Pencocokan String dengan Algoritma Reverse Colussi

Didik Haryadi - 13509601[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
E-mail: if19601@students.if.itb.ac.id[1]

ABSTRAK

Sekarang ini, algoritma pencocokan string sangat banyak macamnya. Meskipun dari satu algoritma ke algoritma lain hanya merupakan pengembangan/perbaikan dari algoritma yang sudah ada sebelumnya. Algoritma hasil dari pengembangan tersebut terkadang menggunakan arah pencarian yang berbeda dari sebelumnya. Arah pencarian itu sendiri dapat dikelompokkan dalam 3 kategori, yaitu dari kiri ke kanan, dari kanan ke kiri, dan arah yang ditentukan secara spesifik. Salah satu contoh dari arah pencarian yang telah ditentukan secara spesifik adalah algoritma reverse colussi. Algoritma reverse colussi ini sendiri berbeda dengan algoritma colussi. Algoritma reverse colussi merupakan perbaikan dari algoritma Boyer-Moore. Sedangkan algoritma colussi merupakan perbaikan dari algorima Knuth, Morris, dan Pratt.

Kata kunci: *pencocokan string, algoritma reverse colussi, algoritma colussi, algoritma Boyer-Moore, algoritma Knuth, Morris, and Pratt*

I. PENDAHULUAN

Algoritma reverse colussi merupakan satu dari sekian banyak algoritma pencocokan string. Algoritma reverse colussi merupakan perbaikan dari algoritma Boyer-Moore dan idenya sendiri berasal dari algoritma colussi. Proses pencocokan tiap-tiap karakter pada algoritma reverse colussi menggunakan sepasang karakter yang telah didefinisikan pada tabel sebelumnya. Algoritma reverse colussi posisi ke dalam 2 kategori yaitu *special position* dan *non-special position.* Dimana *special position* akan diproses terlebih dahulu.

II. ALGORITMA

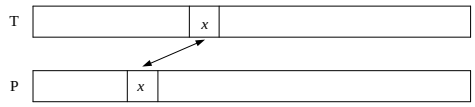
Pada algoritma reverse colussi, terdiri dari 2 fase, yaitu fase pemrosesan awal dan fase pencarian. Pada fase pemrosesan awal, dilakukan pencarian sepasang karakter, *special position,* dan *non-special position* untuk menentukan pergeseran pattern. Pada fase pencarian, dilakukan operasi pencocokan pattern terhadap teks.

1. Fase Pemrosesan Awal

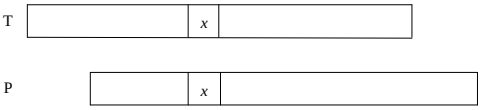
Pada algoritma reverse colussi, harus ditentukan beberapa *point* yang dianggap spesial dan beberapa *point* yang dianggap tidak spesial. Spesial *point* memungkinkan angka pergeseran yang lebih kecil dari pada Bukan spesial *point.* Oleh karena itu, dalam algoritma reverse colussi harus ditentukan *special position* terlebih dahulu.

Misal, T_i merupakan karakter ke-i pada $T(1 \leq i \leq n)$. Dan P_j adalah karakter ke-j pada $P(1 \leq j \leq m)$. Berikut gambaran mengenai apa yang disebut sepasang karakter.

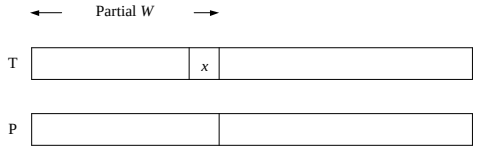
Untuk setiap karakter x pada T, cari karakter x terdekat pada P yang terletak pada sisi kiri dari karakter x pada T.



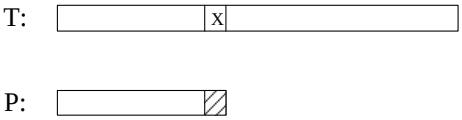
Jika terdapat karakter x pada P yang terletak disebelah kiri dari karakter x pada T, maka geser P sampai kedua karakter x tersebut sejajar.



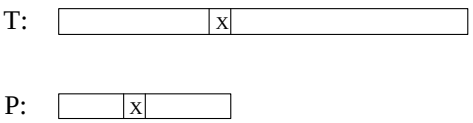
Jika tidak ada, maka buat *partial window* berdasarkan posisi karakter x pada T dan string yang ada di sebelah kirinya. Setelah ini, proses pencarian hanya sebatas ukuran *window* saja.



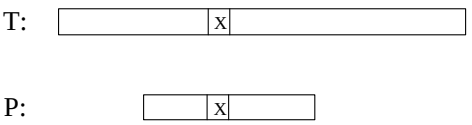
Misalkan karakter x yang terakhir pada *window* T tidak cocok dengan karakter terakhir pada P



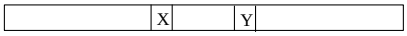
Misalkan terdapat karakter x pada P:

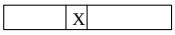


Maka, sejajarkan posisi x pada P dengan posisi x pada T

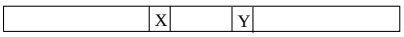


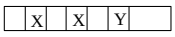
Misalkan karakter terakhir y dari window pada T tidak cocok dengan karakter terakhir dari P:

T: 

P: 

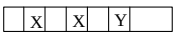
Jika tidak cocok, kemudian cari pasangan x dan y pada P. Misal posisi x dan y sebagai berikut:

T: 

P: 

Maka geser sampai posisi pasangan x dan y pada P sejajar dengan posisi x dan y pada T, sepasang karakter sudah diperoleh.

T: 

P: 

1.1 Pencarian sepasang karakter

Dalam algoritma reverse colussi, pencarian pasangan karakter dapat kita lakukan dengan menggunakan bantuan tabel rcBc. Misalkan:

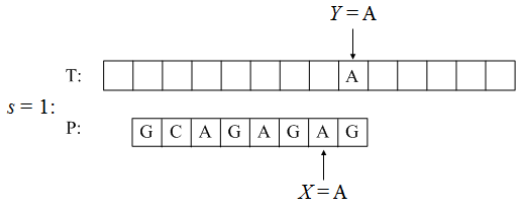
Y adalah karakter terakhir dari *window* pada T
s adalah panjang pergeseran yang akan digunakan pada langkah terakhir
k bilangan integer
m panjang pattern

Kondisi 1:
Jika ditemukan $P_{m-k-1}=Y$ and $P_{m-k-s-1}=P_{m-s-1}$, masukkan k terkecil ke tabel rcBc[Y,s]

Kondisi 2:
Jika ditemukan $P_{m-k-1}=Y$ and $k>m-s-1$, masukkan k terkecil ke tabel rcBc[Y,s]

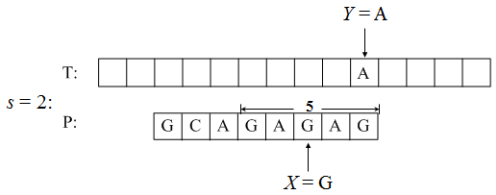
Kondisi 3:
Jika tidak ditemukan, masukkan m ke tabel rcBc[Y,s]

Contoh:



XY = AA tidak terdapat pada *P*.
rcBc[*Y*, 1] = m => *rcBc*[*Y*, 1] = 8

Present matched character of T (*Y*) \ Length of Previous Shifts (*s*)	1	2	3	4	5	6	7	8
A	8							
C								
G								
T								

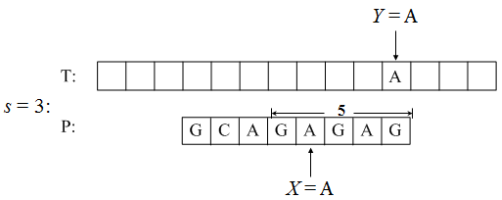


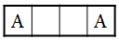
Cari 

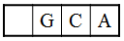
ditemukan 

rcBc[*Y*, 2] = 5

Present matched character of T (*Y*) \ Length of Previous Shifts (*s*)	1	2	3	4	5	6	7	8
A	8	5						
C								
G								
T								



Cari 

, ditemukan 

rcBc[*Y*, 3] = 5

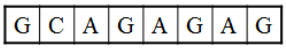
Present matched character of T (*Y*) \ Length of Previous Shifts (*s*)	1	2	3	4	5	6	7	8
A	8	5	5					
C								
G								
T								

Proses tersebut dilakukan hingga tabel terisi semua.

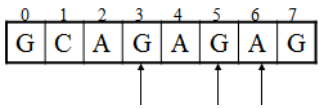
Present matched character of T (*Y*) \ Length of Previous Shifts (*s*)	1	2	3	4	5	6	7	8
A	8	5	5	3	3	3	1	1
C	8	6	6	6	6	6	6	6
G	2	2	2	4	4	2	2	2
T	8	8	8	8	8	8	8	8

1.2 Pencarian *special positions* dan *non-special positions*

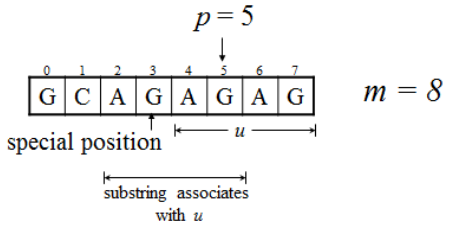
Dalam algoritma reverse colussi, pencarian *special positions* dapat kita lakukan dengan menggunakan bantuan tabel rcGs. Dalam membuat tabel rcGs ini, aturan mengenai *substring matching* (*suffix* / *prefix*) yang digunakan sama dengan aturan yang digunakan pada algoritma Boyer-Moore. Misalkan diberikan *pattern P*, tunjukkan semua posisi *suffix* sering diulang pada P atau disebut sebagai *special positions*.



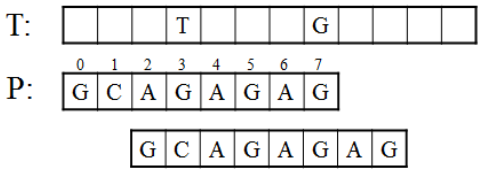
Dari *pattern* diatas, G, AG, AGAG merupakan *substring* yang sering diulang. Berikut *special positions* dari *pattern* tersebut.



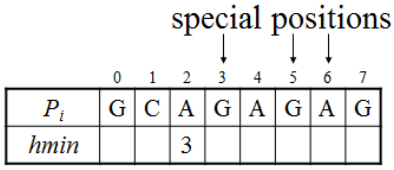
Untuk setiap *suffix* terpanjang *u*, sejajarkan posisi substring yang ada pada P. Jika *special positions* tidak cocok dengan *u*, geser P sejauh m-p-1 langkah dimana m adalah panjang pattern.



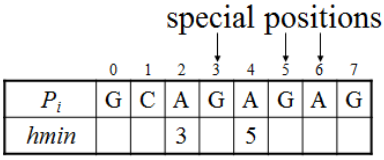
Hasil pergeseran sebesar 8-5-1 sebagai berikut:



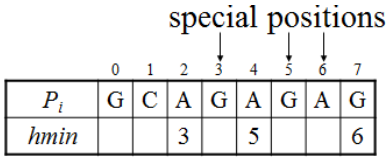
Jumlah pergeseran untuk setiap *special positions* disimpan dalam tabel yang disebut *hmin.*



Special position i = 3, simpan jumlah pergeseran (8-5-1=2) pada tabel *hmin*[2]=3



Special position i = 5, simpan jumlah pergeseran (8-3-1=4) pada tabel *hmin*[4]=5

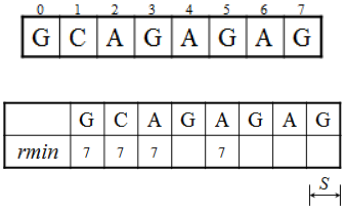


Special position i = 6, simpan jumlah pergeseran (8-0-1=7) pada tabel *hmin*[6]=6

Setelah membandingkan *special positions*, *remainder-position/non-special positions* juga harus dibandingkan. Perbandingan ini dilakukan dari kiri ke kanan. Jumlah pergeseran untuk setiap *non-special positions* disimpan dalam tabel yang disebut *rmin.*

Jika *suffix* S yang terdapat pada sisi kanan dari *non-special positions* ke-i sama dengan *prefix* maka *rmin(i)* = *m*-|*S*|. (|S| panjang S), tetapi jika tidak ada S maka *rmin(i)=m.*

Contoh:



Suffix S sama dengan *prefix* yang ada di sisi kanan dari *non-special positions*, jadi nilai dari *rmin non-special positions* adalah *m*-|*S*| *(8-1)*

Setelah tabel *hmin* dan *rmin* terbuat, sekarang dapat digunakan untuk membuat tabel rcGs.

Contoh: GCAGAGAG

i	0	1	2	3	4	5	6	7	8
P_i	G	C	A	G	A	G	A	G	
hmin[*i*]			3		5			6	
rmin[*i*]	7	7	7		7				
rcGs[*i*]	0								

Pertama-tama, isi indeks dari *special positions* yang tidak kosong kedalam tabel rcGs, menghasilkan:

i	0	1	2	3	4	5	6	7	8
P_i	G	C	A	G	A	G	A	G	
hmin[*i*]			3		5			6	
rmin[*i*]	7	7	7		7				
rcGs[*i*]	0	2	4						

Kemudian isikan semua nilai dari tabel *rmin* yang tidak kosong kedalam tabel rcGs.

i	0	1	2	3	4	5	6	7	8
P_i	G	C	A	G	A	G	A	G	
hmin[*i*]			3		5			6	
rmin[*i*]	7	7	7		7				
rcGs[*i*]	0	2	4	7	7	7	7	7	

Jika *P exact match* dengan *T*, maka geser *P*. jadi *rcGs*[8]=*m*-|*S*| (8-1).

i	0	1	2	3	4	5	6	7	8
P_i	G	C	A	G	A	G	A	G	
hmin[*i*]			3		5			6	
rmin[*i*]	7	7	7		7				
rcGs[*i*]	0	2	4	7	7	7	7	7	7

2. Fase Pencocokan Pattern

Setelah tabel rcBc dan rcGs terbentuk, pencocokan *pattern* dilakukan berdasarkan nilai dari kedua tabel tersebut. Pada algoritma reverse colussi, perbandingan karakter dilakukan dengan menggunakan urutan tertentu yang diberikan oleh tabel *h*.

Untuk setiap i dimana $0 \le i \le m$ definisikan dua himpunan yang disjoint:

- $Pos(i)=\{k : 0 \le k \le i \text{ dan } x[i] = x[i\text{-}k]\}$
- $Neg(i)=\{k : 0 \le k \le i \text{ dan } x[i] \ne x[i\text{-}k]\}$

For $1 \le k \le m$, biarkan $hmin[k]$ menjadi nilai terkecil ℓ dimana $\ell \ge k$-1 dan k bukan elemen dari $Neg(i)$ untuk semua i dimana $\ell < i \le m$-1.

For $0 \le \ell \le m$-1, biarkan $kmin[\ell]$ menjadi nilai terkecil k dimana $hmin[k]= \ell \ge k$ jika ada k dan $kmin[\ell]=0$ jika tidak ada k.

For $0 \le \ell \le m$-1, biarkan $rmin[\ell]$ menjadi nilai terkecil k dimana $r > \ell$ dan $hmin[r]=r$-1.

Nilai dari $h[0]$ diisi dengan m-1. Setelah itu, increment $kmin[\ell]$, semua indeks $h[1]$, ..., $h[d]$ dimana $kmin[h[i]] \ne 0$ dan isi $rcGs[i]$ dengan $kmin[h[i]]$ untuk $1 \le i \le d$. lalu pilih indeks $h[d+1]$, ... , $h[m\text{-}1]$ untuk diincrement dan isi $rcGs[i]$ dengan $rmin[h[i]]$ untuk $d < i < m$.

Berikut algoritma reverse colussi dalam bahasa C.

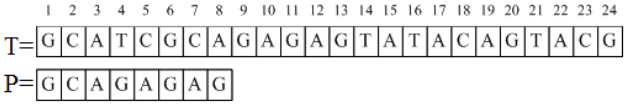
```
void RC(char *x,int m, char *y, int n)
{
   int i, j, s, rcBc[ASIZE][XSIZE],
                rcGs[XSIZE], h[XSIZE];

   /* Preprocessing */
   preRc(x, m, h, rcBc, rcGs);

   /* Searching */
   j = 0;
   s = m;
   while (j <= n - m) {
      while (j <= n - m && x[m - 1] !=
                           y[j + m - 1]) {
         s = rcBc[y[j + m - 1]][s];
         j += s;
      }
      for (i = 1; i < m && x[h[i]] ==
                      y[j + h[i]]; ++i);
      if (i >= m)
         OUTPUT(j);
      s = rcGs[i];
      j += s;
   }
}
```

III.PERCOBAAN

Diketahui:
Teks T, *Pattern* P:

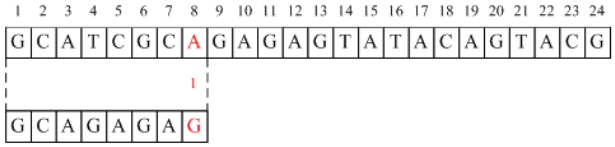


Pemrosesan awal akan menghasilkan tabel rcBc dan rcGs.

rcBc	1	2	3	4	5	6	7	8
A	8	5	5	3	3	3	1	1
C	8	6	6	6	6	6	6	6
G	2	2	2	4	4	2	2	2
T	8	8	8	8	8	8	8	8

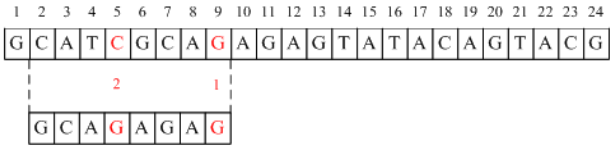
i	0	1	2	3	4	5	6	7	8
rcGs[i]	0	2	4	7	7	7	7	7	7

Fase Pencocokan Pattern:
Percobaan ke-1



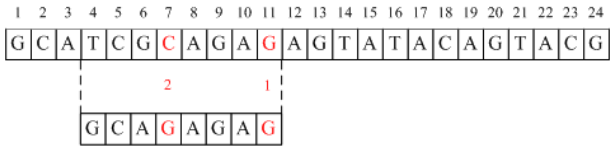
Shift by 1 (*rcBc*[A][*s*], *s* = 8), and change s = 1

Percobaan ke-2



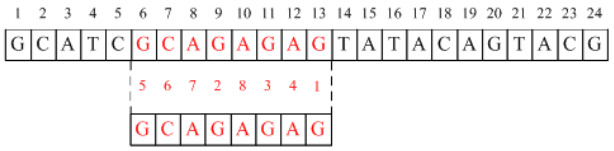
Shift by 2 (*rcGs*[1]), and change s = 2

Percobaan ke-3



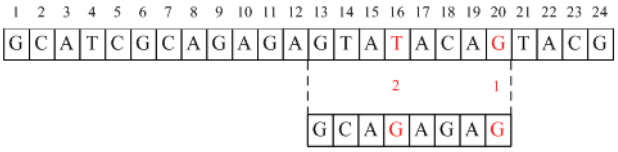
Shift by 2 (*rcGs*[1]), and change s = 2

Percobaan ke-4



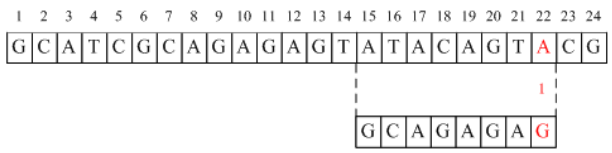
Shift by 7 (*rcGs*[8]), and change s = 7

Percobaan ke-5



Shift by 2 (*rcGs*[1]), and change s = 2

Percobaan ke-6



Shift by 5 (*rcBc*[A][*s*], *s* = 2), and change *s* = 5

Untuk mencocokkan *pattern* GCAGAGAG pada

GCATCGCAGAGAGTATACAGTACG memerlukan percobaan sebanyak 6 kali percobaan dengan jumlah perbandingan sebanyak 16 kali.

IV. PERBANDINGAN

Untuk teks dan *pattern* yang sama dengan percobaan diatas, berikut perbandingan algoritma reverse colussi dengan algoritma lainnya.

Algoritma	Waktu pemrosesan awal	Waktu pencarian	Jumlah Percobaan	Jumlah perbandingan
KMP	$O(m)$	$O(n+m)$	8	18
BM	$O(m+\sigma)$	$O(mn)$	5	17
Colussi	$O(m)$	$O(n)$	8	20
Reverse Colussi	$O(m^2)$	$O(n)$	6	16

V. KESIMPULAN

Algoritma reverse colussi menghasilkan jumlah perbandingan tiap karakter yang lebih sedikit dari pada algoritma Boyer-Moore, Knuth, Morris, and Pratt, Dan Colussi. Dengan kompleksitas waktu pemrosesan awal $\boldsymbol{O}(m^2)$ dan kompleksitas ruang $\boldsymbol{O}(m\sigma)$. Sedangkan untuk kompleksitas waktu pencocokan $\boldsymbol{O}(n)$. Untuk kasus terburuk, algoritma reverse colussi akan melakukan 2n perbandingan karakter.

Algoritma reverse colussi sangat cepat dalam melakukan pencarian string, tetapi membutuhkan waktu pemrosesan awal yang kurang baik. Hal ini dikarenakan harus menghasilkan 2 tabel terlebih dahulu sebelum memulai pencocokan.

REFERENCES

http://alg.csie.ncnu.edu.tw/course/StringMatching/Reverse Colussi.ppt, Tanggal akses : 1 Desember 2010 pukul : 20.00 WIB

http://www-igm.univ-mlv.fr/~lecroq/string/, Tanggal akses : 1 Desember 2010 pukul : 20.00 WIB

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, 1 Desember 2010

Didik Haryadi
13509601