Pencocokan *String* dalam Fitur *Autocompletion* pada *Text Editor* atau *Integrated Development Environment (IDE)*

Muhammad Wachid Kusuma 13510074[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13510074@itb.ac.id

***Abstrak*—Text Editor dan Integrated Development Environment (IDE) mempunyai banyak fasilitas yang disediakan oleh pengembangnya. Salah satunya adalah fasilitas *autocompletion* atau *autocomplete* yang berguna memberikan prediksi kata terhadap karakter yang pengguna tuliskan. Fasilitas ini menggunakan algoritma pencocokan *string* dalam mencari prediksi dari karakter yang diketikkan oleh pengguna. Algoritma pencocokan *string* yang digunakan ada banyak jenisnya.**

Kata kunci—algoritma, *IDE*, pencocokan string, *text editor*

I. PENDAHULUAN

Dalam dunia pengembangan perangkat lunak, kita mengenal kakas yang berupa *text editor* dan *Integrated Development Environment (IDE)*. Kakas ini digunakan untuk menulis *source code* program ketika mengembangkan sebuah perangkat lunak. Tiap-tiap *text editor* dan *Integrated Development Environment (IDE)* mempunyai fasilitas/fitur yang disediakan oleh pengembangnya untuk mempermudah pengguna dalam menulis *source code* program dalam mengembangkan suatu perangkat lunak.

Fasilitas/fitur dalam *text editor* dan *Integrated Development Environment (IDE)* ada banyak macamnya. Mulai dari *autosave*, *syntax highlighting*, *autoindentation*, hingga *autocomplete* atau *autocompletion*. *Autosave* adalah fasilitas/fitur yang menyimpan *source code* program secara otomatis setelah jangka waktu tertentu. *Syntax highlighting* merupakan fasilitas/fitur yang akan memberikan warna berbeda pada kata atau kode yang merupakan *reserved words* dari bahasa pemrograman yang dipakai pengguna dalam menulis *source code*.

Autoindentation adalah fasilitas/fitur yang akan mengotomasi indentasi *source code* program. Biasanya indentasi akan muncul ketika *source code* memasuki blok baru. Sedangkan *autocomplete* atau *autocompletion* merupakan fasilitas/fitur ini merupakan alat bantu yang disediakan untuk membantu pengguna dalam menulis *source code* program dengan memberikan prediksi kata atau kode apa yang akan pengguna tulis sehingga pengguna dapat meminimalisasi pengetikan karakter yang membentuk kata atau kode tersebut.

Dalam makalah ini, penulis akan memfokuskan penggunaan salah satu algoritma pencocokan *string* dalam pengembangan fasilitas/fitur *autocomplete* atau *autocompletion* ini. Penggunaan algoritma pencocokan string dalam pengembangan fasilitas/fitur ini nantinya tidak hanya membahas tentang implementasi algoritma pencocokan *string*-nya saja namun juga struktur data dari fasilitas/fitur *autocomplete* atau *auto completion* ini.

II. DASAR TEORI

A. *TEXT EDITOR*

Text Editor merupakan tipe kakas yang digunakan untuk mengedit *file* teks yang berjenis *plain text*. *Text editor* sering tersedia di dalam sistem operasi atau paket pengembangan perangkat lunak.

Text editor dapat mengubah *file* konfigurasi dan *source code* suatu bahasa pemrograman. *Text editor* berbeda dengan *word processor*. *Text editor* menghasilkan *plain text* yang menggunakan set karakter yang sederhana seperti ASCII. *Word processor* menghasilkan *text file* yang telah terformat.

Menyimpan *plain text* di *word processor* akan menambah informasi pemformatan tambahan. Sedangkan menyimpan *formatted text* dalam *text editor* dapat menghilangkan informasi pemformatan.

Contoh dari *text editor* adalah Vim, Notepad, Notepad++.

B. *INTEGRATED DEVELOPMENT ENVIRONMENT (IDE)*

Integrated Development Environment (IDE) adalah aplikasi perangkat lunak yang menyediakan fasilitas yang komprehensif untuk pengembangan perangkat lunak. Sebuah *Integrated Development Environment (IDE)* secara normal terdiri atas *source code editor*, *build automation*, serta *debugger*. Beberapa *Integrated Development Environment (IDE)* mengandung *compiler*, *interpreter*, ataupun keduanya seperti Microsoft Visual Studio dan Eclipse.

Integrated Development Environment (IDE) didesain untuk memaksimalkan produktivitas *programmer* dengan menyediakan komponen yang terintegrasi dengan *user interface* yang serupa. *Integrated Development*

Environment (IDE) menyediakan sebuah program yang mana semua pengembangannya sudah selesai.

Program ini menyediakan banyak fasilitas/fitur untuk menciptakan, memodifikasi, meng-*compile*, men-*deploy*, serta men-*debug* sebuah perangkat lunak.

Beberapa *Integrated Development Environment (IDE)* hanya didedikasikan untuk bahasa pemrograman yang spesifik seperti Free Pascal (FPC) tetapi banyak juga *Integrated Development Environment (IDE)* yang dapat untuk mengembangkan perangkat lunak dalam beberapa bahasa pemrograman seperti Microsoft Visual Studio, Eclipse, NetBeans, dan Oracle JDeveloper.

C. *AUTOCOMPLETE* ATAU *AUTOCOMPLETION*

Autocomplete atau *autocompletion* atau *word completion* adalah fasilitas yang disediakan oleh berbagai *web browser*, *email-programs*, *search engine interface*, *source code editors*, *database query tools*, *word processor*, dan *command line interpreters*. *Autocomplete* juga tersedia untuk atau sudah terintegrasi di dalam *text editor*.

Autocomplete atau *autocompletion* melibatkan program yang dapat melakukan prediksi terhadap sebuah kata atau frasa yang pengguna ingin tulis tanpa harus menulis keseluruhan kata atau frasa secara lengkap. Fasilitas/fitur ini efektif ketika proses prediksi itu mudah saat kata yang diprediksi berdasarkan pada kata yang telah ditulis sebelumnya.

Autocomplete atau *autocompletion* bekerja ketika penulis menulis huruf pertama atau beberapa huruf/karakter dari sebuah kata. Program yang melakukan prediksi akan mencari satu atau lebih kemungkinan kata sebagai pilihan. Jika kata yang dimaksud ada dalam pilihan itu, maka penulis dapat memilih itu. Jika kata yang dimaksud tidak ada dalam pilihan prediksi maka penulis harus menulis huruf/karakter selanjutnya. Ketika penulis memilih pilihan kata yang ada dalam daftar pilihan kata prediksi maka kata yang dipilih tersebut akan disisipkan pada teks.

Dalam *source code editor*, *autocomplete* atau *code completion* menyederhanakan struktur regular dari sebuah bahasa pemrograman. Biasanya hanya ada sejumlah kata yang berarti dalam konteks saat ini atau *namespace*, seperti nama dari sebuah variable atau nama dari sebuah fungsi. Contoh dari *code completion* adalah desain IntelliSense dari Microsoft. Dia menampilkan sebuah *pop-up list* yang berisi prediksi yang mungkin dari karakter yang dimasukkan oleh pengguna untuk dipilih pengguna yang sesuai dengan apa yang diinginkan pengguna.

Fitur ini sangat berguna dalam pemrograman berorientasi objek karena seringkali *programmer* tidak tahu secara benar *member* apa saja yang sebuah *class* punya.

D. ALGORITMA PENCOCOKAN STRING

1. Algoritma *Brute Force*

Algoritma *brute force* merupakan algoritma pencocokan *string* yang ditulis tanpa memikirkan performansi dari algoritma itu sendiri. Algoritma ini sangat jarang dipakai dalam praktik, namun berguna dalam studi pembanding dan studi-studi lainnya.

Secara sistematis, langkah-langkah yang dilakukan algoritma *brute force* pada saat mencocokkan *string* adalah :

a. Algoritma *brute force* mulai mencocokkan pattern pada awal teks.

b. Dari kiri ke kanan, algoritma ini akan mencocokkan karakter per karakter pattern dengan karakter di teks yang bersesuaian, sampai salah satu kondisi berikut dipenuhi:

i. Karakter di pattern dan di teks yang dibandingkan tidak cocok (*mismatch*).

ii. Semua karakter di pattern cocok. Kemudian algoritma akan memberitahukan penemuan di posisi ini.

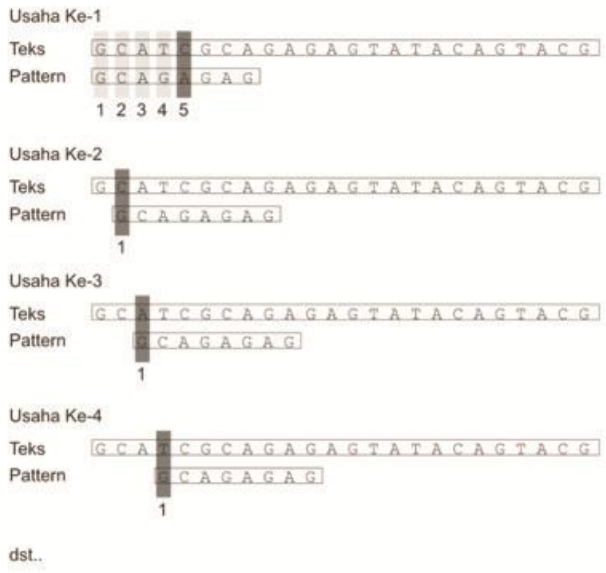
c. Algoritma kemudian menggeser pattern sebesar satu karakter ke kanan dan mengulangi langkah b. sampai pattern berada di ujung teks.

Dalam *pseudocode*, algoritma *brute force* adalah sebagai berikut:

```
procedure BruteForceSearch(
   input m, n : integer,
   input P : array[0..n-1] of char,
   input T : array[0..m-1] of char,
   output ketemu : array[0..m-1] of
boolean
)

Deklarasi:
   i, j: integer

Algoritma:
   for (i:=0 to  m-n) do
      j:=0
      while (j < n and T[i+j] = P[j])
do
         j:=j+1
      endwhile
      if(j >= n) then
         ketemu[i]:=true;
      endif
   endfor
```



Algoritma *brute force* ini mempunyai kompleksitas pada *worst case* sebesar O(mn) di mana m adalah panjang dari teks dan n adalah panjang dari pattern.

2. Algoritma Knuth-Morris-Pratt

Algoritma Knuth-Morris-Pratt adalah salah satu algoritma pencarian string, dikembangkan secara terpisah oleh Donald E. Knuth pada tahun 1967 dan James H. Morris bersama Vaughan R. Pratt pada tahun 1966, namun keduanya mempublikasikannya secara bersamaan pada tahun 1977. Jika kita melihat algoritma brute force lebih mendalam, kita mengetahui bahwa dengan mengingat beberapa perbandingan yang dilakukan sebelumnya kita dapat meningkatkan besar pergeseran yang dilakukan. Hal ini akan menghemat perbandingan, yang selanjutnya akan meningkatkan kecepatan pencarian.

Perhitungan penggeseran pada algoritma ini adalah sebagai berikut, bila terjadi ketidakcocokkan pada saat pattern sejajar dengan $teks[i..i+n-1]$, kita bisa menganggap ketidakcocokan pertama terjadi di antara $teks[i+j]$dan $pattern[j]$, dengan $0 < j < n$. Berarti, $teks[i..i+j-1] = pattern[0..j-1]$ dan $a = teks[i+j]$ tidak sama dengan $b = pattern[j]$. Ketika kita menggeser, sangat beralasan bila ada sebuah awalan v dari pattern akan sama dengan sebagian akhiran u dari sebagian teks. Sehingga kita bisa menggeser pattern agar awalan v tersebut sejajar dengan akhiran dari u.

Secara sistematis, langkah-langkah yang dilakukan algoritma Knuth-Morris-Pratt pada saat mencocokkan string:

a. Algoritma Knuth-Morris-Pratt mulai mencocokkan pattern pada awal teks.

b. Dari kiri ke kanan, algoritma ini akan mencocokkan karakter per karakter pattern dengan karakter di teks yang bersesuaian, sampai salah satu kondisi berikut dipenuhi:

i. Karakter di pattern dan di teks yang dibandingkan tidak cocok (mismatch).

ii. Semua karakter di pattern cocok. Kemudian algoritma akan memberitahukan penemuan di posisi ini.

c. Algoritma kemudian menggeser pattern berdasarkan tabel next, lalu mengulangi langkah 2 sampai pattern berada di ujung teks.

Pseudocode algoritma Knuth-Morris-Pratt pada tahap pra-pencarian adalah:

```
procedure preKMP(
  input P : array[0..n-1] of char,
  input n : integer,
  input/output kmpNext : array[0..n]
of integer
)
Deklarasi:
  i,j: integer

Algoritma
  i := 0;
  j := kmpNext[0] := -1;
  while (i < n) {
     while (j > -1 and not(P[i] =
P[j]))
        j := kmpNext[j];
     i:= i+1;
     j:= j+1;
     if (P[i] = P[j])
        kmpNext[i] := kmpNext[j];
     else
        kmpNext[i] := j;
     endif
  endwhile
```

Sedangkan *pseudocode* algoritma Knuth-Morris-Pratt pada tahap pra-pencarian adalah:

```
procedure KMPSearch(
  input m, n : integer,
  input P : array[0..n-1] of char,
  input T : array[0..m-1] of char,
  output ketemu : array[0..m-1] of
boolean
)

Deklarasi:
  i, j,next: integer
  kmpNext : array[0..n] of interger

Algoritma:
  preKMP(n, P, kmpNext)
  i:=0
  while (i<= m-n) do
    j:=0
    while (j < n and T[i+j] = P[j]) do
       j:=j+1
    endwhile
    if(j >= n) then
```

```
        ketemu[i]:=true;
    endif
    next:= j - kmpNext[j]
    i:= i+next
  endwhile
```

Kompleksitas dari algoritma Knuth-Morris-Pratt adalah O(m+n) di mana m adalah panjang dari teks dan n adalah panjang dari pattern.

III. PEMBENTUKAN *AUTOCOMPLETE* ATAU *AUTO COMPLETION* PADA *TEXT EDITOR ATAU INTEGRATED DEVELOPMENT ENVIRONMENT (IDE)*

Di sini penulis akan menggunakan dua buah algoritma pencocokan *string* untuk membentuk fitur *autocomplete* atau *autocompletion* pada *text editor* atau *Integrated Development Environment (IDE).* Fitur *autocomplete* atau *autocompletion* ini didesain sebagai fitur prediksi terhadap *reserved words*, variable, dan fungsi dari sebuah bahasa pemrograman yang dipakai oleh pengguna ketika sedang menulis *source code* program.

1. Pembentukan *autocomplete* atau *autocompletion* menggunakan algoritma *brute force.*

Fitur *autocomplete* atau *autocompletion* dapat dibuat dengan menggunakan algoritma *brute force*. Pada pembentukan fitur *autocomplete* atau *autocompletion*, struktur data yang diperlukan adalah *array* yang berisi *reserved words* dan nama fungsi dari berbagai bahasa pemrograman yang didukung oleh *text editor* atau *Integrated Development Environment (IDE)* serta daftar kata apa aja yang telah ditulis oleh pengguna. *Array* tersebut sudah terurut secara alfabetik.

Ketika pengguna mengetikkan karakter/huruf pertama, fitur *autocomplete* atau *autocompletion* akan bekerja dengan mencari kata yang berada dalam *array* yang karakter pertamanya sama dengan karakter yang diketikkan oleh pengguna. Setelah semua elemen *array* sudah diperiksa, elemen *array* yang bersesuaian dengan karakter pertama yang diketikkan oleh pengguna ditampilkan dalam *pop-up list*. Proses pemrediksian berulang ketika pengguna mengetikkan karakter kedua, namun *array* yang diperiksa adalah *array* hasil dari pemeriksaan sebelumnya.

```
Procedure completion_bruteforce(string:
pattern, integer: length_pattern){

Kamus
   words : array [1..m] of string
   list : array [1..n] of string
   yes : boolean
   indeks, i : integer

Algoritma
   indeks = 0
   for i = 1 to m do
      yes = words[m] indeks ke 1 sampai
length_pattern sama dengan pattern
      if (yes)
         indeks = indeks + 1
         list[indeks] = words[m]
      endif
   endfor
}
```

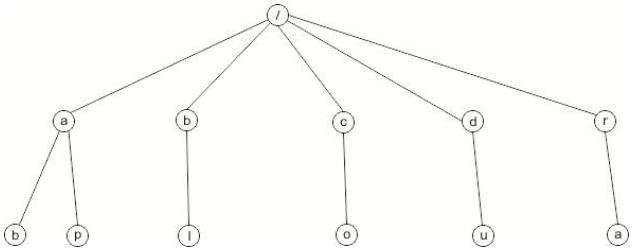
Algoritma di atas memiliki kompleksitas sebesar O(nm) untuk pencocokan *string*-nya dan O(p) untuk pemeriksaan tiap elemen *array* di mana p adalah jumlah elemen *array* sehingga memiliki total kompleksitas sebesar O(nmp).

2. Pembentukan *autocomplete* atau *autocompletion* menggunakan algoritma *brute force* versi penulis

Pembentukan fitur *autocomplete* atau *autocompletion* dengan algoritma *brute force* versi penulis adalah mengganti struktur data yang digunakan pada algoritma sebelumnya. Pada algoritma sebelumnya, struktur data yang digunakan untuk menyimpan *reserved words*, variable, fungsi, serta daftar kata apa saja yang pengguna telah tuliskan di dalam sebuah *array*. Dalam algoritma ini, penulis menggunakan struktur data pohon/*tree* untuk menyimpan *reserved words*, variable, fungsi, serta daftar kata apa saja yang pengguna telah tuliskan tadi.

Struktur data pohon yang dipakai adalah sebagai berikut: Akar dari pohon adalah karakter sembarang, simpul berikutnya dari akar adalah huruf ‘a’ hingga’z’. Simpul anak dari masing-masing simpul huruf tadi juga berisi huruf ‘a’ hingga ‘z’. Namun, untuk menyederhanakan bentuk pohon, huruf yang tidak ada pada *reserved words*, variable, fungsi, serta daftar kata apa saja yang pengguna telah tuliskan tidak dimasukkan dalam pohon. Contoh: *reserved words* tidak terdapat kata yang berawalan dengan huruf ‘x’, maka pada simpul anak pertama dari struktur data pohon yang digunakan tidak terdapat simpul ‘x’.

Contoh *tree*-nya adalah:



Pada pohon di atas, *reserved words* hanya memiliki daftar kata dengan huruf pertama ‘a’, ‘b’, ‘c’, ‘d’, dan ‘r’. Dari simpul pohon ‘a’ terdapat dua simpul anak yaitu ‘b’ dan ‘p’. Hal ini menunjukkan bahwa dalam *reserved words* tersebut hanya ada kata yang mempunyai dua karakter awal ‘ab’ dan ‘ap’. Pada simpul lain juga demikian.

Cara kerja algoritma pemrediksiannya adalah sebagai berikut:

a. Pengguna mengetikkan karakter pertama

b. Program akan mencari apakah karakter pertama tersebut ada di simpul pertama tingkat pertama pohon
c. Jika ada maka keluarkan seluruh kata yang berada di seluruh anak simpul tersebut
d. Jika tidak ada maka cari di simpul selanjutnya tingkat pertama hingga seluruh simpul tingkat pertama sudah dikunjungi.
e. Ulangi langkah b untuk karakter selanjutnya pada simpul tingkat kedua dan seterusnya.

REFERENCES

[1] Lecroq, Thierry Charras, Christian. 2001. Handbook of Exact String Matching Algorithm. ISBN 0-9543006-4-5

[2] Knuth, Donald E. Morris, James H. Pratt, Vaughan R. 1977. Fast Pattern Matching in Strings, SIAM Journal of Computing Vol 6 No.2.

[3] Breslauer, Dany. 1992. Efficient String Algorithms.

[4] Tam, C. & Wells, D. (2009). Evaluating the Benefits of Displaying Word Prediction Lists on a Personal Digital Assistant at the Keyboard Level. Assistive Technology, 21, 105-114.

[5] Anson, D., Moist, P., Przywara, M., Wells, H., Saylor, H. & Maxime, H. (2006). The Effects of Word Completion and Word Prediction on Typing Rates Using On-Screen Keyboards. Assistive Technology, 18, 146-154.

[6] Trnka, K., Yarrington, J.M. & McCoy, K.F. (2007). The Effects of Word Prediction on Communication Rate for AAC. Proceedings of NAACL HLT 2007, Companion Volume, 173-176.

[7] Beukelman, D.R. & Mirenda, P. (2008). Augmentative and Alternative Communication: Supporting Children and Adults with Complex Communication Needs. (3rd Ed.) Baltimore, MD: Brookes Publishing, p. 77.

[8] Witten, I. H.; Darragh, John J. (1992). *The reactive keyboard* Cambridge, UK: Cambridge University Press. pp. 43–44. ISBN 0-521-40375-8.

[9] Jelinek, F. (1990). Self-Organized Language Modeling for Speech Recognition. In Waibel, A. & Kai-Fulee, Ed. Morgan, M.B. Readings in Speech Recognition (pp. 450). San Mateo, California: Morgan Kaufmann Publishers, Inc.

[10] Dabbagh, H. H. & Damper, R. I. (1985). Average Selection Length and Time as Predictors of Communication Rate. Proceedings of the RESNA 1985 Annual Conference, RESNA Press, 104-106.

[11] Goodenough-Trepagnier, C., & Rosen, M.J. (1988). Predictive Assessment for Communication Aid Prescription: Motor-Determined Maximum Communication Rate. In L.E. Bernstein (Ed.), The vocally impaired: Clinical Practice and Research (pp. 165-185).Philadelphia: Grune & Stratton.; as cited in Tam & Wells (2009), pp. 105-114.

[12] Swiffin, A. L., Arnott, J. L., Pickering, J. A., & Newell, A. F. (1987). Adaptive and predictive techniques in a communication prosthesis. Augmentative and Alternative Communication, 3, 181–191; as cited in Tam & Wells (2009).

[13] Tam, C., Reid, D., Naumann, S., & O' Keefe, B. (2002). Perceived benefits of word prediction intervention on written productivity in children with Spina Bifida and hydrocephalus. Occupational Therapy International, 9, 237–255; as cited in Tam & Wells (2009).

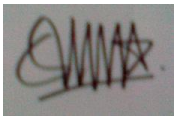
[14] http://stevedaskam.wordpress.com/2009/05/17/autocomplete-data-structures/

[15] Davison, Dr. Andrew. (2006). Pattern Matching.ppt

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 21 Desember 2012



Muhammad Wachid Kusuma
13510074