IMPLEMENTASI PERANGKAT LUNAK DENGAN PENERAPAN PENCARIAN RELATIF *(HASH SEARCH)*

Sri Winarno , Sumardi
Program Studi Teknik Informatika, Fakultas Ilmu Komputer,
Universitas Dian Nuswantoro Semarang
Jl. Nakula I No. 5-11 Semarang
Telp : (024) 3517261, Fax : (024)3520165
E-mail : *wiwin@dosen.dinus.ac.id, sumardi@dosen.dinus.ac.id*

Abstract

Searching a list of values is a common task. An application program might retrieve a student record, bank account record, credit record, or any other type of record using a search algorithm. Some of the most common search algorithms are serial search, binary search and search by hashing. Search method (searching) there is a wide - range. Each - each method has advantages and disadvantages. One search method (searching), which has the efficiency of a better place is a relative search (Hash Search). Search relative (Hash Search) using the formula for doing the placement process and search data. Search relative (Hash Search) divided into 2 types, namely Hash Closed (Closed Hashing) and Hash Open (Open Hashing). On Closed Hash is possible more than one data has the same function value resulting in a collision (collision). How to cope with this collision can be done with some strategies, such as Linear Resolution (Linear Resolution), Overflow and Double Hashing.

Keywords: *hash search, linier resolusion, overflow, double hashing*.

1. PENDAHULUAN

Pencarian (*searching*) merupakan pekerjaan yang sering kita lakukan dalam kehidupan sehari – hari. Dalam *text editor*, kita sering melakukan pekerjaan mencari kata, atau mencari kata1 dan menggantikannya dengan kata-kata, atau mencari kata tertentu dan menghitung frekuensi kemunculan kata tersebut dalam dokumen. *Windows explorer* dan *internet explorer* menggunakan prinsip pencarian dalam melakukan pekerjaan tersebut.

Metoda pencarian (*searching*) ada bermacam – macam. Masing – masing metoda memiliki kelebihan dan kekurangan. 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.

Berdasarkan uraian di atas, penelitian bermaksud untuk merancang suatu perangkat lunak pembelajaran yang mampu melakukan analisa terhadap metoda pencarian relatif (*Hash Search*) dari data – data numerik yang di-*input*.

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*.

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*

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

Sedangkan proses penempatan data 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.

Tabel utama

0	1	2	3	4

Tabel overflow

0	1

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.

Tabel utama

	1			
0	1	2	3	4

Tabel overflow

0	1

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.

Tabel utama

	1			4
0	1	2	3	4

Tabel overflow

0	1

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.

Tabel utama

5	1			4
0	1	2	3	4

Tabel overflow

0	1

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*.

Tabel utama

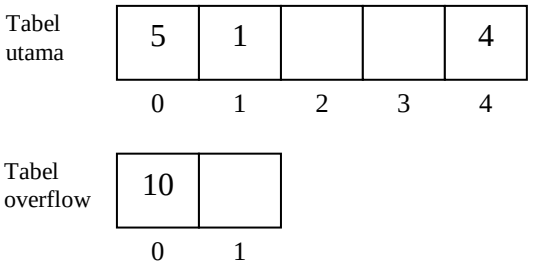
5	1			4
0	1	2	3	4

Tabel overflow

10	
0	1

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

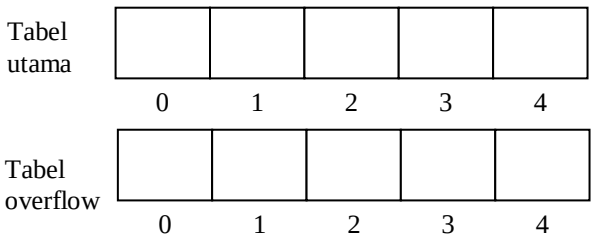
ditemukan kotak yang kosong. Jika tidak ada lagi kotak yang kosong, maka data ditolak.
Untuk contoh di atas, posisi data 10 pada tabel *overflow* dapat dicari dengan menggunakan rumus g(10) = 10 mod 2 = 0. Data 10 ditempatkan pada kotak ke – 0 pada tabel *overflow*.



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

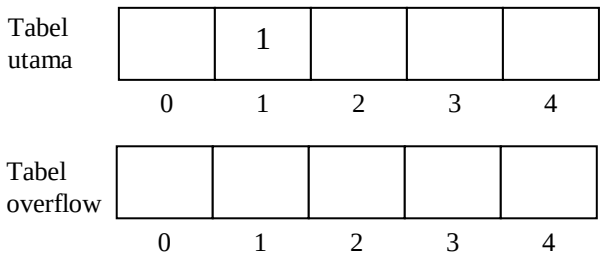
Proses penempatan data dengan strategi *Double Hashing* adalah sebagai berikut :

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.

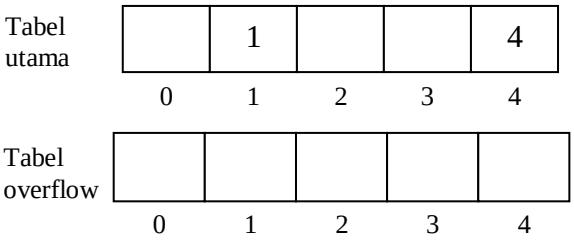


2. Data dimasukkan satu per satu ke dalam tabel dengan dimulai dari data pertama yaitu data 1.

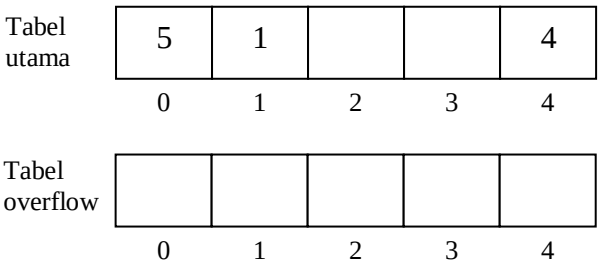
h(1) = 1 mod 5 = 1. Data 1 ditempatkan pada kotak ke – 1 pada tabel utama.



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.

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-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-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-data lainnya dapat dilihat pada tabel berikut ini,

Tabel 3. Waktu akses metoda pencarian Hash Tertutup dengan Double Hashing

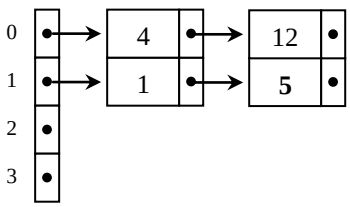
X	**1**	**4**	**5**	**10**
T	1	1	1	2

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-data lainnya dapat dilihat pada tabel berikut ini,

Tabel 4. Waktu akses metoda pencarian Hash Terbuka

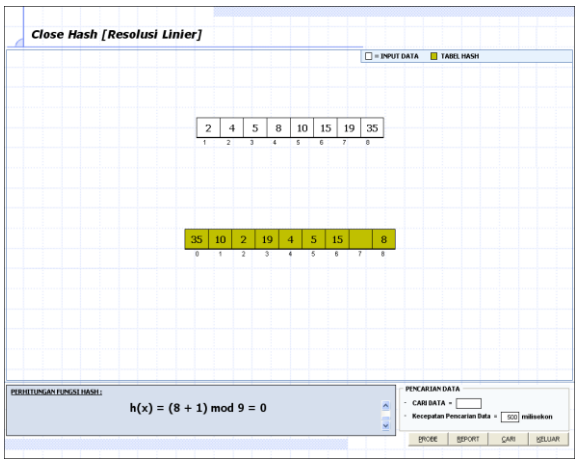
X	**1**	**4**	**5**	**12**
T	1	1	2	2

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.

3. IMPLEMENTASI

Sebagai contoh, penulis meng-*input* data sebagai berikut.

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



Gambar 1. Penempatan data pada close hash dengan resolusi linier dan ukuran tabel utama=9

Langkah – langkah yang dilakukan program:

PENEMPATAN DATA PADA 'Close Hash [Resolusi Linier]'

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

Data - 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'

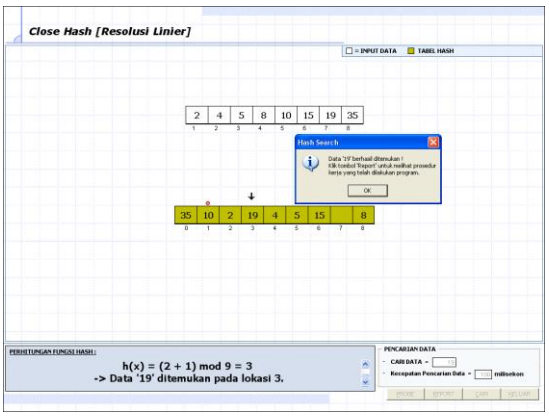
TABEL UTAMA→ h(x) = 10 mod 9 = 1, lokasi 1 kosong sehingga data '10' ditempatkan pada lokasi 1
---
6. Data '15'

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, didapat



Gambar 2. Hasil Pencarian Data ‘19’ pada tabel close hash dengan resolusi linier dan ukuran tabel utama = 9

Langkah – langkah yang dilakukan program :

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.

4. KESIMPULAN

Setelah menyelesaikan perancangan perangkat lunak pembelajaran *Hash Search*, kesimpulan dari penelitian ini adalah bahwa Perangkat lunak mampu menampilkan tahap-tahap penempatan dan pencarian secara langkah demi langkah dan *probe* untuk setiap data yang di-*input*. Hasil proses penempatan dan pencarian data dapat disimpan ke dalam bentuk *text file*, sehingga dapat dibuka dan dicetak dengan aplikasi lainnya seperti *Notepad*. Pada metoda *Closed Hash*, ukuran tabel utama minimal harus sama dengan jumlah data agar semua data dapat ditempatkan. Jika tabel *overflow* telah penuh, maka data yang masuk ke tabel *overflow* akan ditolak, walaupun tabel utama masih belum penuh.

5. DAFTAR PUSTAKA

Wilton, Rex (1996) *Mathematics for Computer Students.* 2nd ed. England: NCC Blackwell.

Munir, R, *Matematika Diskrit*, Vol 2, Penerbit Informatika Bandung, 2003.

Wikipedia (http://www.wikipedia.org/) entry on *Hash table*
*(*http://en.wikipedia.org/wiki/*Hash*_table).