PEMANFAATAN PRINSIP SARANG MERPATI UNTUK MEMBUAT PERFECT HASH

Dannis Muhammad Mangan NIM : 13507112
Program Studi Teknik Informatika, Institut Teknologi Bandung
Jl. Ganesha 10, Bandung
email : dannis_m@students.itb.ac.id

Abstrak *– Makalah ini berisi tentang penggunaan Prinsip Sarang Merpati (Pigeonhole Principle) untuk membuat fungsi hash yang ideal. fungsi hash dan tabel hash sering digunakan untuk penyimpanan dalam database. Penggunaan fungsi hash bisa dimaksimalkan jika tidak ada bentrokan (collision), makalah ini secara tidak berbelit-belit akan menjelaskan topik tersebut.*

Kata Kunci*: Prinsip Sarang Merpati, Hash, Perfect, Bentrokan*

1. PENDAHULUAN

Dunia dewasa ini sangat bergantung pada komputer. Data yang akan diolah oleh komputer bertambah banyak seiring dengan naiknya kecepatan *clock* komputer. Data yang disimpan di dalam memori komputer di dalam memori komputer perlu ditempatkan dalam suatu cara yang membuat pencariannya dilakukan dengan cepat. Fungsi hash adalah salah satu solusi cerdas untuk mengatasi masalah tersebut.

Namun demikian, sulit untuk membuat fungsi hash yang benar-benar sempurna dan ideal, karena cenderung akan terjadi collision. Dengan Prinsip Sarang Burung dapat dihitung sedemikian sehingga dapat dibuat ‘Perfect Hash’, *hash* yang ideal dan mangkus.

2. FUNGSI HASH

Fungsi *Hash* (dilambangkan dengan *h(k)*) adalah salah satu cara enkripsi dengan untuk mengubah k (*key*) menjadi suatu nilai dalam interval [0....X], dimana "X" adalah jumlah maksimum dari *record-record* yang dapat ditampung dalam tabel.

Jumlah maksimum ini bergantung pada ruang memori yang tersedia. Fungsi *Hash* yang ideal adalah mudah dihitung dan bersifat *random*, agar dapat menyebarkan semua *key*. Dengan *key* yang tersebar, berarti data dapat terdistribusi secara seragam bentrokan dapat dicegah. Sehingga kompleksitas waktu model *Hash* dapat mencapai O(1), di mana kompleksitas tersebut tidak ditemukan pada struktur model lain.

2.1 Membuat Tabel Hash dengan Fungsi Hash

1) Dengan cara menggunakan sisa pembagian

Jumlah lokasi memori yang tersedia dihitung, kemudian jumlah tersebut digunakan sebagai pembagi untuk membagi nilai yang asli dan menghasilkan sisa. Sisa tersebut adalah nilai hashnya. Secara umum, rumusnya adalah

h(k)= k mod m

Dalam hal ini m adalah jumlah lokasi memori yang tersedia pada array. Contoh algoritmanya

```
1  unsigned add_hash ( void *key, int len )
2 {
3  unsigned char *p = key;
4  unsigned h = 0;
5  int i;
6
7  for ( i = 0; i < len; i++ )
8    h += p[i];
9
10  return h;
11 }
```

Fungsi *hash* tersebut menempatkan *record* dengan kunci k pada suatu lokasi memori yang beralamat h(k). Hasilnya ditempatkan pada tabel yang disebut tabel *hash*.
Metode ini sering menghasilkan nilai *hash* yang sama dari dua atau lebih nilai aslinya atau disebut dengan bentrokan (collision).

2) XOR hash

```
1 unsigned xor_hash ( void *key, int len )
2 {
3  unsigned char *p = key;
4  unsigned h = 0;
5  int i;
6
7  for ( i = 0; i < len; i++ )
8    h ^= p[i];
```

```
 9
10   return h;
11 }
```

Namun, algoritma ini terlalu simpel untuk ukuran data yang besar.

3)Rotating hash

```
 1 unsigned rot_hash ( void *key, int len )
 2 {
 3   unsigned char *p = key;
 4   unsigned h = 0;
 5   int i;
 6
 7   for ( i = 0; i < len; i++ )
 8     h = ( h << 4 ) ^ ( h >> 28 ) ^ p[i];
 9
10   return h;
11 }
```

Algoritma ini bagus dan cukup efektif, algoritma ini banyak dipakai untuk fungsi hash.

4) Bernstein hash

Algoritma ini sangat bagus untuk data yang kecil:

```
 1 unsigned djb_hash ( void *key, int len )
 2 {
 3   unsigned char *p = key;
 4   unsigned h = 0;
 5   int i;
 6
 7   for ( i = 0; i < len; i++ )
 8     h = 33 * h + p[i];
 9
10   return h;
11  }
```

5)Shift-Add-XOR hash

Walaupun algortima inis angat efektif untuk data yang kecil, untuk data yang besar tidak begitu efektif

```
1 unsigned sax_hash ( void *key, int len )
 2 {
 3   unsigned char *p = key;
 4   unsigned h = 0;
 5   int i;
 6
 7   for ( i = 0; i < len; i++ )
 8     h ^= ( h << 5 ) + ( h >> 2 ) + p[i];
 9
10   return h;
11 }
```

6) FNV hash

```
1 unsigned fnv_hash ( void *key, int len )
 2 {
 3   unsigned char *p = key;
 4   unsigned h = 2166136261;
 5   int i;
 6
 7   for ( i = 0; i < len; i++ )
 8     h = ( h * 16777619 ) ^ p[i];
 9
10   return h;
11 }
```

7) One-at-a-Time hash

Algoritma ini dibuat juga oleh Jenkins

```
1 unsigned oat_hash ( void *key, int len )
 2 {
 3   unsigned char *p = key;
 4   unsigned h = 0;
 5   int i;
 6
 7   for ( i = 0; i < len; i++ ) {
 8     h += p[i];
 9     h += ( h << 10 );
10     h ^= ( h >> 6 );
11   }
12
13   h += ( h << 3 );
14   h ^= ( h >> 11 );
15   h += ( h << 15 );
16
17   return h;
18  }
```

8) JSW hash

```
1 unsigned jsw_hash ( void *key, int len )
 2 {
 3   unsigned char *p = key;
 4   unsigned h = 16777551;
 5   int i;
 6
 7   for ( i = 0; i < len; i++ )
 8     h = ( h << 1 | h >> 31 ) ^ tab[p[i]];
 9
10   return h;
11 }
```

Algoritma ini dikenal sebagai algoritma terbaik dalam hal efektifitas waktu dalam data yang besar. Algoritma ini sangat disarankan

9)ELF hash

ELF function sebenarnya lebih dipakai untuk testing database karena kemampuannya untuk meng-kover kemungkinan-kemungkinan input untuk menghindari error.

```
unsigned elf_hash ( void *key, int len )
{
  unsigned char *p = key;
  unsigned h = 0, g;
  int i;

  for ( i = 0; i < len; i++ ) {
    h = ( h << 4 ) + p[i];
    g = h & 0xf0000000L;

    if ( g != 0 )
      h ^= g >> 24;

    h &= ~g;
  }

  return h;
}
```

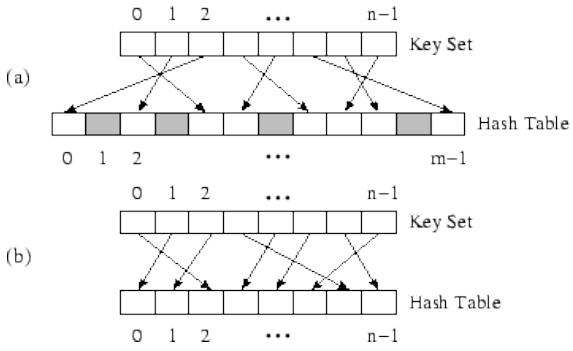
10)Jenkins hash

Teknik hash dengan cara Jenkins sudah teruji dan terbukti, namun kelemahannya adalah waktu eksekusinya

```
#define hashsize(n) ( 1U << (n) )
#define hashmask(n) ( hashsize ( n ) - 1 )

#define mix(a,b,c) \
{ \
  a -= b; a -= c; a ^= ( c >> 13 ); \
  b -= c; b -= a; b ^= ( a << 8 ); \
  c -= a; c -= b; c ^= ( b >> 13 ); \
  a -= b; a -= c; a ^= ( c >> 12 ); \
  b -= c; b -= a; b ^= ( a << 16 ); \
  c -= a; c -= b; c ^= ( b >> 5 ); \
  a -= b; a -= c; a ^= ( c >> 3 ); \
  b -= c; b -= a; b ^= ( a << 10 ); \
  c -= a; c -= b; c ^= ( b >> 15 ); \
}

unsigned jen_hash ( unsigned char *k,
unsigned length, unsigned initval )
{
  unsigned a, b;
  unsigned c = initval;
  unsigned len = length;

  a = b = 0x9e3779b9;

  while ( len >= 12 ) {
    a += ( k[0] + ( (unsigned)k[1] << 8 )
      + ( (unsigned)k[2] << 16 )
      + ( (unsigned)k[3] << 24 ) );
    b += ( k[4] + ( (unsigned)k[5] << 8 )
      + ( (unsigned)k[6] << 16 )
      + ( (unsigned)k[7] << 24 ) );
    c += ( k[8] + ( (unsigned)k[9] << 8 )
      + ( (unsigned)k[10] << 16 )
      + ( (unsigned)k[11] << 24 ) );

    mix ( a, b, c );

    k += 12;
    len -= 12;
  }

  c += length;

  switch ( len ) {
  case 11: c += ( (unsigned)k[10] << 24 );
  case 10: c += ( (unsigned)k[9] << 16 );
  case 9 : c += ( (unsigned)k[8] << 8 );
  /* First byte of c reserved for length */
  case 8 : b += ( (unsigned)k[7] << 24 );
  case 7 : b += ( (unsigned)k[6] << 16 );
  case 6 : b += ( (unsigned)k[5] << 8 );
  case 5 : b += k[4];
  case 4 : a += ( (unsigned)k[3] << 24 );
  case 3 : a += ( (unsigned)k[2] << 16 );
  case 2 : a += ( (unsigned)k[1] << 8 );
  case 1 : a += k[0];
  }

  mix ( a, b, c );

  return c;
}
```

2.2. Perfect Hash

Suatu perfect hash dapat dibuat jika angka dan struktur dari kunci enkripsi dari hash tersebut diketahui.



Gambar 1. A) Perfect Hash, B) Minimal Perfect hash

Fungsi hash yang ideal berbentuk injektif, yaitu semua input dipetakan ke dalam nilai hash yang berbeda-beda. Dengan cara tersebut, pencarian dapat dilakukan dengan cepat tanpa mengulangi pencarian lagi

Semua fungsi hash yang bersifat injektif dan mempunyai range dari bilangan buluat antara 0 dan n-1, dimana n adalah jumlah dari input yang dapat diterima, disebut perfect hash. Jadi, selain hasilnya dapat langsung ditemukan, suatu perfect hash juga mempunyai tabel hash yang kompak tanpa slot yang kosong.

2.3. Prinsip Sarang Merpati

Prinsip Sarang Merpati menyebutkan bahwa, jika kita punya dua buah bilangan cacah n dan m, dengan n > m, kemudian sebanyak n buah burung ditaruh di m buah sarangnya, maka setidaknya ada satu sarang yang berisi lebih dari satu burung.

Gambar 2. Sepuluh ekor Merpati dalam 9 sarangnya

Formalnya, teorema ini menyatakan bahwa tidak ada fungsi injektif pada himpunan berhingga yang domainnya lebih besar dari kodomainnya.

Teorema ini ternyata dapat menjelaskan fakta sehari-hari yang bisa saja tidak terduga, misalnya:

1. Misalkan kita mengambil 367 orang secara acak dari belahan dunia manapun, maka dengan teorema ini, *minimal* ada 2 orang yang mempunyai tanggal ulang tahun (masehi) yang sama.

2. Misalkan ada beberapa orang yang dapat ‘berjabat tangan’ dengan orang lainnya, dengan Prinsip Sarang Merpati, pasti ada pasangan orang yang mempunyai jumlah ‘jabatan tangan’ yang sama.

3. HASIL DAN PEMBAHASAN

Untuk membuat perfect hash kita perlu menghindari adanya bentrokan. Suatu bentrokan terjadi jika dalam fungsi hash tersebut ada 2 kunci hash yang mempunyai hasil yang sama. Untuk itulah perlu dipakai Prinsip Sarang Burung,

Prinsip Sarang Burung membuktikan bahwa tidak ada algoritma hash yang dapat meng-hash setiap kunci hash ke dalam indeks-indeks yang unik jika kemungkinan kuncinya lebih besar daripada ukuran tabel hash tersebut.

Contohnya studi kasus dengan persoalan perkalian 10 bilangan yang dimana bilangan ke-4 dan ke-5nya berbeda. Fungsi perfect hashnya:

```
1 unsigned hash ( unsigned pid )
2 {
3   return pid / 1000 % 100;
4 }
```

Teknik yang dipakai adalah, walaupun hanya perkalian 10 bilangan, tabel yang dipakai haruslah berisi 100 slot karena hasilnya setidaknya mempunyai dua digit. Jika kita hanya memakai tabel dengan 10 slot maka bisa terjadi bentrokan yang menyebabkan fungsi hash tersebut tidak ‘perfect’.

4. KESIMPULAN

Untuk membuat *perfect hash*, perlu diterapkan Prinsip Sarang Merpati (*Pigeonhole Principle*) untuk menghindari terjadinya bentrokan (*collision*), yaitu dengan memanfaatkan pernyataan dari teorema tersebut yang menyatakan bahwa tidak ada fungsi injektif pada himpunan berhingga yang domainnya lebih besar dari kodomainnya.

Diantara banyaknya fungsi hash yang ada, keefektifannya tergantung dari data-data yang akan

ditangani, jadi harus dipilih fungsi yang tepat agar fungsi hash yang dibentuk adalah sempurna.

DAFTAR REFERENSI

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

[2] Pigeonhole Principle. http://en.wikipedia.org/wiki/Pigeonhole_principle. Tanggal akses : 30 Desember 2008 pukul 20:00

[3]The Art of Hashing. http://www.eternallyconfuzzled.com/tuts/algorithms/jsw_tut_hashing.aspx Tanggal akses : 6 Januari 2009 pukul 19.00