Mempercepat Algoritma Brute Force Dalam Permasalahan Pencocokan String

Goppy Gita Gustaman[1], Akhmad Mukhammad[2]

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

E-mail : if14146@students.if.itb.ac.id[1],if14152@students.if.itb.ac.id[2]

Abstrak

Pencocokan string merupakan salah satu masalah yang terus menjadi hal yang menggelitik untuk dicari algoritma termangkusnya. Telah banyak algoritma pencocokan string yang memiliki keunggulan sendiri – sendiri. Beberapa algoritma yang cukup terkenal adalah algoritma brute force, algoritma KMP, dan algoritma Boyer-Moore. Algoritma pencocokan string yang mangkus dapat mendukung kinerja suatu software atau OS misalnya pencarian string pada word processor, text editor, pencarian file berdasarkan nama misal pada perintah locate pada OS UNIX dan search file pada OS WINDOWS, dan search engine pada website. Dalam makalah ini kami akan mencoba mengembangkan algoritma brute force dalam pencocokan string dengan prinsip mengurangi jumlah pembandingan karakter.

Kata kunci *brute force, string matching, pencocokan string.*

1. Pendahuluan

1.1 Latar Belakang

Konsep dasar dalam pencocokan string adalah traversal dari awal teks sampai ditemukan string pattern yang dicari atau sampai akhir teks. Jika T adalah teks yang direpresentasikan dalam tabel berindeks 0..m dan P adalah string pattern yang dicari yang juga direpresentasikan dalam tabel berindeks 0..n, maka traversal dimulai dari T[0] sampai ketemu atau sampai T[m - n]. Dalam membahas algoritma yang kami buat, kami juga merepresentasikan teks ke dalam tabel T dengan indeks 0..m.

1.2 Tujuan penulisan makalah:

1. Mencoba menemukan algoritma baru yang dapat menangani masalah pencocokan string, meskipun mungkin kurang mangkus dari algoritma yang sudah ada.
2. Berusaha mengembangakan algoritma brute force sehingga dapat memperkecil kompleksitasnya.

1.3 Metode penyusunan :

studi literatur, eksperimen.

2. Ruang Lingkup

Algoritma yang coba kami uraikan merupakan pengembangan dari algoritma brute force dalam masalah pencocokan string agar mengurangi kompleksitasnya. Pengembangan dilakukan dengan melakukan penambahan fitur – fitur yang mengurangi jumlah traversal.

3. Mempercepat Algoritma Brute Force Dalam Masalah Pencocokan String

Prinsip utama pada algoritma brute force dalam mencocokan string adalah traversal satu persatu mulai dari karakter pertama teks T, kemudian membandingkan setiap karakter dengan pattern P yang dicari, jika ditemukan satu karakter yang tidak sama, maka traversal akan maju satu karakter saja.
Berikut adalah pseudocode algoritma bruteforce tersebut:

```
procedure BruteForceSearch(input m,n : integer, input P : array[1..n] of char, input T : array[1..m] of char, output idx : integer)

Deklarasi
s, j : integer
ketemu : boolean

Algoritma
s ← 0
ketemu ← false
while (s ≤ m-n) and (not ketemu) do
        j ← 1
        while( j ≤ n ) and (P[j] = T[s+j]) do
                j ← j + 1
        endwhile
        {j > n atau P[j] != T[s+j]}
        if(j = n) then
                ketemu ← true
        else
                s ← s+ 1
        endif
endwhile
```

```
{s > m-n atau ketemu}

if (ketemu) then
        idx ← s+ 1
else
        idx ← -1
endif
```

dari pseudocode di atas dapat kita ketahui bahwa increment traversal hanya terjadi sebanyak satu karakter . Kemudian muncul algoritma KMP yang menangani lompatan terjauh. Algortima ini memiliki prinsip bahwa informasi yang digunakan untuk melakukan jumlah pergeseran tetap dipelihara.[1]

Dalam algoritma brute force yang kami kembangkan, kami menggunakan prinsip yang sama dengan algoritma KMP yaitu memelihara informasi untuk menentukan jumlah lompatan. Namun algoritma yang kami buat tidak memerlukan prosedur HitungPinggiran[1] seperti halnya pada algoritma KMP.

Dalam algoritma yang kami tawarkan, pembandingan karakter secara penuh (pembandingan 1 pattern) hanya dilakukan jika T[x] = P[0] dan T[x + panjangpattern - 1] = P[panjangpattern -1] (indeks tabel dimulai dari 0). Jika syarat itu dipenuhi kita akan perbandingan string, jika tidak kita akan maju satu langkah. Dalam perbandingan string tersebut, jika kita menemukan karakter yang sama dengan karakter awal pattern untuk pertama kali, kita akan menyimpan informasi indeksnya untuk menjadi acuan lompatan berikutnya. Algoritma yang kami tawarkan berbasis pada karakter pertama pattern dan karakter terakhir pattern.

Berikut adalah pseudecode algoritma yang kami tawarkan :

```
procedure AdvancedBruteForce(input m,n : integer, input P : array[0..n] of char, input T : array[0..m] of char, output idx : integer)

Deklarasi
char awal,akhir;
integer i,j,lompat;
boolean ketemu, sudahterisi;
{sudahterisi berguna saat menentukan jumlah lompatan}

Algoritma
i = 0;
ketemu = false;
awal = P[0];
akhir = P[n];
while((i <= m - n) and not ketemu) do
  if ((T [i] = awal)and(T [i+n] = akhir))
  then
        j = 0;
        sudahterisi = false;
        lompat = 0;
        while((j < N)and(T[i]= P[j]))do
         if ((j>0) and (not sudahterisi) and
(Teks[i] = awal))
          then
            lompat = i;
            sudahterisi = true;
         endif
          j++;
          i++;
        endwhile
        {j>=N or T[i] != P[j]}
        if (T[i] == P[j])
        then
          ketemu = true;
        else {(T[i] != P[j])}
          if (lompat != 0)
          then
            i = lompat;
          endif
        endif
          else {(T[i]/=awal)||(T[i+N-1] !=
akhir)}
              i++;
          endif
        endwhile
        { (i > m - n) || ketemu}
        if (ketemu)
        then idx = i - n
          {indeks awal pattern yang ditemui
dimulai dari i-n}
        else
           idx = -1
        endif
```

Alasan mengapa algoritma kami memakai basis karakter terakhir adalah mengurangi jumlah pengulangan yang ada di dalam. Algoritma ini mungkin cukup efektif dalam pencarian kata pada word processor, di mana kata yang dicari memiliki pelafalan dalam bahasa natural yang mirip dengan yang terdapat dalam teks memiliki peluang kejadian yang lebih besar, dan kemiripan tersebut umumnya ditentukan pada akhir string, misal kata “banak” dengan kata “banag”. Algoritma kami tidak akan mengecek satu – satu dari awal pattern. Ilustrasi berikut menunjukkan perbandingan algoritma brute force dengan algoritma brute force tyang dipercepat

Teks : a n a b a r b k a k b a r a k
string pattern : “b a r a k”
secara brute force :

a n a b a r b k a k b a r a k	15 karakter
b a r a k	1
b a r a k	2
b a r a k	3
b a r a k	4,5,6,7
b a r a k	8
b a r a k	9
b a r a k	10
b a r a k	11
b a r a k	12
b a r a k	13

untuk contoh ini memerlukan 13 kali perbandingan

dengan algoritma bruteforce yang dipercepat :

```
a n a b a r b k a k b a r a k
b       k           1
  b       k         2
    b       k       3
      b       k (awalan & akhiran sama,cek,sampai
                  karakter ke 4 tidak sama, tidak
                  kembali ke karakter a, langsung ke
                  indeks b yang telah disimpan)
                  4,5,6
            b       k           7
              b       k         8
                b       k       9
                  b       k     10
                    b       k   11
```

untuk contoh ini hanya dilakukan 11 kali perbandingan.

Untuk banyak kasus mungkin algoritma ini akan berjalan seperti algoritma brute force, namun algoritma ini tidak akan lebih tidak mangkus daripada brute force, karena peluang terjadinya kasus di atas tetap ada.

4. Analisis Kompleksitas

Melihat peluang terjadinya kasus seperti pada contoh tidak dapat ditentukan maka kompleksitas terbaik algoritma kami bisa dikatakan sama dengan algoritma brute force, yakni dengan kompleksitas terbaik O(n). Contoh kasus terbaik adalah

Teks : a b j f d h k i e g o e p t d h g
Pattern : p t d h g

Sedangkan pada kasus terburuknya, kompleksitas belum dapat kami tentukan. Sebagai perbandingan untuk kasus terburuk :

Teks : a b c d e a b c d e a b c s e (15 char)
Pattern : a b c s e (5 char)
Pengulangan :

```
a b c d e a b c d e a b c d g
a b c s e {awalan+akhiran sama, bandingkan,
sampai karakter ke 4 tidak sama,kembali,jumlah =
4}
  a       e
    a       e
      a       e
        a       e
          a b c s e
```

{awalan + akhiran sama, bandingkan, sampai karakter ke 4 tidak sama,kembali,jumlah = 4}

```
            a       e
              a       e
                a       e
                  a       e
                    a b c s e
```

{sama, jumlah perbandingan = 4}

Total jumlah perbandingan : 4x3 + 8 = 20 kali

5. Kesimpulan

Algoritma brute force yang kami kembangkan mungkin tidak semangkus algoritma KMP, namun algoritma kami (brute force yang dipercepat)juga tidak lebih tidak mangkus daripada algoritma brute force. Algoritma kami mungkin dapat menjadi salah satu algoritma yang dapat menyelesaikan masalah pencocokan string.

Untuk kasus tertentu, seperti pada contoh yang kami berikan, algoritma brute force yang dipercepat tersebut dapat menghemat jumlah perbandingan daripada algoritma brute force.

6. Daftar Pustaka

[1] Munir, Rinaldi, *Strategi Algoritmik,* Program Studi Teknik Informatika, Januari 2005