BAB III

METODE HASHING UNTUK PENCARIAN

Sebuah kamus (*dictionary*), dapat dipertimbangkan sebagai suatu tipe data abstrak (*abstract data type / ADT*). Contoh kamus dapat ditemukan dalam banyak aplikasi. Misalnya dalam sebuah pengolah kata (*word processor*) yang dilengkapi dengan sebuah kamus pengeja kata (*the spelling checker*), *the thesaurus*, kamus data (*the data dictionary*) yang ada dalam penerapan manajemen database (*the database management*), dan juga tabel simbol (*the symbol table*) yang dibangkitkan oleh *loader*, *assembler* dan *compiler*.

Dalam Ilmu Komputer, istilah tabel simbol lebih umum digunakan dari pada kamus, apabila dihubungkan dengan tipe data abstrak. Dipandang dari perspektif ini, suatu simbol tabel didefinisikan sebagai himpunan pasangan nama-atribut (*name-attribute*). Karakteristik dari nama dan atribut berbeda menurut penerapannya. Sebagai contoh, dalam thesaurus, nama adalah suatu kata dan atribut adalah daftar sinonim untuk kata itu; dalam tabel simbol untuk compiler, nama adalah suatu kata untuk pengenal (*identifier*) dan atribut mungkin mengandung suatu harga awal dan suatu daftar baris yang menggunakan pengenal itu.

Operasi-operasi yang umum dilakukan pada suatu tabel simbol adalah :

1. menentukan apakah suatu nama tertentu ada dalam tabel.
2. mendapatkan lagi (*retrieve*) atribut-atribut dari nama itu.
3. memodifikasi atribut-atribut dari nama itu.
4. menyisipkan (*insert*) suatu nama baru dan atribut-atributnya.
5. menghapus (*delete*) suatu nama dan atributnya.

Dari kelima operasi tabel simbol di atas, tiga diantaranya merupakan operasi utama, yaitu : pencarian, penyisipan dan penghapusan.

Teknik paling sederhana yang bekerja dengan baik untuk operasi tabel simbol adalah dengan tabel pengalamatan langsung (*direct-address table*). Hal ini mungkin dapat dilakukan apabila himpunan semesta U-nya relatif kecil.

Andaikan bahwa suatu aplikasi membutuhkan himpunan dinamis yang tiap elemen mempunyai satu kunci yang diambil dari $U = \{0,1,\ldots,m-1\}$, dengan m tidak terlalu besar dan tidak ada dua elemen mempunyai kunci sama.

Untuk merepresentasikan himpunan dinamis seperti itu digunakan larik (*array*). Dengan demikian, apabila kunci berupa integer, maka kunci tersebut bisa digunakan sebagai indeks dari larik yang dimaksud.

Contoh 1 :

Misalkan terdapat n rekord pekerja yang masing-masing diidentifikasi dengan maksimal dua digit pengenal. Maka seorang pekerja dapat diidentifikasi dengan satu nomor pekerja yang harganya terletak antara 0 dan $n-1$. Dengan pengalamatan langsung, data tersebut disimpan dengan bentuk

```
type NomorPekerja = array [0..n-1] of string;
var Pekerja : NomorPekerja;
```

dimana Pekerja[i] menunjukkan pekerja yang mempunyai pengenal (NomorPekerja) i. □

Operasi-operasi tabel simbol pengalamatan langsung seperti di atas mudah untuk diimplementasikan.

Yaitu :

```
DIRECT-ADDRESS-SEARCH(T,k)
    return T[k]
DIRECT-ADDRESS-INSERT(T,x)
    T[key[x]] <-- x
DIRECT-ADDRESS-DELETE(T,x)
    T[key[x]] <-- NIL
```

3.1. Tabel Hash

Kesulitan pengorganisasian simbol tabel dengan pengalamatan langsung adalah apabila himpunan semesta U berukuran besar, penempatan tabel T berukuran $|U|$ mungkin

menjadi tidak praktis atau bahkan tidak mungkin mengingat ketersediaan memori pada suatu jenis komputer tertentu.

Contoh 2 :

Misal diasumsikan bahwa pengenal-pengenal yang digunakan untuk mengidentifikasi suatu rekord itu panjangnya dibatasi paling banyak enam karakter, dengan karakter pertama adalah sebuah huruf dan sisanya berupa huruf atau digit desimal, maka ada

$$\sum_{0 \le i \le 5} 26 \times 36^{i}$$

$$= 26 + 26 \times 36^{1} + 26 \times 36^{2} + 26 \times 36^{3} + 26 \times 36^{4} + 26 \times 36^{5}$$

$$> 1{,}6 \times 10^{9}$$

harga-harga berbeda yang mungkin.
Mengingat bahwa sebarang aplikasi hanya menggunakan suatu bagian yang sangat kecil dari jumlah ini dengan penggunaan memori yang lebih sedikit, penggunaan semua harga-harga berbeda yang mungkin itu tampak tidak praktis dan efisien.

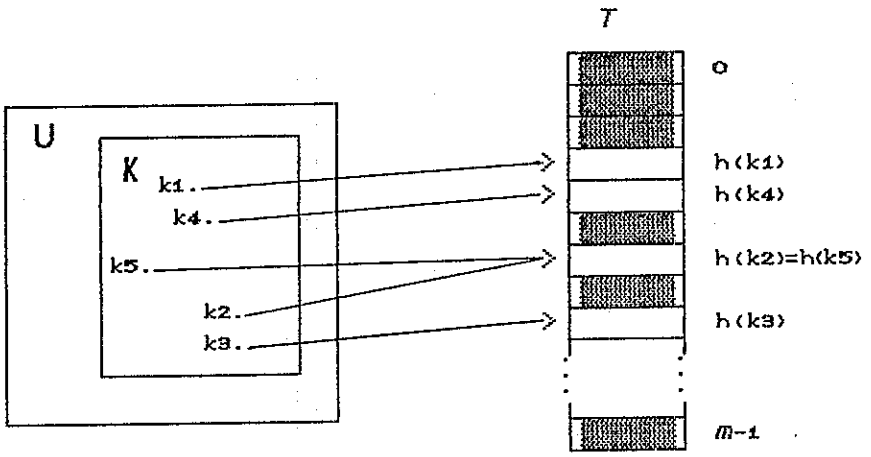
□

Selanjutnya, misalkan K adalah himpunan kunci-kunci yang sebenarnya ditempatkan (*keys actually stored*) berukuran relatif kecil dibandingkan dengan U, maka banyak ruang yang dialokasikan untuk tabel pengalamatan langsung menjadi boros.

Apabila himpunan kunci-kunci yang ditempatkan, K,dalam suatu kamus berukuran sangat lebih kecil dibanding dengan himpunan semesta atau himpunan semua kunci-kunci yang mungkin, U, suatu tabel hash membutuhkan ruang penempatan yang jauh lebih kecil lagi daripada tabel pengalamatan langsung. Tegasnya, kebutuhan ruang penempatan dapat dikurangi menjadi $\Theta(|K|)$, sedangkan pencarian untuk suatu elemen dalam tabel hash itu tetap membutuhkan waktu hanya $O(1)$ pada rata-rata, padahal untuk pengalamatan langsung ia merupakan waktu worst-case.

Pada pengalamatan langsung suatu elemen dengan kunci k diletakkan dalam slot s, sedangkan dengan hashing, elemen ini diletakkan dalam slot $h(k)$; yaitu suatu fungsi hash h digunakan untuk menghitung slot dari kunci k itu. Disini h memetakan himpunan semesta kunci-kunci U ke dalam slot-slot suatu tabel hash $T[0..m-1]$:

$$h : U \longrightarrow \{0,1,\ldots,m-1\}.$$



Gambar 3.1. Penggunaan suatu fungsi hash h untuk memetakan kunci-kunci ke slot-slot tabel hash.

Suatu elemen dengan kunci *k* di-hash ke slot *h(k)*; juga *h(k)* adalah harga hash (*hash value*) dari kunci *k*. Gambar 3.1 mengilustrasikan ide dasar ini; tampak bahwa kunci k_1 dan k_5 dipetakan ke slot yang sama. Disini, tugas utama dari fungsi hash adalah untuk mengurangi range dari indeks larik yang perlu ditangani. Sebagi pengganti harga $|U|$, jumlah yang ditangani hanyalah *m* harga, dan secara bersamaan kebutuhan ruang penempatan berkurang pula.

Definisi 1 :

Kepadatan pengenal (*the identifier density*) dari suatu tabel hash adalah $|K|/|U|$, dengan $|K|$ adalah jumlah pengenal dalam tabel dan $|U|$ adalah total jumlah pengenal yang mungkin . ■

Himpunan pengenal yang mungkin diistilahkan pula dengan himpunan semesta U dari pengenal-pengenal itu sedangkan himpunan pengenal dalam tabel (K) disebut pula dengan himpunan kunci aktual (*actual key*).

Definisi 2 :

Kepadatan beban (*the loading density*) atau faktor beban (*the loading factor*) dari suatu tabel hash adalah

$$\alpha = |K|/(sm)$$

dengan *m* adalah jumlah kotak suatu tabel hash dan tiap kotak mampu menampung *s* slot. ■

Karena jumlah pengenal $|K|$ dalam penggunaan biasanya beberapa tingkat besaran kurang dari total jumlah pengenal yang mungkin $|U|$, maka jumlah kotak m dalam tabel hashnya juga kurang dari $|U|$.

Sedang total jumlah U dapat diambil sebagian kecil dari harga-harga berbeda yang mungkin seperti dalam Contoh 2 dengan jumlah kunci aktual sebanyak yang dibutuhkan, misalnya 100 atau 200 buah.

Definisi 3 :

Dua kunci pengenal k_1 dan k_2 dikatakan sinonim (*synonym*) terhadap fungsi hash h apabila $h(k_1) = h(k_2)$.

Overflow terjadi apabila suatu kunci pengenal k dipetakan / di-hash-kan oleh suatu fungsi hash h ke dalam kotak yang penuh.

Tabrakan (*collision*) terjadi apabila dua kunci pengenal yang tidak identik di-hash-kan ke dalam slot yang sama.

■

Dengan demikian, sinonim-sinonim yang berbeda dimasukkan ke dalam kotak yang sama sepanjang slot-slot dalam kotak itu belum penuh. Apabila slot-slot suatu kotak telah penuh, penyisipan kunci pengenal baru ke dalam kotak itu mengakibatkan terjadinya overflow. Tentu saja, apabila ukuran slot s adalah 1, tabrakan dan overflow terjadi bersamaan.

Contoh 3 :

Perhatikan tabel hash T dengan $m = 26$ kotak dan $s = 2$ (Gambar 3.2).

Diasumsikan bahwa ada $|K| = 10$ pengenal berbeda dan tiap pengenal dimulai dengan suatu huruf. Faktor beban untuk untuk tabel ini adalah $\alpha = 10/(26.2) = 10/52 = 0{,}19$.

Fungsi hash h harus memetakan tiap pengenal yang mungkin ini ke dalam satu dari bilangan 0 sampai dengan 25. Jika perwakilan biner internal (*the internal binary representation*) untuk huruf-huruf A sampai Z berturut-turut berkorespondensi dengan bilangan 0 sampai 25, maka fungsi h yang didefinisikan dengan

$h(k)$ = karakter pertama k

akan meng-hash-kan semua pengenal k ke dalam tabel hash itu. Pengenal-pengenal GA, D, A, G, L, A2, A1, A3, A4 Dan E berturut-turut akan di-hash-kan ke dalam kotak 6, 3, 0, 6, 11, 0, 0, 0, 0 dan 4 oleh fungsi hash ini. □

	slot 1	slot 2
0	A	A2
1		
2		
3	D	
4		
5		
6	GA	G
⋮	⋮	⋮
25		

Gambar 3.2. Tabel hash dengan 26 kotak dan dua slot per kotak.

Tampak bahwa kunci GA dan G berada dalam kotak yang sama dan tiap kotak mempunyai dua slot. Pengenal berikutnya, A1, di-hash-kan ke dalam kotak T[0]. Kotak ini penuh, sehingga terjadi overflow dan suatu pencarian kotak itu mengindikasikan bahwa A1 tidak berada dalam kotak.

3.2. Tahap Persiapan

Penggunaan metode hashing untuk pencarian memerlukan persiapan awal sebelum fungsi hash tertentu digunakan. Hal ini dikarenakan tiap elemen kunci sering mengandung karakter-karakter alfanumerik. Beberapa karakter-karakter ini secara aritmetika ataupun logika sulit dimanipulasi, sehingga tepat untuk mengkonversi kunci-kunci yang demikian agar dapat lebih mudah dimanipulasi oleh suatu fungsi hash. Proses konversi ini sering disebut dengan persiapan (*preconditioning*).

Contoh 4 :

Misalkan kunci yang akan digunakan adalah RATE1*. Satu kemungkinan adalah menyandikan huruf latin sebagai bilangan 11, 12, ..., 36, dan himpunan karakter-karakter khusus (seperti +, -, *, /, ...) sebagai bilangan 37, 38, 39, ... Dengan menggunakan pendekatan ini, RATE1 disandikan sebagi bilangan 281130150139 (yaitu karakter-karakter R, A, T, E, 1 dan * berturut-turut diganti dengan bilangan-bilangan bulat 28, 11, 30, 15, 01, 39). □

Cara persiapan yang paling efisien dilakukan adalah dengan menggunakan perwakilan internal yang disandikan secara numerik (*the numerically coded internal representation*) (misalnya EBCDIC atau ASCII) dari tiap karakter kuncinya.

Contoh 5 :

Untuk kunci A1, dalam ASCII, karakter A mempunyai nilai 65_{10} = 1000001_2 dan karakter 1 mempunyai nilai 31_{10} = 0110001_2. Dengan menginterpretasikan kunci-kunci sebagai bilangan basis 2 14-digit, kunci A1 mempunyai kode biner 10000010110001_2. Nilai ini sama dengan 8.345_{10}. Secara sama, perwakilan EBCDIC dari A1 disandikan biner sebagai 1100000111110001_2 atau 49.649_{10}. □

Umumnya hasil persiapan sebuah kunci mungkin tidak cocok dengan ukuran suatu *word* memori. Dalam kejadian seperti itu, digit-digit tertentu hasil persiapan tersebut dapat dibatalkan, misalnya dengan menggunakan suatu fungsi hash yang melakukan transformasi pengurangan ukuran. Hal ini dilakukan mengingat bahwa seringkali satu fungsi hash membangun hasil persiapan itu dan kemudian fungsi hash kedua memetakan hasil ini ke dalam lokasi tabel.

Akhirnya, tentu saja bagi kunci yang tiap elemennya hanya terdiri dari karakter-karakter numerik tahap persiapan ini dapat tidak diperlukan (yaitu apabila elemen-elemen kuncinya ingin diinterpretasikan sebagai

bilangan bulat). Sebagai contoh himpunan kunci U = {23, 45, 18, 52, 67} tidaklah perlu dilakukan persiapan apabila digunakan fungsi hash division.

Dalam suatu program implementasi penggunaan metode hashing, biasanya proses persiapan ini disatukan bersama dengan fungsi hashnya dalam suatu fungsi atau prosedur.

Dalam pembahasan Tugas Akhir ini asumsi yang dipakai adalah bahwa kunci-kunci yang akan dihitung harga hashnya merupakan kombinasi dari karakter alfanumerik.

3.3. Fungsi Hash

Ada beberapa kriteria yang harus diperhatikan untuk pemilihan suatu fungsi hash yang baik, yaitu :

1. Kecepatan dan kemudahan perhitungannya.

   Suatu fungsi hash yang baik haruslah sangat cepat dan mudah memanipulasi suatu kunci.

2. Ketergantungan pada bit-bit kunci.

   Fungsi hash yang baik harus bergantung pada semua bit-bit kunci agar tidak ada informasi yang hilang. Kejadian kehilangan informasi ini dapat memperbesar tabrakan yang terjadi, sehingga dapat pula menyebabkan kegagalan suatu operasi tabel simbol.

   Tampak fungsi seperti pada Contoh 3 bukan merupakan pilihan fungsi hash yang baik untuk operasi tabel simbol, meskipun mudah untuk menghitungnya. Hal ini disebabkan karena fungsi itu hanya bergantung pada karakter pertama dalam pengenal tersebut.

3. Fungsi hash seharusnya menghasilkan nilai acak (*random*). Dengan demikian akan menghindari tabrakan atau paling tidak meminimumkan.

   Fungsi hash mengambil sebuah elemen kunci untuk diletakkan dalam suatu tabel dan mentransformasikannya ke dalam sebarang lokasi dalam tabel itu. Jika transformasi ini membuat lokasi-lokasi tabel tertentu lebih mungkin terjadi dari lainnya, maka kesempatan tabrakan bertambah dan efisiensi operasi tabel simbol berkurang. Fenomena sebarang lokasi-lokasi tabel menjadi lebih sering terjadi ini dinamakan *primary clustering.*

   Fungsi hash ideal menyebarkan elemen-elemen itu (dengan menghasilkan nilai random) secara seragam (*uniform*) diseluruh tabelnya, yaitu tidak menunjukkan adanya primary clustering. Hal ini mengarahkan pada sifat fungsi hash ideal berikut.

4. Fungsi hash ideal seharusnya menghentikan secara almiah kejadian cluster (*break up naturally occuring clusters*) kunci-kuncinya. Sifat ini berkaitan dengan karakteristik ketiga.

Kenyataanya, tidaklah mungkin mendapatkan suatu fungsi hash yang memilih suatu lokasi random dimana kuncinya diletakkan. Hal ini dikarenakan fungsi hash *h* tidak dapat bersifat kemungkinan (*probabilistic*) tetapi ia harus bersifat menentukan (*deterministic*) yaitu

menghasilkan lokasi yang sama setiap waktu diterapkan pada elemen yang sama.

Fungsi hash yang baik memenuhi (mendekati) asumsi hashing seragam sederhana : tiap kunci berkesempatan sama (*equally likely*) untuk di-hash-kan ke sebarang m buah kotak. Secara lebih formal misal diasumsikan bahwa tiap kunci yang diambil secara independen dari semesta U menurut suatu distribusi probabilitas P; yaitu $P(k)$ adalah probabilitas bahwa kunci k terambil. Maka asumsi dari hashing seragam sederhana adalah :

$$\sum_{k:h(k)=j} P(k) = \frac{1}{m} \quad \text{untuk } j = 0, 1, \ldots, m-1.$$

Sehingga suatu kunci k acak mempunyai kesempatan hashing yang sama ke dalam sebarang lokasi dari m buah kotak.

Beberapa fungsi hash mendasarkan perhitungannya pada operasi aljabar aritmetika sederhana yaitu pembagian dan perkalian, disamping ada pula yang merupakan manipulasi biasa digit-digit dari kunci-kunci itu setelah dihitung harga hashnya.

3.3.1. Fungsi Hash Mid-Square

Definisi 4 :

Diberikan suatu kunci k.

Fungsi hash Mid-Square didefinisikan sebagai

$$h_m(k) = l,$$

dimana l adalah sejumlah bit-bit (digit-digit) yang tepat dari tengah hasil pengkuadratan kunci k. ■

Apabila pengambilan bit-bit (digit-digit) tengah menyebabkan banyaknya digit kiri dan kanan hasil pengkuadratan kunci *k* tidak sama, maka pada digit kiri ditambahkan sejumlah *leading / trailing zero*, sehingga menghasilkan alamat yang benar.

Contoh 6 :

Misalkan elemen-elemen kunci berupa huruf yang diinterpretasikan sebagai bilangan basis 8 (*octal*) menurut kode ASCII dimana karakter kunci adalah input right-justified zero filled sedangkan kunci-kunci berupa angka 0 sampai dengan 9 meneruskan nilai kunci huruf itu namun masih dalam bilangan basis 8. Tabel 1 menunjukkan konfigurasi-konfigurasi bit hasil pengkuadratan beberapa contoh pengenal. Apabila diambil dua digit tengah zero-filled (lihat Tabel 1), maka untuk kunci A1 mempunyai alamat 04. □

Jumlah bit-bit yang digunakan untuk menghasilkan alamat kotak bergantung pada ukuran tabel, agar dapat pas dengan satu *word* memori komputer. Jika digunakan r bit, range harga-harganya adalah 2^r. Sehingga ukuran tabel hashnya dipilih yang merupakan suatu kuadrat dua.

Tabel 1. Pervakilan internal dari k dan k^2 dalam notasi oktal.

Pengenal	Representasi Internal	
k	k	k^2
A	1	1
A1	134	20420
A2	135	20711
A3	136	21204
A4	137	21501
A9	144	23420
B	2	4
C	3	11
G	7	61
DMAX	4150130	21526443617100
DMAX1	415013034	5264473522151420
AMAX	1150130	135423617100
AMAX1	115013034	3454246522151420

3.3.2. Fungsi Hash Division

Fungsi hash division menggunakan operator modulo (mod). Harga hashnya diperoleh dari sisa pembagian kuncinya oleh suatu bilangan m.

Definisi 5 :

Diberikan suatu kunci k. Fungsi hash division didefinisikan sebagai sisa pembagian harga kunci k oleh bilangan m, yaitu

$$h_D(k) = k \bmod m.$$ ■

Fungsi hash ini memberikan alamat-alamat kotak dalam range 0 sampai $m-1$, sehingga tabel hashnya sekurang-kurangnya berukuran m.

Contoh 7 :

Misalkan ukuran tabel hashnya m = 32 dan kuncinya KEY. Hasil persiapan untuk kunci KEY seperti pada Contoh 5 adalah 756989. Maka alamat kotaknya h_D(KEY) = 756989 mod 32 = 29. □

Penentuan harga m adalah kritis. Beberapa harga m sebaiknya dihindari. Pangkat-pangkat 10 untuk harga m harus dihindari apabila penerapannya menghadapi bilangan-bilangan desimal sebagai kuncinya, karena fungsi hashnya tidak bergantung pada semua digit-digit desimal k.

Harga m yang merupakan suatu bilangan kuadrat dua, $m = 2^i$, juga harus dihindari. Sebab $h_D(k)$ akan hanya bergantung pada bit-bit signifikan paling sedikit dari k (hanya merupakan i bit orde terendah (i *lowest-order bits*) dari k), apabila tidak diketahui sebelumnya bahwa distribusi probabilitas pada kunci menjadikan semua pola-pola i bit orde rendah adalah equally likely.

Contoh 8 :

Misalkan tiap karakter diwakili oleh enam bit, dan pengenal-pengenal diletakkan rata kanan (*right-justified*) dalam suatu kata 60-bit dengan mendahulukan bit-bit yang

diisi dengan nol (*zero-filled*) (Gambar 3.3).

Maka dengan $m = 2^i$, $i \leq 6$, pengenal A1, B1, C1, X41, DNTXY1, dan lain-lainnya semuanya akan mempunyai alamat kotak yang sama, yaitu 1.

Gambar 3.3. Pengisian A1 rata kanan dan kiri (enam bit per karakter).

Misalkan sekarang pengenal k diletakkan rata kiri dan kemudian diisi dengan nol. Maka untuk $m = 2^i$, $i \leq 54$, semua pengenal satu-karakter akan dipetakan ke kotak yang sama,yaitu 0; untuk $m = 2^i$, $i \leq 48$, semua pengenal dua-karakter akan dipetakan ke kotak 0; dan seterusnya. □

Harga m seharusnya juga bukan merupakan suatu kelipatan 3. Sebab apabila kunci-kuncinya adalah alfabet, maka dua kunci yang berbeda satu sama lain, hanya dengan permutasi huruf-huruf akan berbeda dalam harga numerik dengan suatu kelipatan 3. (Hal ini terjadi karena $10^n \bmod 3 = 4^n \bmod 3 = 1$).

Contoh 9 :

Misalkan $x = x_1x_2$ dan $y = x_2x_1$ adalah dua pengenal, yang masing-masingnya mengandung karakter x_1 dan x_2.

Apabila perwakilan biner internal dari x_1 mempunyai harga $C(x_1)$ dan untuk x_2 mempunyai harga $C(x_2)$, maka jika tiap karakter diwakili dengan enam bit, harga numerik x adalah $2^6C(x_1)+C(x_2)$ dan untuk y adalah $2^6C(x_2)+C(x_1)$.

Misalkan p adalah bilangan prima yang membagi m. Maka :

$$\begin{aligned}(h_D(x)-h_D(y)) \bmod m &= (h_D(x)-h_D(y)) \bmod p \\ &= (2^6C(x_1) \bmod p + C(x_2) \bmod p \\ &\quad - (2^6C(x_2) \bmod p + C(x_1) \bmod p) \bmod p\end{aligned}$$

Ambil $p = 3$, maka

$$\begin{aligned}(h_D(x)-h_D(y)) \bmod p &= (64C(x_1) \bmod 3 + C(x_2) \bmod 3 \\ &\quad - (64C(x_2) \bmod 3 + C(x_1) \bmod 3) \bmod 3 \\ &= C(x_1) \bmod 3 + C(x_2) \bmod 3 \\ &\quad - (x_2) \bmod 3 + C(x_1) \bmod 3) \bmod 3 \\ &= 0 \bmod 3\end{aligned}$$

□

Tampak bahwa permutasi-permutasi dari himpunan karakter-karakter yang sama di-hash-kan pada suatu jarak kelipatan tiga. Program dimana banyak variabel adalah permutasi-permutasi satu dengan yang lain menghasilkan suatu penggunaan tabel bias yang besar dan karenanya banyak terjadi tabrakan. Ini terjadi karena (pada Contoh 9) misal r =64, 64 mod 3 = 1. Sifat yang sama dapat diharapkan apabila 7 membagi m, mengingat 64 mod 7 = 1.

Secara umum, harga m yang harus dihindari adalah yang membagi $r^k \pm a$, dimana k dan a adalah bilangan-bilangan

kecil, dan r adalah bilangan pokok dari himpunan karakter alfabet (biasanya r = 32, 64, 256 atau 100).

Berdasarkan uraian diatas, harga m yang baik adalah suatu bilangan prima sedemikian hingga $r^k \equiv \pm a$ (modulo m) untuk k dan a kecil dan tidak terlalu dekat dengan sebarang tepat kuadrat 2.

Contoh 10 :

Andaikan ingin dialokasikan suatu tabel hash dengan pemecahan tabrakan dengan penggandengan, untuk menangani kasarnya n = 2000 karakter string, dimana satu karakter mempunyai 8 bit. Tidak mengapa menguji rata-rata 3 elemen (tiap kotak tiga slot) dalam suatu pencarian tidak berhasil, sehingga dialokasikan suatu tabel hash berukuran m = 701.

Bilangan 701 dipilih karena merupakan bilangan prima dekat dengan α = 2000/3 tetapi tidak terlalu dekat dengan sebarang kuadrat 2. Sehingga didapat fungsi hash division

$$h_D(k) = k \bmod 701.$$

□

Sebagai ukuran pencegahan, dapat dicek seberapa meratanya fungsi hash ini mendistribusikan himpunan-himpunan kunci diantara slot-slotnya, dimana kunci-kuncinya dipilih dari data nyata (*real data*).

3.3.3. Fungsi Hash Multiplicative

Definisi 6 :

Diberikan bilangan real ε, $0 < \varepsilon < 1$.

Untuk suatu kunci bulat nonnegatif (*nonnegative integer key*) k, fungsi hash multiplication $h_M(k)$ didefinisikan sebagai :

$$h_M(k) = \lfloor m(k\varepsilon \bmod 1) \rfloor$$

dimana $m = 2^i$, i = banyaknya digit yang digunakan untuk 1-karakter pengenal. ■

Dengan demikian fungsi hash multiplicative melakukan dua langkah :

1. Mengalikan kunci k dengan konstanta ε, $0 < \varepsilon < 1$, dan menyadap bagian pecahan dari $k\varepsilon$.
2. Mengalikan harga ini dengan m dan mengambil *the floor* dari harga itu.

Keuntungan metode multiplicative adalah harga m tidak kritis, apabila dipilih sebagai kuadrat dari dua untuk suatu integer i; sehingga dapat dengan mudah mengimlementasikannya pada banyak komputer.

Perhitungan $h_M(k) = \lfloor m(k\varepsilon \bmod 1) \rfloor$ dapat disederhanakan menjadi perkalian integer-tunggal yang diikuti dengan pengutipan suatu blok bit dari perkalian itu.

Misalnya ukuran word komputernya adalah ω-bit (yaitu pada komputer biner) -masalah yang sama juga berlaku untuk komputer desimal-; ε ditulis $q/2^{\omega}$ dan memilih integer ω-bit, q, sedemikian hingga hasil ε tidak memperlihatkan *primary clustering*. Dengan memilih $m = 2^i$ menyebabkan

$$\lfloor m(k\varepsilon \bmod 1) \rfloor$$

merupakan suatu blok i bit dalam integer produk qk.

$$k\varepsilon = k \frac{q}{2^{\omega}}$$

$$= (\leftarrow t \text{ bit} \rightarrow)\ (\leftarrow \omega \text{ bit} \rightarrow)/2^{\omega}$$

$$= (\leftarrow t + \omega \text{ bit} \rightarrow)/2^{\omega}$$

$$= (\leftarrow t \text{ bit} \rightarrow . \leftarrow \omega \text{ bit} \rightarrow)$$

radix point

$k\varepsilon \bmod 1$

Jelaslah, $\lfloor 2^i (k\varepsilon \bmod 1) \rfloor$ paling banyak hanya i bit kiri dari segmen yang ditandai dengan $k\varepsilon \bmod 1$.

Sehingga didapat,

$kq = (\leftarrow t + v \text{ bit} \rightarrow).$

$\leftarrow v \text{ bit} \rightarrow$

$\leftarrow i \text{ bit} \rightarrow$

$h_M(k)$

Hashing metode multiplicative ini bekerja dengan memilih suatu konstanta ε .

Beberapa pedoman pemilihan harga ε :

(i). Harga ε tidak terlalu dekat dengan 0 atau 1. Karena

akan menyababkan elemen-elemen berharga kecil akan meng-cluster ke akhir dari tabel hashnya.

Contoh 11 :

Misalkan huruf-huruf kunci dikodekan dengan 5-bit biner menurut kode ASCII dan ukuran tabel hash m = 19. Apabila diambil harga ε = 0,99999, maka semua kunci yang terdiri dari satu dan dua huruf akan meng-hash ke satu persen terakhir tabel itu (lihat Tabel 2). □

Tabel 2. Contoh hasil penggunaan metode multiplicative dengan harga ε = 0,99999.

Kunci	Harga Kunci	$h_M(k)$
A	1	18
I	9	18
OF	48	18
TO	671	18
AND	1476	18
THE	20741	15
WITH	763528	6

(ii). Harga ($r^k\varepsilon$ mod 1) tidak terlalu dekat dengan 0 atau 1 dengan r adalah bilangan pokok himpunan karakternya, karena akan menyebabkan elemen-elemen berbentuk ar^k+b akan meng-cluster, untuk harga a kecil.

Contoh 12 :

Mengacu pada representasi huruf-huruf kunci dalam Contoh 11, dengan harga ε = 0,3066407. Kunci-kunci yang terdiri dari 3 huruf berakhiran AT akan mempunyai lokasi hash berbentuk

$$m \left[(32^2 a + 52)\varepsilon \bmod 1 \right]$$

, dengan $2 \le a \le 22$.

Hal ini disebabkan karena, misal untuk r = 32,

$(r^k \varepsilon \bmod 1) = (32^2 \times 0{,}3066407 \bmod 1)$

$\approx 0{,}000077$ (mendekati 0).

Sehingga kunci-kunci berupa kata BAT, CAT , EAT, dan seterusnya akan meng-hash sekitar $0.94m$. Tabel 3 memperlihatkan kelakuan ini. □

Tabel 3. Contoh hasil penggunaan fungsi hash multiplicative dengan ε = 0,3066407.

Kunci	Harga Kunci	$\lfloor m\ (r^k \varepsilon \bmod 1) \rfloor$
BAT	$32^2 \times 2 + 52 = 2100$	$0{,}94547m$
CAT	$32^2 \times 3 + 52 = 3124$	$0{,}9455468m$
EAT	$32^2 \times 5 + 52 = 5172$	$0{,}9457m$

(iii). Harga-harga ε yang mendekati

$$\frac{i}{(r^k - 1)}$$

, $1 \le i \le r^k - 1$, k = 1 atau 2; akan memperlihatkan suatu jenis clustering.

Untuk $k = 1$; harga ε mendekati $i/(r-1)$, $1 \leq i \leq r-1$ akan mengakibatkan harga $k\varepsilon$ mod 1 cenderung meng-cluster element-elemen berbeda hanya dengan suatu permutasi dari karakter-karakternya.

Hal ini terjadi karena

$$\frac{1}{(r-1)} = \sum_{i=1}^{\infty} \frac{1}{r^i}$$

$$= \frac{1}{r} + \frac{1}{r^2} + \frac{1}{r^3} + \frac{1}{r^4} + \ldots.$$

ditulis $(0,111\ldots)_r$ dan produk k dan ε yang demikian akan mempunyai digit-digit basis r, yaitu pada dasarnya jumlah digit-digit basis r dari k - jumlah karakter-karakter individu dari k.

Contoh 13 :

Dari representasi karakter seperti Contoh 11, diambil $i = 6$, $r = 32$ sehingga harga $\varepsilon = \frac{6}{31} \approx 0,193548387$. Untuk karakter-karakter kunci EAT, TEA, ETA, dan ATE mempunyai harga $k\varepsilon$ mod 1 sekitar 0,0322581 (lihat Tabel 4), sehingga tentu saja penggunaan harga ε ini dihindari. □

Untuk $k = 2$; harga ε yang mendekati $i/(r^2-1)$, $1 \leq i \leq r^2-1$, akan cenderung meng-cluster elemen-elemen yang berbeda hanya dengan permutasi-permutasi independen dari posisi-posis kesatu, kedua, ketiga keempat, ...; dan posisi-posisi kedua, keempat, keenam, ...

Tabel 4. Contoh hasil penggunaan fungsi hash multiplicative dengan $\varepsilon = 6/31$.

Kunci	Harga Kunci	$k\varepsilon$ mod 1
EAT	5172	0,032258
TEA	20641	0,032258
ETA	5761	0,032258
ATE	1669	0,0322581

Contoh 14 :

Lagi, dari representasi karakter seperti Contoh 11; dengan $i = 473$ dan $r = 32$, sehingga $\varepsilon = \frac{479}{1023} \approx 0{,}468230694$. Karakter-karakter kunci META, TAME, dan MEAT mempunyai harga $k\varepsilon$ mod 1 sekitar 0,261; sedangkan karakter kunci MEAT dan TEAM mempunyai harga $k\varepsilon$ mod 1 berturut-turut sekitar 0,4731 dan 0,792. □

Theorema 1 : *The Three-Distance Theorem*

Misalkan ϕ sebarang bilangan irrasional.

Jika titik-titik $\{\phi\}$, $\{2\phi\}$, $\{3\phi\}$, ..., $\{n\phi\}$ diletakkan dalam segmen garis [0,1], maka (n+1) segmen-segmen garis yang dibentuk mempunyai paling banyak tiga panjang berbeda.

Lagi pula, titik $\{(n+1)\phi\}$ berikutnya akan terletak pada salah satu dari segmen-segmen terbesar yang ada.

Bukti :

Misalkan ϕ adalah bilangan irrasional antara 0 dan 1, $0 < \phi < 1$ yang direpresentasikan sebagai pecahan bersambung (*continued fraction*), yaitu

$$/a_1,a_2,a_3,.../ = 1/(a_1+1/(a_2+1/(a_3+1/(...+1/(a_n)...)))).$$

$$= \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \cfrac{1}{\ddots + \cfrac{1}{a_n}}}}}$$

Misalkan $q_0 = 0$, $p_0 = 1$, $q_1 = 1$, $p_1 = 0$ dan

$$q_{k+1} = a_k q_k + q_{k-1},$$

$$p_{k+1} = a_k p_k + p_{k-1},$$

untuk $k \geq 1$.

Misalkan $\{x\}$ menotasikan $x \bmod 1 = x - \lfloor x \rfloor$ dan misal $\{x\}^+$ menotasikan $x - \lceil x \rceil + 1$.

Misalkan titik-titik $\{\phi\}$, $\{2\phi\}$, $\{3\phi\}$, ... yang disisipkan berturut-turut ke dalam interval [0,1] itu dinomori yaitu segmen pertama dari suatu panjang yang diberikan adalah nomor 0, berikutnya adalah 1 dan seterusnya.

Untuk interval nomor 0 dari suatu panjang baru dibentuk.

Kenyataan bahwa :

$$\{(-1)^k(rq_k + q_{k-1})\phi\} = (-1)^k(r(q_k\phi - p_k) + (q_{k-1}\phi - p_{k-1}))$$

, untuk $0 \leq r \leq a_k$.

Harga-harga "rekord rendah" (*record low*) dari $\{n\phi\}$ terjadi untuk

$n = q_1, q_2+q_1, 2q_2+q_1, \ldots, a_2q_2+q_1 = 0q_4+q_3, a_4q_4+q_3 =$

$0q_6+q_5$, ...;

harga-harga "rekord tinggi" (*record high*) terjadi untuk

$n = q_0, q_1+q_0, \ldots, a_1q_1+q_0 = 0q_3+q_2, \ldots$.

Maka untuk interval nomor s dari panjang $\{t\phi\}$, dimana $t = rq_k + q_{k-1}$ dan $0 \le r < a_k$,dan k genap, $0 \le s < q_k$ mempunyai titik akhir kiri $\{s\phi\}$ dan titik akhir kanan $\{(s+t)\phi\}^+$.

Interval nomor s dari panjang $1 - \{t\phi\}$, dimana $t = rq_k + q_{k-1}$ dan $0 \le r < a_k$,dan k genap, $0 \le s < q_k$ mempunyai titik akhir kiri $\{(s+t)\phi\}$ dan titik akhir kanan $\{s\phi\}^+$.

Pandang setiap bilangan integer positif n direpresentasikan dengan unik sebagai

$n = rq_k + q_{k-1} + s$, untuk suatu $k \ge 1$, $1 \le r \le a_k$, $0 \le s \le q_k$.

Sehingga tepat sebelum titik $\{n\phi\}$ disisipkan, n buah interval itu sekarang adalah

- interval-interval s pertama (dinomori $0, \ldots, s-1$) dari panjang $\{(-1)^k(rq_k + q_{k-1})\phi\}$;
- interval-interval $n - q_k$ pertama (dinomori $0, \ldots, n - q_k - 1$) dari panjang $\{(-1)^k q_k\phi\}$;
- interval-interval $q_k - s$ terakhir (dinomori $s, \ldots, q_k - 1$) dari panjang $\{(-1)^k((r-1)q_k + q_{k-1})\phi\}$. Operasi penyisipan $\{n\phi\}$ itu memindahkan interval nomor s dari tipe sebelumnya dan mengkonversinya kedalam interval nomor s dari tipe pertama, nomor $n - q_k$ dari tipe kedua.

■

Teorema 1 menyatakan bahwa segmen-segmen dari suatu panjang yang diberikan dibentuk dan dihapus dalam cara First-In-First-Out (FIFO).

Akibat 1 :

Bilangan $\Phi^{-1} = (\sqrt{5} - 1)/2$ dan $\Phi^{-2} = 1 - \Phi^{-1}$ menuju ke barisan yang paling terdistribusi secara seragam diantara semua bilangan-bilangan irrasional antara 0 dan 1.

Bukti :

Teorema 1 menyatakan bahwa setiap titik baru selalu memisahkan satu interval interval-interval sisa terbesar. Jika interval-interval [a,c] dengan cara seperti itu dipecah menjadi 2 bagian [a,b] dan [b,c], maka akan dikatakan "pecahan buruk" (*bad breaks*) apabila salah satu bagiannya dua kali lebih panjang dari yang lainnya, yaitu jika (b-a) > 2(c-b) atau (c-b) > 2 (b-a).

Akan dibuktikan bahwa pecahan buruk akan terjadi unuk suatu {nϕ} jika ϕ mod 1 tidak sama dengan Φ^{-1} atau Φ^{-2}.

Dipunyai Φ^{-1} = /1,1,1,.../ dan Φ^{-2} = /2,1,1,.../.

Misalkan $\phi = /a_1,a_2,a_3,.../$, $\phi_k = /a_{k+1},a_{k+2},a_{k+3},.../$ dan $Q_k = q_k + q_{k-1}\phi_{k-2}$, seperti dalam Teorema 1.

Jika $a_1 > 2$, pecahan pertama itu juga buruk.

Tiga ukuran interval-interval dalam Teorema 1. berturut-turut adalah

$(1-r\phi_{k-1})/Q_k$, ϕ_{k-1}/Q_k, dan $(1-(r-1)\phi_{k-1})/Q_k$,

sehingga perbandingan panjang pertama dan kedua adalah

$(a_k - r) + \phi_k$.

Ini akan kurang dari $\frac{1}{2}$ jika $r = a_k$ dan $a_{k+1} \geq 2$; karenanya $\{a_2, a_3, \ldots\}$ semuanya haruslah sama dengan 1. ■

Dari Teorema 1 dan Akibat 1 didapat bahwa harga Φ^{-1} dan Φ^{-2} untuk ε fungsi hash metode multiplicative diharapkan akan menyebarkan kunci-kunci merata keseluruh tabel hashnya. Dengan demikian harga ε yang baik untuk metode multiplicative ini adalah $\Phi^{-1} \approx 0{,}6180339887$ atau $\Phi^{-2} \approx 0{,}3819660113$.

Walaupun harga Φ^{-1} ini cukup baik namun tetap harus berhati-hati untuk $r = 1000$ dengan $k > 2$, sebab

$r^k \varepsilon \bmod 1 = r^k \Phi^{-1} \bmod 1 = 10^6 \times 0{,}3819660113 \bmod 1$

$\approx 0{,}0113$ (mendekati 0).

3.3.4. Fungsi Hash Polynomial / Algebraic Coding

Definisi 7 :

Suatu kunci biner n-digit $K = (k_0 k_1 \ldots k_{n-1})_2$ dipandang sebagai polinomial

$$K(x) = \sum_{i=0}^{n-1} k_i x^i .$$

Jika suatu alamat dengan range 0 sampai $M = 2^m - 1$ dibutuhkan, maka $K(x) \bmod P(x)$ memberikan alamat $h_P(K) = (h_0 h_1 \ldots h_{m-1})_2$, dengan

$$P(x) = x^m + \sum_{i=0}^{m-1} p_i x^i . \quad ■$$

Contoh 15 :

Untuk n = 15, m = 10, dan $P(x) = x^{10} + x^8 + x^5 + x^4 + x^2 + x + 1$, maka kunci-kunci yang berbeda kurang dari 7-bit posisi akan mempunyai harga hash yang berbeda.

Polinomial P(x) itu mengkonversi n-bit kunci ke m-bit alamat sedemikian higga kunci-kunci berbeda yang berbeda dalam t-bit atau kurang akan meng-hash ke alamat-alamat yang berbeda.

Diberikan n dan $t \leq n$, dan diberikan sebarang integer k sedemikian hingga n membagi $2^k - 1$. Dibentuk polinomial derajat m yaitu suatu fungsi dari n, t, dan k (jika perlu, biasanya n ditambah, sehingga k dapat dipilih agak kecil).

Misalkan S adalah himpunan bilangan bulat terkecil, sedemikian hingga $\{1,2,\ldots,t\} \subseteq S$, dan $(2j) \bmod n \in S$, untuk semua $j \in S$. Untuk contoh jika n = 15, k = 4 dan t = 6, $S = \{1,2,3,4,5,6,8,9,10,12\}$.

Sekarang didefinisikan polinomial

$$P(x) = \prod_{j \in S} (x - \alpha^j),$$

dimana α adalah suatu alamat orde n dalam field finite $\mathcal{GF}(2^k)$ dan koefisien-koefisien P(x) dihitung dalam field ini.

Derajat m dari P(x) adalah jumlah elemen-elemen S.

Karena α^{2j} adalah suatu akar P(x) bilamana α^j adalah suatu akar, maka koefisien-koefisien p_i dari P(x) memenuhi $p_i^2 = p_i$. Dengan demikian semua koefisien P(x) adalah 0 atau 1.

Misalkan $R(x) = r_{n-1}x^{n-1} + \ldots + r_1x + r_o$ adalah sebarang polinomial tak kosong modulo 2 dengan paling banyak t koefisien tak kosong.

Andaikan R(x) adalah suatu kelipatan dari P(x), maka $R(\alpha^j) = 0$ dalam $\mathcal{GF}(2^k)$ untuk semua $j \in S$. Misalkan sekarang

$$R(x) = x^{a_1} + \ldots + x^{a_s},$$

dimana $a_1 > \ldots > a_s \geq 0$ dan $s \leq t$, dan dipilih harga-harga t-s selanjutnya , $a_s + 1, \ldots, a_t$, sedemikian hingga $a_1, \ldots, a_t$ adalah bilangan bulat nonnegatif berbeda yang kurang dari n.

Matriks

$$\begin{bmatrix} \alpha^{a_1} & \ldots & \alpha^{a_s} \\ \alpha^{2a_1} & \ldots & \alpha^{2a_t} \\ \vdots & & \vdots \\ \alpha^{ta_1} & \ldots & \alpha^{ta_t} \end{bmatrix}$$

adalah *singular*, karena jumlah s kolom-kolom pertamanya adalah nol. Kotradiksi dengan kenyataan bahwa $\alpha^{a_1}, \ldots, \alpha^{a_t}$ adalah elemen-elemen berbeda dari $\mathcal{GF}(2^k)$. Jadi R(x) bukan kelipatan dari P(x) modulo 2. □

3.3.5. Fungsi Hash Pelipatan

Fungsi hash metode pelipatan (*folding method*) didefinisikan sebagai berikut :

Definisi 8 :

Suatu kunci integer *k* dipartisi menjadi beberapa bagian dengan panjang yang sama menurut kebutuhan alamat hashnya dengan pengecualian bagi bagian terakhir. Kemudian bagian-bagian itu semuanya dijumlahkan untuk mendapatkan alamat hash. Metode ini dinamakan *shift-folding*.

Jika bagian-bagian itu dilipat pada batas-batas partisi dan digit-digit yang terletak dalam posisi yang sama dijumlahkan, metodenya dinamakan *folding at the boundaries* atau *fold-boundary*. ■

Contoh 16 :

Misalkan kunci k = 123203241111220 dipartisi menjadi 3 digit desimal. Maka didapat partisi $p_1 = 123$, $p_2 = 203$, $p_3 = 241$, $p_4 = 112$, $p_5 = 20$.

Dengan metode shift-folding didapat alamat

$$h_F(k) = \sum_{i=1}^{5} p_i = 123 + 203 + 241 + 112 + 20 = 699.$$

Dengan metode fold-boundary, partisi-partisi itu dilipat pada batas partisinya didapat $p_2 = 302$ dan $p_4 = 211$, sedangkan p_1, p_3, dan p_5 tidak berubah harganya. Alamat hash yang diperoleh adalah

$$h_F(k) = \sum_{i=1}^{5} p_i = 123 + 302 + 241 + 211 + 20 = 897.$$ □

3.3.6. Fungsi Hash Analisis Digit

Definisi 9 :

Suatu kunci *r*-digit dipilih (*selected*) kemudian ditukar (*shifted*) posisi-posisi digit yang dipilih itu, untuk menghasilkan alamat hash. ■

Contoh 17 :

Kunci k = |7|5|4|6|1|2|3| ditransformasikan ke alamat 2164 dengan memilih digit-digit di posisi 3 sampai dengan 6 dan membalik urutannya. □

Untuk suatu kumpulan kunci yang diberikan, posisi-posisi yang sama dalam kunci itu beserta pola penyusunan kembali (*rearrangement*) yang sama harus digunakan secara konsisten.

Pertama kali, satu analisis pada sampel dari kumpulan kunci dilakukan untuk menentukan posisi-posisi kunci mana yang seharusnya digunakan dalam pembentukan suatu alamat. Kemudian dipilih posisi-posisi digit yang paling berdistribusi seragam (*most uniformly distribution*).

Contoh 18 :

Perhatikan analisis digit dari sampel kunci yang ditunjukkan dalam Tabel 5. Sejumlah 5000 kunci 10-digit dianalisis untuk menentukan posisi-posisi kunci mana yang seharusnya digunakan dalam pembentukan alamat-alamat dalam ruang alamat A = { 0, 1, ..., 9999}. Dari tabel itu tampak

bahwa posisi-posisi 2, 4, 5, dan 9 paling berdistribusi seragam, sehingga posisi-posisi tersebut dipilih. Jadi, untuk contoh, kunci 1234567890 ditransformasikan ke alamat 9542 dengan memilih digit-digit di posisi 2, 4, 5, 9 dan membalik urutannya. □

Metode analisis digit ini berguna terutama sekali untuk himpunan kunci-kunci statis yaitu himpunan kunci yang tidak berubah setiap waktu, misalnya dalam kasus file statis (*static file*) dimana semua pengenal dalam tabelnya diketahui terlebih dulu.

Tabel 5. Analisis digit suatu himpunan sampel dari 10-digit bagian bilangan (Horovitz and Sahni, hal 610).

Digit	Posisi Kunci									
	1	2	3	4	5	6	7	8	9	10
0	5000	531	594	499	590	721	1565	1133	562	2540
1	0	582	568	536	467	905	874	759	612	1581
2	0	571	620	531	563	553	657	606	542	557
3	0	546	565	511	512	277	555	482	522	332
4	0	518	529	495	461	0	284	521	546	0
5	0	503	503	500	463	673	276	469	472	0
6	0	488	456	469	510	629	263	296	426	0
7	0	449	411	500	459	0	212	365	425	0
8	0	422	431	470	457	501	159	310	455	0
9	0	390	323	489	518	741	155	59	438	0

Contoh 19 :

Misalkan kunci-kunci pengenal A, A1, A2, A3, A9 B, C3, G, DMAX dan DMAX1 akan ditempatkan dalam tabel hash berukuran m = 11. Tahap persiapan yang dilakukan adalah dengan menginterpretasikan tiap karakter kunci-kunci pengenal itu sebagai bilangan desimal menurut kode ASCII.

Apabila digunakan metode Mid-Square, harga hash yang tepat didapat dengan mengambil satu bilangan desimal tengah hasil pengkuadratan tahap persiapannya. Sehingga diperoleh alamat tabel hashnya (Tabel 6). Untuk fungsi hash Pelipatan, harga numerik hasil persiapan dipartisi menjadi satu bagian kemudian dijumlahkan, didapat alamat tabelnya (Tabel 6).

Tabel 6. Penggunaan fungsi hash Mid-Square dan Pelipatan dengan tahap persiapan pengkodean desimal karakter ASCII.

Kunci k	Harga Kunci	k^2	$h_m(k)$	$h_F(k)$
A	65	4225	2	4
A1	113	12769	7	5
A2	114	12996	9	6
A3	115	13225	2	7
A9	122	14884	8	5
B	66	4356	3	3
C3	118	13924	9	10
G	71	5041	0	8
DMAX	298	88804	8	10
DMAX1	347	120409	0	5

Tabel 7. Penggunaan fungsi hash Division dan Multiplicative dengan tahap persiapan pengkodean desimal karakter ASCII.

Kunci	Harga Kunci	$h_D(k)$	$h_M(k)$
A	65	10	1
A1	113	3	9
A2	114	4	5
A3	115	5	8
A9	122	1	4
B	66	0	8
C3	118	8	10
G	71	5	9
DMAX	298	1	1
DMAX1	347	6	5

Misalkan sekarang digunakan fungsi hash Division dan fungsi hash Multiplicative dengan harga ε = 0,6180339887 untuk menghitung harga hashnya. Maka didapat alamat tabel hash seperti dalam Tabel 7.

Fungsi hash Polynomial tidak dapat digunakan untuk menghitung alamat tabel hash untuk kunci-kunci pengenal yang direpresentasikan sebagai bilangan desimal menurut kode ASCII. Untuk fungsi hash metode Analisis Digit, mengingat terlebih dahulu dilakukan analisis posisi-posisi digit kunci sehingga kurang tepat apabila hanya digunakan representasi karakter kunci seperti ini.

□

3.3.7. Universal Hashing

Universal hashing adalah suatu metode yang berhubungan dengan masalah utama hashing yaitu *liner worst-case*, dimana n kunci semuanya di-hash-kan ke slot yang sama, sehingga mempunyai waktu rata-rata retrieval $\Theta(n)$. Disamping mengingat bahwa walaupun semua himpunan kunci-kunci $K \subseteq \{0, \ldots, n\}$, $|K| = n$, dengan suatu fungsi hash h tertentu menghasilkan alamat-alamat dalam himpunan ruang alamat $A = \{h(k)\}$ dengan cukup baik; tetapi selalu ada beberapa input yang sangat buruk untuk h. Sehingga selalu riskan menggunakan hashing apabila distribusi aktual dari input-input itu tidak diketahui oleh perancang fungsi hashnya.

Definisi 10 :

Misalkan $\mathcal{H}$ adalah koleksi berhingga fungsi-fungsi hash yang memetakan himpunan kunci universal U ke dalam range $\{0, 1, \ldots, m-1\}$, dinotasikan

$$\mathcal{H} \subseteq \{h \mid h : [0,\ldots,n-1] \longrightarrow [0,\ldots,m-1]\}.$$

$\mathcal{H}$ disebut universal jika untuk setiap pasangan kunci berbeda; $x, y \in U$, jumlah fungsi-fungsi hash $h \in \mathcal{H}$ untuk mana $h(x) = h(y)$ adalah tepat $|\mathcal{H}|/m$.

Dengan perkataan lain, dengan suatu fungsi hash yang dipilih secara random dari $\mathcal{H}$, kesempatan satu tabrakan antara x dan y, $x \neq y$ adalah tepat $1/m$, yakni tepat suatu kesempatan tabrakan terjadi jika $h(x)$ dan $h(y)$ dipilih secara random dari $\{0, 1, \ldots, m-1\}$. ■

Definisi diatas memberikan gagasan utama dari universal hashing yaitu fungsi hash tertentu yang akan digunakan dipilih secara random dari koleksi $\mathcal{H}$. Jika $\mathcal{H}$ dipilih secara tepat, yaitu untuk setiap subset $K \subseteq U$ hampir semua $h \in \mathcal{H}$ mendistribusikan K hampir merata menyebar ke seluruh tabel hashnya, maka ini diharapkan akan mengarahkan ke waktu akses tabel hash yang kecil untuk setiap himpunan K.

Teorema 2 :

Jika h dipilih dari suatu koleksi fungsi-fungsi hash $\mathcal{H}$ dan digunakan untuk meng-hash-kan n kunci kedalam suatu tabel berukuran m, dimana $n \leq m$, jumlah tabrakan yang diharapkan dengan melibatkan suatu kunci tertentu k adalah kurang dari 1.

Bukti :

Untuk setiap pasangan kunci berbeda y, z, misal c_{yz} adalah variabel random yang didefinisikan dengan

$$c_{yz} = \begin{cases} 1, \text{ jika } h(y) = h(z) & (y \text{ dan } z \text{ bertabrakan dengan menggunakan } h) \\ 0, \text{ untuk yang lain.} \end{cases}$$

Karena menurut definisi bahwa suatu pasangan tunggal kunci-kunci bertabrakan mempunyai probabilitas $1/m$, maka

$$\mathbb{E}[c_{yz}] = 1/m.$$

Misal C_x adalah jumlah total tabrakan yang melibatkan elemen kunci x dalam suatu tabel hash T berukuran m

mengandung n kunci.

Maka

$$\mathbb{E}[C_x] = \sum_{\substack{y \in T \\ y=x}} \mathbb{E}[c_{xy}]$$

$$= \frac{n-1}{m}$$

Karena $n \leq m$, maka didapat $\mathbb{E}[C_x] < 1$. ■

Mendisain suatu kelas universal dari fungsi-fungsi hash

Misalkan tabel hashnya dipilih berukuran m, dengan m bilangan prima. Suatu kunci x didekomposisi ke dalam $r + 1$ byte (yaitu karakter-karakter atau karakter-karakter substring biner *fixed-width*), sehingga

$x = (x_0, x_1, \ldots, x_r)$.

Misalkan $a = (a_0, a_1, \ldots, a_r)$ menotasikan suatu $r + 1$ elemen yang dipilih secara random dari himpunan $\{0, 1, \ldots, m-1\}$.

Maka fungsi hash $h_a \in \mathcal{H}$ yang berkorespondensi adalah

$$h_a(x) = \sum_{i=0}^{r} a_i x_i \bmod m \qquad (1)$$

Dengan ini, maka

$$\mathcal{H} = \bigcup \{h_a\} \text{ mempunyai } m^{r+1} \text{ anggota} \qquad (2)$$

Teorema 3 :

Kelas $\mathcal{H}$ yang didefinisikan oleh (1) dan (2) adalah suatu kelas universal fungsi-fungsi hash.

Bukti :

Perhatikan sebarang pasangan kunci-kunci berbeda x, y. Diasumsikan bahwa untuk posisi byte O, $x_o \neq y_o$. (alasan yang sama dapat diberlakukan untuk perbedaan dalam sebarang posisi byte lain).

Untuk sebarang harga-harga tertentu dari a_1, a_2, ...,a_r; ada tepat satu harga a_o yang memenuhi persamaaan $h(x) = h(y)$, a_o ini adalah solusi untuk

$$a_o(x_o - y_o) \equiv -\sum_{i=1}^{r} a_i(x_i - y_i) \pmod{m}.$$

Karena m adalah bilangan prima kuantitas tak kosong $x_o - y_o$ mempunyai invers perkalian modulo m, dan karenanya thus ada solusi unik untuk a_o modulo m. Oleh karena itu, tiap pasangan kunci x dan y bertabrakan untuk tepat m^r harga-harga dari a, karena kunci-kunci itu bertabrakan tepat sekali untuk tiap harga yang mungkin dari (a_1, a_2, ...,a_r) (yaitu untuk harga-harga unik dari a_o yang dicatat di atas).

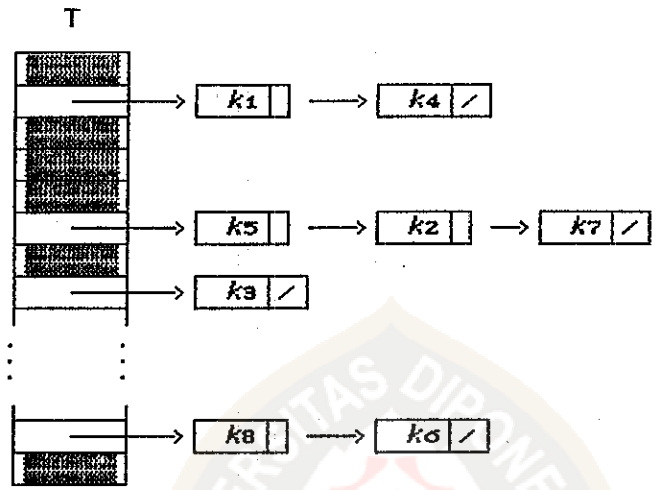
Karena ada m^{r+1} harga-harga yang mungkin untuk barisan a, kunci-kunci x dan y bertabrakan dengan probabilitas tepat $m^r/m^{r+1} = 1/m$.

Karena itu, $\mathcal{H}$ universal. ■

3.4. Menangani Tabrakan dengan Penggandengan

Dalam penggandengan, semua elemen yang meng-hash ke slot yang sama diletakkan dalam suatu linked-list. Slot j mengandung suatu pointer ke kepala list dari semua elemen

$k \in K$. Gambar 3.4 mengilustrasikan cara menangani tabrakan dengan pengandengan.



Gambar 3.4. Menangani tabrakan dengan penggandengan.

Running time worst-case untuk operasi penyisipan adalah $O(1)$. Untuk pencarian, running time worst-casenya sebanding dengan panjang listnya. Yang pertama kali dilakukan untuk menghapus suatu elemem k jika list-listnya single-linked adalah harus harusditemukan k dalam list $T[h(\text{key}[k])]$. Sehingga link berikut dari pendahulu k dapat ditetapkan sebagaimana mestinya untuk menyambung k. Dalam hal ini, penghapusan dan pencarian pada dasarnya mempunyai running-time yang sama.

Kelakuan worst-case hashing dengan penggandengan adalah buruk sekali : semua n kunci meng-hash ke slot yang sama, membentuk list dengan panjang n. Waktu worst-case untuk pencarian menjadi $\Theta(n)$ ditambah waktu untuk

menghitung fungsi hashnya. Ini berarti tidak lebih baik apabila digunakan satu linked-list untuk semua elemen-elemen itu.

3.4.1. Penggandengan di Bawah Asumsi Hashing Seragam Sederhana

Misalkan harga hash $h(k)$ dapat dihitung dalam waktu $O(1)$, sehingga waktu yang dibutuhkan untuk mencari suatu elemen k bergantung secara linier terhadap panjang list $T[h(k)]$. Dengan menyimpan waktu $O(1)$ yang dibutuhkan untuk menghitung fungsi hash itu dan mengakses slot $h(k)$, misal jumlah yang diharapkan dari elemen-elemen yang diuji oleh algoritma pencarian itu adalah jumlah elemen-elemen dalam list $T[h(k)]$ yang dicek untuk melihat apabila sama dengan k.

Teorema 4 :

Rata-rata pencarian berhasil dan tidak berhasil dalam tabel hash dengan penggandengan di bawah asumsi hashing seragam sederhana memerlukan waktu $\Theta(1+\alpha)$.

Bukti :

Dibawah asumsi hashing seragam sederhana (lihat halaman 31), waktu rata-rata pencarian tidak berhasil, untuk suatu kunci k menjadi waktu rata-rata untuk mencari akhir salah satu dari m slot.

Panjang rata-rata list yang demikian adalah faktor beban α

$= n/m$.

Sehingga jumlah yang diharapkan dari elemen-elemen yang diuji dalam suatu pencarian tidak berhasil adalah α dan total waktu yang dibutuhkan (termasuk waktu untuk menghitung $h(k)$ adalah $\Theta(1+\alpha)$.

Misalkan bahwa prosedur penyisipan suatu elemen barunya pada akhir list (teorema ini berlaku juga apabila prosedur penyisipan elemen barunya pada awal list).

Jumlah yang diharapkan dari elemen-elemen yang diuji selama suatu pencarian berhasil adalah lebih 1 dari jumlah elemen-elemen yang diuji ketika pencarian untuk elemen yang disisipkan.

Jumlah yang diharapkan dari elemen-elemen yang diuji diambil rata-ratanya meliputi n buah item dalam tabel itu, dari 1 ditambah panjang rata-rata list untuk mana elemen ke-i ditambahkan. Panjang yang diharapkan dari list itu adalah $(i-1)/m$, sehingga jumlah rata-rata elemen-elemen yang diuji dalam pencarian yang berhasil adalah

$$\frac{1}{n} \sum_{i=1}^{n} \left(1 + \frac{i-1}{m}\right) = 1 + \frac{1}{nm} \sum_{i=1} (i-1)$$

$$= 1 + \left[\frac{1}{nm}\right] \left[\frac{(n-1)n}{2}\right]$$

$$= 1 + \frac{\alpha}{2} - \frac{1}{2m}$$

Jadi, total waktu yang diperlukan untuk pencarian berhasil (termasuk waktu yang diperlukan untuk menghitung fungsi hashnya) adalah

$$\Theta\left(1 + \frac{\alpha}{2} - \frac{1}{2m}\right) = \Theta(1 + \alpha).$$ ■

Di bawah asumsi hashing seragam sederhana kunci-kunci masuk ke dalam lokasi random dari tabel, sehingga masing-masing dari $\binom{m}{n}$ konfigurasi dari n slot yang diduduki dan $m - n$ slot kosong adalah equally likely. Probabilitas tepat r buah probe yang dibutuhkan untuk mengnyisipkan item (n+1) adalah jumlah konfigurasi-konfigurasi dimana $r-1$ slot-slot yang diberikan diduduki dan lainnya kosong. yaitu

$$P_r = \binom{m-r}{n-r+1} \Big/ \binom{m}{n}.$$

Misalkan masing-masing m^n "barisan-barisan hash" (*hash sequence*) yang mungkin

$$a_1, a_2, \ldots, a_n, \quad 0 \le a_j < m \qquad (3)$$

adalah equally likely, dimana a_j menotasikan alamat hash awal dari kunci ke-j yang disisipkan ke dalam tabel. Dan misal

C_n = jumlah rata-rata probe dalam suatu pencarian sukses, untuk sebarang algoritma pencarian tertentu. Nilai ini diasumsikan merupakan jumlah rata-rata probe yang diperlukan untuk menemukan kunci ke-k, dirata-ratakan melalui $1 \le k \le n$, dan dirata-ratakan keseluruh barisan-barisan hash (3).

Secara sama,

C'_n = Jumlah rata-rata probe yang diperlukan apabila kunci ke-n+1 disisipkan dengan mempertimbangkan semua barisan (3). Nilai ini adalah jumlah

rata-rata probe dalam suatu pencarian tidak berhasil mulai n elemen-elemen dalam tabel.

Teorema 5 :

Jumlah rata-rata probe pencarian tidak berhasil untuk hashing seragam sederhana adalah

$$U(\alpha) \approx \frac{1}{1-\alpha} ,$$

sedangkan untuk pencarian yang berhasil rata-rata probenya adalah

$$S(\alpha) \approx \frac{1}{\alpha} \ln \frac{1}{1-\alpha} ,$$

dengan α adalah faktor beban.

Bukti :

Dari asumsi hashing seragam sederhana

$$C_n' = \sum_{1 \le r \le m} r P_r$$

$$= m + 1 - \sum_{1 \le r \le m} (m + 1 - r) P_r$$

$$= m + 1 - \sum_{1 \le r \le m} (m + 1 - r) \binom{m-r}{m-n-1} \Big/ \binom{m}{n}$$

$$= m + 1 - \sum_{1 \le r \le m} (m - n) \binom{m+1-r}{m-r} \Big/ \binom{m}{n}$$

$$= m + 1 - (m - n) \binom{m+1}{m-n+1} \Big/ \binom{m}{n}$$

$$= m + 1 - (m - n) \frac{m+1}{m-n+1}$$

$$= \frac{m+1}{m-n+1}$$

dengan $\alpha = \frac{n}{m}$, didapat $U(\alpha) \approx \frac{1}{1-\alpha}$.

Untuk pencarian berhasil, ada dua cara pembuktian :

I. $C_n = \frac{1}{n} \sum_{0 \le k \le n} C_k'$

$= \frac{m+1}{n} \sum_{k=m+2-n} \frac{1}{k}$

$= \frac{m+1}{n} \left(\frac{1}{m+1} + \frac{1}{m} + \ldots + \frac{1}{m-n+2} \right)$

Misalkan $H_j = \sum_{j=1}^{i} \frac{1}{j}$ adalah bilangan harmonik ke-i, dengan batas-batas $ln\ i \le H_i \le ln\ i + 1$.

Maka

$C_n = \frac{m+1}{n} (H_{m+1} - H_{m-n+1})$

$= \frac{m+1}{n} (ln\ (m+1) - ln\ (m-n+1))$

dengan $\alpha = n/m$ didapat rumus eksak untuk C_n adalah

$$S(\alpha) \approx \frac{1}{\alpha}\ ln\ \frac{1}{1-\alpha}.$$

II. Jumlah probe yang diperlukan untuk menemukan k setelah berada dalam tabel adalah sama dengan dengan jumlah probe yang digunakan pada pencarian tidak berhasil apabila x disisipkan.

Apabila tabel hashnya dipertimbangkan seperti sedang dikkonstruksinya elemen-elemen disisipkan satu demi satu, maka faktor beban bertambah dalam pertambahan diskrit yang kecil dari 0 menuju harga akhir α. Dengan perkataan lain, keadaaan ini dapat diperkirakan dengan memisalkan faktor beban bertambah secara kontinu dari 0 sampai harga akhir α.

Sehingga $S(\alpha)$ adalah harga rata-rata $U(\alpha)$ untuk $0 \leq x \leq \alpha$, jadi

$$S(\alpha) = \frac{1}{\alpha} \int_0^{\alpha} U(x)\, dx$$

$$= \frac{1}{\alpha} \int_0^{\alpha} \frac{dx}{1 - x}$$

$$= \frac{1}{\alpha} \ln \frac{1}{1 - \alpha}$$ ■

Sebagai interpretasi kasar untuk pencarian tidak berhasil dari teorema hashing seragam adalah bahwa dengan probabilitas α dinginkan satu probe, dengan probablitas α^2 diinginkan dua probe dan seterusnya.

3.4.2. Penggandengan Terpisah

Dalam penggandengan terpisah (*separate chaining*), tiap lokasi tabel $T[i]$ adalah kepala list (Gambar 3.4), menunjuk ke linked-list dari elemen-elemen k, dengan $h(k) = i$. Dengan demikian membutuhkan m pointer dari tabelnya. Jika listnya tidak terurut (*unordered*), elemen k disisipkan tepat setelah kepala list $T[h(k)]$ sebelum elemen pertama pada list itu.

Teorema 6 :

Jumlah rata-rata probe dalam suatu pencarian berhasil apabila digunakan penggandengan terpisah untuk menangani tabrakan dengan list-list terurut adalah

$$S(\alpha) \approx 1 + \frac{1}{2}\alpha ,$$

sedangkan untuk pencarian tidak berhasil adalah

$$U(\alpha) \approx e^{-\alpha} + \alpha \ ,$$

dimana α adalah faktor beban.

Bukti :

Misalkan diberikan suatu list dengan panjang k dan m^n barisan-barisan hash (3) adalah equallly likely. Maka ada $\left[\begin{matrix} n \\ k \end{matrix} \right]$ cara memilih himpunan j sedemikian hingga a_j mempunyai suatu harga khusus dan $(m-1)^{k-1}$ cara menetapkan harga-harga a yang lain. Sehingga probabilitas bahwa suatu list yang diberikan mempunyai panjang k, apabila m^n barisan-barisan hash (3) adalah equally likely adalah

$$P_{nk} = \left[\begin{matrix} n \\ k \end{matrix} \right] (m - 1)^{n-k} \ / \ m^n$$

Sehingga

$$\begin{aligned} P_n(z) &= \sum_{k \geq o} P_{nk} z^k \\ &= \sum_{k \geq o} \left[\begin{matrix} n \\ k \end{matrix} \right] (m - 1)^{n-k} \ / \ m^n \ z^k \\ &= \frac{1}{m^n} \sum_{k \geq o} \left[\begin{matrix} n \\ k \end{matrix} \right] (m - 1)^{n-k} \ z^k \\ &= \frac{1}{m^n} ((m - 1) + z)^n \\ &= \left(1 + \frac{z - 1}{m} \right)^n \end{aligned}$$

Diasumsikan bahwa suatu pencarian tidak berhasil dari pada list dengan panjang k membutuhkan $k + \delta_{ko}$ probe.

Sehingga

$$C'_n = \sum_{k \geq o} (k + \delta_{ko})\, P_{nk}$$

$$= P'_n(1) + P_n(0)$$

$$= \frac{n}{m} + \left(1 - \frac{1}{m}\right)^n$$

$$= \frac{n}{m} + \left(1 + \frac{-\left(\frac{n}{m}\right)}{n}\right)^n$$

dengan $\alpha = n/m$, didapat

$$U(\alpha) \approx e^{-\alpha} + \alpha .$$

Pertimbangakan jumlah total probe untuk menemukan semua kunci. Suatu list dengan panjang k mengkonstribusi $\binom{k+1}{2}$ kepada total probe itu; karenanya

$$C_n = m \sum_{k \geq o} \binom{k+1}{2} P_{nk} / n$$

$$= \frac{m}{n} \left(\frac{1}{2} P''_n(1) + P'_n(1) \right)$$

$$= \frac{m}{n} \left[\frac{1}{2} \left(\frac{n\,(n-1)}{m} \right) + \frac{n}{m} \right]$$

$$= 1 + \frac{n-1}{2m},$$

dengan $\alpha = n/m$, didapat

$$S(\alpha) \approx 1 + \frac{1}{2}\alpha .$$ ■

3.4.3. Penggandengan Bersatu

Dalam penggandengan bersatu (*coalesced chaining*), rekord-rekord yang diletakkan dalam list diberi tambahan field link yang dimasukkan ke dalamnya, tanpa tambahan m kepala-kepala list yang dibutuhkan oleh penggandengan

terpisah. Yakni tiap lokasi tabel *T*[i] digunakan untuk meletakkan suatu rekord, di dalamnya mengandung suatu field Berikut[i].

Apabila *T*[*h*(k)] ditemukan ternyata mengandung elemen lain pada suatu usaha penyisipan *k*, field-field Berikut ditelusuri sampai mencapai nil; maka diambil suatu lokasi tabel kosong *T*[*kosong*], memasang field Berikut nil terakhir menunjuk kepada *T*[*kosong*] dan meletakkan *k* dalam *T*[*kosong*].

Pencarian lokasi-lokasi kosong bermula dari *T*[*m*-1] kembali ke belakang ke arah *T*[0]. Tiap waktu lokasi kosong dibutuhkan, pencarian dilanjutkan ke belakang dari mana pada saat berhenti sebelumnya. Untuk menghentikan pencarian dikenalkan elemen *dummy T*[0] yang selalu kosong. Tabelnya akan *overflow* apabila semua lokasi penuh. Algorithma proses ini diberikan oleh Algoritma C.

Algoritma C. mencari tabel *m*-simpul, menemukan suatu kunci yang diberikan *k*. Jika *k* tidak ada dalam tabel dan tabel tidak penuh, maka *k* disisipkan.
Simpul-simpul tabel dinotasikan dengan *T*[i], $0 \leq i \leq m$, terdiri dari dua tipe berbeda kosong (*empty*) dan ditempati (*occupied*). Simpul yang ditempati mengandung field kunci *K*[i], field link Berikut[i] dan mungkin field-field lainnya. Peubah bantu *R* digunakan untuk membantu menemukan ruang-ruang kosong; apabila tabelnya kosong, maka didapat $R = m + 1$, dan seperti penyisipan-penyisipan dibuat akan

selalu benar bahwa $T[j]$ ditempati untuk semua j dalam range $R \leq j \leq m$. Dengan perjanjian, $T[0]$ akan selalu kosong.

Algoritma C. *(Chained scatter table search and insertion)*

C1. [Hash.]

Menetapkan i := $h(k)$ + 1 (sekarang $1 \leq i \leq m$).

C2. [Apakah ada list ?]

Jika $T[i]$ kosong, lompat ke C6.

(Yang lain $T[i]$ diduduki; akan melihat list dari simpul-simpul yang diduduki itu dimulai disini.)

C3. [Membandingkan.]

Jika $k = K[i]$, Algoritma ini berhenti dengan berhasil.

C4. [Lanjutkan ke berikut.]

Jika Berikut[i] $\neq 0$, tetapkan i := Berikut[i] dan kembali ke langkah C3.

C5. [Menemukan simpul kosong.]

(Pencarian tidak berhasil, dan ingin menemukan posisi kosong dalam tabel itu.

Kurangi R satu atau beberapa kali sampai ditemukan suatu harga sedemikian hingga $T[R]$ kosong. Jika $R = 0$, Algoritma berhenti dengan overflow (tidak ada simpul kosong yang tersisa); yang lain tetapkan Berikut[i] := R, i := R.

C6. [Menyisipkan kunci baru.]

Tandai $T[i]$ sebagai simpul yang diduduki , dengan $K[i]$:= k dan Berikut[i] := 0. ■

Tampak bahwa dalam Algoritma C -untuk penggandengan bersatu ini-, elemen-elemen dengan alamat hash yang sama terletak dalam list-list yang bercampur baur.

Contoh 20 :

Perhatikan penyisipan 30 kata ke dalam 31 lokasi seperti diperlihatkan dalam Gambar 3.5.

Penyisipan kata-kata kunci THE, OF, AND, TO, A dan IN tidak terjadi tabrakan, sehingga dengan mudah diletakkan dalam $T[h(k)]$. Pada penyisipan THAT terjadi tabrakan sebab h[THAT] = h[OF] = 9. Sehingga tabel di-*scan* ke belakang dari T[30], mencari lokasi kosong.

Ditemukan T[30], kemudian ditetapkan

Berikut[9] := 30;

T[30] := THAT; (Gambar 3.5.(a))

Selanjutnya IS, WAS dan HE disisipkan tanpa tabrakan, tetapi untuk penyisipan FOR,

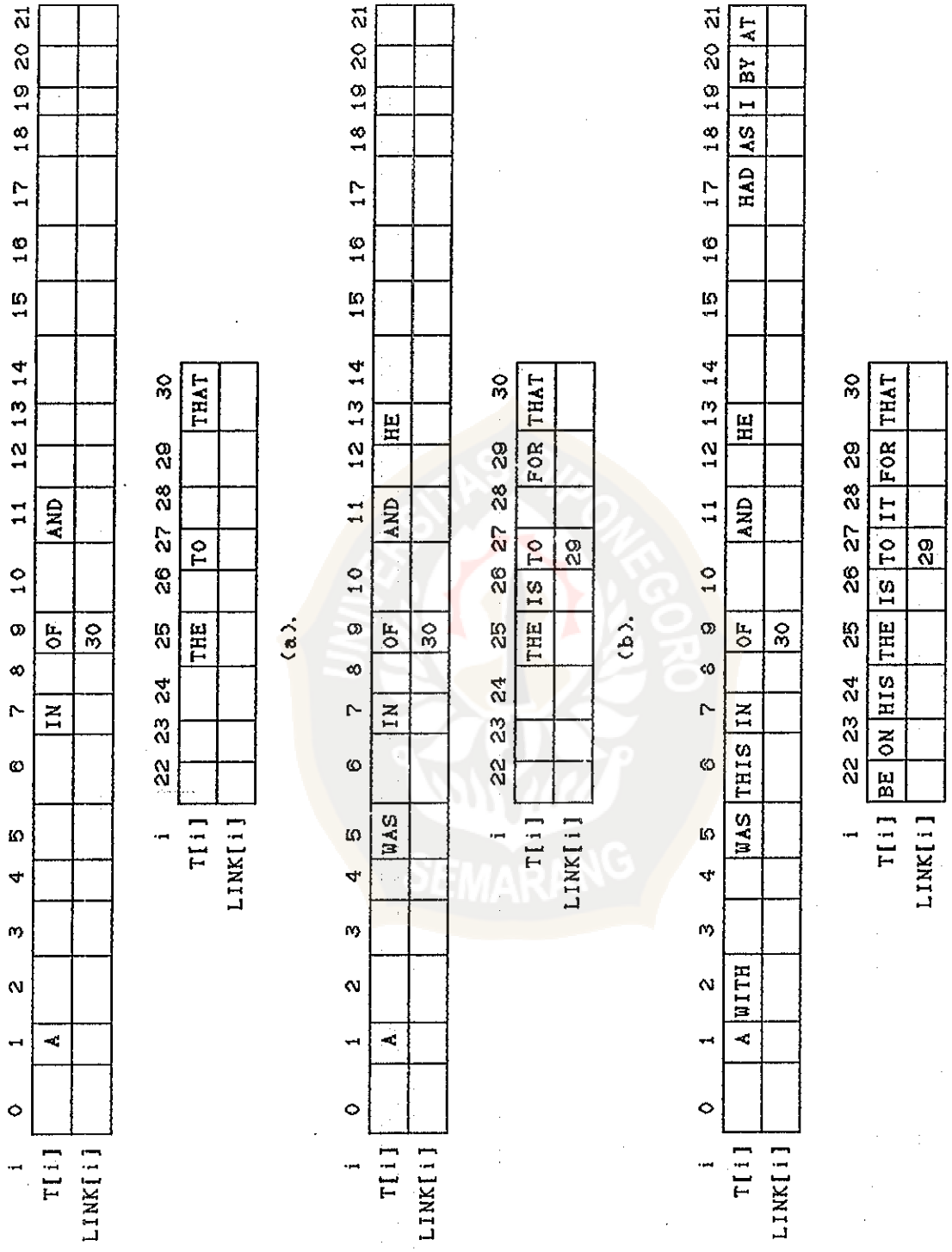
h[FOR] = 27,

h[27] telah terisi oleh T[TO], sehingga kembali mencari dari belakang satu lokasi kosong, saat ini dari T[28]. ditemukan T[29] kosong dan meletakkan FOR di dalamnya.

Berikut[27] := 29;

T[29] := FOR; (Gambar 3.5.(b))

dan seterusnya. □



Gambar 3.5. Tabel Hash yang dibangun oleh penggandengan bersatu.

Menghitung jumlah probe penggandengan bersatu

Dengan menguji Gambar 3.5.(a), tujuh dari 31 lokasi tabel terisi, sehingga tabelnya 7/31 ≈ 23% penuh. Pada suatu pencarian berhasil dimana masing-masing dari 7 elemen itu sekarang adalah equally likely, jumlah rata-rata lokasi yang di-probe dalam loop C5 Algoritma C adalah 1,14; dihitung sebagai berikut :

enam dari 7 elemen itu ditemukan pada probe pertama dan satunya (THAT) ditemukan pada probe kedua.

Jumlah rata-rata probe menjadi 8/7 ≈ 1,14.

Untuk suatu pencarian pencarian tidak berhasil probe dihitung dengan asumsi bahwa : masing-masing dari 31 lokasi tabel sama-sama mungkin (*equally probable*).

Jika alamat hash itu adalah 0, 2, melompat ke 6, 8, 10, 12, melompat ke 24, 26, 28 atau 29; maka probe tunggal memenuhi karena lokasi-lokasi itu kosong.

Secara sama, untuk lokasi-lokasi 1, 7, 11, 25, 27 dan 30 -karena field-field Berikutnya nil, probe tunggal memenuhi.

Untuk alamat hash 9 maka 2 probe dibutuhkan, pertama pada T[9] dan kemudian pada T[Berikut[9]].

Jumlah rata-rata probe dalam suatu pencarian tidak berhasil menjadi 32/31 ≈ 1,03. Demikian juga untuk menghitung probe di Gambar 3.5.(b) dan (c).

Teorema 7 :

Jumlah rata-rata probe untuk pencarian tidak berhasil apabila digunakan penggandengan bersatu untuk menangani tabrakan adalah

$$U(\alpha) \approx 1 + \frac{1}{4}\left[e^{2\alpha} - 1 - 2\alpha \right],$$

dimana α adalah faktor beban.

Bukti :

Misalkan $c(k_1,k_2,k_3, \ldots)$ adalah jumlah dari barisan-barisan hash (3), yang membuat Algoritma C membentuk tepat k_1 list dengan panjang 1, k_2 list dengan panjang 2 dan seterusnya jika $k_1 + 2k_2 + 3k_3 + \ldots = n$.

Relasi pengulangan dimana mendefinisikan bilangan-bilangan : $c(k_1,k_2,k_3, \ldots)$ ini adalah

$$\begin{aligned} c(k_1,k_2,k_3, \ldots) = {} & (m - n + 1)\, c(k_1 - 1, k_2, \ldots) + \\ & (k_1 + 1)c(k_1 + 1, k_2 - 1, k_3, \ldots) + \\ & 2(k_2 + 1)c(k_1, k_2 + 1, k_3 - 1, \ldots) + \ldots. \end{aligned}$$

dan

$$S_n = \sum_{\substack{j \geq 1 \\ k_1+k_2+\ldots=n}} \left\lceil \frac{j}{2} \right\rceil k_j\, c(k_1,k_2, \ldots).$$

Karenanya,

$$\begin{aligned} S_{n+1} = {} & (m - n) \sum_{\substack{j \geq 1 \\ k_1+2k_2+\ldots=n+1}} \left\lceil \frac{j}{2} \right\rceil k_j\, c(k_1 - 1,k_2, \ldots) + \\ & \sum_{\substack{j \geq 1 \\ k_1+2k_2+\ldots=n+1}} \left\lceil \frac{j}{2} \right\rceil k_j(k_1 + 1)c(k_1 + 1,k_2 - 1, \ldots) + \ldots \end{aligned}$$

$$= (m-n)S_n + \sum_{\substack{j\geq 1 \\ k_1+2k_2+\ldots=n}} c(k_1,k_2,\ldots)\ X$$

$$\left[\left(\binom{j}{2}k_j - \binom{1}{2} + \binom{2}{2}\right)k_1 + \left(\binom{j}{2}k_j - \binom{2}{2} + \binom{3}{2}\right)2k_2 + \ldots\right]$$

$$= mS_n + \sum_{\substack{j\geq 1 \\ k_1+2k_2+\ldots=n}} j^2 k_j c(k_1,k_2,\ldots)$$

$$= mS_n + 2S_n + nm^n.$$

Akibatnya,

$$S_n = (n-1)m^{n-1} + (n-2)m^{n-2}(m+2) + m(m+2)^{n-2}.$$

$$= \frac{1}{4}(m(m+2)^n - m^{n+1} - 2nm^n).$$

Perhatikan bahwa jumlah total probe dalam pencarian tidak berhasil, dijumlahkan meliputi semua harga-harga m dari $h(k)$; masing-masing list dari panjang k mengkonstribusi $k + \delta_{ko} + \binom{k}{2}$ kepada total itu, karenanya

$$m^{n+1} C_n' = m^{n+1} + S_n$$

$$C_n' = 1 + \frac{S_n}{m^{n+1}}$$

$$= 1 + \frac{\frac{1}{4}(m(m+2)^n - m^{n+1} - 2nm^n)}{m^{n+1}}$$

$$= 1 + \frac{1}{4}\left[\left(\frac{m+2}{m}\right)^n - 1 - 2\frac{n}{m}\right]^n$$

$$= 1 + \frac{1}{4}\left[\left(1+\frac{2}{m}\right)^n - 1 - 2\frac{n}{m}\right]^n$$

Dengan $\alpha = n/m$, didapat

$$U(\alpha) \approx 1 + \frac{1}{4}\left[e^{2\alpha} - 1 - 2\alpha\right].$$

■