PENERAPAN ALGORITMA *STRING MATCHING* PADA PERMAINAN "WORD SEARCH PUZZLE"

Desi Hadiati

Program Studi Informatika, Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung
Jalan Ganesa No 10 Bandung
e-mail: if15007@students.if.itb.ac.id

ABSTRAK

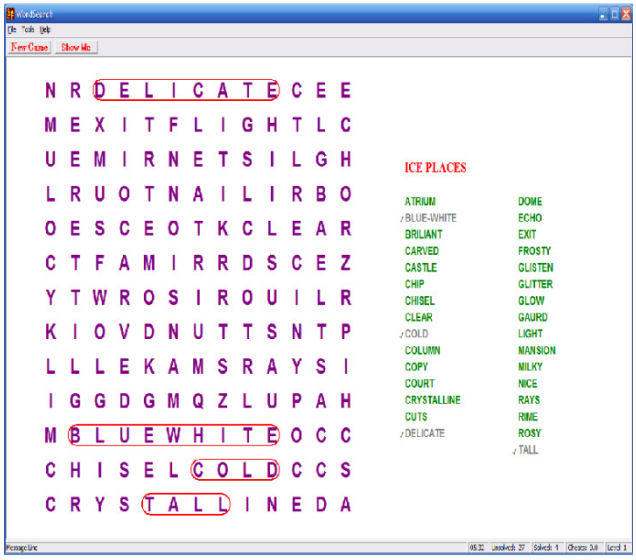
Pada kehidupan kita sekarang ini, keberadaan *game* atau kita kenal dengan istilah permainan sangatlah penting. Permainan bukan hanya sebagai pengisi waktu senggang, tetapi juga sebagai sarana hiburan bagi sebagian besar dari kita yang memiliki banyak kesibukan. Salah satu permainan yang banyak digemari saat ini adalah *game* komputer, diantaranya adalah permainan "Word Search Puzzle". Sebagian besar orang memainkan permainan ini dengan hanya mengandalkan penglihatan dan intuisi untuk menemukan kata-kata yang tersembunyi pada papan permainan. Dalam makalah ini akan dipaparkan penerapan algoritma *string matching* untuk menemukan solusi bagi permainan ini.

Kata kunci: permainan, "Word Search Puzzle", *string matching*

1. PENDAHULUAN

Sejak zaman dahulu, permainan adalah suatu hal yang penting dalam kehidupan manusia. Permainan, baik tradisional maupun modern (menggunakan teknologi atau teknik komputerisasi), dimainkan oleh semua kalangan orang sebagai pengisi waktu senggang, sekedar mencari hiburan, atau bahkan sebagai sarana pendidikan. Di antara orang-orang tersebut, ada yang rela duduk berjam-jam hanya untuk menyelesaikan permainan dan ingin melihat akhir dari permainan tersebut.

Dalam makalah ini, penulis akan memaparkan analisa penulis terhadap penerapan algoritma *string matching* pada permainan "Word Search Puzzle". Pada permainan ini, pemain harus menemukan kata-kata yang tersembunyi di antara banyak karakter dalam papan permainan.



Gambar 1. Tampilan Utama Permainan "Word Search Puzzle"

Pada tampilan permainan "Word Search Puzzle" di atas, di bagian kanan terdapat kata-kata yang harus dicari di antara karakter-karakter di bagian kiri. Kata-kata yang diminta dapat dicari pada setiap baris ataupun kolom dari kumpulan karakter-karakter yang ada pada papan permainan. Jika pemain menemukan kata yang dimaksud, pemain harus menandai kata-kata yang sudah ditemukan dan secara otomatis pada bagian kiri kata yang sudah ditemukan akan berubah warna.

Untuk mencari keseluruhan kata yang ada memang tidak mudah, oleh karena itu aplikasi permainan menyediakan sarana pencarian solusi secara otomatis. Algoritma *string matching* akan diterapkan pada sarana pencarian solusi otomatis tersebut.

2. ALGORITMA *STRING MATCHING*

Algoritma *string matching* dalam bahasa Indonesia dikenal dengan istilah algoritma pencocokan *string*. [1]

Persoalan pencarian *string* dirumuskan sebagai berikut :
Diberikan :

1. Sebuah teks (*text*), yaitu sebuah (*long*) *string* yang panjangnya n karakter
2. *Pattern*, yaitu sebuah *string* dengan panjang m karakter (m < n) yang akan dicari dalam teks

Carilah lokasi pertama di dalam teks yang bersesuaian dengan *pattern*.

Aplikasi permasalahan pencocokan *string* biasa ditemukan dalam pencarian sebuah kata dalam dokumen (misalnya menu *Find* dalam Microsoft Word).

Contoh 1

Pattern : not
Teks : nobody noticed him
↑target

Dalam algoritma pencocokan *string*, teks diasumsikan berada di dalam memori, sehingga bila kita mencari *string* di dalam sebuah arsip, maka semua isi arsip perlu dibaca terlebih dahulu kemudia disimpan di dalam memori. Jika *pattern* muncul lebih dari sekali di dalam teks, maka pencarian hanya akan memberikan keluaran berupa lokasi *pattern* ditemukan pertama kali.

Ada tiga buah algoritma yang umum digunakan dalam melakukan pencocokan *string*, algoritma Brute Force, algoritma Knuth-Morris-Pratt, serta algoritma Boyer Moore.

2.1 Algoritma Brute Force

Algoritma brute force untuk pencocokan *string* adalah sebagai berikut :
(diasumsikan teks berada dalam array T [1..n] dan *pattern* berada dalam array P[1..m])

1. Mula-mula *pattern* P dicocokan pada awal teks T
2. Dengan bergerak dari kiri ke kanan, bandingkan setiap karakter di dalam *pattern* P dengan karakter yang berkesesuaian di dalam teks T sampai semua karakter yang dibandingkan cocok atau sama (pencarian berhasil), atau dijumpai sebuah ketidakcocokan karakter (pencarian belum berhasil)
3. Bila *pattern* P belum ditemukan kecocokannya dan teks T belum habis, geser *pattern* P satu karakter ke kanan dan ulangi langkah 2. [1]

Contoh 2

Teks : nobody noticed him
Pattern : not

nobody **not**iced him
s=0 not
s=1 not
s=2 not
s=3 not
s=4 not
s=5 not
s=6 not
s=7 **not**

Pattern **not** ditemukan pada posisi indeks ke delapan dari awal teks.

Kompleksitas algoritma pencocokan *string* yang dihitung dari jumlah perbandingan yang dilakukan untuk kasus terbaik adalah *O(n)*. Kasus terbaik ini terjadi apabila karakter pertama *pattern* P tidak pernah sama dengan karakter pada teks T yang dicocokan. Pada kasus ini dilakukan paling banyak n buah operasi perbandingan. Untuk kasus terburuk, kompleksitas algoritma ini adalah *O(mn)* karena dibutuhkan m(n-m+1) perbandingan.

2.2 Algoritma Knuth-Morris-Pratt

Pada algoritma Knuth-Morris-Pratt (KMP) informasi ketidakcocokan *pattern* dengan teks digunakan disimpan untuk menentukan jumlah pergeseran. Algoritma KMP melakukan pergeseran lebih jauh sesuai dengan informasi yang disimpan, tidak seperti pada algoritma Brute Force di mana pergeseran dilakukan setiap satu karakter, sehingga waktu pencarian dapat dikurangi secara signifikan.

AlgoritmaKnuth-Morris-Pratt dikembangkan oleh D. E. Knuth, bersama-sama dengan J.H Morris dan V. R. Pratt[1].

Dalam algoritma KMP, ada beberapa definisi yang digunakan :

Misalkan A adalah alfabet dan $x = x_1x_2\ldots x_k$, $k \in$ **N**, adalah *string* yang panjangnya k yang dibentuk dari karakter-karakter di dalam alfabet A.

Awalan (*prefix*) dari x adalah upa-*string* (*substring*) u dengan

$$u = x_1x_2\ldots x_{k-1}\text{ , } k \in \{1, 2, \ldots, k-1\}$$

dengan kata lain, x diawali dengan u.

Akhiran (*suffix*) dari x adalah upa-*string*(*substring*) u dengan

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

dengan kata lain, x diakhiri dengan v.

Pinggiran (*border*) dari x adalah upa-*string* r sedemikian sehingga

$r = x_1 x_2 \ldots x_{k-1}$ dan $u = x_{k-b} x_{k-b+1} \ldots x_k$,
$k \in \{1, 2, \ldots, k-1\}$
dengan kata lain, pinggiran dari *x* adalah upa-*string* yang keduanya awalan dan juga akhiran sebenarnya dari *x*. [1]

Algoritma KMP melakukan proses awal terhadap *pattern* P dengan menghitung fungsi pinggiran yang mengindikasikan pergeseran s terbesar yang mungkin dengan menggunakan perbandingan yang dibentuk sebelum pencarian *string* [1]. Hal ini dimaksudkan untuk mencegah pergeseran yang tidak berguna seperti pada algoritma Brute Force.

Fungsi pinggiran hanya bergantung pada karakter-karakter di dalam *pattern*. Fungsi pinggiran *b(j)* didefinisikan sebagai ukuran awal terpanjang dari P yang merupakan akhiran dari P[1..j].
Algoritma untuk menghitung fungsi pinggiran adalah sbb :

```
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]}

Deklarasi
  k,q : integer

Algoritma:
  b[1]←0
  q←2
  k←0
  for q←2 to m do
    while ((k > 0) and
          (P[q] ≠ P[k+1])) do
      k←b[k]
    endwhile
    if P[q]=P[k+1] then
      k←k+1
    endif
    b[q]=k
  endfor
```

Untuk melakukan pencocokan *string*, mula-mula kita harus menghitung fungsi pinggiran untuk *pattern* tersebut. Selanjutnya dilakukan pencocokan pertama, samakan ujung kiri *pattern* dengan ujung kiri teks. Misalkan karakter cocok pada posisi[1..5] dan pada posisi 6 tidak ditemukan kecocokan, jumlah pergeseran selanjutnya ditentukan oleh pinggiran awalan P yang berkesesuaian.

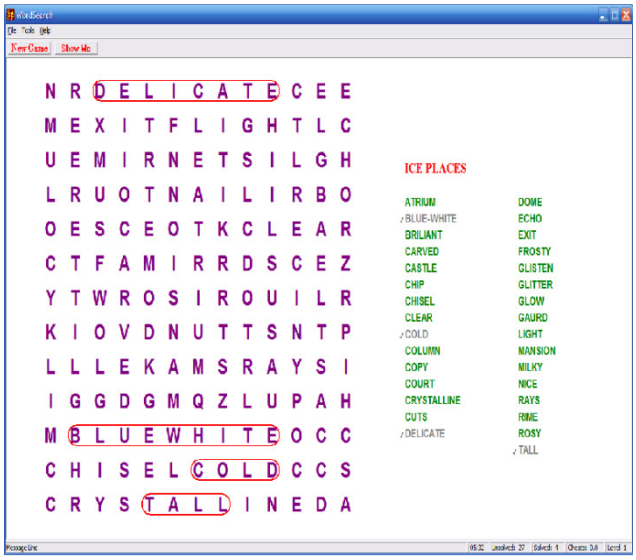
Kompleksitas algoritma KMP dihitung dengan menggabungkan kompleksitas waktu untuk menghitung fungsi pinggiran, yaitu *O(m),* dan kompleksitas waktu untuk melakukan pencarian *string*, *O(n)*, sehingga kompleksitas waktu keseluruhan algoritma ini adalah *O(m+n).*

2.3 Algoritma Boyer-Moore (BM)

Pencarian *string* dengan menggunakan algoritma ini berbeda dengan algoritma KMP. Algoritma BM melakukan perbandingan *pattern* mulai dari kanan / karakter paling belakang dan lompatan maju dilakukan sejauh mungkin.

3. PENERAPAN DALAM PERMAINAN "WORD SEARCH PUZZLE"

Untuk mencari seluruh solusi yang mungkin pada permainan wsp, pencarian setiap kata yang diminta oleh program dilakukan per baris.



Gambar 2 Permainan "Word Search Puzzle"

Contohnya saja pada tampilan permainan (Gambar 2), kata pertama yang diminta oleh program adalah kata "ATRIUM". Dengan menggunakan algoritma pencocokan *string*, kata tersebut akan dicari pada baris pertama, jika tidak ditemukan pencarian dilanjutkan dengan baris kedua dan jika tidak ditemukan juga, pencarian akan dilanjutkan pada baris-baris selanjutnya dan jika pada baris tidak ditemukan, pencarian dilanjutkan pada setiap kolom.

Jika kata "ATRIUM" telah ditemukan, kata berikutnya yaitu "BLUEWHITE" akan dicari pada papan permainan dengan cara yang sama. Setiap setelah kata yang diminta ditemukan pada kumpulan karakter di papan permainan, pencarian akan dilanjutkan untuk kata-kata berikutnya.

Pada permainan ini, yang didefinisikan sebagai *pattern* adalah kata-kata yang diminta oleh program (yang terletak pada bagian kanan dari tampilan permainan), sedangkan yang didefinisikan sebagai teks adalah kumpulan karakter-karakter pada setiap baris atau kolomnya.

Algoritma yang lebih baik digunakan pada proses pencarian solusi dari permainan "Word Search Puzzle" ini adalah algoritma Knuth-Morris-Pratt atau KMP. Dengan menggunakan *loop* pencarian tiap baris dan kolom, algoritma pencarian solusi tersebut secara umum adalah sebagai berikut :

```
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. 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 direpresentasika 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:
for ((setiap baris) and (not ketemu))
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
endfor
if (not ketemu)
   for ((setiap kolom) and (not ketemu))
    {lakukan yang sama seperti pada baris}
   endfor
endif
```

4. KESIMPULAN

Algoritma *string matching* atau pencocokan *string* yang paling tepat digunakan adalah algoritma Knuth-Morris-Pratt (KMP), karena jika dibandingkan dengan algoritma Brute Force kompleksitas waktu yang dibutuhkan lebih sedikit.

Penerapan algoritma pencocokan *string* pada permainan "Word Search Puzzle" ini belum diimplementasikan dalam bentuk sebuah program yang bisa dimainkan. Sejauh ini, penulis baru hanya membuat konsep kasar dari penerapan algoritma tersebut. Keoptimalan penggunaan algoritma pencocokan *string* pada permainan "Word Search Puzzle" ini juga belum dapat diukur karena belum dibuat perbandingan dengan algoritma lain yang mungkin dapat diterapkan.

Untuk ke depannya, penulis akan mencoba mengimplementasikan algoritma pencocokan *string* ini untuk mencari solusi dalam permainan "Word Search Puzzle" dengan melakukan perbaikan-perbaikan untuk dapat meminimalisasi kompleksitas waktu yang dibutuhkan.

REFERENSI

[1] Munir, Rinaldi.Diktat Kuliah IF2251 *Strategi Algoritmik*. Institut Teknologi Bandung. 2007.

[2] Abel, Ute. "*String Matching*". Proseminar "Algorithmen". Sommersemester 2001.