Aplikasi Algoritma Pencarian String Knuth-Morris-Pratt dalam Permainan Word Search

Gahayu Handari Ekaputri[1], Yulie Anneria Sinaga[2]

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

E-mail : if14031@students.if.itb.ac.id[1], if14085@students.if.itb.ac.id[2]

Abstrak

Permainan *word search* adalah permainan untuk menemukan sejumlah kata dari sebuah *matrix of char*. Kata adalah kumpulan dari karakter yang memiliki suatu makna tertentu. Kata-kata yang dicari pada permainan ini dapat dibaca dari kiri ke kanan, atas ke bawah, diagonal dan sebaliknya tergantung letak kata tersebut pada *matrix of char*. Pencarian sejumlah kata tersebut pada permainan ini dapat menggunakan algoritma Knuth-Morris-Pratt (KMP). Algoritma KMP adalah algoritma untuk melakukan pencocokan string dari sebuah teks. Algoritma ini merupakan algoritma pencocokan string yang cukup mangkus dan sangkil.

***Kata kunci**: algoritma KMP, pencocokan string , word search*

1. Pendahuluan

Permainan *Word Search* merupakan permainan pencarian sejumlah kata dari sebuah *matrix of char*. Kata adalah kumpulan karakter yang memiliki makna. Kata yang dicari terletak di dalam *matrix of char* tersebut dapat memiliki makna jika dibaca dari atas ke bawah, bawah ke atas, diagonal, kanan ke kiri atau kiri ke kanan.

Permainan ini merupakan salah satu aplikasi dari algoritma Knuth-Morris-Pratt (KMP). Karena pada permainan ini dilakukan pencarian suatu *pattern*, yaitu kata , pada teks, yaitu *matrix of char*. Algoritma KMP dapat secara baik digunakan untuk menyelesaikan permainan ini. Algoritma ini akan selalu menghasilkan solusi.

Tujuan penulisan makalah:
1. Memahami algoritma pencocokan *string* dengan memakai algoritma KMP;
2. Mempelajari penggunaan algoritma KMP pada permainan *Word Search*.

2. Algoritma Knuth-Morris-Pratt (KMP)

Algoritma KMP merupakan algoritma yang digunakan untuk melakukan proses pencocokan string. Algoritma ini merupakan jenis *Exact String Matching Algorithm* yang merupakan pencocokan *string* secara tepat dengan susunan karakter dalam *string* yang dicocokkan memiliki jumlah maupun urutan karakter dalam string yang sama [1]
Contoh : kata *algoritmik* akan menunjukkan kecocokan hanya dengan kata *algoritmik*.

Pada algoritma KMP, kita simpan informasi yang digunakan untuk melakukan pergeseran lebih jauh, tidak hanya satu karakter seperti algoritma *Brute Force*. Algoritma ini melakukan pencocokan dari kiri ke kanan[3].

Terdapat beberapa definisi pada algoritma KMP :
1. Misalkan *A* adalah alfabet dan $x = x1x2\ldots xk$ adalah *string* yang panjangnya k yang dibentuk dari karakter-karakter di dalam alfabet *A*.
   - Awalan (*prefix*) dari *x* adalah upa-string (*substring*) *u* dengan $u = x1x2\ldots xk-1$, $k \in \{1, 2, \ldots, k-1\}$ dengan kata lain, *x* diawali dengan *u*.
   - Akhiran (*suffix*) dari *x* adalah upa-string (*substring*) *u* dengan $u = xk-b\,xk-b+1\ldots xk$, $k \in \{1, 2, \ldots, k-1\}$ dengan kata lain, *x* di akhiri dengan *v*
   - Pinggiran (*border*) dari *x* adalah upa-string *r* sedemikian sehingga $r = x1x2\ldots xk-1$ dan $u = xk-b\,xk-b+1\ldots xk$, $k \in \{1, 2, \ldots, k-1\}$, dengan kata lain, pinggiran dari *x* adalah upa-string yang keduanya awalan dan juga akhiran sebenarnya dari *x*.
2. Fungsi Pinggiran b(j) didefinisikan sebagai ukuran awalan terpanjang dari P yang merupakan akhiran dari P[1..j]

Berikut adalah pseudo-code dari algoritma KMP secara umum :

```
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]}
```

```
Kamus
  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
```

```
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 KMP.
Jika ditemukan P di dalam T, maka lokasi awal kecocokan disimpan di dalam peubah idx}
Kamus
  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)

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
  else
     idx ← -999      {tidak ketemu}
  endif
```

Kompleksitas algoritma pencocokan string dengan KMP ini dihitung dari kompleksitas untuk menghitung fungsi pinggiran dan pencarian string. Untuk menghitung fungsi pinggiran dibutuhkan waktu *O(m)*, sementara untuk pencarian string dibutuhkan *O(n)*, sehingga kompleksitas waktu algoritma KMP adalah *O(m + n)*[2].

3. Penggunaan Algoritma KMP dalam Permainan Word Search

Permainan ini menggunakan modifikasi dari algoritma KMP. Karena algoritma KMP tidak hanya digunakan untuk teks mendatar saja tapi juga untuk teks menurun dan diagonal, serta untuk melakukan pembacaan secara terbalik.

Langkah-langkah yang akan kita lakukan adalah :
Cari keberadaan kata pada tiap baris, jika tidak ketemu ubah cara baca, lakukan pencocokan kata pada tiap baris,
jika tidak ketemu lakukan pencocokan kata per kolom, jika tidak ketemu ubah cara baca, lakukan pencocokan kata pada tiap kolom,
jika tidak ketemu lakukan pencocokan kata secara diagonal dari kiri bawah ke kanan atas, jika tidak ketemu ubah cara baca, lakukan pencocokan kata secara diagonal dari kiri bawah ke kanan atas,
jika tidak ketemu lakukan pencocokan kata secara diagonal dari kiri atas ke kanan bawah yang panjang diagonalnya ≥ panjang kata yang dicari, jika tidak ketemu ubah cara baca, lakukan pencocokan kata secara diagonal dari kiri atas ke kanan bawah bawah yang panjang diagonalnya ≥ panjang kata yang dicari.
Ulangi hingga tiap kata sudah ditemukan posisinya pada matrix of char.

Berikut adalah *pseudo-code* modifikasi algoritma KMP untuk mencari suatu kata pada permainan ini secara lengkap :

```
procedure changeDirection(input/output kata : string)
{prosedur untuk membalik kata}
Kamus
  temp : string
  i : integer
Algoritma
  for (i = 0) to kata.length() do
     temp[i] = kata[length - i + 1]
  kata <- temp
```

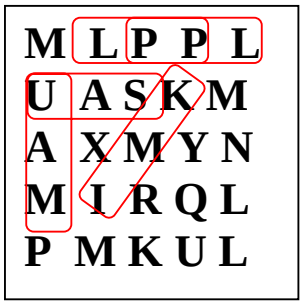
```
procedure changeLineType(input : matriks : array[1..n][1..n]of char,output line : array of char)
{dari baris ubah jadi kolom}
if (line = baris) then
  line <- matrix[][]
else
  if (line = kolom) then
    line <- diagonal endif
endif
{dari kolom ubah jadi diagonal}
```

```
procedure find1Word(input kata : string, input matrix : array[1..n][1..n] of char)
ketemu <- false
while(not ketemu) do
while(not ketemu) do

  while ((not ketemu) and (y < n)) do
     while ((not ketemu) and (x < n)) do
```

```
      KMPsearch(m.length(),kata.length()
, matrix[x], kata,idx)
      if (idx != -999) then
         ketemu <- true
      endif
      x <- x + 1
    endwhile
   changeDirection(kata)    {ubah    cara
baca dengan menukar urutan kata yang
dicari}
  endwhile
  ubah dari teks dari baris dari matrix
of char menjadi kolom dari matrix of
char
endwhile
  ubah dari teks dari kolom menjadi
diagonal dari matrix of char
endwhile
```

Untuk penerapan algoritma tersebut, akan diperlihatkan dengan contoh berikut :



Kata-kata yang berada di sebelah kanan adalah kata-kata yang harus dicari pada *matrix of char* yang berada di sebelah kiri.

Misal kita lakukan pencarian kata pertama PPL
Kita hitung pinggiran dari kata PPL

j	1	2	3
P[j]	P	P	L
B(j)	0	1	0

maka nilai pinggiran PPL adalah 1

Kita cari pada baris pertama, maka T = MLPPL sementara P = PPL

```
          1 2 3  4   5
Teks    : M L P P L
Pattern : P  P  L
```

ternyata tidak ditemukan , maka pattern digeser sebanyak panjang patterm – nilai pinggiran = 3 -1 =2. Jadi patterm P digeser sejauh 2 karakter

```
          1 2 3  4   5
Teks    : M L P P L
Pattern :      P  P L
```

Setelah kata PPL ditemukan pada baris pertama, program akan mencari kata PPL sampai baris terakhir dan kembali lagi ke baris pertama dengan pencarian kata PPL yang dibalik menjadi LPP

Kita hitung pinggiran dari kata LPP

j	1	2	3
P[j]	L	P	P
B(j)	0	0	0

maka nilai pinggiran LPP adalah 0

```
          1 2 3  4   5
Teks    : M L P P L
Pattern : L  P  P
```

ternyata tidak ditemukan , karena nilai pinggiran LPP adalah 0 maka patterm P digeser sejauh 1 karakter

```
          1 2 3  4   5
Teks    : M L P P L
Pattern :    L P P
```

kata PPL berhasil ditemukan, maka program melanjutkan pencarian kata-kata berikutnya.

4. Kesimpulan

Kesimpulan dari topik ini adalah algoritma KMP menyimpan sebuah informasi yang digunakan untuk melakukan jumlah pergeseran, sehingga algoritma ini melakukan pergeseran lebih jauh (tidak hanya bergeser satu karakter seperti dalam *brute force*). Dengan ini penggunaan algoritma KMP dapat mempersingkat waktu pencocokan *string*.

5. Referensi

[1]http://www.cs.cmu.edu/afs/andrew.cmu.edu/course/15/354/www/postscript/kmp.pdf, diakses pada tanggal 15 Mei 2006 Pukul 12.00

[2]http://www.ics.uci.edu/~eppstein/161/960227.html, diakses pada tanggal 15 Mei 2006 Pukul 12.00

[3] Rinaldi Munir. *Diktat Kuliah Strategi Algoritmik*. Program Studi Teknik Informatika ITB : 2006.