Algoritma Pencarian *String* Knuth-Morris-Pratt Dalam Pengenalan Tulisan Tangan

Andri Rizki Aminulloh

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung
Jalan Ganesha No 10 Bandung Jawa Barat
e-mail: andrizki@students.itb.ac.id

ABSTRAK

Makalah ini akan membahasa tentang pengenalan tulisan tangan(*handwriting recognition)* yang dilakukan oleh sebuah komputer atau sebuah *gadget* sebagai aplikasi dari algoritma pencarian *string* Knuth-Morris-Pratt. Pengenalan tulisan tangan adalah teknologi yang sudah lama dikembangkan oleh para ahli di bidang teknologi informasi. Tetapi beberapa tahun belakangan teknologi ini kembali menjadi sorotan seiring semakin berkembang dan digemarinya *device* yang dilengkapi dengan layar sentuh(*touchscreen*). Dengan adanya layar sentuh pada *device*, seperti PDA, *tablet* PC, bahkan ponsel, maka sangat memungkinkan bagi *device* tersebut untuk mengenali tulisan tangan. Ditambah lagi penggunaan *keyboard* saat ini sudah dianggap kurang praktis oleh beberapa pengguna *device-device* ini terutama pada PDA dan ponsel. Algoritma Knuth-Morris-Pratt untuk pencarian string sendiri adalah algoritma pencarian string yang cukup populer dan banyak digunakan oleh para pengembang perangkat lunak. Algoritma ini sangat mangkus dan sangkil sehingga sangat baik untuk digunakan sebagai solusi dalam pencarian string.

Kata kunci: . Pencarian *string,* pengenalan tulisan tangan

1. PENDAHULUAN

Pengenalan tulisan tangan (*handwriting* recognition) pertama kali dikenal pada awal tahun 1980-an dimana pada saat itu seorang bernama Dr. Charles Elbaum mendirikan sebuah perusahaan bernama "Nestor" dan membuat sebuah alat bernama "NestorWriter handwriting recognizer" dan mendapatkan nobel dari hasil karyanya ini. Penemuan ini menimbulkan euphoria tersendiri, banyak orang mengganggap bahwa penggunaan alat ini untuk komputasi, yang selanjutnya disebut *pen computing*, dapat menggantikan penggunaan komputer dengan *keyboard* dan *mouse* konvensional dan akan semakin banyak orang yang akan akrab dengan komputer karena lebih banyak orang yang bisa menggunakan pena(*pen*) daripada yang bisa menggunakan *keyboard*.

Pada tahun 1991 *pen computing* berada pada puncaknya. Sebuah pena(yang digunakan untuk *handwriting recognition*) dianggap sebagai tandingan untuk mouse, dan *pen computer*(komputer yang menggunakan teknologi *handwriting recognition*) akan bisa mengalahkan *desktop*. Hal ini tentu saja mengancam Microsoft yang jelas-jelas mengembangkan Windows untuk *desktop*. Microsoft pun akhirnya mau tidak mau mengikuti euphoria ini dengan meluncurkan sebuah *extension pack* untuk *pen computing*. Setelah peluncuran ini perusahaan-perusahaan produsen *hardware* berlomba-lomba menciptakan alat pengenal tulisan tangan yang *compatible* dengan versi Windows ini.

Namun ternyata pada tahun 1992, perusahaan-perusahaan ini harus menelan pil pahit dan mengalami kerugian setelah mengetahui kenyataan bahwa produk yang mereka tawarkan tidak terjual. *Pen Computing* ternyata tidak digemari oleh pengguna komputer. Beberapa orang beropini hal ini dikarenakan sulitnya penggunaan *hardware* pengenal tulisan ini, selain itu ada yang juga berpendapat bahwa hal ini karena banyak kekurangan pada teknologi ini dan terkadang *hardware* yang mereka gunakan tidak berfungsi sama sekali.

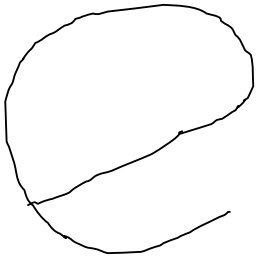
Euphoria pun meredup dan produsen-produsen tidak lagi memproduksi *pen comp[uters* maupun *hardware* yang mendukungnya. Sampai akhirnya pada tahun 2002 dunia teknologi komputer terhenyak pada peluncuran *tablet PC* oleh microsoft. Bill Gates ternyata masih percaya bahwa teknologi ini bisa berkembang, dan ia benar, *tablet PC* menjadi fenomena.

Pen Computing akhirnya terus berkembang sampai saat ini, bahkan penggunaanya telah meluas sampai ke level ponsel. Saat ini *device* dengan *touchscreen*(*hardware*

yang memungkinkan input pada layar) sangat diminati dan hampir semua perusahaan teknologi informasi di dunia berusaha mengembangkannya.

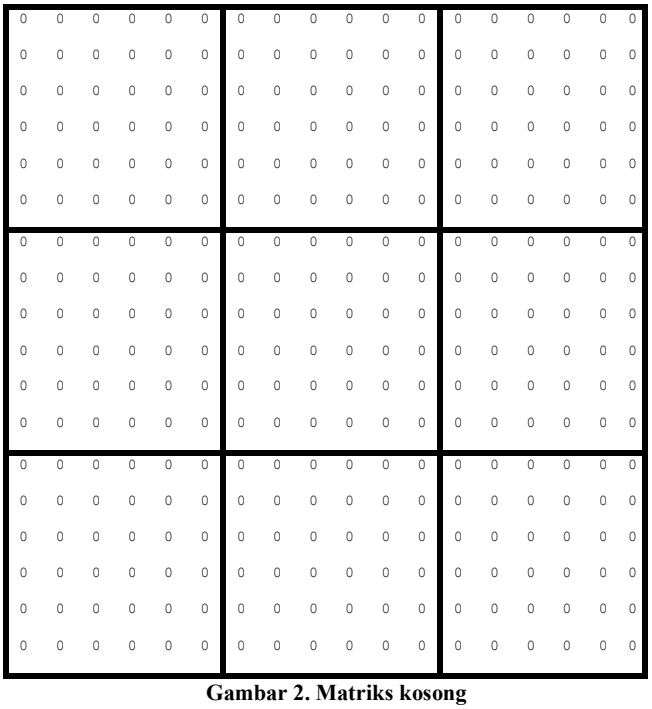
2. PENGENALAN TULISAN TANGAN (*HANDWRITING RECOGNITION*)

Seperti yang telah disebutkan sebelumnya pengenalan tulisan tangan memungkinkan karena adanya *touch screen*. *Touchscreen* mampu mengirimkan informasi posisi tempat dirinya disentuh. Informasi posisi ini kemudian direpresentasikan menjadi matriks vektor. Misalkan seorang *user* menuliskan huruf 'e' seperti berikut:



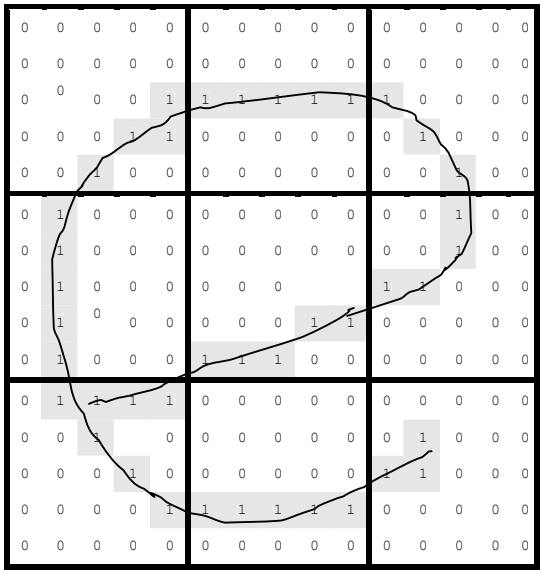
Gambar 1. Contoh tulisan tangan

Sebelumnya telah layar akan direpresentasikan seperti matriks kosong seperti berikut:



Gambar 2. Matriks kosong

Kemudian hasil pencitraan(pencatatan posisi) tulisan yang digoreskan akan tercatat ke dalam matriks. Bagian yang tergores akan berubah angka menjadi 1. seperti berikut:

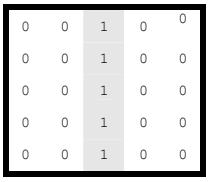


Gambar 3. Matriks hasil pencitraan

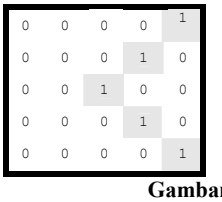
Jika anda perhatikan matriks diatas terdiri dari 9 upa-matriks. Langkah selanjutnya adalah mengubah tiap-tiap upa-matriks diatas menjadi list, contoh upa-matriks pertama diubah menjadi seperti berikut:

000000000000000000000001000011000100

List ini kemudian dicocokkan dengan *pattern-pattern* yang sudah disimpan sebelumnya. *Pattern-pattern* mewakili bentuk-bentuk dasar dari huruf-huruf. Contoh :



100001000010000100000100



10001000100000100000 1

Gambar 4. Contoh *pattern* tersimpan

Jika hasil pencocokkan sama maka list tersebut disimpan, jika tidak maka list tersebut akan dibah menjadi *pattern* yang paling mirip / hampir sama. Lalu list tersebut disimpan. Begitu seterusnya sampai upa-matriks ke sembilan. List-list yang disimpan ini kemudian dibuat kombinasi berdasarkan *pattern* kemudian kombinasi ini akan menjadi sebuah *string*, misal list pertama adalah *pattern* a , list kedua adalah c, ketiga e, keempat m, kelima n, keenam y, ketujuh g, kedelapan p, maka akan terbentuk *string*

a c e m n y g

String ini kemudian dicocokkan lagi dengan *string* baku sebuah huruf yang terdapat di dalam memori sebuah *device*.

Ada cara lain dalam pengenalan tulisan tangan ini yaitu mengubah langsung seluruh matriks menjadi sebuah list. Untuk contoh diatas maka akan menjadi:

```
00000000000000000000000000000000011111110000000110000001000001000000000100010000000001000100000000001000100000000011000010000001100000010001110000000001111000000000001000000001000000010000001100000011111110000000000000000000
```

Kemudian list ini akan langsung dicocokkan dengan *pattern-pattern* kemungkinan cara penulisan huruf yang telah disimpan di dalam memori.

3. ALGORITMA KNUTH-MORRIS-PRATT (KMP)

3.1 Pengertian

Algoritma Knuth-Morris-Pratt adalah salah satu algoritma pencari string yang dikembangkan oleh D.E. Knuth bersama dua orang rekannya J. H. Morris dan V. R. Pratt. Dengan menggunakan algoritma ini kita dapat mencari sebuah sub-string dari sebuah string dengan cara mencocokkan masing-masing karakter. Berbeda dengan algoritma *brute force* dimana jika pada saat pencocokkan *pattern* yang dicocokkan tidak sesuai maka bergeser satu karakter ke kanan, pada algoritma KMP sebelum pencocokkan dimulai algoritma ini menyimpan informasi yang dapat digunakan untuk mengetahui dimana pencocokkan berikutnya akan dilakukan, sehingga pergeseran yang tidak perlu tidak akan terjadi.[1]

Sebagai contoh, misalkan kita akan mencari *pattern* ‘al’ di dalam teks ‘alasan algoritma mangkus’:

Teks : a l a s a n a l g o r i t m a m a n g k u s
Pattern : a l p h a

Dapat terlihat bahwa pencocokkan akan gagal di karakter ke-3. Jika kita melakukan pencarian dengan *brute force* maka pencocokkan selanjutnya akan dilakukan mulai karakter ke-2. Tetapi dengan algoritma KMP yang akan terjadi adalah sebagai berikut:

Teks : a l a s a n a l g o r i t m a m a n g k u s
Pattern : a l p h a

Pencocokkan langsung dimulai dari karakter yang paling mungkin berhasil. Selain itu pencocokan juga langsung dimulai dari karakter *pattern* ke-2 karena pencocokkan karakter pertama telah dilakukan saat pengambilan informasi.

Secara definitif algoritma KMP dapat dituliskan seperti berikut:

Misalkan A adalah alfabet dan $x = x_1x_2 \ldots x_4$, $k \in$ N adalah *string* yang panjangnya k yang dibentuk dari karakter-karakter di dalam alfabet A.[1]
Awalan dari x adalah *substring* u dengan

$$u = x_1x_2 \ldots x_4,\ k \in \{1, 2, ..., k-1\}$$

dengan kata lain, x diawali dengan u.

Akhiran dari x adalah *substring* u dengan

$$u = x_{k-b}\, x_{k-b+1} \ldots x_k,\ k \in \{1, 2, ..., k-1\}$$

dengan kata lain, x diakhiri u.

Awalan u dari x atau akhiran u dari x disebut awalan atau akhiran sebenarnya (*proper*) jika $u \neq x$, yaitu panjangnya, b, lebih kecil daripada k.

Pinggiran (*border*) dari x adalah *substring* r sedemikian sehingga

$$r = x_1x_2 \ldots x_{k-1} \text{ dan}$$

$$r = x_{k-b}\, x_{k-b+1} \ldots x_k,\ k \in \{1, 2, ..., k-1\}$$

dengan kata lain, pinggiran dari x adalah *substring* yang keduanya awalan dan juga akhiran sebenarnya dari x.

3.2 Fungsi Pinggiran

Menghitung fungsi pinggiran adalah langkah awal dan langkah terpenting dari algoritma KMP, karena fungsi inilah kunci dari kemangkusan algoritma ini Hasil perhitungan fungsi pinggiran akan mengindikasikan

pergeseran yang harus dilakukan pattern. Dengan adanya fungsi pinggiran ini pergeseran yang tidak perlu dapat dicegah.

Fungsi pinggiran ini hanya bergantung pada karakter-karakter di dalam *pattern*, oleh karena itu kita dapat melakukan perhitungan sebelum proses pencarian string dimulai.

Adapun cara penghitungan fungsi pinggiran adalah sebagai berikut. Kita ambil contoh *pattern* 'ababc', kita dapat menghitung fungsi pinggiran *b*(*j*)

j	1	2	3	4	5
P[*j*]	A	b	a	b	c
b(*j*)	0	0	1	2	2

Lalu mari kita coba gunakan hasil penghitungan ini untuk pencarian dalam teks 'abababcd'.

Teks : a b a b a b c d
Pattern : a b a b c

Percobaan pertama yang terjadi adalah ujung kiri *pattern* dan ujung kiri teks disamakan. Karakter pada posisi 1...4 sama, tetapi karakter pada posisi 5 tidak sama. Percobaan selanjutnya ditentukan oleh pinggiran dari awalan *pattern* yang bersesuaian. Dapat dilihat bahwa awalan yang bersesuaian pada contoh diatas adalah 'abab', panjangnya adalah $l = 4$. Lalu pinggiran yang terpanjang dari awalan tersebut adalah ab yang panjangnya $b(4) = 2$, maka jarak pergeseran adalah $l - b = 4 - 2 = 2$. Jadi, *pattern* akan digeser 2 karakter langsung dan pencocokan akan langsung dimulai dari karakter ke-2 dari awal *pattern*.

Teks : a b a b a b c d
Pattern : a b a b c

Algoritma selengkapnya:

```
procedure KMPsearch(input m, n : integer,
                    input P : array[1..m] of
                    char, input T :
                    array[1..n] of char,
                    output idx : integer)
{ Mencari kecocokan pattern P di dalam teks T
dengan algoritma Knuth-Morris-Pratt (KMP). Jika
ditemukan P di dalam T, lokasi awal kecocokan
disimpan di dalam peubah idx.       Masukan:
pattern P yang panjangnya m dan teks T yang
panjangnya n.Teks T direpresenasikan sebagai
string (array of character)
Keluaran: posisi awal kecocokan (idx). Jika
P tidak ditemukan, idx = -1. }
 Deklarasi
  i, j : integer
  ketemu : boolean
  b : array[1..m] of integer

procedure HitungPinggiran(input m : integer,
P : array[1..m] of char, output b :
array[1..m] of integer)
  { Menghitung nilai b[1..m] untuk pattern


P[1..m] }

Algoritma:
   HitungPinggiran(m, P, b)
  j←0
  i←1
  ketemu←false
  while (i ≤ n and not ketemu) do

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

    if P[j+1]=T[i] then
      j←j+1
    endif
    if j = m then
      ketemu←true
    else
      i←i+1
    endif
 endwhile
 if ketemu then
   idx←i-m+1  { catatan: jika indeks array
dimulai dari 0, maka idx←i-m }
 else
   idx←-1
 endif
```

3.3 Kompleksitas Waktu

Kompleksitas waktu dari algoritma KMP ditentukan oleh waktu yang dibutuhkan saat penghitungan fungsi pinggiran dan pada saat pencarian string.

Untuk menghitung fungsi pinggiran dibutuhkan waktu $O(m)$, sedangkan untuk pencarian string dibutuhkan waktu $O(n)$, maka kompleksitas waktu dari algoritma KMP adalah $O(m+n)$.

4. ALGORITMA KNUTH – MORRIS – PRATT (KMP) DALAM PENGENALAN TULISAN TANGAN

Seperti yang telah tertulis diatas bahwa dalam pengenalan tulisan tangan banyak sekali ditemukan kasus

pencocokan string. Kasus-kasus inilah yang dapat diselesaikan dengan algoritma KMP dengan mangkus dan sangkil.

Kita ambil contoh kasus pencocokan sebuah upa-matriks dengan sebuah pattern. Mula-mula kita hitung funsi pinggiran dari sebuah *pattern* dengan string : 100010001000001000001 sesuai dengan algoritma yang telah dijelaskan sebelumnya.

j	1	2	3	4	5	6	7	8	9
P[*j*]	1	0	0	0	1	0	0	0	1
b(*j*)	0	0	0	0	1	2	3	4	5

.

j	10	11	12	13	14	15	16	17	18
P[*j*]	0	0	0	0	0	1	0	0	0
b(*j*)	6	7	8	0	0	1	2	3	4

j	19	20	21
P[*j*]	0	0	1
b(*j*)	0	0	1

Setelah fungsi pinggiran dihitung, pencarian dapat dimulai. Misalkan pencarian dilakukan pada hasil pencitraan dengan string:

000100010001000100010000010000010000

Pencarian pertama akan terjadi seperti berikut:

String : 000100010001000100010000010000010000
Pattern : 100010001000001000001

Ketidakcocokkan langsung terjadi pada pencocokkan karakter pertama, sehingga awalan yang bersesuaian adalah $l = 0$ dan pinggiran $b = -1$ (sesuai algoritma, lihat *pseudo code*). Maka pergeseran yang terjadi $l - b = 0 - (-1) = 1$. Dilihat dari string yang dicocokkan hal ini akan terjadi sampai karakter ke-3 dimana setelah itu keadaan akan menjadi seperti berikut:

String : 000100010001000100010000010000010000
Pattern : 100010001000001000001

Pada percobaan kali ini terjadi kecocokkan pada 12 karakter pertama kemudian ketidakcocokkan pada karakter ke 13, sehingga tercipta awalan dengan panjang $l = 12$ dan pinggiran dengan panjang b = 8. Maka untuk percobaan berikutnya *pattern* akan bergeser sebanyak $l - b = 12 - 8 = 4$ karakter. Sehingga keadaan yang terbentuk akan menjadi seperti berikut:

String : 000100010001000100010000010000010000
Pattern : 100010001000001000001

Pada percobaan ini hal yang sama seperti sebelumnya terjadi kembali maka, *pattern* akan bergeser lagi sebanyak 4 karakter menjadi seperti berikut:

String : 000100010001000100010000010000010000
Pattern : 100010001000001000001

Pada percobaan ini ternyata terjadi kecocokan pada seluruh pattern maka pencarian pun selesai.

4. KESIMPULAN

Adanya fungsi pinggiran dalam algoritma pencocokan *string* Knuth-Morris-Pratt membuat algoritma ini dapat menyelesaikan pencarian dengan waktu yang lebih singkat dari yang dibutuhkan oleh algoritma *brute force*.

Algoritma pencocokan *string* dapat digunakan untuk menyelesaikan banyak persoalan sejauh persoalan tersebut dapat direpresentasikan ke dalam sebuah *string*, tidak hanya untuk pencarian kata dan semacamnya,.

REFERENSI

[1] Rinaldi Munir, "Diktat Kuliah Strategi Algoritmik", Program Studi Teknik Inofrmatika ITB, 2006.

[2] http://en.wikipedia/wiki/Knuth_Morris_Pratt_algoritm, diakses pada tanggal 19 Mei 2008 pukul 23:00

[3] http://www.webpronews.com/expertarticles/2005/12/20/a-brief-history-of-tablet-pcs, diakses pada tanggal 20 Mei 2008 pukul 09.00

[4] http://en.wikipedia.org/wiki/Handwriting_recognition, diakses pada tanggal 20 Mei 2008 pukul 08.00

[5] http://en.wikipedia.org/wiki/Knuth-Morris-Pratt_algorithm, diakses pada 20 Mei 2008 pukul 22.00

[6] http://www.dontveter.com/basisofai/char.html diakses pada tanggal 20 Mei 2008 pukul 09.00