Pencocokan Pola Majemuk Dengan Algoritma Rabin - Karp

Edwin Wijaya - 13513040
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
edwinwijaya94@yahoo.com

***Abstrak*—Pencocokan pola atau pencocokan string merupakan salah satu permasalahan dalam dunia komputasi. Setiap algoritma pencocokan string memiliki teknik / metode sendiri dalam menyelesaikan permasalahan. Jumlah pola yang dicocokan pada permasalahan tertentu juga bisa lebih dari satu. Pada makalah ini, akan dibahas salah satu algoritma pencocokan pola yaitu algoritma Rabin – Karp yang bisa menjadi salah satu solusi pada pencocokan pola majemuk.**

***Kata kunci*—pencocokan pola, hashing, rolling hash, algoritma Rabin – Karp.**

I. PENDAHULUAN

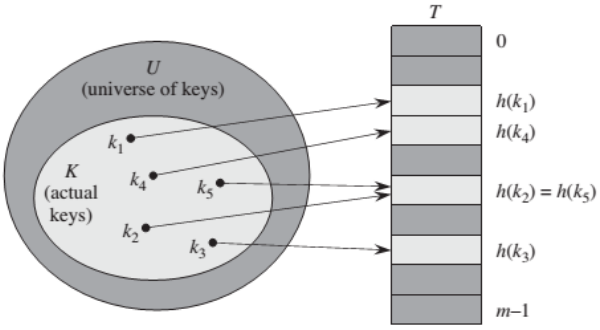
Dalam kehidupan sehari – hari ataupun dalam dunia komputasi, pencocokan pola atau string merupakan salah satu permasalahan yang sudah ada sejak dulu. Sudah banyak pula algoritma pencocokan pola string yang ditemukan. Mulai dari algoritma paling natural yaitu bruteforce hingga algoritma yang menggunakan pre-processing untuk mempercepat proses pencocokan. Sebagian besar algoritma pencocokan pola menggunakan *finite automata* untuk membantu proses pencocokan.

Pencocokan pola yang dilakukan tidak selalu tunggal, namun memungkinkan terjadinya pencocokan dengan jumlah pola lebih dari satu (*multiple pattern matching*). Pada makalah ini, akan dibahas salah satu algoritma pencocokan pola /string yaitu algoritma Rabin – Karp yang menggunakan hashing sebagai metode utamanya untuk menyelesaikan permasalahan. Hal ini juga yang membuatnya menjadi salah satu solusi dalam pencocokan pola majemuk.

II. DASAR TEORI

A. Fungsi Hash

Fungsi hash atau biasa disebut *hash function* adalah fungsi yang memetakan sekumpulan data yang ukurannya besar menjadi sekumpulan data yang ukurannya lebih kecil. Hasil dari fungsi hash disebut nilai hash (*hash value*) yang disimpan ke dalam sebuah tabel hash. Dengan menggunakan fungsi hash, elemen ke –k dalam sebuah array disimpan pada slot ke – h(k) dengan h(k) sebagai fungsi hash, berbeda dengan *direct addressing table* yang menyimpan elemen ke –k pada slot ke – k. Salah satu permasalahan yang dihadapi dengan menggunakan fungsi hash adalah terjadinya *collision* yaitu ketika 2 nilai kunci berbeda dipetakan ke nilai hash yang sama. Untuk menghindari *collision* diperlukan pemilihan fungsi hash yang tepat. Salah satu caranya adalah dengan membuat fungsi h(k) sebagai fungsi random sehingga mengurangi jumlah *collision*. Pada makalah ini kita akan menggunakan fungsi hash yang menggunakan bilangan prima sebagai salah satu base nya untuk mengurangi terjadinya pemetaan substring berbeda ke nilai hash yang sama (*collision*). Jika pattern yang dicari cukup panjang, maka akan dilakukan modulus terhadap fungsi hash agar nilai hash tidak terlalu besar.



Gambar 1 – Hashing dan Collision
Sumber : Introduction to Algorithms 3ed.

B. Rolling Hash

Rolling hash adalah sebuah metode yang digunakan pada algoritma Rabin – Karp dalam menentukan nilai hash dari setiap substring yang ada di teks dengan menggunakan nilai hash dari substring sebelumnya sehingga proses *hashing* untuk setiap substring yang akan diperiksa dapat dilakukan dalam kompleksitas O(1). Misalkan kita memiliki string "abracadabra" dan akan memetakan semua substring yang panjangnya 3 karakter menggunakan fungsi hash.

Substring pertama pada string adalah “abr”. Pada contoh ini kita menggunakan fungsi hash berdasarkan aturan Horner sebagai berikut :

H(j) = (j[0] * basis^(M-1)) + (j[1] * basis^(M-2)) + … + (j[M-1] * basis^0)

Dengan j sebagai array yang berisi pemetaan setiap karakter dalam substring “abr” menjadi nilai ASCII (a=97, b=98, r=114) , basis = 101, dan M adalah panjang substring,

Nilai H(j) untuk substring “abr” :

```
// ASCII a = 97, b = 98, r = 114.
hash("abr") = (97 × 101^2) + (98 × 101^1) + (114 × 101^0) = 999,509
```

Menggunakan nilai H(j) untuk “abr”, kita bisa menghitung substring berikutnya yaitu “bra” dengan rumus :
H(k) = (basis * (H(j) - (j[0] * basis^(M-1)))) + (k[M-1] * basis^0)

Dengan k sebagai array yang berisi pemetaan setiap karakter dalam substring “bra” menjadi nilai ASCII (b=98, r=114, a=97) , basis = 101, dan M adalah panjang substring,

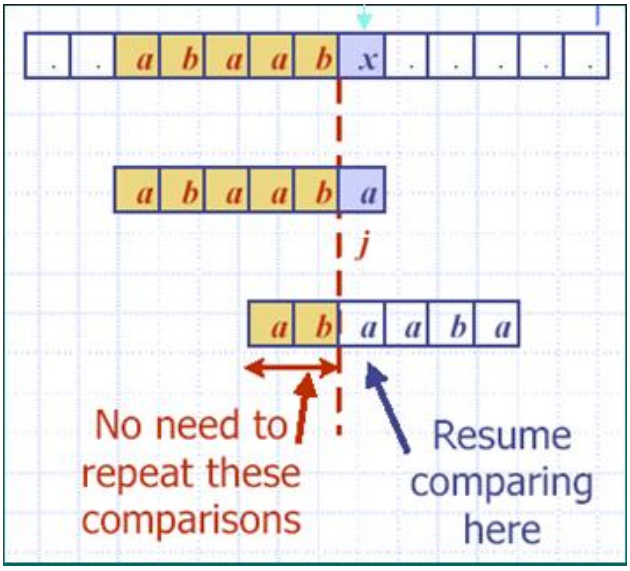
Nilai H(k) untuk substring “bra”:

```
hash("bra") = [101 × (999,509 - (97 × 101^2))] + (97 × 101^0) = 1,011,309
```

Menggunakan rolling hash, nilai hash dari sebuah substring pada teks dapat dihitung dari nilai hash substring sebelumnya dengan kompleksitas O(1).

C. Pencocokan String

Pencocokan string (*string matching*) atau yang lebih umum disebut sebagai pencocokan pola (*pattern matching*) adalah upaya untuk mencari pola (dalam hal ini sebuah substring) pada suatu teks / string. Jika ditemukan, maka pencarian akan mengembalikan indeks ditemukannya pola yang sesuai. Variasi dari algoritma pencocokan string banyak sekali. Salah satu algoritma pencocokan string yang paling sederhana adalah brute-force (*naïve string matching algorithm*) dengan membandingkan satu persatu karakter pada string dengan karakter pada pattern. Jika terjadi *mismatch*, maka pattern akan digeser satu karakter ke kanan, kemudian pencocokan diulang dengan membandingkan ulang karakter pertama pada pattern dengan karakter ke –i pada string. Hal ini menyebabkan pencocokan karakter yang sama dilakukan berulang –ulang. Kompleksitas waktu terburuk dari bruteforce adalah O((n-m+1)m). Walaupun algoritma bruteforce masih cukup cepat untuk string yang mempunyai alphabet yang banyak seperti a-z, 0-9, bruteforce akan buruk untuk memeriksa string biner yang hanya memiliki alphabet {0,1}. Oleh karena itu diperlukan algoritma pencocokan string yang lebih cepat antara lain algoritma Knuth-Morris-Pratt (KMP) dan algoritma Boyer-Moore yang dapat berjalan dalam kompleksitas waktu rata rata O(m+n). Algoritma KMP dan Boyer-Moore berfokus agar pengecekan karakter yang berulang seminimal mungkin sehingga dapat menghemat waktu pencocokan. Kebanyakan algoritma pencocokan string menggunakan *finite automata* dalam melakukan pencocokan pola. Pada makalah ini akan dibahas salah satu algoritma pencocokan string yaitu algoritma Rabin-Karp yang menggunakan *hashing* agar. Walaupun algoritma Rabin-Karp masih menggeser pattern satu persatu karakter ke kanan seperti bruteforce, dengan adanya hashing proses perbandingan karakter bisa lebih cepat dilakukan. Algoritma Rabin-Karp sering digunakan pada pencocokan pola majemuk (*multiple pattern matching*). Pembahasan lebih detail mengenai algoritma Rabin – Karp akan ada di bab selanjutnya.



Gambar 2 – Ilustrasi algoritma KMP
Sumber : slide kuliah

III. ALGORITMA RABIN – KARP

A. Ide Dasar

Berbeda dengan algoritma pencocokan string lainnya yang menggunakan *finite automata* atau

regular expression untuk mempercepat proses pencocokan dengan cara mengurangi pengecekan karakter yang sama secara berulang, algoritma Rabin – Karp menggunakan fungsi hash untuk mempercepat perbandingan pola dengan string yang akan diperiksa. Setiap substring pada teks yang panjangnya sama dengan pola akan dipetakan oleh fungsi hash. Hal serupa dilakukan pada pola dengan memetakan pola menggunakan fungsi hash yang sama.

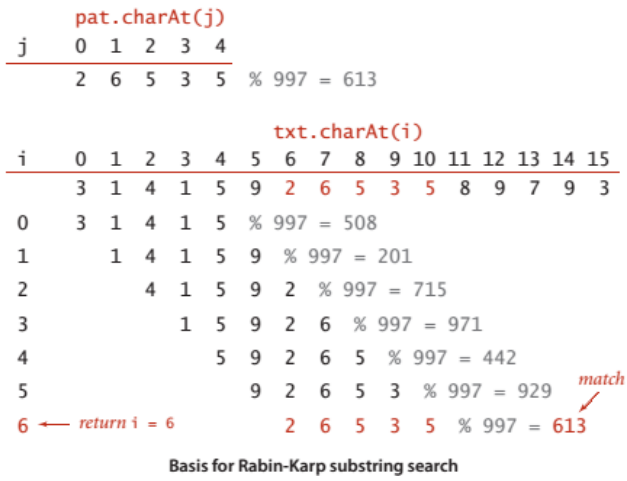
Dengan menggunakan fungsi hash, panjang string yang akan diperiksa umumnya akan lebih pendek, bergantung pada panjang pola yang akan dicari. Hal ini sama saja dengan menyimpan semua substring pada string ke dalam tabel hash, hanya saja kita tidak memerlukan memori untuk tabel hash karena kita hanya perlu memeriksa satu substring pada setiap pencocokan. Proses pencocokan dilakukan dengan cara membandingkan nilai hash dari setiap substring dengan nilai hash dari pola. Jika nilai hash ditemukan sama, maka pencocokan per karakter dilakukan, sama seperti pada algoritma bruteforce. Pencocokan per karakter ini perlu dilakukan untuk memastikan nilai hash yang sama bukan karena terjadinya *collision* atau sering disebut juga sebagai *spurious hit*. Jika terjadi *spurious hit* maka akan dilakukan pembandingan nilai hash untuk substring berikutnya.

Implementasi algoritma ini secara langsung tidak akan menghasilkan waktu komputasi yang lebih cepat dari algoritma bruteforce karena menghitung nilai hash dari setiap substring membutuhkan *cost* yang lebih mahal daripada perbandingan langsung per karakter seperti pada algoritma bruteforce. Akan tetapi Rabin dan Karp berhasil membuktikan bahwa untuk menghitung nilai hash dari setiap substring dapat dilakukan dalam waktu linear. Hal ini dilakukan akan dijelaskan pada sub-bab berikutnya.

B. Menghitung Nilai Hash

Misalkan kita memiliki string dengan panjang M dan kita akan menggunakan fungsi hash dengan base-R =10 serta tabel hash dengan ukuran Q. Untuk mendapatkan nilai hash yang berada di antara 0 dan Q-1, kita dapat menggunakan *modular hashing* yaitu dengan cara mengambil sisa pembagian (modulus) nilai hash dengan Q. Pada umumnya kita dapat memilih bilangan acak prima Q yang cukup besar untuk menghindari *collision*.

Berikut contohnya : Akan dicari pola 2 6 5 3 5 pada string 3 1 4 1 5 9 2 6 5 3 5 8 9 7 9 3, nilai Q yang digunakan adala 997, hitung nilai hash dari pola = 26535 % 997 = 613. Bandingkan nilai hash dari pola dengan masing – masing nilai hash dari setiap substring yaitu 508, 201, 715, 971, 442, dan 929, sebelum akhirnya kita menemukan nilai hash 613 pada string yang sama denga nilai hash milik pola. Ilustrasi yang lebih jelas dapat dilihat pada gambar berikut :



Basis for Rabin-Karp substring search

Gambar 3 – Ilustrasi hashing pada algoritma Rabin - Karp
Sumber : Algorithms 4th ed.

Fungsi hash yang dibentuk pada contoh di atas akan terlihat cukup mudah untuk pola yang tidak terlalu panjang karena nilai hash masih berupa integer. Akan tetapi prnghitungan nilai hash ke dalam integer tidak mungkin dilakukan pada pola yang memiliki panjang 100 atau 1000 misalnya. Hal ini diatasi dengan penggunaan aturan Horner yang menghitung nilai hash per karakter pada sebuah substring. Untuk setiap karakter atau digit pada substring, kalikan dengan bilangan basis R, tambahkan dengan nilai hash digit berikutnya. Lalu pada akhir penjumlahan, hitung dengan modulus Q.

Algoritma Rabin – Karp memiliki kelebihan dalam menghitung nilai hash untuk substring ke i+1 dengan cara memanfaatkan nilai hash untuk substring ke –i. Hal ini didasari oleh rumus matematis yang memanfaatkan aturan Horner seperti berikut :

$$x_i = t_i R^{M-1} + t_{i+1} R^{M-2} + \ldots + t_{i+M-1} R^0$$

Keterangan :
X_i = nilai hash substring ke – i
t_i = karakter ke – i pada string
M = panjang substring (sama dengan panjang pola)
R = bilangan basis

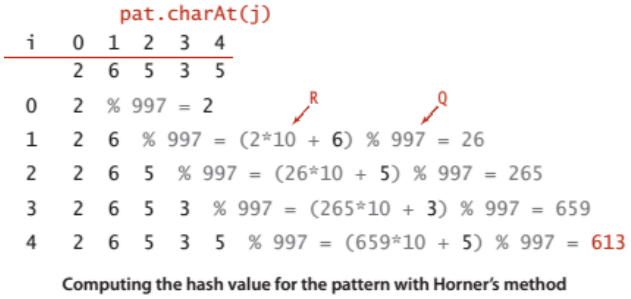
Nilai hash substring berikutnya yaitu X_{i+1} dapat dihitung dengan memanfaatkan nilai hash sebelumnya. Metode ini disebut *rolling hash* yang sudah dijelaskan pada bab II.B.

$$x_{i+1} = (x_i - t_i R^{M-1}) R + t_{i+M}$$

Keterangan :
X_{i+1} = nilai hash substring ke – i+1
X_i = nilai hash substring ke – i
t_i = karakter ke – i pada string
M = panjang substring (sama dengan panjang pola)

R = bilangan basis

Setelah mendapatkan nilai hash dari substring, kita dapat memanfaatkan *modular hashing* untuk memperkecil nilai hash, yaitu dengan mengambil sisa pembagian dari nilai hash ketika dibagi dengan Q (modulus). Keuntungannya, kita dapat menghitung nilai hash dengan waktu komputasi konstan berapapun nilai M (panjang pola). Berikut perhitungan nilai hash untuk masing – masing substring :



Computing the hash value for the pattern with Horner's method

Gambar 4 – Perhitungan nilai hash untuk setiap substring
Sumber : Algorithms 4th ed.

C. Implementasi

Implementasi algoritma Rabin – Karp dapat dilakukan dengan langkah – langkah sebagai berikut:

1. Hitung nilai hash dari pola. Basis R yang biasa digunakan adalah ukuran dari alfabetnya, misal jika string berisi karakter ASCII, nilai R yang digunakan adalah 256.
2. Ambil substring yang panjangnya sama dengan panjang pola.
3. Hitung nilai hash dari substring yang akan diperiksa.
4. Jika nilai hash dari substring sama denga nilai hash dari pola lanjut ke langkah 5. Jika tidak ulangi langkah 3 untuk substring berikutnya yang akan diperiksa.
5. Periksa apakah isi substring sama dengan isi pola, atau hanya berupa *spurious hit / collision*. Jika sama, lanjut ke langkah 6. Jika tidak ulangi langkah 3 untuk substring berikutnya yang akan diperiksa.
6. Jika ditemukan substring yang sama dengan pola, kembalikan indeks substring tersebut. Jika tidak kembalikan -1.

Contoh implementasinya menggunakan bahasa Java adalah sebagai berikut :

```
public class RabinKarp
{
 private String pat;
 private long patHash; // nilai hash pola
 private int M; // panjang pola
 private long Q; // bilangan prima Q
 private int R = 256; // ukuran alfabet, ASCII
 private long RM; // R^(M-1) % Q
 public RabinKarp(String pat)
 {
   this.pat = pat; // simpan pola
   this.M = pat.length();
   Q = longRandomPrime();
   RM = 1;
   for (int i = 1; i <= M-1; i++) // hitung R^(M-1) %
Q
     RM = (R * RM) % Q;
   patHash = hash(pat, M);
 }

 //cek apakah pat vs txt(i..i-M+1)
 public boolean check(int i)
 {
   int k = 0;
   for(int j=i; j<=i-M+1;j++){
     if(pat[k]!=txt[j])
       return false;
     k++;
    }
   return true;
 }

 private long hash(String key, int M){
   //implementasi fungsi hash
 }

private int search(String txt)
 { // lakukan pencocokan nilai hash
   int N = txt.length();
   long txtHash = hash(txt, M);
   for (int i = M; i < N; i++)
   { // implementasi rolling hash
     txtHash = (txtHash + Q - RM*txt.charAt(i-M)
% Q) % Q;
     txtHash = (txtHash*R + txt.charAt(i)) % Q;
     // cek apakah spurious hit atau valid
     if (patHash == txtHash)
       if (check(i - M + 1)) return i - M + 1; // cocok
    }
       return -1; // pola tidak ditemukan
  }
}
```

IV. PENCOCOKAN POLA MAJEMUK

A. Pembahasan

Pencocokan pola tunggal (*single-pattern matching*) menggunakan algoritma Rabin – Karp tidak memberikan hasil yang lebih baik jika dibandingkan dengan algoritma lain seperti KMP dan Boyer-Moore. Akan tetapi, pada pencocokan pola majemuk (*multi-*

pattern matching), algoritma Rabin – Karp bisa menjadi salah satu alternatif solusi. Pencocokan yang dimaksud di sini adalah pencocokan majemuk dengan setiap pola memiliki panjang yang sama.

Misalkan kita mempunyai himpunan pola pencocokan dan sebuah string yang akan diperiksa. Hitung terlebih dahulu nilai hash untuk setiap pola yang ada di himpunan pola. Ambil sebuah substring, hitung nilai hash nya. Lalu bandingkan nilai hash dari substring dengan nilai hash untuk setiap pola. Jika sama, cek apakah terjadi *spurious hit* atau memang valid. Jika valid, tambahkan ke himpunan solusi. Periksa semua substring dengan langkah yang sama.

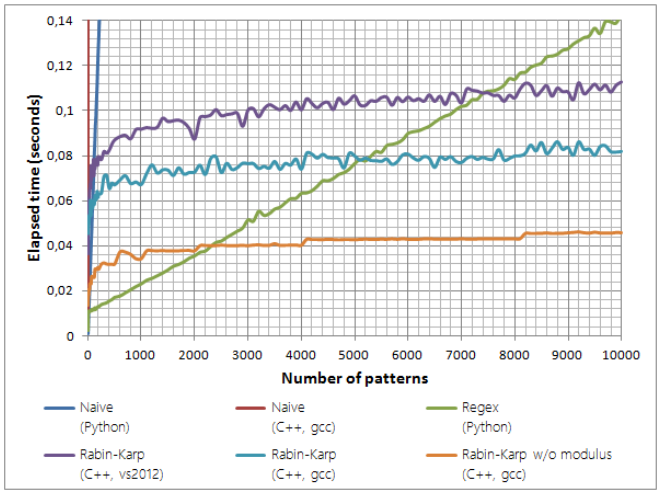
Pseudo-code

```
matches = {}
patterns_checksums = [checksum(pattern) for
pattern in patterns]
substring_checksum = checksum(s[0 : k[)

for position from 0 to n - k - 1 do
    update substring_checksum
    if substring_checksum belongs to
pattern_checksum  then
    add position to matches

return matches
```

Berdasarkan pseudocode di atas, pencocokan pola majemuk dengan algoritma Rabin – Karp memiliki kompleksitas waktu komputasi yang tidak berbeda jauh dengan pencarian pola tunggal. Kompleksitas waktu algoritma akan dibahas pada bab V. Perbandingan beberapa algoritma ketika melakukan pencocokan pola majemuk dapat diamati melalui grafik di bawah ini :



Gambar 5 – Perbandingan waktu eksekusi beberapa algoritma pada pencocokan pola majemuk

Sumber : http://pit-claudel.fr/clement/blog/linear-time-probabilistic-pattern-matching-and-the-rabin-karp-algorithm/

Berdasarkan grafik dapat diamati bahwa algoritma Rabin – Karp dengan penggunaan modulus (berwarna oranye) memiliki rata – rata waktu eksekusi yang paling rendah dan cenderung konstan terhadap penambahan jumlah pola (*patterns*) jika dibandingkan dengan bruteforce (warna biru tua) dan *regular expression* (warna hijau).

B. Aplikasi

Pencocokan pola majemuk antara lain banyak diterapkan pada pemrosesan gambar , bio-informatika, pemrosesan teks. Pada pemrosesan gambar, pencocokan pola majemuk bisa digunakan untuk melakukan face recognition, digital signatures. Di bidang bio- informatika misalnya pada pencocokan pola pada DNA. Adapun di bidang pemrosesan teks, pencocokan pola majemuk bisa digunakan untuk mengecek terjadinya plagiarisme pada sebuah dokumen atau karya tulis.

V. Kompleksitas Algoritma

Algoritma Rabin – Karp membutuhkan kompleksitas waktu O(m) untuk menghitung nilai hash dari semua substring karena menggunakan metode *rolling hash* dan m adalah panjang pola. Sedangkan kompleksitas waktu terburuk untuk melakukan perbandingan nilai hash adalah O((n-m+1)m) yaitu jika terjadi *spurious hit* untuk setiap substring yang diperiksa.

Jadi kompleksitas waktu terburuk untuk algoritma Rabin – Karp adalah

$$O((n-m+1)m) + O(m) = O((n-m+1)m)$$

Walaupun kompleksitas waktu terburuk sama dengan algoritma bruteforce, kompleksitas waktu rata – rata algoritma Rabin – Karp adalah O(n) karena menggunakan kemungkinan terjadinya *spurious hit* / *collision* pada fungsi hash seharusnya kecil asalkan kita membangun fungsi hash yang baik.

VI. Simpulan

Berdasarkan pembahasan pada makalah ini, bisa ditarik beberapa kesimpulan berikut:

1. Algoritma Rabin – Karp sebagai salah satu algoritma pencocokan string yang menggunakan teknik hashing dalam menyelesaikan permasalahan

2. Algoritma Rabin – Karp menjadi salah satu solusi dalam pencocokan pola majemuk
3. Kompleksitas waktu rata – ratanya adalah O(n) dengan anggapan fungsi hash yang dibangun baik. Sedangkan kompleksitas waktu terburuknya adalah O((n-m+1)m).

VII. UCAPAN TERIMA KASIH

Saya mengucapkan syukur kepada Tuhan Yang Maha Esa karena dapat menyelesaikan makalah ini tepat waktu. Saya juga mengucapkan terima kasih kepada Bu Ulfa dan Pak Rinaldi atas bimbingan dan pengajaran yang telah diberikan pada mata kuliah Strategi Algoritma. Terima kasih saya sampaikan juga kepada keluarga dan teman-teman yang telah mendukung saya dalam kuliah ini.

REFERENSI

[1] Cormen, Thomas H., et.al. *Introduction to Algorithms, Third Edition*. The MIT Press, 2009, ch.11, ch.32.
[2] Sedgewick, Robert; Wayne, Kevin. *Algorithms , Fourth Edition*. Addison-Wesley, 2009, ch.5
[3] MIT 6.006: Introduction to Algorithms 2011- Lecture Notes - Rabin–Karp Algorithm/Rolling Hash
[4] http://pit-claudel.fr/clement/blog/linear-time-probabilistic-pattern-matching-and-the-rabin-karp-algorithm/
Diakses pada 4 Mei 2015 pukul 19.30
[5]http://www.dcc.uchile.cl/~gnavarro/workshop07/lsalmela.pdf
Diakses pada 3 Mei 2015 pukul 16.00
[6] http://www.cs.mun.ca/~kol/courses/6783-w12/scribe-rabin-karp.pdf
Diakses pada 2 Mei 2015 pukul 22.00
[7] http://www.sci.unich.it/~acciaro/Rabin_Karp.pdf
Diakses pada 2 Mei 2015 pukul 21.30

PERNYATAAN

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

Bandung, 5 Mei 2015

Edwin Wijaya
13513040