Perbandingan Kinerja Berbagai Algoritma Fungsi Hash pada Algoritma Rabin-Karp

Rizki Alif Salman Alfarisy
Teknik Informatika / Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
Bandung, Indonesia
13516005@std.stei.itb.ac.id

***Abstrak* — Algoritma pencarian string merupakan salah satu algoritma yang paling umum digunakan di dunia dikarenakan kegunaannya yang banyak. Salah satu dari algoritma pencarian string yang terkenal adalah Rabin-Karp Algorithm yang memanfaatkan fungsi Hash di dalamny. Makalah ini dibuat untuk melakukan analisis untuk mendesain suatu fungsi hash yang cocok untuk digunakan di dalam Algoritma Rabin-Karp. Selain itu, makalah ini juga membahas beberapa algoritma yang sudah ada dan digunakan untuk algoritma Rabin-Karp dan melakukan analisis terhadap kinerjanya. Penulis berharap makalh ini dapat digunakan sebagai acuan untuk mengembangkan fungsi Hash yang dapat mengoptimasi Algoritma Rabin-Karp.**

***Kata Kunci* — *Hash*, Robin Karp, Kinerja, Perbandingan**

I. Pendahuluan

Fungsi hash adalah semuah fungsi yang dapat digunakan untuk memetakan data berukuran apapun menjada data dengan ukuran tertentu yang biasa disebut dengan nilai hash. Salah satu bentuk penggunaan fungsi hash yang umum digunakan adalah *hash* table, struktur data yang biasa diguankan untuk mengoptimasi pencarian data. Biasanya hash table digunakan untuk mencari *record* yang sama pada suatu basis data atu tabel dengan sangat cepat. Pada kriptografi, fungsi hash biasa digunakan untuk melakukan verifikasi tergadap integritas suatu data. Suatu data pasti memiliki nilai hash yang sama jika dimasukkan ke fungsi hash yang sama, oleh karena itu fungsi hash dapat digunakan untuk mendeteksi apakah terdapat perubahan terhadap data yang tersebar dibandingkan dengan data yang sebenarnya. Namun pada makalah ini, kegunaan fungsi hash yang akan dibahas adalah penggunaan fungsi hash untuk mengoptimasi beberapa algoritma pencarian, pada khususnya algoritma pencarian string. Algoritma pencarian yang memanfaatkan fungsi hash ini disebut dengan Algoritma Rabin-Karp.

Algoritma Rabin-Karp adalah salah satu dari algoritma pencarian string yang umum digunakan. Algoritma ini memanfaatkan fungsi hash untuk membandingkan nilai hash dari pattern yang dicari dengan nilai hash dari substring-substring yang terdapat di dalam teks sumber. Substring yang memiliki nilai hash yang sama dengan pattern merupakan substring jawabannya. Salah satu penggunaan algoritma ini yang umum digunakan adalah pendeteksian plagiarisme.

```
    01234567890123456789012345678
S = HACKHACKHACKHACKITHACKEREARTH
P =     HACKHACKIT
P =         HACKHACKIT...[match!]
P =                     HACKHACKIT
```

Gambar 1 Contoh Algoritma Pencarian String

Pada saat ini, ada banyak sekali fungsi hash yang dikembangkan oleh berbagai pihak di dunia. Setiap fungsi hash tersebut memilki kinerja dan cara kerjanya masing-masing. Pada makalah ini, penulis akan melakukan analisis dan perbandingan beberapa fungsi hash dalam kinerjanya di algoritma Robin-Karp yang telah dijelaskan di atas. Analisis dilakukan utnuk memahami fungsi hash yang paling tepat untuk digunakan pada algortima Robin-Karp. Kemudian perbandingan beberapa fungsi hash yang dilakukan diharapkan dapat menjadi acuan untuk pengembangan fungsi hash yang dapat meningkatkan kinerja dari algoritma Robin-Karp dengan lebih baik lagi,

II. Dasar Teori

A. Algoritma Pencarian String

Pencocokan String atau yang biasa disebut String Matching merupakan algoritma yang digunakan untuk mencari kemunculan sebuah string (pattern) tertentu pada sebuah teks yang lebih besar dari pattern tersebut. Berikut contoh dari pencarian string :

Pattern : ini
Teks : Olahraga harus dilakukan sejak d**ini** untuk mem**ini**malisir kemungkinan munculnya penyakit saat tua nanti.
Jumlah pattern yang ditemukan : 2

Banyak algoritma string matching yang telah ditemukan dan dikembangkan oleh orang-orang. Jenis-jenis dari algoritma string matching ini dapat diklasifikasikan berdasarkan berbagai klasifikasi, salah satunya yaitu berdasarkan strategi yang digunakan. Berdasarkan strategi yang digunakan, stringnmatching dapat diklasifikasikan menjadi:

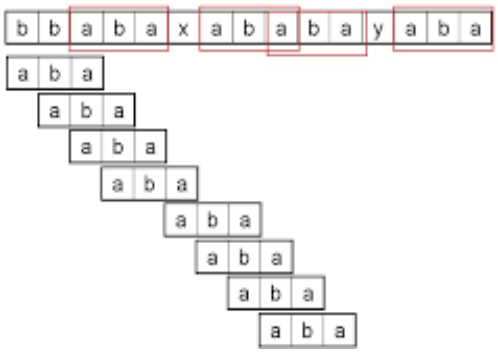
1. From left to right : Pencarian dilakukan dengan mencocokkan karakter dari kiri ke kanan dimulai dari awal teks

2. From right to left : Pencarian dilakukan dengan mencocokkan karakter dari kanan ke kiri dimulai dari akhir teks

3. In a specific order : Pencarian dilakukan dengan membuat aturan tertentu, misal mencari dari kedua ujung teks secara bersamaan.

4. In any order : Tidak ada aturan khusus mengenai aturan pencarian di dalam teks.

Algoritma paling dasar dari pencarian string biasa disebut *naive string matching*, algoritma ini melakukan pencocokan string dengan cara melakukan pencocokan karakter setiap karakter di teks dengan karakter di pattern. Pencocokan di mulai dari awal teks hingga sepanjang panjang pattern atau hingga ditemukan ketidak cocokan. Apabila algoritma berhenti karena ditemuka kecocokan hingga panjang pattern maka pattern terletak di indeks tersebut. Jika tidak, maka algoritma akan memulai kembali pencocokan dimulai dari karakter betikutnya. Hal ini terus diulang hingga algoritma menemukan pattern atau sudah mencapai penghujung teks. Kompleksitas algoritma ini adalah $O(n^2)$



Gambar 2 *Naive string matching*

B. *Fungsi Hash*

Fungsi hash adalah fungsi yang menerima masukan sembarang string dengan panjang bebas dan menghasilkan sebuah string dengan panjang yang selalu sama yang biasa disebut dengan *hash value*. Pada umumnya, pesan yang dimasukkan ke dalam fungsi hash jauh lebih panjang dibandingkan dengan *hash value*. Fungsi hash biasanya dinotasikan dalam bentuk persamaan h = H(M).

Berikut adalah syarat-syarat suatu fungsi H dapat dikatakan sebagai fungsi hash yang baik :

1. Fungsi H dapat diterapkan pada blok data berukuran berapa saja.
2. H menghasilkan nilai h dengan panjang yang selalu sama.
3. H(x) mudah dihitung untuk setiap nilai x yang diberikan.
4. Untuk setiap h yang dihasilkan, tidak mungkin dikembalikan nilai x sedemikian hingga H(x) = h.
5. Untuk setiap x yang diberikan, tidak mungkin mencari y , x sedemikian sehingga H(y) = H(x).
6. Tidak mungkin mencari pasangan x dan y sedemikian sehingga H(x) = H(y).

Syarat yang pertama dan kedua merupakan syarat wajib untuk seluruh fungsi hash, kedua syarat tersebut merupakan definisi fungsi hash. Sedangkan syarat keempat biasa disebut dengan *pre-image* resistance, suatu hash value tidak boleh diketahui nilai sebelum dimasukkan ke fungsi hash-nya, hal ini cukuplah penting untuk fungsi hash dalam kriptografi. Pada kenyataanya, nilai sebelum fungsi hash dapat dicari dengan menggunakan *brute force*, namun funsgi hash yang baik harusnya memastikan bahwa dengan *brute force* waktu yang akan dibutuhkan sangatlah lama. Terakhir, syarat yang kelima dan keenam disebut dengan *non-collision*, yaitu tidak boleh ada 2 nilai berbeda yang jika dimasukkan ke fungsi hash menghasilkan nilai yang sama. Pada kenyataanya, syarat ini cukup sulit untuk dicapai dikarenakan beberapa fungsi hash lama masih memiliki kolisi di dalamnya. Namun, beberapa fungsi hash yang baru-baru ini dikembangkan sudah berhasil mengurangi jumlah collision sebanyank mungkin.

C. *Algoritma Rabin-Karp dan Rolling Hash*

Algoritma Rabin-Karp adalah algoritma pencarian string yang dibuat oleh Richard M. Karp dan Michael O. Rabin pada tahun 1987. Algoritma ini memanfaatkan fungsi hash untuk mencari susbtring tertentu di dalam teks. Algoritma ini berfungsi dengan cara mencocokkan nilai hash dari pattern dengan nilai hash substring di teks, substring yang memiliki nilai has sama dengan nilai hash pattern diasumsikan sama dengan pattern. Untuk teks dengan panjang n dan pattern dengan panjang m, kasus terbaik dari algoritma ini adalah O(n+m), sedangkan kasus terburuknya adalah O(nm).

Pada algoritma rabin-karp, fungsi hash yang digunakan untuk mengoptimalkan algoritma ini bukanlah fungsi hash sembarangan tetapi fungsi yang disebut dengan *rolling hash*. *Rolling hash* adalah fungsi yang dapat menghitung nilai hash sebuah string yang mirip dengan cepat. Hal ini dapat dimungkinkan dikarenakan rolling hash menggunakan prinsip yang mirip dengan *sliding window*. Ketika nilai hash suatu window telah dihitung, maka window akan bergeser. Window yang bergeser menyebabkan ada karakter yang terbuang dan ada juga karakter yang baru muncul pada jendela. *Rolling hash* mampu mengkomputasi nilai hash baru dengan mengubah

nilai hash lama, yaitu dengan ‘membuang’ karakter lama dan ‘menambah’ karakter baru.

$$H(\text{“}bcd\text{”}) = H(\text{“}abc\text{”}) - H(\text{“}a\text{”}) + H(\text{“}d\text{”})$$

Gambar 3 Rumus umum Rolling Hash

Rolling Hash biasanya hanya melibatkan penjumlahan dan perkalian di dalam perhitungannya untuk mempermudah penghapusan dan penambahan nilai baru. Akan tetapi, karena fungsi hash yang dibuat sederhana tersebut banyak terjadi kemungkinan kolisi pada hasil fungsi hash. Untuk mencegah kesalahan pencarian string karena hal tersebut, algoritma rabin-karp melakukan pengcekan ulang pattern dengan cara naif apabila berhasil menemukan pattern dengan nilai hash yang sama.

Pada makalah ini, penulis menggunakan beberapa alternatif dari Rolling Hash, alternatif tersebut adalah :

1. Rolling Hash Rabin-Karp

   Rolling Hash ini disebut juga dengan Rolling Hash Polynomial. Pada mode ini nilai c diisi dengan orde dari karakter yang akan dihash misal untuk “abc”, c1 bernilai 1, c2 bernilai 2, dan c3 bernilai 3. Dan a merupakan jumlah kemungkinan karakter, misal untuk alfabet maka digunakan angka 26. Penggeseran window akan menyebabkan penghapusan nilai sebelumnya dengan cara melakukan pembagian.

$$H = c_1 a^{k-1} + c_2 a^{k-2} + c_3 a^{k-3} + \cdots + c_k a^0$$

Gambar 4 Rumus Polinomial Rabin-Karp Rolling Hash

2. Rolling Hash berbasis modulo

   Rolling hash ini hanya memanfaatkan perkalian dan modulo utnuk menghitung nilai hash. Apabila terjadi penggeseran maka nilai baru akan ditambahkan dnegan mengalikan orde dari karakter baru dengan base (jumlah karakter yang mungkin) dan membaginya dengan modulo. Jika ada karakter yang dihilangkan, hal yang sebaliknya akan dilakukan

3. Rolling Hash berbasis Perfect Hash

   Rolling Hash ini menggunakan hash sederhana namun memastikan tidak adanya kolisi dengan cara memetakan semua hasil yang mungkin dengan jumlah input yang mungkin. Pada kasus ini, angka yang digunakn adalah 26. Dengan perfect hash, algoritma tidak perlu melakukan pengecekan ulang kesamaan string dengan naive string matching karena tidak ada kemungkinan kolisi.

III. PENGUJIAN

A. *Rancangan Pengujian*

Pengujian dilakukan terhadap berbagai jenis rolling hash yang telah dibahas sebelumnya, non-cryptographic hash dengan waktu eksekusi yang sangat cepat, dan *naive string matching*. Hash sederhana non-cryptograpghic digunakan sebagai perbandingan apakah rolling hash lebih baik dibandingkan melakukan hashing sederhana berulang kali. Kemudian *naive string matching* digunakan sebagai batas bawah dari performa pencarian string. Metode yang lebih lambat dibandingkan dengan *naive string* matching tidak layak digunakan di kehidupan sehari-hari. Pengujian dilakukan dengan cara memmbandingkan kecepatan hashing dari berbagai metode tersebut dengan menggunakan program python. Kasus uji yang digunakan berupa random teks dalam berbagai ukuran dan pattern yang dicari juga dipilih secara random. Teks dibatasi untuk uppercase saja untuk memperkecil jumlah karakter sehingga memperbanyak jumlah substring yang mirip.

Berikut kode yang digunakan untuk melakukan pengujian.

```
import time
from random import choice, randint, seed
from string import ascii_uppercase


class RollingHash:
def __init__(self, text, sizeWord):
self.text = text
self.hash = 0
self.sizeWord = sizeWord

for i in range(0, sizeWord):
#ord maps the character to a number
#HASH INITIALIZATION
self.hash += (ord(self.text[i]) - ord("a")
+1)*(26**(sizeWord - i -1))

#start index of current window
self.window_start = 0
#end of index window
self.window_end = sizeWord

def move_window(self):
if self.window_end <= len(self.text) - 1:
#HASH VALUE CHANGE HERE
self.hash -= (ord(self.text[self.window_start]) -
ord("a")+1)*26**(self.sizeWord-1)
self.hash *= 26
self.hash += ord(self.text[self.window_end])- ord("a")
```

```
+1
self.window_start += 1
self.window_end += 1

def window_text(self):
return self.text[self.window_start:self.window_end]

def rabin_karp(word, text):
if word == "" or text == "":
return None
if len(word) > len(text):
return None

rolling_hash = RollingHash(text, len(word))
word_hash = RollingHash(word, len(word))
#word_hash.move_window()

for i in range(len(text) - len(word) + 1):
if rolling_hash.hash == word_hash.hash:
if rolling_hash.window_text() == word:
return i
rolling_hash.move_window()
return None

seed(13516005)
text = ''.join(choice(ascii_uppercase) for i in
range(100))
word_length = 10
word_index = randint(0, len(text)- word_length)
word = text[word_index:word_index+word_length]
t1 = time.clock()
results = rabin_karp(word, text)
t2 = time.clock()
print(text)
print(word)
print(results)
print('Run Time: %0.5f ms' % ((t2 - t1) * 1000.0))
```

Pada kode terebut terdapat bagian kode Rolling Hash yang dapat digunakan untuk mengganti implmentasi rolling hash yang digunakan. Selain itu, algoritma yang digunakan dapat diganti di bagian main dari program untuk melakukan pengujian *naive string matching* Teks yang dirandom dan substring yang dipilih dipastikan ama karena menggunakan seed yang sama.

IV. Analisis

A. Hasil Pengujian

Berikut hasil pengujian untuk berbagai metode yang digunakan

Rabin-Karp Rolling Hash

Panjang Teks	Panjang pattern	Waktu
500	50	0.68800 ms
	100	0.13600 ms
	300	1.25400 ms
5000	50	3.02600 ms
	100	6.21500 ms
	300	13.55900 ms
100000	50	78.46300 ms
	100	179.82200 ms
	300	178.55700 ms

Rolling Hash Berbasis Modulo

Panjang Teks	Panjang pattern	Waktu
500	50	0.32400 ms
	100	0.11900 ms
	300	0.29500 ms
5000	50	1.12900 ms
	100	1.75500 ms
	300	2.33600 ms
10000	50	25.93700 ms
	100	57.48900 ms
	300	24.31000 ms

Rolling Hash Berbasis Perfect hash

Panjang Teks	Panjang pattern	Waktu
500	50	0.58000 ms
	100	0.10000 ms
	300	0.64200 ms
5000	50	2.46100 ms
	100	4.11800 ms
	300	11.17600 ms
10000	50	49.26700 ms
	100	115.71500 ms
	300	136.43000 ms

Naive Hash String Matching dengan Adler 32

Panjang Teks	Panjang pattern	Waktu
500	50	0.25400 ms
	100	0.03500 ms
	300	0.13200 ms
5000	50	2.09700 ms
	100	2.96900 ms
	300	1.94100 ms
10000	50	27.23900 ms
	100	53.08500 ms
	300	38.36200 ms

Naive String Matching

Panjang Teks	Panjang pattern	Waktu
500	50	0.76900 ms
	100	0.14500 ms
	300	1.27600 ms
5000	50	2.09700 ms
	100	3.45200 ms
	300	5.72400 ms
10000	50	14.21200 ms
	100	83.70200 ms
	300	182.79600 ms

B. Analisis

Hasil pengujian yang didapatkan penulis beragam dan cukup mengejutkan. Hal ini disebabkan sebagian besar dari Rolling Hash Algorithm ternyata lebih lambat dari Naive Hash String Matching dan bahkan oleh algoritma Naive String Matching. Kedua algoritma yang memiliki waktu eksekusi paling buruk adalah rolling hash berbasis perfect hash dan Rollong hash buatan rabin-karp. Kinerja yang buruk ini kemungkinan besar disebabkan oleh persamaan dari implmentasi kedua algoritma itu, yaitu menggunakan operasi perpangkatan. Operasi perpangkatan yang dilakukan pun makin banyak jika panjang pattern semakin banyak, menyebabkan waktu eksekusi yang berbanding lurus dengan panjang pattern.

Naive Hash String Matching dapat mencapai eksekusi yang cukup cepat dikarenakan algoritma menggunakan fungsi hash paling sederhana walaupun memungkinkan banyak kolisi. Perbandingan string tambahan untuk mengecek kolisi ternyata tidak mempengaruhi kinerja begitu banyak. Untuk Naive String Matching, algoritma ini memiliki keuntungan apabila panjang pattern tidak terlalu besar. Namun dapat dilihat bahwa di panjang pattern 300, algoritma ini memiliki waktu eksekusi terlama dibandingkan yang lain. Hal ini membuktikan fungsi hash yang memiliki kinerja sama dengan input apapun berperan baik pada Algoritma Rabin-Karp.

Berdasarkan hasil pengujian, ternyata rolling Hash yang paling baik berfungsi adalah Rolling hash berbasis modulo. Analisi yang penulis dapatkan adalah algoritma ini unggul dikarenakan hanya melakukan operasi perkalian dan modulo dalam seluruh algoritmanya.

V. KESIMPULAN DAN SARAN

Kesimpulan yang penulis dapatkan adalah penggunaan fungsi hash dalam algoritma rabin-karb terbukti dapat mempercepat pencarian string matching dikarenakan algoritma tidak lagi tergantung pada panjang pattern. Namun *cost* yang harus diperhatikan adalah *cost* penghitungan fungsi hash itu sendiri. Beberapa alternatif rolling hash memiliki kinerja yang kurang baik karena banyak menggunakan operasi perpangkatan. Sedangkan rolling hash yang menggunakan operasi sederhana terbukti menjadi lebih cepat.

Saran dari pengguna mengenai pengembangan fungsi hash untuk algorima Rabin-Karp adalah pengembang funsgi hash harus sangat memperhatikan *cost* dari penghitungan hash. Selain itu, jangan takut menggunakan algoritma yang memiliki banyak kolisi karena penghitungan ulang menggunakan *naive string matching* untuk mendeteksi *false positive* tidak berdampak terlalu besar.

Untuk peneliti lainnya yang akan melakukan pengujian terhadap perbandingan fungsi Hash. Penulis meyarankan utnuk menyiapkan kasus uji yang dapat mensiumalsikan terjadinya kolisi di dalam teks yang dibuat. Hal ini dapat digunakan untuk mengukur dampak kolisi ke algoritma *rolling hash*.

UCAPAN TERIMA KASIH

Pertama-tama, saya mengucapkan syukur dan terima kasih kepada Tuhan Yang Maha Esa yang telah melimpahkan berkatnya selama pengerjaan makalah ini hingga selesai. Saya juga berterima kasih kepada teman-teman saya yang membantu memberikan inspirasi kepada saya untuk mengerjakan tugas ini.Saya juga ingin mengucapkan terima kasih kepada bapak Rinaldi Munir selaku dosen pengajar IF4020 Kriptografi yang telah memberikan materi mata kuliah kriptografi yang membantu dalam proses penyelesaian makalah ini.

REFERENSI

[1] Karp, Richard M.; Rabin, Michael O. (March 1987). "Efficient randomized pattern-matching algorithms". IBM Journal of Research and Development. 31 (2): 249–260. CiteSeerX 10.1.1.86.9502. doi:10.1147/rd.312.0249.

[2] Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford (2001-09-01) [1990]. "The Rabin–Karp algorithm". Introduction to Algorithms (2nd ed.). Cambridge, Massachusetts: MIT Press. pp. 911–916. ISBN 978-0-262-03293-3.

[3] https://www.strchr.com/hash_functions, diakses pada 9 Mei 2019, 19.20

[4] https://codereview.stackexchange.com/questions/178865/rolling-hash-algorithm, diakses pada 8 Mei 2019, 22.34

[5] https://brilliant.org/wiki/rabin-karp-algorithm/, diakses pada 8 Mei 2019, 20.43

[6] http://informatika.stei.itb.ac.id/~rinaldi.munir/, diakses ada 8 Mei 2019, 14.04

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang kami tulis ini adalah tulisan kami sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 29 April 2012

Rizki Alif S.A
13516005