Fungsi Hash dan Metode *Collision Resolution*

Riffa Rufaida (13507007) [1)]

1) Jurusan Teknik Informatika ITB, Bandung 40132, email: if17007@students.if.itb.ac.id

Abstract – *Setiap record data memiliki kunci unik yang membedakan data tersebut dengan yang lain. Hashing –menggunakan metode hash- menawarkan metode penyimpanan data maupun file dengan waktu akses yang cepat tanpa pencarian yang berlama-lama. Fungsi hash dapat pula digunakan dalam pengecekan autentikasi suatu pesan. Pada implementasinya, akan terjadi tabrakan pada penggunaan fungsi hash, saat data yang berbeda mempunyai nilai hash yang sama. Untuk menanggulanginya, ada beragam collision resolution.*

Kata Kunci*: Fungsi hash, modulo, indeks address, collision, kriptografi.*

1. PENDAHULUAN

Data yang kita simpan di computer tentu memiliki beragam ukuran dan jumlahnya sangat banyak. Penyimpanan yang tidak terstruktur tentu akan membuat mesin bekerja lambat pada saat pencarian. Karena itu dibutuhkan suatu cara penyimpanan pada memori agar pencarian bisa berjalan lebih cepat.

Hashing merupakan metode pengaksesan yang dilakukan dengan cara mengonversi himpunan kunci rekaman menjadi himpunan alamat pengingat menggunakan fungsi hash.

Setiap data yang berupa record mempunyai field kunci yang unik yang membedakan suatu record dengan record lainnya. Fungsi hash akan memetakan suatu record dengan nilai kunci k ke sebuah home address. Jika kita ingin mencari suatu data record tertentu, maka kita menggunakan fungsi hash kembali. Dengan fungsi hash kita hitung nilai hash dari data record tertentu, yang merupakan home address, lokasi memori tempat data tersebut tersimpan. Hal ini akan mempersingkat waktu dibandingkan dengan melakukan pencarian satu persatu pada memori.

2. HASHING

Metode penempatan dan pencarian yang memanfaatkan metode hash disebut hashing atau 'hash addressing' dan fungsi yang digunakan disebut fungsi hashing / fungsi hash.

Fungsi hashing atau fungsi hash inilah yang dapat menjadi salah satu alternatif dalam menyimpan atau mengorganisasi file dengan metode akses langsung. Fungsi hash berupaya menciptakan "fingerprint" dari berbagai data masukan. Fungsi hash akan mengganti atau mentransposekan data tersebut untuk menciptakan fingerprint, yang biasa disebut hash value (nilai hash). Hash value biasanya akan digambarkan sebagai suatu string pendek yang terdiri atas huruf dan angka yang terlihat random (data biner yang ditulis dalam notasi heksadesimal).

Berkaitan dengan upayanya untuk menciptakan "fingerprint", fungsi hash digunakan juga pada algoritma enkripsi untuk menjaga integritas sebuah data.

Dalam konsepnya modern ini –selain digunakan pada penyimpanan data-, fungsi hash adalah sebuah fungsi matematika, yang menerima masukan string yang panjangnya sebarang, mengambil sebuah panjang variable dari string masukan tersebut –yang disebut pre-image-, lalu mekonversinkannya ke sebuah string keluaran dengan ukuran tetap (fixed), dan umumnya lebih pendek dari ukuran string semula, yang disebut message digest.

Pada penggunaan fungsi hash, saat keadaan tertentu dapat terjadi tabrakan (coallision) pada home address yang dihasilkan. Yaitu saat munculnya nilai hash yang sama dari beberapa data yang berbeda. Untuk mengantisipasi keadaan ini ada beberapa metode yang dapat digunakan, seperti perubahan fungsi hash atau mengurangi perbandingan antara jumlah data yang tersimpan dengan slot address yang tersedia. Hal-hal tersebut dapat meminimalisir tabrakan, tetapi tidak menghilangkannya. Kita tetap memerlukan collision resolution -sebuah prosedur untuk menempatkan data yang memiliki address yang sama-.

3. MACAM-MACAM FUNGSI HASH

Fungsi hash diimplementasi untuk mengkonversi himpunan kunci rekaman (K) menjadi himpunan alamat memori (L). Bisa dinotasikan dengan
H : K -> L

Aspek yang perlu dipertimbangkan dalam pemilihan fungsi hash adalah :

- fungsi hash harus mudah dan cepat dihitung
- fungsi hash sebisa mungkin mendistribusikan posisi yang dimaksud secara uniform sepanjang himpunan L sehingga collision yang mungkin terjadi dapat diminimalkan.

Ada beberapa fungsi hash yang dapat digunakan, yaitu :

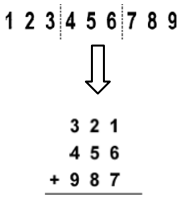
1. Kunci mod N
   F(kunci) = kunci mod N
   N merupakan banykanya kunci yang biasanya dijadikan sebagai ukuran table. Alamat index dalam table dari fugsi ini adalah 0 sampai N-1.

2. Kunci mod P
   F(kunci) = kunci mod P
   P merupakan bilangan prima terkecil yang lebih besar dari N dan N merupakan jumlah kunci. Hashing dengan fungsi ini akan memberikan resolusi tabrakan (collision) yang lebih rendah untuk metode hashing yang sama.

3. Truncation
   Nama lainnya adalah substringing, dilakukan dengan cara pemangkasan / pemotongan. Pemotongan dapat dilakukan pada posisi manapun, dengan catatan bahwa aturan diberlakukan untuk seluruh kunci dalam himpunan.

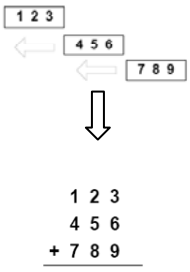
4. Folding
   Terdiri atas dua macam, yaitu :
   a. folding by boundary,
   b. folding by shifting.

   Foulding by boundary dapat dilakukan dengan membagi digit kunci tersebut dengan cara seolah-olah melipat batas pembagian digit seperti berikut :



   Tiap kelompok digit lalu dijumlahkan dengan atau tanpa carry.

   Foulding by shifting dapat dilakukan dengan cara seolah-olah menggeser batas pembagian digit seperti berikut :



   Tiap kelompok digit dijumlahkan dengan atau tanpa carry.

5. Radix Convertion
   Kunci ditransformasikan menjadi bilangan basis lain untuk mendapatkan nilai hashnya. Umumnya basis yang digunakan di luar dari basis 2-10.Misalnya jika kunci 38652 akan ditempatkan dalam table berukuran 10000 dengan basis 11, maka:
   3x114+8x113+6x112+5x111+2x110 = 5535411
   Nilai hash 55354 telah melampaui batas hash, makan pecehan terbesar dari harsh tersebut akan dibuang sehingga diapatkan harsh 5354.

6. Mid-square
   Kunci ditransformasikan dengan cara dikuadratkan dan diambil bagian tengahnya (asalkan jumlah digit kiri dan kanan sama))sebagai nilai hash. Misalnya jika kunci = 3121 akan ditempatkan pada table berukuran 1000, maka 31212 = 9740641, diambil 406 sebagai nilai hashnya.

7. Penambahan Kode ASCII
   Jika kunci bukan kode numeric, home address didapatkan dari penjumlahan kode ASCII setiap huruf pembentuk kunci.

4. TABRAKAN

Dengan menggunakan hashing, maka hubungan korespondensi satu-satu antara record key dengan alamat record akan hilang. Selalu timbul kemungkinan dimana terdapat dua buah record dengan kunci yang berbeda namun memiliki home address yang sama, dan terjadi tabrakan (collision).

Tabrakan dapat diminimalisir dengan melakukan penggantian pada fungsi hash yang digunakan, atau mengurangi packing factor.

4.1 Packing Factor

Packing factor, bisa disebut juga dengan packing density ataupun load factor adalah perbandingan antara jumlah data yang tersimpan terhadap jumlah slot address yang tersedia.

$$Packing\ Factor = \frac{Number\ of\ Record\ Stored}{Total\ Number\ of\ Storage\ Location}$$

Penggantian fungsi hash dan pengurangan packing factor hanya meminimasisasi tabrakan, tetap dibutuhkan collision resolution.

4.2 Collision Resolution

Pada hashing untuk penempatan data, output dari fungsi hash tidak selalu unik, namun hanya berupa kemungkinan suaru alamat yang dapat ditempati. Jika suatu home address sudah ditempati oleh record lain, maka harus dicarikan alamat lain. Proses pencarian

alamat lain inilah yang disebut sebagai prosedur collision resolution.

4.2.1 Metode Collision Resolution

a. Open addressing

Metode dengan pencarian alamat alternative di alamat-alamat selanjutnya yang masih kosong.

Cara :

- Linear probing

Pencarian dilakukan dengan jarak pencarian tetap

- Quardratic probing

Pencarian dilakukan dengan jarak pencarian berubah dengan perubahan tetap

- Double hashing

Pencarian dilakukan menggunakan dua fungsi hash, yaitu fungsi H1 untuk menentukan home address dan fungsi H2 untuk menentukan increment jika terjadi tabrakan. Syarat metode ini adalah ukuran table merupakan bilangan prima sehingga kemungkinan terjadinya siklus pencarian pada slot yang sama dapat dihindari.

Algoritma :

Tentukan home address dari key dengan fungsi H1.
IF home address kosong THEN
  Sisip record pada home address.
ELSE
  Hitung increment dengan fungsi H2 → misalnya H2 (key) = x
  Temukan slot kosong dengan cara increment sejauh x dari home address.
  IF slot kosong ditemukan THEN
    Sisip record pada slot kosong.
  ELSE
    Tabel telah penuh.

b. Computed chaining

Menggunakan "pseudolink" untuk menemukan next address jika terjadi collision.

Tidak menyimpan actual address pada pseudolink, tapi address ditemukan dengan menghitung apa yang tersimpan pada pseudolink.

Kinerja pseudolink lebih baik dibandingkan non-link karena menghilangkan penebakan lokasi (address).

Algoritma :

Temukan home address dari key.
IF home address kosong THEN
  Sisip record baru ke home address.
ELSE
  Set 3 prioritas increment untuk mencari new address :
  1 : Tentukan increment (new key).
  2 : Tentukan increment (key pada current address).
  3 : Penjumlahan hasil prioritas 1 dan 2.
  WHILE new address belum kosong dan tabel belum penuh DO
    Cek posisi mulai dari home address untuk ke – 3 prioritas untuk mencari new address yang kosong.
  IF new address belum kosong THEN
    Set ke – 3 nilai prioritas dengan kelipatannya.
  END WHILE
  IF tabel penuh THEN
    Proses sisip tidak dilakukan, keluarkan pesan "Tabel Penuh".
  ELSE
    Sisip record baru pada new address.
    Set field pseudolink pada home address dengan kode urut prioritas yang digunakan.

c. Coalesced hashing

Algoritma :

Tentukan home address dari key.
IF home address kosong THEN
  Sisip record pada home address.
ELSE
  Temukan record terakhir dari data yang telah menempati home address, dengan mengikuti link.
  Temukan slot kosong mulai dari yang terletak pada address paling bawah.
  IF slot kosong tidak ditemukan THEN
    File telah penuh.
  ELSE
    Sisip record pada slot kosong.
    Set link field dari record terakhir yang ber-home address sama ke alamat dari record yang baru disisip.

d. Chained progressive overflow

Algoritma :

Tentukan home address dari key.
IF home address kosong THEN
  Sisip record pada home address.
ELSE
  Temukan slot kosong yang terletak setelah home address.
  IF slot kosong ditemukan THEN
    Sisip record pada slot kosong.
  ELSE
    Tabel telah penuh.

e. Binary tree

Metode yang menggunakan struktur binary tree untuk pencarian address ketika erjadi tabrakan dengan memberikan ua pilihan langkah :

- Continue : melanjutkan pencarian address berikutnya yang mungkin ditempati oleh record yang akan disisipkan.
- Move : memindahkan record yang menempati address ke address berikutnya yang memungkinkan untuk ditempati record lama.

Algoritma :

Tentukan home address dari key yang akan di-sisipkan (new key).
IF home address kosong THEN
  Sisip record pada home address.
ELSE
  WHILE new address tidak kosong dan tabel belum penuh DO
  Generate binary tree untuk mendapatkan new address :

CREATE cabang kiri dengan menjumlahkan increment (new key) + current address.
CREATE cabang kanan dengan menjumlahkan increment (key pada node) + current address.
END WHILE
{Tabel penuh atau new address kosong}
IF tabel penuh THEN
    Proses sisip tidak dilakukan, keluarkan pesan "Tabel Penuh".
ELSE
    Sisip record pada new address dengan cara :
    IF new address didapatkan di cabang kiri THEN
        Sisip record baru ke posisi new address.
    ELSE
        Pindahkan parent (new address) ke new address.
        Sisip record baru ke posisi parent (new address).

f. Cuckoo hashing

Metode yang dapat menggunakan satu atau dua table hash. Dua table digunakan untuk melakukan pertukaran atau pemindahan kunci. Apabila memiliki 1 tabel, maka H1 dan H2 akan mengacu pada table yang sama, tetapi dengan fungsi yang berbeda. Untuk 1 tabel, fungsi H1 menghasilkan hash untuk ½ table dan H2 menghasilkan hash untuk ½ table yang tersisa. Apabila dengan 2 tabel, maka H1 adalah fungsi untuk tabel – 1, dan H2 adalah fungsi untuk tabel – 2.

Kunci baru akan menyisihkan kunci lama pada posisi yang ditunjuk. Kunci lama yang tersisih akan menyisihkan lagi Kunci lama pada posisi yang ditunjuk berikutnya. Proses ini akan berjalan terus hingga ditemukan tempat kosong atau tabel telah penuh.

Dapat terjadi deadlock apabila alamat hash yang dihasilkan dari H1 dan H2 adalah sama untuk 3 kali perulangan pencarian tempat kosong secara berturut-turut.

Penanganan deadlock dapat dilakukan dengan 2 cara yakni :

- Menambahkan fungsi hash modifikasi dari H1 atau H2 sedemikian hingga alamat hash yang dihasilkan dari H1 dan H2 menjadi berbeda.
- Membatasi perulangan hingga suatu "MaxLoop" sedemikian hingga apabila jumlah perulangan telah mencapai MaxLoop dan masih belum menemukan tempat kosong maka proses penyisipan selesai.

Algoritma dengan 1 tabel :

Nyatakan x = new key.
Tentukan home address dari x dengan H1 dan H2 → H1 (x) dan H2 (x).
WHILE new address belum kosong dan tabel belum penuh DO
IF key [H1 (x)] atau key [H2 (x)] = x THEN
    Proses sisip tidak dilakukan, keluar pesan "key telah ada".
ELSE
    pos = H1 (x).
    WHILE tabel belum penuh dan belum terjadi deadlock DO
    IF key (pos) kosong THEN
        Sisip record baru pada pos.
        Proses sisip selesai.
    ELSE
        Tukarkan key (pos) ←→ x.
        IF pos = H1(x) THEN
            pos = H2 (x).
        ELSE
            pos = H1 (x).
    END WHILE
END WHILE

Algoritma dengan 2 tabel :

Nyatakan x = new key.
Tentukan home address dari x dengan H1 dan H2 → H1 (x) dan H2 (x).
WHILE new address belum kosong dan tabel belum penuh DO
    IF key T1 [H1 (x)] atau key T2 [H2 (x)] = x THEN
        Proses sisip tidak dilakukan, keluar pesan "key telah ada".
    ELSE
        WHILE belum terjadi deadlock DO
        IF key T1 [H1 (x)] kosong THEN
            Sisip record baru pada T1 [H1 (x)].
            Proses sisip selesai.
        ELSE
            Tukarkan key T1 [H1 (x)] ←→ x.
            IF key T2 [H2 (x)] kosong THEN
                Sisip record baru pada T2 [H2 (x)].
                Proses sisip selesai.
            ELSE
                Tukarkan key T2 [H2 (x)] ←→ x.
        END WHILE
END WHILE

5. FUNGSI HASH SATU ARAH

Fungsi hash satu arah adalah fungsi hash yang bekerja dalam satu arah. Maksud dari satu arah disini adalah bahwa pesan yang sudah diubah menjadi message digest tidak dapat dikembalikan lagi menjadi pesan semula (irreversible).

Sifat-sifat fungsi *hash* satu-arah adalah sebagai berikut:

1. Fungsi *H* dapat diterapkan pada blok data berukuran berapa saja.
2. *H* menghasilkan nilai (*h*) dengan panjang tetap (*fixed-length output*).
3. *H*(*x*) mudah dihitung untuk setiap nilai *x* yang diberikan.

4. Untuk setiap *h* yang dihasilkan, tidak mungkin dikembalikan nilai *x* sedemikian sehingga $H(x) = h$. Itulah sebabnya fungsi *H* dikatakan fungsi *hash* satu-arah (*one-way hash function*).
5. Untuk setiap *x* yang diberikan, tidak mungkin mencari $y \neq x$ sedemikian sehingga $H(y) = H(x)$.
6. Tidak mungkin mencari pasangan *x* dan *y* sedemikian sehingga $H(x) = H(y)$.

Beberapa fungsi *hash* satu-arah yang sudah dibuat, antara lain:
- *MD2*, *MD4*, *MD5*,
- *Secure Hash Function* (*SHA*),
- *Snefru*,
- *N-hash*,
- *RIPE-MD*.

6. KESIMPULAN

Fungsi hash memiliki beragam kegunaan, salah satunya dapat kita gunakan sebagai metode pengaksesan suatu data pada memori.

Fungsi hash -satu arah- juga memiliki kegunaan lain mengingat sifatnya yang memberi “fingerprint” pada sebuah pesan, yaitu menjamin integritas suatu pesan.

Pada penggunaan fungsi hash dapat dipastikan akan terjadi tabrakan, karena itu telah ada berbagai macam collision resolution untuk meminimalkan tabrakan.

DAFTAR REFERENSI

[1] Munir, Rinaldi. (2008). Bahan Kuliah IF2091 Matematika Diskrit. Departemen Teknik Informatika, Institut Teknologi Bandung.

[2] Organisasi File Relatif (Hash) www.mikroskil.ac.id/~poiwong/Berkas/Sesi%209%20-%20File%20Hash.ppt Diakses pada tanggal 5 Januari 2009.

[3] Fungsi Hash www.informatika.org/~rinaldi/Kriptografi/2006-2007/Fungsi%20Hash.ppt Diakses pada tanggal 5 Januari 2009.

[4] Struktur Data http://lecturer.ukdw.ac.id/anton/download/TIstrukdat12.ppt Diakses pada tanggal 5 Januari 2009.

[5] Hashing www.yk-edu.org/e-refleksi/sharefile/files/11112008182510_Berkas_6.pdf Diakses pada tanggal 5 Januari 2009.