ANALISA METODE PENCARIAN HASH

Imelda Sri Duma Yanti

Dosen Fakultas Ilmu Komputer Universitas Methodist Indonesia
imensinaga@yahoo.co.id

Abstract

Hash Methods Search efficiency use of a better place. Search relative (*Hash Search*) is divided into two kinds, namely Hash Closed (Closed Hashing) and Hash Open (Open Hashing). On Closed Hash there is the possibility of more than one data has a value the same function, causing a collision (*collision*). The way to overcome this collision can be done with some strategies such as, Resolution Linear (*Linear Resolution*), and Double Hashing Overflow. Each - each strategy has its advantages and disadvantages

***Keywords** : Hash Methods. Linear Resolution*

1. Pendahuluan

Pencarian (*searching*) merupakan hal yang sering dilakukan dan menggunakan banyak waktu. Seperti saat menggunakan *text editor*, untuk mencari sebuah kata, atau mencari kata dan kemudian menggantikannya dengan kata yang lainnya. Atau mencari kata tertentu dan menghitung frekuensi kemunculan kata tersebut dalam dokumen, atau juga melakukan prinsip kerja *Windows explorer* dan *internet explorer* menggunakan prinsip pencarian kata yang sejenis.

Salah satu metoda pencarian (*searching*) yang memiliki efisiensi penggunaan tempat yang lebih baik adalah pencarian relatif (*Hash Search*). Pencarian relatif (*Hash Search*) menggunakan rumus tertentu untuk melakukan proses penempatan dan pencarian data. Pencarian relatif (Hash Search) terbagi atas 2 macam, yaitu Hash Tertutup (*Closed Hashing*) dan Hash Terbuka (*Open Hashing*). Pada Hash Tertutup terdapat kemungkinan lebih dari satu data memiliki nilai fungsi yang sama sehingga terjadi tabrakan (*collision*). Cara untuk mengatasi tabrakan ini dapat dilakukan dengan beberapa strategi seperti, Resolusi Linier (*Linear Resolution*), *Overflow* dan *Double Hashing*. Masing – masing strategi memiliki kelebihan dan kekurangan.

2. Tujuan dan Manfaat

Tujuan dari penelitian ini adalah untuk melakukan analisa terhadap kinerja metoda pencarian relatif (*Hash Search*). Dan manfaat yang diperoleh dari penelitian ini untuk mengetahui proses kerja pencarian relatif (*Hash Search*) yang dapat dijadikan pendukung dalam proses pembelajaran.

3. Metode Pencarian Relatif (Hash Search)

Metoda pencarian Relatif (Hash Search) ini hampir mirip dengan metoda pencarian langsung (*Direct Search*), yaitu dengan menggunakan rumus tertentu baik pada saat penempatan maupun pencarian data. Pencarian Relatif (*Hash Search*) memiliki efisiensi penggunaan tempat yang lebih baik daripada pencarian langsung (*Direct Search*). Fungsi (I – 1) yang digunakan oleh metoda pencarian langsung (*Direct Search*) memiliki efisiensi penggunaan tempat yang buruk, sehingga metoda pencarian Relatif (*Hash Search*) memperbaikinya dengan menggunakan fungsi operasi modulo (mod). Fungsi operasi modulo (mod) ini sering disebut sebagai Fungsi *Hash* dan tempat penampungan data disebut Tabel *Hash*. Fungsi *Hash* bukan merupakan fungsi satu – satu seperti fungsi bagi dari metoda pencarian Langsung (*Direct Search*) sehingga ada kemungkinan beberapa data memiliki hasil fungsi yang sama. Hal ini mengakibatkan terjadinya tabrakan (*collision*) pada saat penempatan data ke dalam tabel sehingga diperlukan strategi untuk mengatasi tabrakan (*collision*) ini. Strategi untuk mengatasi tabrakan (*collision*) ini ada bermacam – macam dan masing – masing memiliki kelebihan dan kekurangannya masing – masing. Fungsi *Hash* ini menyebabkan data yang tersimpan dalam Tabel *Hash* memiliki 2 jenis alamat (*address*) yaitu,

1. *Home Address*, adalah lokasi (*address*) yang diperoleh dengan menggunakan Fungsi *Hash*.
2. *Real (Physical) Address*, adalah lokasi (*address*) dimana data tersimpan dalam tabel.

Metoda pencarian Relatif (*Hash Search*) terdiri dari 2 macam yaitu,

1. Hash Tertutup (*Closed Hash*)
2. Hash Terbuka (*Open Hash*)

3.1 Hash Tertutup (Closed Hash)

Hash Tertutup (*Closed Hash*) didefinisikan sebagai cara *Hash* dimana data langsung disimpan dalam tabel *Hash* dengan ukuran tabel tertentu (dapat ditentukan sendiri). Fungsi *Hash* tersebut dapat didefinisikan sebagai berikut:

$H(X) = X \bmod [\text{ukuran tabel}]$ Rumus 2.1

dengan : X = nilai data
H(X) = hasil fungsi *Hash*.

Dalam penerapan fungsi *Hash* di atas dalam perhitungan, terdapat kemungkinan beberapa data memiliki hasil fungsi *Hash* yang sama. Hal ini menyebabkan terjadinya tabrakan (*collision*) pada saat penempatan data. Tabrakan (*collision*) dapat diatasi dengan menggunakan beberapa strategi antara lain,

1. Resolusi Linier (*Linear Resolution*)
2. *Overflow*
3. *Double Hashing*

a. Resolusi Linier (Linear Resolution)

Strategi ini mengatasi tabrakan (*collision*) dengan cara mencari lokasi lain sebagai tempat dari data yang mengalami tabrakan (*collision*) dengan menghitung nilai fungsi *Hash* yang baru (*Rehashing*) dengan menggunakan rumus berikut,

$h'_i(X) = (h'_{i-1}(X) + 1) \bmod [\text{ukuran tabel}]$

Rumus 2.2

b. Overflow

Strategi ini mengatasi / mengurangi tabrakan (*collision*) dengan cara membagi tabel menjadi 2 buah, yaitu tabel utama dan tabel *overflow* dengan ketentuan ukuran tabel utama lebih besar daripada ukuran tabel *overflow*.

Cara penempatan data pada tabel utama dilakukan dengan menggunakan fungsi *Hash*. Jika terjadi tabrakan (*collision*), maka data ditempatkan pada tabel *overflow*. Cara penempatan data pada tabel *overflow* dapat dilakukan dengan cara sekuensial atau menggunakan fungsi *Hash* yang baru. Jika tabel *overflow* telah penuh, maka data yang masuk ke dalam tabel *overflow* akan ditolak.

c. Double Hashing

Strategi *Double Hashing* ini hampir mirip dengan strategi *Overflow*, yaitu dengan menggunakan dua buah tabel sebagai tempat pengisian data. Perbedaannya adalah strategi ini menggunakan dua buah tabel yang sama besar ukurannya. Proses penempatan dan pencarian data untuk strategi ini juga sama persis dengan strategi *overflow*.

3.2 Hash Terbuka (Open Hash)

Hash Terbuka (*Open Hash*) menggunakan tabel *Hash* hanya sebagai tempat penyimpanan *pointer* untuk merangkai elemen dalam *Linked List*. Elemen data yang memiliki nilai fungsi yang sama (*Home Address* yang sama) akan dirangkai membentuk satu *Linked List*. Penambahan elemen ke dalam *Linked List* dapat dilakukan di depan *List* atau di belakang *List*. *Hash* Terbuka (*Open Hash*) mampu menampung data dengan jumlah yang belum diketahui sebelumnya. Hal ini dicapai karena alokasi dinamis dalam *Linked List* tidak mengharuskan untuk menentukan besar memori yang disediakan, cukup menentukan ukuran tabel yang merupakan *Array of Pointer*.

4. Metode Penelitian

Penelitian ini dilakukan dengan menggunakan beberapa algoritma perancangan perangkat lunak penerapan metoda pencarian relatif (*hash search*) yang dibagi menjadi 5 bagian yaitu,

1. Algoritma Pengecekan Data *Input* dan Tabel *Hash*.
2. Algoritma Penempatan Data pada *Close Hash*, terdiri atas,
   a. Algoritma Penempatan Data pada *Close Hash* dengan Resolusi Linier.
   b. Algoritma Penempatan Data pada *Close Hash* dengan Resolusi *Overflow*.
   c. Algoritma Penempatan Data pada *Close Hash* dengan *Double Hashing*.
3. Algoritma Penempatan Data pada *Open Hash*, terdiri atas,
   a. Algoritma Penempatan Data pada *Open Hash* dengan penambahan data di depan *list*.
   b. Algoritma Penempatan Data pada *Open Hash* dengan penambahan data di belakang *list*.
4. Algoritma Pencarian Data pada *Close Hash*, terdiri atas,
   a. Algoritma Pencarian Data pada *Close Hash* dengan Resolusi Linier.
   b. Algoritma Pencarian Data pada *Close Hash* dengan Resolusi *Overflow*.
   c. Algoritma Pencarian Data pada *Close Hash* dengan *Double Hashing*.
5. Algoritma Pencarian Data pada *Open Hash*, terdiri atas,
   a. Algoritma Pencarian Data pada *Open Hash* dengan penambahan data di depan *list*.
   b. Algoritma Pencarian Data pada *Open Hash* dengan penambahan data di belakang *list*.

4. Analisa dan Pembahasan

4.1 Analisa

Pada bagian ini akan dilakukan analisa kinerja metode Hash dengan studi kasus data sebagai berikut:

1. *Data* : 2, 4, 5, 8, 10, 15, 19, 35.
2. Proses penempatan data pada tabel *close hash* dengan resolusi linier dan ukuran tabel utama = 9, didapat :

Langkah – langkah yang dilakukan:

PENEMPATAN DATA PADA 'Close Hash [Resolusi Linier]'

Ukuran Tabel Utama = 9, Fungsi hash = h(x) mod 9

Data : 2, 4, 5, 8, 10, 15, 19, 35
---
1. Data '2'

TABEL UTAMA -> h(x) = 2 mod 9 = 2, lokasi 2 kosong sehingga data '2' ditempatkan pada lokasi 2
---
2. Data '4'

TABEL UTAMA -> h(x) = 4 mod 9 = 4, lokasi 4 kosong sehingga data '4' ditempatkan pada lokasi 4
---
3. Data '5'

TABEL UTAMA -> h(x) = 5 mod 9 = 5, lokasi 5 kosong sehingga data '5' ditempatkan pada lokasi 5
---
4. Data '8'

TABEL UTAMA -> h(x) = 8 mod 9 = 8, lokasi 8 kosong sehingga data '8' ditempatkan pada lokasi 8
---
5. Data '10'

TABEL UTAMA -> h(x) = 10 mod 9 = 1, lokasi 1 kosong sehingga data '10' ditempatkan pada lokasi 1
---
6. Data '15'

TABEL UTAMA -> h(x) = 15 mod 9 = 6, lokasi 6 kosong sehingga data '15' ditempatkan pada lokasi 6
---
7. Data '19'

TABEL UTAMA -> h(x) = 19 mod 9 = 1 -> terjadi tabrakan dengan data '10' -> h(x) = (1 + 1) mod 9 = 2 -> terjadi tabrakan dengan data '2'-> h(x) = (2 + 1) mod 9 = 3, lokasi 3 kosong sehingga data '19' ditempatkan pada lokasi 3
---
8. Data '35'

TABEL UTAMA -> h(x) = 35 mod 9 = 8 -> terjadi tabrakan dengan data '8' -> h(x) = (8 + 1) mod 9 = 0, lokasi 0 kosong sehingga data '35' ditempatkan pada lokasi 0

Proses pencarian data '19' pada tabel *close hash* dengan resolusi linier dan ukuran tabel utama = 9, didapat

PENCARIAN DATA '19' PADA 'Close Hash [Resolusi Linier]'

Ukuran Tabel Utama = 9, Fungsi hash = h(x) mod 9

TABEL UTAMA -> h(x) = 19 mod 9 = 1 -> Data '19' tidak ditemukan.
-> h(x) = (1 + 1) mod 9 = 2 -> Data '19' tidak ditemukan.
-> h(x) = (2 + 1) mod 9 = 3 -> Data '19' ditemukan pada lokasi 3.

Proses penempatan data pada tabel *close hash* dengan resolusi *overflow* dan fungsi hash baru (*rehashing*), ukuran tabel utama = 8 dan ukuran tabel *overflow* = 5, didapat

```
PENEMPATAN DATA PADA 'Close Hash [Resolusi Overflow - Rehash]'
----------------------------------------------------------------
Ukuran Tabel Utama    = 8, Fungsi hash = h(x) mod 8
Ukuran Tabel Overflow = 5, Fungsi hash = g(x) mod 5
----------------------------------------------------------------
Data : 2, 4, 5, 8, 10, 15, 19, 35
--------------------------
1. Data '2'
--------------------------
TABEL UTAMA   -> h(x) = 2 mod 8 = 2, lokasi 2 kosong
sehingga data '2' ditempatkan pada lokasi 2
--------------------------
2. Data '4'
--------------------------
TABEL UTAMA   -> h(x) = 4 mod 8 = 4, lokasi 4 kosong
sehingga data '4' ditempatkan pada lokasi 4
--------------------------
3. Data '5'
--------------------------
TABEL UTAMA   -> h(x) = 5 mod 8 = 5, lokasi 5 kosong
sehingga data '5' ditempatkan pada lokasi 5
--------------------------
4. Data '8'
--------------------------
TABEL UTAMA   -> h(x) = 8 mod 8 = 0, lokasi 0 kosong
sehingga data '8' ditempatkan pada lokasi 0
--------------------------
5. Data '10'
--------------------------
TABEL UTAMA   -> h(x) = 10 mod 8 = 2 -> terjadi tabrakan
dengan data '2'
Pencarian dilanjutkan ke Tabel Overflow
TABEL OVERFLOW -> g(x) = 10 mod 5 = 0, lokasi 0 kosong
sehingga data '10' ditempatkan pada lokasi 0
--------------------------
6. Data '15'
--------------------------
TABEL UTAMA   -> h(x) = 15 mod 8 = 7, lokasi 7 kosong
sehingga data '15' ditempatkan pada lokasi 7
--------------------------
7. Data '19'
--------------------------
TABEL UTAMA   -> h(x) = 19 mod 8 = 3, lokasi 3 kosong
sehingga data '19' ditempatkan pada lokasi 3
--------------------------
8. Data '35'
--------------------------
TABEL UTAMA   -> h(x) = 35 mod 8 = 3 -> terjadi tabrakan
dengan data '19'
Pencarian dilanjutkan ke Tabel Overflow
TABEL OVERFLOW -> g(x) = 35 mod 5 = 0 -> terjadi tabrakan
dengan data '10' g(x) = (0 + 1) mod 5 = 1, lokasi 1 kosong
sehingga data '35' ditempatkan pada lokasi 1
```

Proses pencarian data ‘35’ pada tabel *close hash* dengan resolusi *overflow – rehashing*, ukuran tabel utama = 8 dan ukuran tabel *overflow* = 5, didapat :

```
PENCARIAN DATA '35' PADA 'Close Hash [Resolusi Overflow - Rehash]'
----------------------------------------------------------------
Ukuran Tabel Utama    = 8, Fungsi hash = h(x) mod 8
Ukuran Tabel Overflow = 5, Fungsi hash = g(x) mod 5
----------------------------------------------------------------

TABEL UTAMA   -> h(x) = 35 mod 8 = 3 -> Data '35' tidak
ditemukan.
Pencarian dilanjutkan ke Tabel Overflow
TABEL OVERFLOW -> g(x) = 35 mod 5 = 0 -> Data '35' tidak
ditemukan.
g(x) = (0 + 1) mod 5 = 1 -> Data '35' ditemukan pada lokasi 1.
```

Proses penempatan data 2, 4, 5, 8, 10, 15, 19, 35 pada tabel *close hash* dengan *double hash - rehashing*, ukuran tabel = 6, didapat :

```
PENEMPATAN DATA PADA 'Close Hash [Double Hash - Rehash]'
----------------------------------------------------------------
Ukuran Tabel Utama    = 6, Fungsi hash = h(x) mod 6
Ukuran Tabel Overflow = 6, Fungsi hash = g(x) mod 6
----------------------------------------------------------------
Data: 2, 4, 5, 8, 10, 15, 19, 35
--------------------------
1. Data '2'
--------------------------
TABEL UTAMA   -> h(x) = 2 mod 6 = 2, lokasi 2 kosong
sehingga data '2' ditempatkan pada lokasi 2
--------------------------
2. Data '4'
--------------------------
TABEL UTAMA   -> h(x) = 4 mod 6 = 4, lokasi 4 kosong
sehingga data '4' ditempatkan pada lokasi 4
--------------------------
3. Data '5'
--------------------------
TABEL UTAMA   -> h(x) = 5 mod 6 = 5, lokasi 5 kosong
sehingga data '5' ditempatkan pada lokasi 5
--------------------------
4. Data '8'
--------------------------
TABEL UTAMA   -> h(x) = 8 mod 6 = 2 -> terjadi tabrakan
dengan data '2'
          Pencarian dilanjutkan ke Tabel Overflow
TABEL OVERFLOW -> g(x) = 8 mod 6 = 2, lokasi 2 kosong
sehingga data '8' ditempatkan pada lokasi 2
--------------------------
5. Data '10'
--------------------------
TABEL UTAMA   -> h(x) = 10 mod 6 = 4 -> terjadi tabrakan
dengan data '4'
Pencarian dilanjutkan ke Tabel Overflow
TABEL OVERFLOW -> g(x) = 10 mod 6 = 4, lokasi 4 kosong
sehingga data '10' ditempatkan pada lokasi 4
--------------------------
6. Data '15'
--------------------------
TABEL UTAMA   -> h(x) = 15 mod 6 = 3, lokasi 3 kosong
sehingga data '15' ditempatkan pada lokasi 3
--------------------------
7. Data '19'
--------------------------
TABEL UTAMA   -> h(x) = 19 mod 6 = 1, lokasi 1 kosong
sehingga data '19' ditempatkan pada lokasi 1
--------------------------
8. Data '35'
--------------------------
TABEL UTAMA   -> h(x) = 35 mod 6 = 5 -> terjadi tabrakan
dengan data '5'
Pencarian dilanjutkan ke Tabel Overflow
TABEL OVERFLOW -> g(x) = 35 mod 6 = 5, lokasi 5 kosong
sehingga data '35' ditempatkan pada lokasi 5
```

Proses pencarian data ‘16’ pada tabel *close hash* dengan *double hash – rehashing*, ukuran tabel = 6, didapat

PENCARIAN DATA '16' PADA 'Close Hash [Double Hash - Rehash]'

Ukuran Tabel Utama = 6, Fungsi hash = h(x) mod 6
Ukuran Tabel Overflow = 6, Fungsi hash = g(x) mod 6

TABEL UTAMA -> h(x) = 16 mod 6 = 4 -> Data '16' tidak ditemukan.
Pencarian dilanjutkan ke Tabel Overflow
TABEL OVERFLOW -> g(x) = 16 mod 6 = 4 -> Data '16' tidak ditemukan.
g(x) = (4 + 1) mod 6 = 5 -> Data '16' tidak ditemukan.
g(x) = (5 + 1) mod 6 = 0 -> lokasi 0 masih kosong
sehingga data '16' tidak ditemukan !

Proses penempatan data 2, 4, 5, 8, 10, 12, 15, 16, 19, 24, 35 pada tabel *open hash* dengan penambahan elemen di depan *list* dan ukuran tabel = 8, didapat :
PENEMPATAN DATA PADA 'Open Hash [Depan List]'

Ukuran Tabel = 8, Fungsi hash = h(x) mod 8

Data: 2, 4, 5, 8, 10, 12, 15, 16, 19, 24, 35
---
1. Data '2'

TABEL UTAMA -> h(x) = 2 mod 8 = 2
---
2. Data '4'

TABEL UTAMA -> h(x) = 4 mod 8 = 4
---
3. Data '5'

TABEL UTAMA -> h(x) = 5 mod 8 = 5
---
4. Data '8'

TABEL UTAMA -> h(x) = 8 mod 8 = 0
---
5. Data '10'

TABEL UTAMA -> h(x) = 10 mod 8 = 2
---
6. Data '12'

TABEL UTAMA -> h(x) = 12 mod 8 = 4
---
7. Data '15'

TABEL UTAMA -> h(x) = 15 mod 8 = 7
---
8. Data '16'

TABEL UTAMA -> h(x) = 16 mod 8 = 0
---
9. Data '19'

TABEL UTAMA -> h(x) = 19 mod 8 = 3
---
10. Data '24'

TABEL UTAMA -> h(x) = 24 mod 8 = 0
---
11. Data '35'

TABEL UTAMA -> h(x) = 35 mod 8 = 3
Proses pencarian data ‘8’ pada tabel *open hash* dengan penambahan elemen di depan *list*, dan ukuran tabel = 8, didapat :
PENCARIAN DATA '8' PADA 'Open Hash [Depan List]'

Ukuran Tabel Utama = 8, Fungsi hash = h(x) mod 8
---
h(x) = 8 mod 8 = 0
Pencarian pada lokasi-0 dimulai !
List ke - 1 - Data '24', -> pencarian dilanjutkan.
List ke - 2 - Data '16', -> pencarian dilanjutkan.
List ke - 3 - Data '8' -> Data '8' ditemukan !
Proses penempatan data 2, 4, 5, 8, 10, 12, 15, 16, 19, 24, 35 pada tabel *open hash* dengan penambahan elemen di belakang *list* dan ukuran tabel = 8, didapat :
PENEMPATAN DATA PADA 'Open Hash [Belakang List]'

Ukuran Tabel = 8, Fungsi hash = h(x) mod 8

Data: 2, 4, 5, 8, 10, 12, 15, 16, 19, 24, 35
---
1. Data '2'

TABEL UTAMA -> h(x) = 2 mod 8 = 2
---
2. Data '4'

TABEL UTAMA -> h(x) = 4 mod 8 = 4
---
3. Data '5'

TABEL UTAMA -> h(x) = 5 mod 8 = 5
---
4. Data '8'

TABEL UTAMA -> h(x) = 8 mod 8 = 0
---
5. Data '10'

TABEL UTAMA -> h(x) = 10 mod 8 = 2
---
6. Data '12'

TABEL UTAMA -> h(x) = 12 mod 8 = 4
---
7. Data '15'

TABEL UTAMA -> h(x) = 15 mod 8 = 7
---
8. Data '16'

TABEL UTAMA -> h(x) = 16 mod 8 = 0
---
9. Data '19'

TABEL UTAMA -> h(x) = 19 mod 8 = 3
---
10. Data '24'

TABEL UTAMA -> h(x) = 24 mod 8 = 0
---
11. Data '35'

TABEL UTAMA -> h(x) = 35 mod 8 = 3
Proses pencarian data ‘16’ pada tabel *open hash* dengan penambahan elemen di belakang list, dan ukuran tabel = 8, didapat
PENCARIAN DATA '16' PADA 'Open Hash [Belakang List]'

Ukuran Tabel Utama = 8, Fungsi hash = h(x) mod 8
---
h(x) = 16 mod 8 = 0
Pencarian pada lokasi-0 dimulai !
List ke - 1 - Data '8', -> pencarian dilanjutkan.
List ke - 2 - Data '16' -> Data '16' ditemukan !

4. 2 Pembahasan

Secara garis besar, proses penyelesaian metoda pencarian Relatif (*Hash Search*) dapat dibagi menjadi 3 bagian yaitu,

1. Proses pengecekan *input* data
2. Proses penempatan data
3. Proses pencarian data

Sedangkan metoda pencarian Relatif (*Hash Search*) terdiri dari 2 macam dengan perincian sebagai berikut,

1. Metoda pencarian *Hash* Tertutup (*Closed Hash*) dengan beberapa strategi untuk mengatasi tabrakan (*collision*) pada saat penempatan data antara lain,
- Resolusi Linier (*Linear Resolution*)
  - *Overflow*, terbagi dua yaitu *Rehashing* dan Sekuensial
  - *Double Hashing*, terbagi dua yaitu *Rehashing* dan Sekuensial.
2. Metoda pencarian *Hash* Terbuka (*Open Hash*) dengan 2 jenis proses penempatan data antara lain,
  - Penempatan data di depan *List*
  - Penempatan data di belakang *List*

Berdasarkan pembagian di atas, maka proses penyelesaian dari metoda pencarian Relatif (*Hash Search*) dapat dibagi menjadi beberapa proses berikut,

1. Proses pengecekan *input* data.
2. Proses penempatan data untuk metoda,
   a. *Hash* Tertutup (*Close Hash)* dengan strategi – strategi untuk mengatasi tabrakan (*collision*) yaitu Resolusi Linier (*Linear Resolution*), *Overflow* dan *Double Hashing*.
   b. *Hash* Terbuka (*Open Hash*) dengan proses penempatan data di depan *List* dan di belakang *List*.
3. Proses pencarian data untuk metoda,
   a. *Hash* Tertutup (*Close Hash)* dengan strategi – strategi untuk mengatasi tabrakan (*collision*) yaitu Resolusi Linier (*Linear Resolution*), *Overflow* dan *Double Hashing*.
   b. *Hash* Terbuka (*Open Hash*) dengan proses penempatan data di depan *List* dan di belakang *List*.

4.2.1 Proses Pengecekan Input Data

Dalam merancang perangkat lunak pembelajaran metoda pencarian *Hash Search* ini, penulis menggunakan metoda peng*input*an data dengan pemisah tanda koma. Data yang di-*input* bertipe data numerik *integer*. Proses pembacaan datanya adalah sebagai berikut,

1. Periksa apakah *input* data masih kosong atau tidak. Jika *input* data masih kosong maka proses tidak dilanjutkan.
2. Checking apakah ada data yang sama, jika ada maka munculkan pesan kesalahan.
3. *Input* data dipisahkan berdasarkan tanda koma dengan menggunakan perintah *Split* dan disimpan ke dalam variabel *Array*.
   Contoh, misalkan *input* data 12,1,4,5; maka data akan dipisahkan dan disimpan ke dalam variabel *Array* (misalkan variabel A) seperti berikut,
   A[1] = 12 A[3] = 4
   A[2] = 1 A[4] = 5
4. *Input* data diperiksa apakah terdapat tanda koma yang berurutan. Jika ada berarti akan terjadi *error* dan proses dihentikan.
   Contoh, 12,1,,4,5 → *error*

4.2.2 Proses Penempatan Data

Penempatan data pada *Hash* Tertutup (*Close Hash*) dilakukan dengan menggunakan fungsi *hash*. Penempatan data pada *Hash* Tertutup (*Close Hash*) direpresentasikan dengan menggunakan tabel. Karena fungsi *hash* memungkinkan beberapa data menghasilkan nilai yang sama, maka proses penempatan data pada *Hash* Tertutup (*Close Hash*) dapat menimbulkan tabrakan (*collision*). Masalah tersebut dapat diatasi dengan menggunakan beberapa strategi yaitu Resolusi Linier (*Linear Resolution*), *Overflow* dan *Double Hashing*.

Untuk lebih jelas dapat dilihat pada contoh berikut ini,

Misalkan data *input* 1,4,5,10 dengan jumlah data = 4.

Proses penempatan data dengan menggunakan strategi Resolusi Linier (*Linear Resolution*) adalah sebagai berikut,

1. Misalkan ukuran tabel = 5 maka fungsi *hash* h(x) = x mod 5. Penomoran tabel dimulai dari 0.

0	1	2	3	4

2. Data dimasukkan satu per satu ke dalam tabel dengan dimulai dari data pertama yaitu data 1.
   h(1) = 1 mod 5 = 1. Data 1 ditempatkan pada kotak ke – 1 pada tabel.

	1			
0	1	2	3	4

3. Proses dilanjutkan dengan menempatkan data ke – 2 yaitu data 4 pada tabel.
   h(4) = 4 mod 5 = 4. Data 4 ditempatkan pada kotak ke – 4 pada tabel.

	1			4
0	1	2	3	4

4. Proses dilanjutkan dengan menempatkan data ke – 3 yaitu data 5 pada tabel.
   h(5) = 5 mod 5 = 0. Data 5 ditempatkan pada kotak ke – 0 pada tabel.

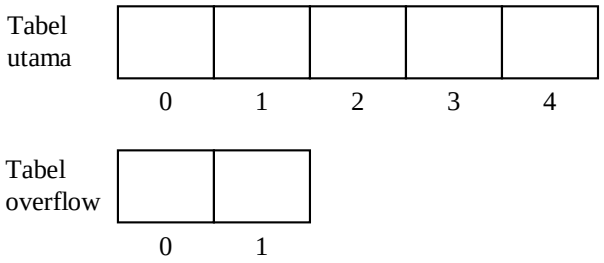
5	1			4
0	1	2	3	4

5. Proses dilanjutkan dengan menempatkan data terakhir yaitu data 10 pada tabel.
   h(10) = 10 mod 5 = 0. Data 10 seharusnya ditempatkan pada kotak ke – 0 pada tabel, namun kotak ke – 0 telah ditempati oleh data 5, sehingga terjadi tabrakan (*collision*). Masalah diatasi dengan membentuk fungsi *hash* yang baru.
   h'(10) = (0 + 1) mod 5 = 1. Data 10 seharusnya ditempatkan pada kotak ke – 1 pada tabel, namun kotak ke – 1 telah ditempati oleh data 1, sehingga terjadi tabrakan (*collision*).
   h'(10) = (1 + 1) mod 5 = 2. Data 10 ditempatkan pada kotak ke – 2 pada tabel.

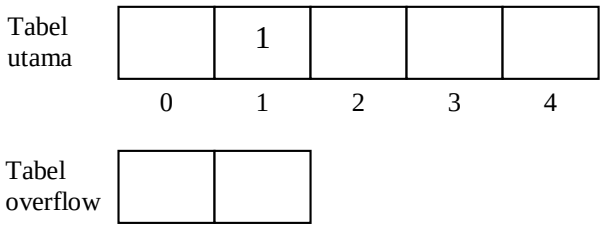
5	1	10		4
0	1	2	3	4

dengan menggunakan strategi *Overflow* adalah sebagai berikut,

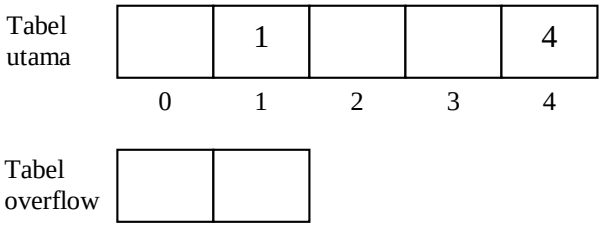
1. Misalkan ukuran tabel utama = 5 dan ukuran tabel overflow = 2 maka fungsi *hash* h(x) = x mod 5 dan fungsi *overflow* g(x) = x mod 2. Penomoran tabel dimulai dari 0.



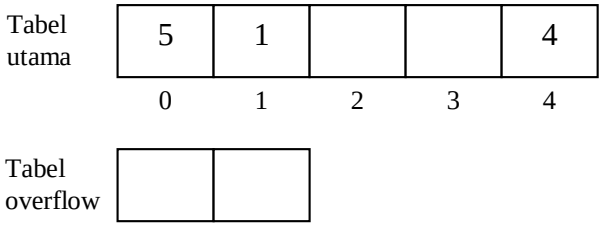
2. Data dimasukkan satu per satu ke dalam tabel dengan dimulai dari data pertama yaitu data 1.
   h(1) = 1 mod 5 = 1. Data 1 ditempatkan pada kotak ke – 1 pada tabel utama.



3. Proses dilanjutkan dengan menempatkan data ke – 2 yaitu data 4 pada tabel.
   h(4) = 4 mod 5 = 4. Data 4 ditempatkan pada kotak ke – 4 pada tabel utama.

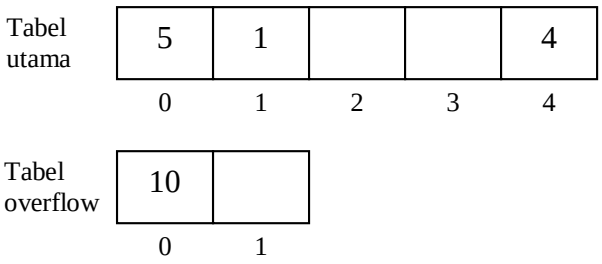


4. Proses dilanjutkan dengan menempatkan data ke – 3 yaitu data 5 pada tabel.
   h(5) = 5 mod 5 = 0. Data 5 ditempatkan pada kotak ke – 0 pada tabel utama.



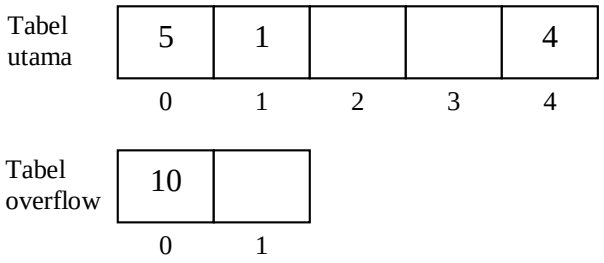
5. Proses dilanjutkan dengan menempatkan data terakhir (10) pada tabel.
   h(10) = 10 mod 5 = 0. Data 10 seharusnya ditempatkan pada kotak ke – 0 pada tabel utama, namun kotak ke – 0 telah ditempati oleh data 5, sehingga terjadi tabrakan (*collision*). Masalah diatasi dengan menempatkan data pada tabel *overflow*. Proses penempatan data pada tabel *overflow* dapat dilakukan dengan cara sekuensial atau membentuk fungsi *hash* yang baru.

   Jika menggunakan cara sekuensial, maka data langsung ditempatkan pada tabel *overflow* dengan pengecekan dimulai dari kotak paling kiri (kotak ke – 0). Jika ketemu kotak yang kosong, maka data langsung ditempatkan pada kotak tersebut. Jika tidak ada lagi kotak yang kosong, maka data ditolak. Untuk contoh di atas maka data 10 ditempatkan pada kotak ke – 0 pada tabel *overflow*.



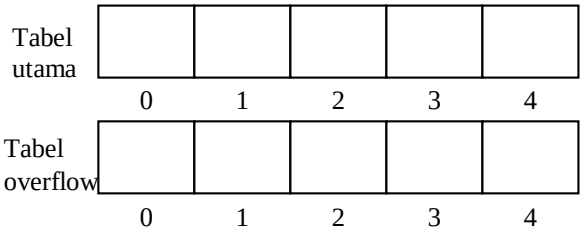
   Jika menggunakan cara membentuk fungsi *hash* yang baru, maka posisi data dicari dengan menggunakan fungsi *hash* tersebut. Jika terjadi tabrakan (*collision*), maka digunakan strategi *Rehashing* lagi sampai ditemukan kotak yang kosong. Jika tidak ada lagi kotak yang kosong, maka data ditolak.

   Untuk contoh di atas, posisi data 10 pada tabel *overflow* dapat dicari dengan menggunakan rumus g(10) = 10 mod 2 = 0. Data 10 ditempatkan pada kotak ke – 0 pada tabel *overflow*.

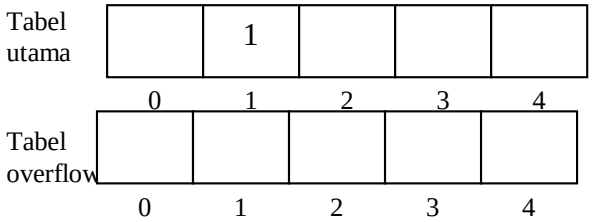


   Proses penempatan data dengan strategi *Double Hashing* sama persis dengan strategi *Overflow*, hanya saja ukuran kedua tabel pada strategi *Double Hashing* sama besar. Proses penempatan data dengan strategi *Double Hashing* adalah sebagai berikut:

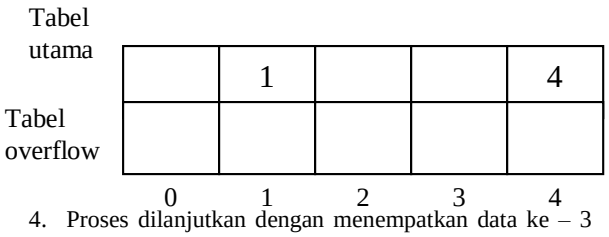
1. Misalkan ukuran tabel *double hash* = 5 maka fungsi *hash* h(x) = x mod 5 dan fungsi *overflow* g(x) = x mod 5. Penomoran tabel dimulai dari 0.



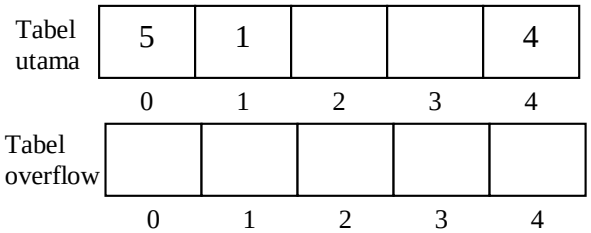
2. Data dimasukkan satu per satu ke dalam tabel dengan dimulai dari data pertama yaitu data 1.
   h(1) = 1 mod 5 = 1. Data 1 ditempatkan pada kotak ke – 1 pada tabel utama.



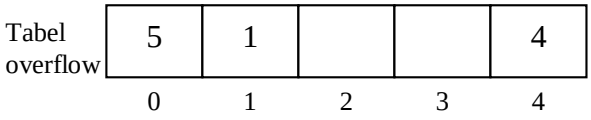
3. Proses dilanjutkan dengan menempatkan data ke – 2 yaitu data 4 pada tabel.
   h(4) = 4 mod 5 = 4. Data 4 ditempatkan pada kotak ke – 4 pada tabel utama.



4. Proses dilanjutkan dengan menempatkan data ke – 3 yaitu data 5 pada tabel.
   h(5) = 5 mod 5 = 0. Data 5 ditempatkan pada kotak ke – 0 pada tabel utama.



5. Proses dilanjutkan dengan menempatkan data terakhir yaitu data 10 pada tabel.
   h(10) = 10 mod 5 = 0. Data 10 seharusnya ditempatkan pada kotak ke – 0 pada tabel utama, namun kotak ke – 0 telah ditempati oleh data 5, sehingga terjadi tabrakan (*collision*). Masalah diatasi dengan menempatkan data pada tabel *overflow*. Proses penempatan data pada tabel *overflow* dapat dilakukan dengan cara sekuensial atau membentuk fungsi *hash* yang baru.
   Jika menggunakan cara sekuensial, maka data langsung ditempatkan pada tabel *overflow* dengan pengecekan dimulai dari kotak paling kiri (kotak ke – 0). Jika ketemu kotak yang kosong, maka data langsung ditempatkan pada kotak tersebut. Jika tidak ada lagi kotak yang kosong, maka data ditolak.



Untuk contoh di atas maka data 10 ditempatkan pada kotak ke – 0 pada tabel *overflow*.
jika menggunakan cara membentuk fungsi *hash* yang baru, maka posisi data dicari dengan menggunakan fungsi *hash* tersebut. Jika terjadi tabrakan (*collision*), maka digunakan strategi *Rehashing* lagi sampai ditemukan kotak yang kosong. Jika tidak ada lagi kotak yang kosong, maka data ditolak.

Untuk contoh di atas, posisi data 10 pada tabel *overflow* dapat dicari dengan menggunakan rumus g(10) = 10 mod 5 = 0. Data 10 ditempatkan pada kotak ke – 0 pada tabel *overflow*.

Sama seperti Hash Tertutup (*Close Hash*), penempatan data pada *Hash* Terbuka (*Open Hash*) juga dilakukan dengan menggunakan fungsi *hash*. Penempatan data pada *Hash* Terbuka (*Open Hash*) direpresentasikan dengan menggunakan *Linked List*. Sesuai dengan karakteristik dari *Linked List* maka proses penempatan data dapat dilakukan dengan 2 cara, yaitu data ditambahkan di depan *List* atau data ditambahkan di belakang *List*.

Contoh, misalkan data *input* 1,4,5,12 dengan jumlah data = 4, ukuran tabel = 4 dan fungsi *hash* h(x) = x mod 4. Proses penempatan data dengan cara penambahan data di depan *List* adalah sebagai berikut,

1. Ukuran tabel untuk menyimpan *pointer* sebanyak 4 kotak dengan penomoran tabel dimulai dari 0.
2. Data ditambahkan satu per satu pada *List* dengan dimulai dari data pertama yaitu data 1.
   h(1) = 1 mod 4 = 1. Data 1 ditambahkan pada *List* ke – 1.
   Proses dilanjutkan dengan menempatkan data ke – 2 yaitu data 4.
   h(4) = 4 mod 4 = 0. Data 4 ditambahkan pada *List* ke – 0.
3. Proses dilanjutkan dengan menempatkan data ke – 3 yaitu data 5.
   h(5) = 5 mod 4 = 1. Data 5 ditambahkan pada *List* ke – 1. Karena proses penambahan data dilakukan di depan *List*, maka semua data pada *List* ke – 1 harus digeser satu langkah ke belakang (samping kanan) dan data 5 diletakkan pada kotak pertama (paling kiri) pada *Linked List* yang bersangkutan.
4. Proses dilanjutkan dengan menempatkan data terakhir yaitu data 12 pada *List*.
   h(12) = 12 mod 4 = 0. Data 12 ditambahkan pada *List* ke – 0. Karena proses penambahan data dilakukan di depan *List*, maka semua data pada *List* ke – 0 harus digeser satu langkah ke belakang (samping kanan) dan data 12 diletakkan pada kotak pertama (paling kiri) pada *Linked List* yang bersangkutan.

Proses penempatan data dengan cara penambahan data di belakang *List* sebenarnya hampir sama dengan cara penambahan data di depan *List*, hanya saja urutannya yang terbalik. Untuk lebih jelas dapat dilihat pada penjelasan di bawah ini,

1. Ukuran tabel untuk menyimpan *pointer* sebanyak 4 kotak dengan penomoran tabel dimulai dari 0.
2. Data ditambahkan satu per satu pada *List* dengan dimulai dari data pertama yaitu data 1.
   h(1) = 1 mod 4 = 1. Data 1 ditambahkan pada *List* ke – 1.
3. Proses dilanjutkan dengan menempatkan data ke – 2 yaitu data 4.
   h(4) = 4 mod 4 = 0. Data 4 ditambahkan pada *List* ke – 0.
4. Proses dilanjutkan dengan menempatkan data ke – 3 yaitu data 5.
   h(5) = 5 mod 4 = 1. Data 5 ditambahkan pada *List* ke – 1. Karena proses penambahan data dilakukan di belakang *List*, maka data pada *List* tidak perlu dilakukan perubahan / pergeseran.

Data 5 langsung ditempatkan pada urutan terakhir (paling kanan) pada *Linked List* yang bersangkutan.

5. Proses dilanjutkan dengan menempatkan data terakhir yaitu data 12 pada *List*.
   h(12) = 12 mod 4 = 0. Data 12 ditambahkan pada *List* ke – 0. Karena proses penambahan data dilakukan di belakang *List*, maka data pada *List* tidak perlu dilakukan perubahan / pergeseran. Data 12 langsung ditempatkan pada urutan terakhir (paling kanan) pada *Linked List* yang bersangkutan.

4.2.3 Proses Pencarian Data

Proses pencarian data pada metoda pencarian Relatif (*Hash Search*) menggunakan fungsi *hash* yang sama dengan proses penempatan data. Proses pencarian pada *Hash* Tertutup (*Close Hash*) untuk strategi Resolusi Linier (*Linear Resolution*) juga dilakukan dengan mencari nilai dari fungsi *hash* untuk data yang dicari. Pencarian akan dilakukan hingga data ditemukan atau menelusuri seluruh tabel dan data tidak ditemukan. Sebagai contoh, diambil contoh penempatan data pada strategi Resolusi Linier di atas. Misalkan data yang dicari adalah data 4, maka proses pencariannya adalah sebagai berikut, h(4) = 4 mod 5 = 4.

Proses pencarian dilakukan pada kotak ke – 4 pada tabel. Data pada kotak ke – 4 pada tabel yaitu data 4 sama dengan data yang dicari, sehingga data ditemukan dan proses pencarian dihentikan.

5	1	10		**4**
0	1	2	3	4

Proses pencarian data 4 dengan menggunakan metoda pencarian *Hash* Tertutup (*Close Hash*) dengan strategi Resolusi Linier melakukan perbandingan sebanyak 1 kali saja sehingga waktu akses dari data 4 adalah sebanyak 1 kali. Waktu akses untuk data lainnya dapat dilihat pada tabel berikut ini,

Tabel 1 Waktu akses metoda pencarian Hash Tertutup dengan Resolusi Linier

X	**1**	**4**	**5**	**10**
T	1	1	1	3

Waktu akses rata – rata dari metoda pencarian *Hash* Tertutup untuk pencarian SUKSES adalah sebesar $^{6}/_{4}$ = 1,5 dan kemungkinan terburuk adalah menelusuri seluruh tabel dan gagal.

Proses pencarian pada *Hash* Tertutup (*Close Hash*) untuk strategi *Overflow* juga dilakukan dengan mencari nilai dari fungsi *hash* untuk data yang dicari. Pencarian akan dilakukan hingga data ditemukan atau maksimal menelusuri seluruh tabel *overflow* dan data tidak ditemukan. Sebagai contoh, diambil contoh penempatan data pada strategi *Overflow* dengan penempatan data pada tabel *Overflow* menggunakan fungsi *hash* yang baru di atas. Misalkan data yang dicari adalah data 10, maka proses pencariannya adalah sebagai berikut,

h(10) = 10 mod 5 = 0. Proses pencarian dilakukan pada kotak ke – 0 pada tabel utama. Data pada kotak ke – 0 pada tabel utama yaitu data 5 tidak sama dengan data yang dicari yaitu data 10, sehingga proses pencarian dilanjutkan ke tabel *overflow*. g(10) = 10 mod 2 = 0. Proses pencarian dilakukan pada kotak ke – 0 pada tabel *overflow*. Data pada kotak ke – 0 pada tabel *overflow* yaitu data 10 sama dengan data yang dicari yaitu data 10, sehingga data ditemukan dan proses pencarian dihentikan.

Tabel utama

5	1			4
0	1	2	3	4

Tabel overflow

10	
0	1

Proses pencarian data 10 dengan menggunakan metoda pencarian *Hash* Tertutup (*Close Hash*) dengan strategi *Overflow* melakukan perbandingan sebanyak 2 kali yaitu terhadap data 5 dan data 10 sehingga waktu akses dari data 10 adalah sebanyak 2 kali. Waktu akses untuk data lainnya dapat dilihat pada tabel berikut ini,

Tabel 2 Waktu akses metoda pencarian Hash Tertutup dengan Overflow

X	**1**	**4**	**5**	**10**
T	1	1	1	2

Waktu akses rata – rata dari metoda pencarian *Hash* Tertutup untuk pencarian SUKSES adalah sebesar $^{5}/_{4}$ = 1,25 dan kemungkinan terburuk adalah menelusuri seluruh tabel *overflow* dan gagal.

Proses pencarian data untuk strategi *Double Hashing* sama persis dengan strategi *Overflow*. Proses pencarian pada *Hash* Tertutup (*Close Hash*) untuk strategi *Double Hashing* juga dilakukan dengan mencari nilai dari fungsi *hash* untuk data yang dicari. Pencarian akan dilakukan hingga data ditemukan atau maksimal menelusuri seluruh tabel *overflow* dan data tidak ditemukan. Sebagai contoh, diambil contoh penempatan data pada strategi *Double Hashing* dengan penempatan data pada tabel *Overflow* menggunakan fungsi *hash* yang baru di atas. Misalkan data yang dicari adalah data 10, maka proses pencariannya adalah sebagai berikut, h(10) = 10 mod 5 = 0. Proses pencarian dilakukan pada kotak ke – 0 pada tabel utama. Data pada kotak ke – 0 pada tabel utama yaitu data 5 tidak sama dengan data yang dicari yaitu data 10, sehingga proses pencarian dilanjutkan ke tabel *overflow*. g(10) = 10 mod 5 = 0. Proses pencarian dilakukan pada kotak ke – 0 pada tabel *overflow*. Data pada kotak ke – 0 pada tabel *overflow* yaitu data 10 sama dengan data yang dicari yaitu data 10, sehingga data ditemukan dan proses pencarian dihentikan.

Tabel utama

5	1			4
0	1	2	3	4

Tabel overflow

10				
0	1	2	3	4

Proses pencarian data 10 dengan menggunakan metoda pencarian *Hash* Tertutup (*Close Hash*) dengan strategi *Double Hashing* melakukan perbandingan sebanyak 2 kali

yaitu terhadap data 5 dan data 10 sehingga waktu akses dari data 10 adalah sebanyak 2 kali. Waktu akses untuk data lainnya dapat dilihat pada tabel berikut ini,

Tabel 3 Waktu akses metoda pencarian Hash Tertutup dengan Double Hashing

X	**1**	**4**	**5**	**10**
T	1	1	1	2

Waktu akses rata – rata dari metoda pencarian *Hash* Tertutup untuk pencarian SUKSES adalah sebesar $^5/_4$ = 1,25 dan kemungkinan terburuk adalah menelusuri seluruh tabel *overflow* dan gagal.

Proses pencarian data pada *Hash* Terbuka (*Open Hash*) juga dilakukan dengan mencari nilai dari fungsi *hash* untuk data yang dicari. Proses pencarian akan dilakukan pada *Linked List* dengan nomor *pointer* pada tabel yang sesuai dengan nilai dari fungsi *hash*. Pencarian akan dilakukan hingga data ditemukan atau sampai menelusuri seluruh *Linked List* dan data tidak ditemukan. Sebagai contoh, diambil contoh penempatan data dengan penambahan data di belakang *List* di atas. Misalkan data yang dicari adalah data 5, maka proses pencariannya adalah sebagai berikut,

h(5) = 5 mod 4 = 1. Proses pencarian akan dilakukan pada *List* ke – 1.

Periksa data pertama pada *List*, apakah sama dengan data dicari. Data pertama pada *List* merupakan data 1 tidak sama dengan data yang dicari yaitu data 5, sehingga proses pencarian dilanjutkan. Data kedua pada *List* berupa data 5 dan sama dengan data yang dicari yaitu data 5, sehingga data ditemukan dan proses pencarian dihentikan. Proses pencarian data 5 dengan menggunakan metoda pencarian *Open Hash* melakukan perbandingan sebanyak 2 kali yaitu terhadap data 1 dan data 5 sehingga waktu akses dari data 5 adalah sebanyak 2 kali. Waktu akses untuk data lainnya dapat dilihat pada tabel berikut ini,

Tabel 4 Waktu akses metoda pencarian Hash Terbuka

X	**1**	**4**	**5**	**12**
T	1	1	2	2

Waktu akses rata – rata dari metoda pencarian *Hash* Terbuka untuk pencarian SUKSES adalah sebesar $^6/_4$ = 1,5 dan kemungkinan terburuk adalah menelusuri satu *Linked List* dan gagal.

5. Simpulan

Kesimpulan yang didapat dari penelitian ini adalah:

1. Penelitian ini menggunakan 5 algoritma yaitu:
   1. Algoritma Pengecekan Data *Input* dan Tabel *Hash*.
   2. Algoritma Penempatan Data pada *Close Hash*, terdiri atas,
      a. Algoritma Penempatan Data pada *Close Hash* dengan Resolusi Linier.
      b. Algoritma Penempatan Data pada *Close Hash* dengan Resolusi *Overflow*.
      c. Algoritma Penempatan Data pada *Close Hash* dengan *Double Hashing*.
   3. Algoritma Penempatan Data pada *Open Hash*, terdiri atas,
      a. Algoritma Penempatan Data pada *Open Hash* dengan penambahan data di depan *list*.
      b. Algoritma Penempatan Data pada *Open Hash* dengan penambahan data di belakang *list*.
   4. Algoritma Pencarian Data pada *Close Hash*, terdiri atas,
      a. Algoritma Pencarian Data pada *Close Hash* dengan Resolusi Linier.
      b. Algoritma Pencarian Data pada *Close Hash* dengan Resolusi *Overflow*.
      c. Algoritma Pencarian Data pada *Close Hash* dengan *Double Hashing*.
   5. Algoritma Pencarian Data pada *Open Hash*, terdiri atas,
      a. Algoritma Pencarian Data pada *Open Hash* dengan penambahan data di depan *list*.
      b. Algoritma Pencarian Data pada *Open Hash* dengan penambahan data di belakang *list*.

2. Berdasarkan penelitian yang telah dilakukan bahwa metode pencarian Hash ini dapat melakukan operasi penyisipan, pencarian, pengeditan dan penghapusan, dan pencarian dengan cepat.

6. Referensi

Cormen, T. H., Leiserson, C. E. & Rivest, R. L., 1994. *Introduction to Algorithms*, McGraw-Hill, Inc., New York,

Sri Winarno , Sumardi, *2011*. Implementasi Perangkat Lunak Dengan Penerapan Pencarian Relatif *(Hash Search)*, Techno.COM, Vol. 10, No. 1, Pg. 7-14