String Matching dengan Menggunakan Algoritma Rabin Karp

Najib Baedlowi[1] , Deka Aditia Adam[2]

Laboratorium Ilmu dan Rekayasa Komputasi
Departemen Teknik Informatika, Institut Teknologi Bandung
Jl. Ganesha 10, Bandung

Email : if14131@students.if.itb.ac.id [1], if14162@students.if.itb.ac.id [2]

Abstrak :

Algoritma Rabin Karp adalah algoritma pencarian string yang ditemukan oleh Michael O. Rabin dan Richard M.Karp, algoritma ini melakukan pencarian string didalam teks dengan memanfaatkan *hash function.* Algoritma ini kurang mangkus apabila diterapkan dalam pencarian *single pattern.* Akan tetapi akan sangat penting dan sangat efektif untuk pencarian *multiple pattern*. Untuk teks dengan panjang n dan pattern dengan panjang m kompleksitas rata rata dan kasus terbaik dari algoritma ini adalah O(n). Akan tetapi untuk kasus terburuk kompleksitasnya adalah O(mn), inilah yang dijadikan alasan kenapa algoritma ini jarang dipakai. Keunggulan algoritma ini terdapat pada pencarian multiple *pattern* . Untuk mencari k string, secara rata-rata kompleksitasnya O(n), tanpa memperhatikan banyaknya k.

Kata kunci : *String Matching, Rabin-Karp, Hash-Table , Hash-Function*

1. Pendahuluan

Pencarian string adalah sebuah masalah yang sangat krusial dalam dunia informatika. Persoalan pencarian string dirumuskan sebagai berikut, Diberikan :
teks yang panjangnya n karakter dan pattern yaitu string yang akan dicari dengan panjang m karakter dengan panjang m < panjang n.
Ada beberapa macam algoritma yang digunakan dalam "*String matching*" diantarany:

1. Algoritma Brute Force
   Algoritma bruto force dalam penerapan pencocokan string didefiniskan sebagai berikut : jika diasumsikan teks adalah sebuah table karakter maka pattern yang juga merupakan table karakter dengan panjang lebih kecil dari teks, maka algoritma bruto force diimplementasikan sebagai berikut :
   1. Mula mula cocokkan pattern diawal teks.
   2. Dengan bergerak dari kiri kekanan bandingkan setiap karakter diadalam pattern dengan karakter yang bersesuaian dengan teks sampai :
      a. Semua karakter yang dibandingkan sesuai dengan pattern yang dicari
      b. Dijumpai sebuah ketidakcocokkan karakter.
      c. Bila dijumpai ketidakcocokkan karakter pada pattern dan teks belum berakhir geser pattern satu karakter ke kanan.
   3. Kompleksitas algoritma BrutoForce dalam String Mathing :
      1. Kasus terbaik O(n)
      2. Kasus terburuk O(mn)

2. Algoritma Knuth Morris Pratt
   Pada algoritma ini kita memelihara informasi yang digunakan untuk melakukan sejumlah pergeseran, Algoritma menggunakan informasi tersebut untuk membuat pergeseran yang lebih jauh, tidak hanya satu karater seperti pada algoritma bruteforce.
   Kompleksitas algoritma ini :
   1. Kasus terbaik O(n)
   2. Kasus terburuk O(m+n)

3. Algoritma Boyer-Moore
   Kompleksitas algoritma :
   1. Kasus terbaik O(n)
   2. Kasus terburuk sama dengan algoritma BrutoForce yaitu O(mn)

4. Algoritma Rabin-Karp
   Algoritma ini adalah algoritma acak sederhana yang cenderung berjalan dalam waktu linier . Kasus terburuk dari algoritma ini seperti pada algoritma brute force

2. Penjelasan algoritma Rabin-Karp

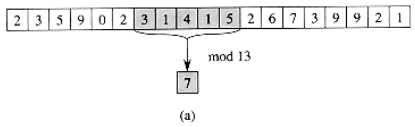
Penjelasan :

1. Asumsikan teks adalah string t yang panjangnya m dan pattern (string yang akan dicari) P panjangnya n.

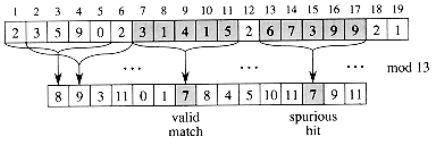
2. Assumsikan Si menyatakan sebuah substring dengan panjang n, yang berkelanjutan pada awal teks- misalkan S0 adalah substring dengan panjang n diawal t
3. Ide utama dari algoritma ini adalah memanfaatka fungsi hash untuk memetakan setiap Si kedalam himpunan.Fungsi hash diugunakan untuk menempatkan suatu record yang mempunyai nilai kunci k. fungsi hash yang paling umum berbentuk ;
   H(k) = k mod m
   Di dalam algoritma ini k dapat berupa string P atau String Si
4. Sketsa Algoritma :
   a. Pertama kita menghitung nilai fungsi hash dari string P.
   b. Kemudian untuk masing-masing Si kita menhitung fungsi hashnya.
   c. Lakukan penelusuran terhadap Si, jika h(Si) = h(P), maka lakukan pencocokan antara String Si dengan string P secara brute force.
   d. jika h(Si) ≠ h(P) kita tidak perlu melakukan pencocokan string, penulusuran dilanjutkan kembali terhadap Si yang berikutnya sampai ditemuka atau sampai string t berakhir.

PseudoCode dari algoritma ini dituliskan sebagai berikut :

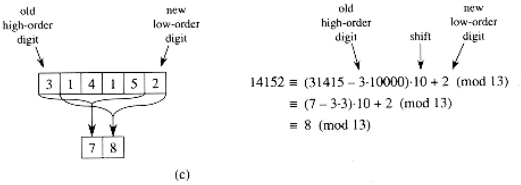
```
Algoritma RabinKarp
 function RabinKarp(string s[1..n],
string sub[1..m])
     hsub := hash(sub[1..m])
     hs := hash(s[1..m])
     for i from 1 to n
         if hs = hsub
             if s[i..i+m-1] = sub
                 return i
         hs := hash(s[i+1..i+m])
     return not found
```



(a)



(b)

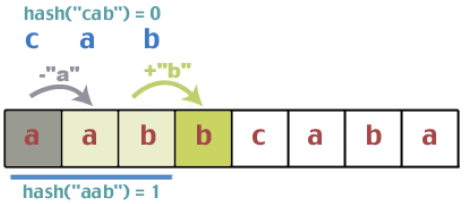


(c)

3. Pemilihan Hash Function

Fungsi Hash adalah sebuah fungsi yang mengkonvert setiap string menjadi bilangan, yang disebut hash value. Sebagai contoh Hash("Hello")=5. Algoritma Rabin-Karp didasarkan pada fakta jika dua buah string sama maka harga hash valuenya pasti sama. Akan tetapi ada dua masalah yang timbul dari hal ini, masalah pertama yaitu ada begitu banyak string yang berbeda, permasalahan ini dapat dipecahkan dengan meng-assign. beberapa string dengan hash value yang sama. Masalah yang kedua belum tentu string yang mempunyai hash value yang sama cocok untuk mengatasinya maka untuk setiap string yang diassign dilakukan pencocokan string secara bruteforce.
Kunci agar algoritma rabin-karp mangkus, terdapat pada pemilihan hash valuenya. Salah satu cara yang terkenal dan efektif adalah memperlakukan setiap substring sebagai suatu bilangan dengan basis tertentu, biasanya yang dijadikan basis adalah bilangan prima berukuran besar. Sebagai contoh , substring "hi" dengan basis 101, akan mempunyai hash value sebesar 104*1011 + 105*1010 = 10609 (nilai ASCII untuk 'h' = 104 dan 'i' = 105)
Seacara teknis, algoritma ini mirip dengan representasi bilangan bulat. Keuntungan yang paling penting yang didapat dengan representasi ini adalah memungkinkan kita untuk menghitung hash value dari substring berikutnya dengan memanfaatkan substring sebelumnya dengan sejumlah operasi yang konstan, terlepas dari panjang setiap substring.

Sebagai contoh kita mempunyai string "abracadabra" dan kita mencari pattern dengan panjang 3. kita bisa menghitung hash "bra" dari hash "abr" dengan membagi dengan hash value dari karakter pertama . contoh 'a' dari "abr" , 97*1012 (97 adalah nilai ASCII dari 'a' dan 101 adalah basis yang kita gunakan)., lalu kalikan dengan bilanganbasis dan jumlahkan substring terkakhir, contoh 'a' dari "bra" , 97*1010 = 97. jika substring yang dicari panjang algoritma ini menghemat waktu untuk mendapatkan hash value.



4. Mulitple pattern Search

Algoritma Rabin-karp apabila digunakan pada pencocokan single pattern masih kurang mangkus dibandingkan dengan algoritma KMP atau Boyer-

Moore, karena kasus terburuknya. Akan tetapi Rabin-Karp adalah sebuah algoritma yang tepat untuk pencocokan multiple pattern.
Misalkan, kita ingin mencari bilangan yang besar (k), kita bisa membuat variant sederhana dari algoritma rabin-karp yang memanfaatkan hashtabel atau struktur data lainnya untuk mengecek apakah string yang kita periksa termasuk himpunan hash value dari pattern yang kita cari.

Pseudocode dari pencocokan multiple pattern dengan menerapkan algoritma Rabin-Karp dapat dijelaskan sebagai berikut :

Pencocokan Multiple Pattern

```
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
         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
 }
```

Disini kita mengasumsikan semua substring mempunyai panjang m, tetapi asumsi ini bisa dieliminir. Secara sederhana kita membandingkan hashvalue sebelumnya dengan hashvalue dari semua substring secara secara berkelanjutan dengan melakukan pencarian didalam himpunan data struktur kita, dan mengecek kecocokan dengan semua substring dengan hash value tersebut.

Algoritma lainnya bisa memiliki kompleksitas O(n) untuk pencocokan single pattern dan kompleksitas O(nk) untuk pencocokan k pattern. Sebaliknya algoritma Rabin-karp diatas bisa mencari k pattern dengan komplekitas sebessar O(n+k).

5. Kesimpulan

Algoritma Rabin-Karp kurang mangkus apabila digunakan dalam pencocokan single pattern, kompleksitas untuk kasus terburuknya adalah O(mn). Akan tetapi dalam pencocokan multiple pattern algoritma ini mempunyai keunggulan dibandingkan algoritma lainnya yaitu dengan kompleksitas rata rata O(n+k).

Daftar Pustaka

1. Munir, Rinaldi. 2005. Diktat Kuliah Strategi Algoritmik. Teknik Informatika ITB : Bandung.
2. Plaxton, Greg. 2005. Theory In Programming Practice. Department of Computer Science University of Texas at Austin.
3. http://en.wikipedia.org/wiki/Rabin-Karp_string_search_algorithm