Penerapan Algoritma Knuth-Morris-Pratt pada Aplikasi Pencarian Berkas di Komputer

Hafni Syaeful Sulun

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
Jalan Ganesha 10 Bandung
if15058@students.if.itb.ac.id

ABSTRAK

Tidak dapat dielak lagi bahwa fitur pencarian berkas di sistem operasi yang paling banyak digunakan saat ini, Windows XP, sangat lambat dalam menjalankan fungsinya. Dengan alasan tersebut, banyak pengembang perangkat lunak membangun kakas pencarian berkas di komputer lokal. Bahkan, tidak sedikit yang mengkombinasikannya dengan pencarian *online*. Google pun, seperti yang kita ketahui memiliki mesin pencari *website* nomor satu di dunia saat ini, menerapkan cara serupa pada kakas pencarian untuk *desktop*. Google menggunakan algoritma unggulannya yang mendukung *file indexing* untuk kedua mesin pencari tersebut.

Dengan alasan tersebut pula, penulis berusaha mengembangkan kakas serupa, yakni XLN Desktop Search. Algoritma yang digunakan dalam kakas ini yaitu algoritma pencarian *string* Knuth-Morris-Pratt atau biasa disebut KMP. Jika dibandingkan dengan algoritma dasar pada hampir semua permasalahan, *brute force*, jelas algoritma ini jauh lebih mangkus. Seperti namanya, algoritma KMP ini dikembangkan oleh tiga ilmuwan ilmu komputer pada masa lampau, yaitu Donald E. Knuth, Joseph H. Morris, dan V. R. Pratt. Donald E. Knuth juga mempunyai sebutan Bapak Ilmu Komputer Modern [2].

Kata kunci: *string*, *pattern*, teks, algoritma, *brute force,* KMP, fungsi pinggiran.

1. PENDAHULUAN

Pencarian *string* yang juga biasa disebut pencocokan *string* (*string matching*) sangat banyak digunakan dalam pemrograman perangkat lunak. Bahkan hampir setiap perangkat lunak mengandung algoritma pencarian *string*. Contoh yang sederhana yaitu membaca masukan dari pengguna. Dalam bahasa pemrograman tingkat tinggi kita bisa menemukan fungsi atau *method* untuk mencocokkan *string*. Namun, penulis tidak menggunakannya untuk menjalankan fungsi utama dari perangkat lunak ini. Penulis membuat algoritma dengan berdasarkan kepada referensi yang penulis dapatkan, yaitu algoritma pencarian *string* Knuth-Morris-Pratt.

Algoritma yang paling sederhana dalam permasalahan pencarian *string* adalah algoritma brute force. Algoritma ini memang algoritma paling dasar dalam berbagai permasalahan. Dari penelusuran algoritma brute force kita bisa menemukan algoritma yang lebih mangkus, seperti yang dilakukan oleh Donald E. Knuth, Joseph H. Morris, dan V. R. Pratt dalam mengembangkan algoritma untuk pencarian *string*.

Banyak langkah-langkah yang dilakukan algoritma brute force dalam menemukan solusi permasalahan tidak berguna atau sia-sia. Dalam algoritma brute force, tiap kali ditemukan ketidakcocokan dalam suatu langkah, *pattern* (*string* yang dicari) hanya digeser satu karakter ke belakang. Berbeda dengan algoritma KMP yang menyimpan informasi yang sangat diperlukan dalam melakukan pergeseran *string pattern* agar proses pencarian dapat dioptimalkan. Dalam hal ini, kita ingin proses tersebut memakan waktu seminimal mungkin. Untuk mengaplikasikannya, algoritma KMP menyimpan informasi tersebut untuk melakukan pergeseran *string* pattern yang lebih jauh, tidak hanya satu karakter seperti pada algoritma *brute force*. Dengan algoritma ini, waktu pencarian dapat dikurangi secara signifikan [3].

2. METODE

2.1 Algoritma KMP

Algoritma KMP melakukan proses awal (*preprocessing*) terhadap *pattern* P dengan menghitung fungsi pinggiran. Pada beberapa literatur disebut fungsi *overlap*, fungsi *failure,* fungsi awalan, dan sebagainya. Fungsi ini mengindikasikan pergeseran P terbesar yang mungkin dengan menggunakan perbandingan yang dibentuk sebelum pencarian *string* [3]. Dengan demikian,

kita bisa melewati pergeseran atau perbandingan *string* yang tidak berguna, seperti pada algoritma *brute force*.

Fungsi pinggiran dihitung hanya berdasarkan kepada karakter-karakter dalam *pattern*, tidak menyertakan karakter-karakter dalam teks (*string* target). Fungsi pinggiran *b(j)* didefinisikan sebagai ukuran awalan terpanjang dari *pattern P* yang merupakan akhiran dari *P[1..j]* [3]. Untuk lebih jelasnya, berikut ini diberikan sebuah contoh untuk mengitung fungsi piggiran dari sebuah *pattern P* = `xlnxls`. Sebagai catatan, penulis menggunakan nilai 0 (nol) sebagai indeks awal *string* pada permasalahan ini.

Awalan dari *P* adalah
□, x, xl, xln, xlnx, xlnxl

Akhiran dari *P* adalah
□, s, ls, xls, nxls, lnxls

Keterangan : □ = *string* kosong

Nilai fungsi pinggiran *b(j)* untuk setiap karakter dalam *P* adalah

Tabel 1 Fungsi pinggiran

j	0	1	2	3	4	5
P[j]	x	l	n	x	l	s
b(j)	0	0	0	1	2	0

Sebelum menghitung fungsi pinggiran, penulis mendefinisikan terlebih dulu larik penampung nilai fungsi pinggiran, yaitu

```
b : array of integer
```

Agoritma menghitung fungsi pinggiran itu sendiri adalah sebagai berikut.

```
procedure HitungPinggiran(input m :
integer, P : string)
{ Menghitung nilai pinggiran b[1..m]
untuk pattern P[1..m] }

Kamus :
  k, g : integer

Algoritma :
  b[] = array[1..m] of integer
  b[0] = 0
  q = 1
  k = 0

  for q ← 1 to m-1 do
    while ((k > 0) and (P[q] ≠ P[k]))
do
      k ← b[k]
    endwhile

    if P[q] = P[k] then
      k ← k+1
    endif

    b[q] ← k
  endfor
```

Sekarang kita tinjau percobaan mencocokkan *pattern P* tadi dengan teks *T* = `xlnxlnxls`.

```
i  :  0  1  2  3  4  5  6  7  8
T  :  x  l  n  x  l  n  x  l  s

j  :  0  1  2  3  4  5
P  :  x  l  n  x  l  s
```

Langkah pertama yaitu bandingkan ujung kiri pattern P dengan ujung kiri teks T. Karakter-karakter pada posisi 0 sampai dengan 4 sama (*match*), tetapi pada posisi $i = j = 5$ terjadi ketidakcocokan, `n` pada teks *T* dengan `s` pada pattern *P*. Karena terjadi ketidakcocokan, kita lakukan pergeseran pattern *P* dengan jumlah pergeseran sesuai dengan nilai pinggiran dari awalan *pattern P* yang *match.* Pada kasus ini, awalan yang *match* adalah `xlnxl` dengan panjang $l = 5$. Nilai pinggiran yang terpanjang untuk *string P[0..4]* adalah *b(4)* = 2. Jumlah pergeseran adalah $l - b = 5 - 2 = 3$. Jadi, *pattern P* digeser sejauh 3 karakter ke kanan dan perbandingan selanjutnya dilakukan mulai pada posisi $j = l - b - 1 = 5 - 2 - 1 = 2$.

```
i  :  0  1  2  3  4  5  6  7  8
T  :  x  l  n  x  l  n  x  l  s

         j  :  0  1  2  3  4  5
         P  :  x  l  n  x  l  s
```

Algoritma KMP selengkapnya adalah sebagai berikut.

```
function KMPSearch(input m : integer,
n : integer, P : string, T string) →
boolean
{ Mengembalikan true jika pattern P
ditemukan dalam teks T }

Kamus :
  i, j : integer
  ketemu : boolean
```

```
procedure HitungPinggiran(input m :
integer, P : string)
{ Menghitung nilai pinggiran b[1..m]
untuk pattern P[1..m] }

Algoritma :
  HitungPinggiran(m, P)
  i ← 0
  j ← 0
  ketemu = false

  while ((j > 0) and (P[j] ≠ T[i]))
do
    j ← b[j-1]
  endwhile

  if P[j] = T[i] then
    j ← j+1
  endif

  if j = m then
    ketemu ← true
  else
    i ← i+1
  endif

  return ketemu
```

Dengan demikian, untuk menghitung fungsi pinggiran, algoritma KMP membutuhkan waktu *O(m)* dan untuk proses pencarian *string* membutuhkan waktu *O(n)*. Jadi, kompleksitas waktu algoritma ini sebesar *O(m+n)*.

2.2 Implementasi pada Program

Penulis menggunakan kakas pemrograman Visual Studio 2005 Professional Edition dengan bahasa pemrograman C#. Untuk mencari nama berkas tentunya kita harus membaca satu per satu nama berkas yang ada di tempat atau direktori yang kita inginkan. Untuk permasalahan ini, kakas yang digunakan telah mempunyai solusinya, yaitu dengan menggunakan *recursively search directories* [1].

Ada dua tingkat rekursif di sini, yaitu rekursif untuk mendapatkan nama direktori

```
foreach (string d in
Directory.GetDirectories(namaDirektori)
)
```

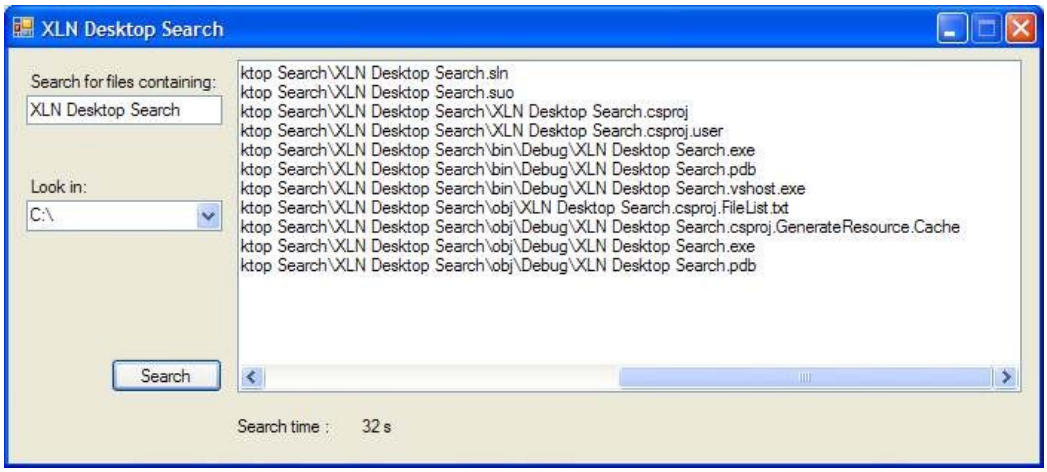
dan rekursif untuk mendapatkan nama berkas

```
foreach (string f in
Directory.GetFiles(d, namaBerkas))
```

[1].

Untuk menghindari terjadinya *error* pada program, penulis memasukkan dua sintaks tersebut ke dalam blok `try` dan untuk penanganan *exception*-nya menggunakan `System.Exception` tanpa menampilkan pesan kesalahannya.

Tentunya sebuah program harus diuji coba terlebih dulu. Penulis pun menguji coba program ini dalam beberapa kasus. Berikut ini salah satu kasus yang telah diuji coba beserta *screenshot*-nya.



Gambar 1 *Screenshot* XLN Desktop Search[1]

[1] XLN Desktop Search bisa diperoleh di http://students.if.itb.ac.id/~if15058.

Spesifikasi *PC* yang digunakan untuk uji coba:

Tabel 2 Spesifikasi *PC* uji coba

Prosessor	Intel P4 LGA775 2.6 GHz
DDR	Team 256 MB
Harddisk	Seagate SATA 80 GB
VGA	Cardex Pro nVIDIA GeForce4 MX 4000 64 MB
Sistem operasi	Windows XP Professional SP2

Dalam kasus ini penulis memasukkan *pattern* = XLN Desktop Search dan memasukkan direktori C:\ sebagai direktori akar tempat berkas yang akan dicari. Direktori ini memiliki kapasitas 23,4 GB dan ruang kosong 13,8 GB. Berarti ruang yang terpakai sebesar 9,6 GB. Hasilnya, waktu yang diperlukan program untuk melakukan proses ini hanya sebesar 32 detik. Algoritma KMP memiliki kompleksitas waktu yang berdasarkan kepada panjang *string pattern* maupun teks, sehingga semakin panjang *string* tersebut, semakin lama pula waktu yang diperlukan.

3. KESIMPULAN

Algoritma serba bisa *brute force* merupakan dasar dari algoritma yang lebih mangkus. Semua permasalahan dapat diselesaikan dengan menggunakan agoritma ini. Dari algoritma ini, banyak ilmuwan yang berhasil menemukan algoritma yang lebih mangkus, seperti algoritma pencarian *string* Knuth-Morris-Pratt yang dibahas pada makalah ini.

Penulis menerapkan algoritma KMP pada sebuah aplikasi pencarian berkas di komputer. Aplikasi ini ternyata sangat bagus. Terbukti waktu yang diperlukan untuk menjalankannya sangat rendah dalam beberapa uji coba yang dilakukan oleh penulis. Bahkan, lebih cepat daripada fitur pencarian berkas bawaan Windows XP sendiri meskipun program yang penulis kembangkan hanya mencari berkas dan menampilkan nama berkas yang ditemukan lengkap dengan *path*-nya.

Semoga program sederhana ini bisa dikembangkan agar bisa lebih berguna lagi. Entah dengan menambah fitur atau mengembangkan algoritma yang lebih mangkus. Sebagai penutup, penulis menyampaikan terima kasih kepada semua pihak yang telah mendukung pengembangan program XLN Desktop Search dan penulisan makalah ini. Terima kasih juga kepada para pembaca dan pengguna XLN Desktop Search.

REFERENSI

[1] http://msdn2.microsoft.com. Diakses pada tanggal 9 Mei 2007 pukul 15.00 WIB.

[2] http://www.cs.cuw.edu. Diakses pada tanggal 22 Mei 2007 pukul 22.00 WIB.

[3] Munir, Rinaldi. *Diktat Kuliah IF2251 Strategi Algoritmik*. Bandung: Program Studi Teknik Informatika, Sekolah Teknik Elektro dan Informatika, ITB. 2007.