Hash(29)=7, Hash(28)=6, Hash(39)=6, Hash(13)=2, Hash(16)=5
- Bentuk gambaran penyisipan setiap langkah Progresive Overflow adalah :



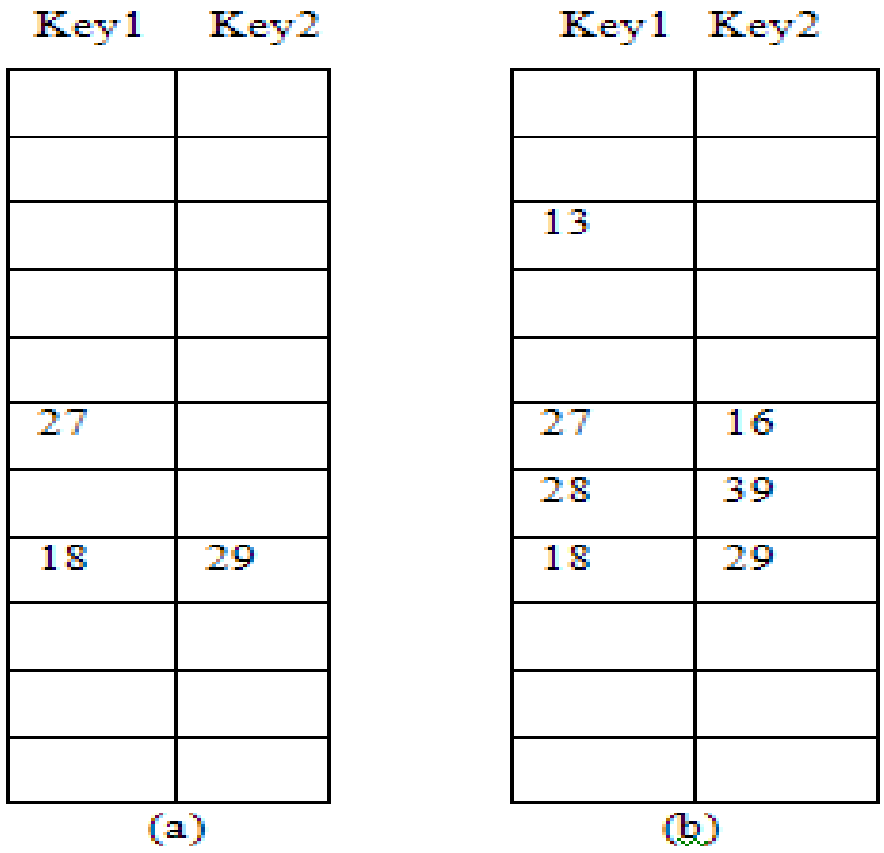
Collision Resolution without Link

2. Use Of Bucket

- Metode ini digunakan untuk menghindari collision dengan cara membuat lokasi penyimpanan diperbolehkan untuk menyimpan multiple record yang berbentuk bucket (atau blok atau page)
- Jumlah record yang boleh disimpan dalam suatu bucket disebut blocking factor
- Contoh : penyimpanan record dengan blocking factor 2, dari record key : ***27, 18, 29, 28, 39, 13 dan 16***
- Dengan menggunakan metode collision dan fungsi hash : Hash (key) = key mod 11, dimana 11 adalah ukuran table
- Hasil address dari 0 sampai 10 dari record tersebut yaitu : Hash(27)=5, Hash(18)=7, Hash(29)=7, Hash(28)=6, Hash(39)=6, Hash(13)=2, Hash(16)=5

Collision Resolution without Link

Dalam bentuk gambaran penyisipan setiap langkah Use Of Bucket adalah :



Collision Resolution without Link

3. Linear Quotient

- Perbedaaan utama dari linear Quotient collision resolution dan progressive overflow adalah bahwa dalam Linear Quotient kita menggunakan variable increment dari sebuah konstanta dengan pertambahan 1
- Sehingga dalam menyisipkan record baru diperlukan dua buah fungsi hash sebagai berikut :

  H1 (Key) = Key Mod P

  H2 = Quotient(Key/P) Mod P atau

  H1 (Key) = Key Mod P

  H2 = (Key Mod (P-2)) , dimana P adalah ukuran tabel
- Contoh dengan menggunakan fungsi hash : $H_1(key) = key \bmod 11$
- Tentukan lokasi dengan menggunakan linear quotient dari record key : **27, 18, 29, 28, 39, 13, dan 16**. Hasil address dari 0 sampai 10, dari record tersebut yaitu: $H_1(27) = 5$, $H_1(18) = 7$, $H_1(29) = 7$, $H_1(28) = 6$, $H_1(39) = 6$, $H_1(13) = 2$, $H_1(16) = 5$, $H_2(29) = 2$, $H_2(39) = 3$, $H_2(16) = 1$

Collision Resolution without Link

Dalam bentuk gambaran penyisipan setiap langkah Linear Quotient adalah :