Penggunaan Algoritma Rabin-Karp dalam Pencocokan *String*

Teddy Pandu Wirawan

Teknik Informatika Institut Teknologi Bandung
Jalan Ganesha no 10, Bandung
e-mail: if16039@students.if.itb.ac.id

ABSTRAK

Pencocokan *string* (*string matching*) adalah suatu persoalan yang sangat mendasar pada ilmu computer. Banyak persoalan yang membutuhkan pencocokan *string*. Algoritma paling sederhana untuk melakukan pencocokan *string* adalah dengan menggunakan algoritma *brute force*. Namun algoritma ini sangatlah tidak mangkus karena algoritma ini mencari semua kemungkinan yang ada. Contoh kongkret penggunaan *string matching* adalah pada compiler. Contoh lain adalah pada pencarian data di computer atau jaringan internet. Pada suatu jaringan internet yang sangatlah besar, sangat sulit untuk mencari website yang kita inginkan. Karena itu Kita memanfaatkan bantuan mesin pencari. Cara kerja mesin pencari seperi Google, atau Yahoo!, adalah dengan melakukan pencocokan *string* yang kita masukkan dengan *string* yang terdapat pada suatu website untuk mendapatkan relevansinya kemudian manampilkan informasi yang diinginkan. Tidak bisa dibayangkan berapa lama waktu yang akan dilakukan untuk melakukan pencarian bila menggunakan algoritma *brute force*. Karena itu, diperlukan suatu algoritma yang mangkus agar waktu komputasi yang diperlukan menjadi lebih singkat, sehingga akan meningkatkan performa suatu komputer. Sudah banyak algoritma yang ditemukan untuk memecahakan masalah ini, diantaranya, algoritma Knuth-Morris-Pratt, algoritma Boyer-Moore, dan algoritma Rabin-Karp. Untuk dua algoritma pertama sudah banyak dibahas. Makalah ini akan membahas Algoritma Rabin-Karp. Walaupun algoritma ini kurang terkenal, tapi cukup menarik untuk mengetahui cara pencocokan *string* yang digunakan. Pendekatan yang digunakan algoritma ini adalah dengan menggunakan fungsi *hash* yang didapat dari nilai ASCII tiap karakter yang terdapat pada *string* sumber dan *string* yang ingin dicari.

Kata kunci: Pencocokan, *string*, *brute force*, nilai *hash*, Boyer-Moore, Rabin-Karp.

1. PENDAHULUAN

Pada suatu teks yang sangat banyak, ada kalanya Kita ingin mencari suatu kata, atau frase dalam teks tersebut. Pasti akan sangat sulit biala pencarian dilakukan secara manual. Karena itu Kita menggunakan bantuan alat pencari. Dengan bantuan itu, kata yang ingin Kita cari dapat ditampilkan dengan lebih cepat. Atau mungkin Kia ingin melakukan pencarian informasi pada suatu situs di internet. Kita menggunakan mesin pencari untuk mencari situs yang mengandung informasi yang Kita inginkan pada jaringan internet. Metode pencarian yang digunaakn pada mesin pencari di internet atau yang ada pada computer adalah dengan melakukan pencocokan *string*. Suatu data pada internet atau computer dianggap sebagai suatu *string* data, kemudian suatu informasi yang ingin Kita cari dianggap sebagai suatu sub*string*. Jadi mesin pencari akan melakukan pencarian sub*string* pada *string* data tersebut. Pencocokan *string* ini tampaknya adalah sesuatu yang sangat sederhana. Namun dengan hanya mengandalkan metode pencarian yang tepat, perusahhan seperti Google dapat menjadi besar. Hal ini membuktikan bahwa pencocokan *string* juga dapat menjadi bisnis jutaan dolar.

Algoritma Rabin-Karp dibuat oleh Michael O. Rabin dan Richard M. Karp. Algoritma ini lebih berguna pada pencarian *multiple pattern* daripada pencarian *single pattern*. Karena algoritma ini tidak memperdulikan huruf besar atau kecil, dan tanda baca yang digunakan. Algoritma Rabin-Karp ini menggunakan fungsi *hash*. Fungsi *hash* adalah fungsi yang digunakan untuk mengubah *string* menjadi untaian integer. Pada algoritma ini untaian *string* akan diubah menjadi integer berdasarkan bilangan ASCII nya. Karena menggunakan bilangan ASCII, proses komputasi menjadi lebih "dekat" ke bahasa mesin. Pendekatan utamanya adalah, *string* yang sama akan memiliki nilai *hash* yang sama.

2. METODE

Pertama, Kita tinjau dulu algoritma *brute force*. Algoritma ini mencari setiap kemungkinan dari pencocokan *string*. Misal, terdapat *string* sumber S yang tiap elemennya berada pada *array* T[1..n], dan kata atau *pattern* K yang ingin dicari ada pada *array* P[1..m], dimana n ≥ m. Secara garis besar cara kerja algoritma ini adalah sebagai berikut:

1. *Pattern* P dicocokkan pada awal teks T.
2. Dari kiri ke kanan, bandingkan setiap karakter dalam P dengan karakter yang bersesuaian pada T sampai semua karakter yang dibandingkan cocok, atau ada dijumpai ketidakcocokkan.
3. Bila *pattern* P belum ditemukan kecocokkannya dan teks T belim habis, geser *pattern* P ke kanan satu karakter, kemudian ulangi langkah 2.

Berikut adala contoh kode yang mengimplementasikan pecarian *string* dengan algoritma *brute force*:

```
function BruteForce(string s[1..n],
string sub[1..m])
      for i from 1 to n-m+1
          for j from 1 to m
              if s[i+j-1] ≠ sub[j]
                 jump to next iteration
      of outer loop
          return i
      return not found
```

Kompleksitas algoritma pada metode ini pada kasus terburuk adalah O(mn).

Sekarang Kita tinjau algoritma Rabin-Karp. Hal utama yang harus dilakukan sebelum melakukan pencocokan dengan algoritma Rabin-Karp adalah mengubah *string* sumber dan *string* yang ingin dicari menjadi sebuah untaian karakter. Contoh paling sederhana adalah dengan menjumlahkan bilangan ASCII nya. Misalnya, terdapat *string* "Strategi", dan Kita ingin mencari kata "rat" pada *string* tersebut.

1. Langkah pertama adalah menentukan panjang *string* sumber dan *string* yang ingin dicari. Dalam kasus ini, panjang *string* sumber n=8. Sedangkan *string* yang ingin dicari panjangnya m=3.
2. Langkah kedua adalah dengan mengubah kata yang ingin dicari dengan menggunakaan fungsi *hash*. Jadi niai *hash* dari "rat" adalah 327 (nilai ASCII r =114, a=97, dan t=116).
3. Kemudian melakukan iterasi dari indeks i=0 sampai i=n-m+1. Pada saat iterasi, dilakukan perbandingan nilai *hash* dari nilai *hash* kata yang ingi dicari dengan nilai *hash* dari *string* sumber pada indeks ke I sampai i+m-1. Bila sama, maka akan mengembalikan nilai true, tapi jika tidak, maka perbandingan akan dilakukan dengan indeks berikutnya. Dalam hal ini, Kita membandingkan 327 dengan nilai *hash* dari *string* "Str" (nilai *Hash* = 313). Karena tidak sama, maka dilakukan perbandingan dengan nilai *hash* dari indeks selanjutnya. Pada kasus ini indeks ke dua sampai ke 4(i+m-1).Pencocokan akan dilakukan terus sampai mendapatkan nilai *hash* yang sama seterusnya.

Berikut ini adalah contoh kode yang mengimplementasikan algoritma Rabin-Karp

```
function RabinKarp(string s[1..n],
string sub[1..m])
    hsub := hash(sub[1..m])
    for i from 1 to n-m+1
        if hs = hsub
            if s[i..i+m-1] = sub
                return i
        hs := hash(s[i+1..i+m])
    return not found
```

Algoritma sederhana ini masih memiliki masalah yaitu untai *string* yang berbeda, bisa saja memiliki nilai *hash* yang sama. Misal saja, *string* "rat" memiliki nilai *hash* yang sama dengan *string* "tar". Hal ini akan mengakibatkan keambiguan dalam pencarian. Misalnya terdapat *string* "kasurrusak" dan kita ingin mencari kata "sak". Dengan fungsi *hash* biasa ini, pmaka pada pencocokan pertama akan langsung menemukan hasil. Karena "kas" dan "sak" memiliki nilai *hash* yang sama (319). Karena itu, diperlukan suatu fungsi *hash* khusus untuk menangani masalah ini.

2.1 Rabin-Karp dengan *Rolling Hash*

Untuk mengatasi masalah itu, maka digunakanlah fungsi *hash* dengan basis yang disebut dengan *Rolling Hash*. Basis biasanya adalah bilangan prima yang cukup besar. Persamaannya adalah sebagai berikut:

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

Dengan H adalah nilai *Hash*,
c1 adalah nilai ASCII suatu karakter
a adalah basis
k adalah banyaknya karakter

Berikut akan dijelaskan penggunaan fungsi *hash* dengan basis. Misalnya, Kita memilih 101 sebagai basis. Sebagai contoh, kita menggunakan *string* "kasurrusak" sebagai *string* sumber dan ingin mencari kata "sak". Dengan metode ini maka kata sak memiliki nilai *hash* 1183019. Ini didapat dari

Nilai has dari *string* ”sak” = 101^2 x 115(nilai ASCII dari s) + 101^1 x 97(nilai ASCII dari a) + 101^0 x 107(nilai ASCII dari k)

Dengan demikian nilai *hash* dari “sak” berbeda dengan “kas”(nila hasah = 1101419). Langkah selanjutnya mirip seperti pencocokan dengan menggunakan fungsi *hash* sederhana.

2.2 Algoritma Rabin-Karp dalam Pencarian *Multiple Pattern*

Algoritma Rabin-Karp ini bekerja lebih baik bila digunakan untuk melakukan pencarian *multiple pattern*. Dengan menggunakan fungsi *hash* biasa, bahkan dengan *rolling hash* sekalipun, akan menghasilkan nilai *hash* yang sangat besar. Namun nini bisa diatasi dengan menggunakan *hashset*. Yaitu nilai *hash* yang merupakan set (setiap elemennya unik).

```
function RabinKarpSet(string s[1..n],
set of string subs, m) {
     set hsubs := emptySet
     for each sub in subs
         insert hash(sub[1..m]) into
hsubs
     hs := hash(s[1..m])
     for i from 1 to n-m+1
         if hs ∈ hsubs
             if s[i..i+m-1] = a
substring with hash hs
                 return i
         hs := hash(s[i+1..i+m])
     return not found
 }
```

Algoritma lain dapat melakukan pencarian *pattern* tunggal dengan kompleksitas O(n) dan melakukan pencarian *multiple pattern* dengan kompleksitas O(n k) dengan k adalah banyaknya *pattern*. Tapi dengan algoritma Rabin-Karp ini, kompleksitas pencarian *multiple pattern* adalah O(n+k).

Berikut ini adalah table yang berisi perbandingan kompleksitas berbagai algoritma pencocokan *string*

Tabel 1 Perbandingan kompleksitas berbagai algoritma pencocokan *string*

Algoritma	Waktu sebelum pencocokan	Waktu pencocokan
Brute Force	Tidak ada proses	$\Theta(n\ m)$
Rabin-Karp	$\Theta(m)$	Rata-rata $\Theta(n+m)$, Terburuk $\Theta(n\ m)$
Finite State automaton	$\Theta(m\ \|\Sigma\|)$	$\Theta(n)$
Knuth-Morris-Pratt	$\Theta(m)$	$\Theta(n)$
Boyer-Moore	$\Theta(m + \|\Sigma\|)$	$\Omega(n/m)$, $O(n)$
Bitap	$\Theta(m + \|\Sigma\|)$	$\Theta(n)$

3. KESIMPULAN

Algoritma Rabin-Karp menggunakan pendekatan yang berbeda dalam melakukan pencocokan *string*. Dengan menggunakan pendekatan pada nilai ASCII dari tiap karakter, algoritma ini mencoba memberikan data yang lebih mudah oleh mesin. Hal ini diharapkan akan meningkatkan performa komputasi.

Pada kasus terburuk algoritma ini memiliki kompleksitas yang sama dengan algoritma *brute force*. Karena itulah algoritma ini jarang digunakan. Tapi algoritma ini telah dikembangkan oleh penemunya sehingga lebih optimal. Namun Penulis tidak menuliskan penjelasan akan hal tersebut pada makalah ini, karena hal tersebut diluar kemampuan penulis. Pada pengembangannya, algoritma ini menggunakan algoritma statistic Rabin, Solovay dan Strassen, juga penggunaan Lemma ***Rosser and Schoenfeld***, penggunaan fungsi *hash* dengan matematika modulo untuk mendapatkan nilai *hash* yang lebih singkat, kemudian menganalisis apakah nilai hasah dari suatu *string* sumber merupakan milik dari *string* yang ingin kita cari, dengan menggunakan bloom filter.

REFERENSI

[1] Munir,Rinaldi.2006.*Diktat Kuliah IF2251 Strategi Algoritmik*.Bandung.

[3] Rabin,Michael O. dan Richard M Karp.Efficient Randomized *Pattern*-Matching algorithms. IBM Journal of Research and Development

[2] http://en.wikipedia.org/wiki/Rabin-Karp_*string*_search_algorithm
Diakses hari Senin 19 Mei 2008 pukul 05.35 WIB