PENGGUNAAN ALGORITMA APOSTOLICO-CROCHEMORE PADA PROSES PENCARIAN *STRING* DI DALAM TEKS

Sindy Gita Ratri

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
Jl. Ganesha No. 10, Bandung
e-mail: if15071@students.if.itb.ac.id

ABSTRAK

Proses pencarian *string* atau disebut juga dengan pencocokan *string* (*string matching* atau *pattern matching*) telah menjadi salah satu kebutuhan dalam pemrosesan teks. Algoritma pencarian *string* digunakan untuk mencari lokasi sebuah atau lebih *string* (disebut *pattern*) dalam kumpulan *string* lain (teks). Algoritma pencarian *string* yang ada saat ini sangat bervariasi dan mungkin masih akan berkembang. Pada makalah ini akan dibahas mengenai salah satu variasi dari algoritma pencarian *string* yaitu algoritma Apostolico-Crochemore dan perbandingannya dengan algoritma Knuth-Morris-Path.

Kata kunci: algoritma Apostolico-Crochemore, algoritma Knuth-Morris-Path, *pattern*, teks, fungsi pinggiran (*border function*).

1 PENDAHULUAN

Proses pencarian *string* atau disebut juga dengan pencocokan *string* (*string matching* atau *pattern matching*) telah menjadi salah satu kebutuhan dalam pemrosesan teks. Proses pencarian *string* dilakukan dengan menggunakan algoritma pencarian *string*.

Algoritma pencarian *string* adalah algoritma yang digunakan untuk mencari lokasi sebuah atau lebih *string* (disebut *pattern*) dalam kumpulan *string* lain (teks). Persoalan pencarian *string* dirumuskan sebagai berikut.

1. Diberikan teks, yaitu *string* yang panjangnya n karakter
2. Diberikan *pattern*, yaitu *string* dengan panjang m karakter (m < n) yang akan dicari di dalam teks

Kemudian dicari lokasi pertama di dalam teks yang bersesuaian dengan *pattern*.

Contoh :
Pattern : hari
Teks : kami pulang saat **hari** mulai malam

2 ALGORITMA PENCARIAN *STRING* YANG UMUM DIGUNAKAN

Saat ini ada banyak variasi algoritma pencarian *string* yang digunakan. Algoritma yang paling dasar adalah algoritma *brute force*. Algoritma lain yang umum digunakan antara lain algoritma Knuth-Morris-Pratt dan algoritma Boyer-Moore. Kebanyakan algoritma-algoritma pencarian *string* yang lain merupakan varian dari kedua algoritma tersebut.

2.1 Algoritma *Brute Force*

Algoritma *brute force* adalah algoritma yang paling sederhana. Teks diasumsikan berada dalam *array* T[1...n] dan *pattern* berada dalam *array* P[1...m] dengan m < n, maka dengan algoritma *brute force* pencarian *string* adalah sebagai berikut.

1. Mula-mula *pattern* dicocokkan pada awal teks T.
2. Dengan bergerak dari kiri ke kanan, setiap karakter pada *pattern* P dibandingkan dengan karakter yang bersesuaian pada teks T sampai :
   a. semua karakter yang dibandingkan cocok (pencarian berhasil), atau
   b. dijumpai sebuah ketidakcocokan karakter (pencarian belum berhasil).
3. Jika *pattern* P belum ditemukan kecocokannya dan teks T belum habis, *pattern* P digeser satu karakter ke kanan dan langkah 2 diulangi.

Kompleksitas waktu algoritma *brute force* untuk kasus terbaik adalah O(n). Sedangkan kompleksitas untuk kasus terburuknya adalah O(mn).

2.2 Algoritma Knuth-Morris-Pratt (KMP)

Algoritma KMP ditemukan oleh Knuth dan Pratt, dan secara independen oleh J. H. Morris pada tahun 1977.

Pada algoritma KMP, jumlah pergeseran karakter yang akan dilakukan apabila terjadi ketidakcocokan saat pencarian *string* telah dicari terlebih dahulu dengan melakukan perhitungan fungsi pinggiran pada *pattern*. Sehingga algoritma ini lebih efisien dibandingkan dengan algoritma *brute force* karena pergeseran dapat dilakukan lebih dari satu karakter.

Fungsi pinggiran yang dihitung didefinisikan sebagai ukuran awalan terpanjang *pattern* P[i] yang merupakan akhiran dari P[i...m]. Pergeseran karakter akan dilakukan sebanyak jumlah karakter yang diperoleh dari fungsi pinggiran tersebut.

Algoritma KMP memliki kompleksitas waktu fase proses awal O(m) dan kompleksitas waktu fase pencarian *string* O(n+m).

2.3 Algoritma Boyer-Moore (BM)

Algoritma BM ditemukan oleh Bob Boyer and J. Strother Moore pada tahun 1977. Perbedaan paling mendasar dari algoritma BM dibandingkan dengan dua algoritma sebelumnya adalah algoritma BM melakukan perbandingan saat pencarian string dari kanan ke kiri (algoritma *brute force* dan KMP melakukan perbandingan dari kiri ke kanan). Prinsip utama algoritma ini adalah melompat sejauh mungkin apabila ditemukan ketidakcocokan saat perbandingan.

Seperti pada algoritma KMP, algoritm BM juga memiliki dua fase. Pada fase proses awal, dilakukan perhitungan pada *pattern* sebagai indikasi pergeseran. Adadua fungsi perhitungan yang dilakukan, yaitu *good-suffix shift* dan *bad-character shift.*

Algoritma BM memiliki kompleksitas waktu fase proses awal sebesar O(m+σ) dan kompleksitas waktu fase pencarian *string* sebesar O(mn). Perbandingan karakter yang terjadi saat kasus terburuk adalah sebanyak 3n. Sedangkan pada kasus terbaiknya, kompleksitas waktu yang dibutuhkan adalah O(n/m).

3 ALGORITMA APOSTOLICO-CROCHEMORE

Algoritma Apostolico-Crochemore adalah salah satu jenis algoritma pencarian *string* dari banyak jenis algoritma pencarian *string*. Algoritma Apostolico-Crochemore adalah algoritma sederhana yang melakukan perbandingan karakter pada teks sebanyak 3n/2 pada kasus terburuknya.

Algoritma Apostolico-Crochemore terdiri dari dua fase, yaitu fase proses awal (*preprocessing*) dan fase pencarian *string*. Pada fase proses awal dilakukan fungsi pinggiran untuk menentukan jumlah langkah pergeseran *pattern* terbesar dengan menggunakan perbandingan sebelum pencarian *string*. Pada fase pencarian *string* dilakukan perbandingan *pattern* pada teks.

Algoritma Apostolico-Crochemore mirip dengan algoritma Knuth-Morris-Path (KMP). Fungsi pinggiran (*border function*) yang digunakan mirip seperti fungsi pinggiran pada algoritma KMP dan proses pencariannya sama-sama dimulai dari kiri ke kanan. Akan tetapi, tahapan pencarian (proses urutan pembandingan) berbeda dengan algoritma KMP.

3.1 Fase Proses Awal

Fase proses awal (*preprocessing*) terhadap *pattern* x adalah dengan menghitung fungsi pinggiran yang mengindikasikan pergeseran x terbesar yang mungkin dengan menggunakan perbandingan yang dibentuk sebelum fase pencarian *string.* Dengan adanya fungsi pinggiran ini dapat dicegah pergeseran yang tidak berguna, seperti halnya pada algoritma *brute force*.

Fungsi pinggiran hanya bergantung pada karakter-karakter di dalam *pattern*, dan bukan pada karakter-karakter di dalam teks. Oleh karena itu, kita dapat melakukan perhitungan fungsi sebelum pencarian *string* dilakukan.

Fungsi pinggiran pada algoritma Apostolico-Crochemore adalah *kmpNext*. Algoritma untuk menghitung fungsi pinggiran kmpNext sebagai berikut (dalam bahasa C).

```
void preKmp(char *x, int m, int kmpNext[]) {
   int i, j;

   i = 0;
   j = kmpNext[0] = -1;
   while (i < m) {
      while (j > -1 && x[i] != x[j])
         j = kmpNext[j];
      i++;
      j++;
      if (x[i] == x[j])
         kmpNext[i] = kmpNext[j];
      else
         kmpNext[i] = j;
   }
}
```

Contoh penerapannya pada *pattern* x dapat dilihat pada tabel berikut.

Tabel 1. Contoh fungsi pinggiran kmpNext[i] pada *pattern* x = GCAGAGAG

i	0	1	2	3	4	5	6	7	8
$x[i]$	G	C	A	G	A	G	A	G	
$kmpNext[i]$	−1	0	0	−1	1	−1	1	−1	1

$\ell = 1$

3.2 Fase Pencarian *String*

Proses pencarian (pencocokan) *pattern* dengan teks pada algoritma Apostolico-Crochemore adalah sebagai berikut. Misalkan :

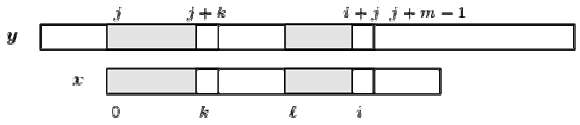
x adalah *string* dari *pattern* yang akan dicari
y adalah teks yang akan dicocokan dengan *pattern*
m adalah panjang x
n adalah panjang y

$\ell = 0$ jika x adalah karakter tunggal yang dipangkatkan ($x = c^m$ dengan c di dalam Σ) dan ℓ adalah posisi karakter pertama dari x yang berbeda dari x[0] ($x = a^{\ell} bu$ untuk a, b di dalam Σ, u di dalam Σ^*, dan $a \neq b$). Setiap pembandingan dilakukan dengan posisi yang berpola dengan urutan sebagai berikut : ℓ, ℓ+1, ..., m-2, m-1, 0, 1, ..., ℓ-1.

Selama fase pencarian, kita mempertimbangkan (i, j, k) di mana:

- jendela karakter diposisikan pada teks y[j .. j+m-1]
- $0 \leq k \leq \ell$ dan x[0 .. k-1] = y[j .. j+k-1]
- $\ell \leq i < m$ dan x[ℓ ..i-1] = y[j+ℓ .. i+j-1]

Inisialisasi awal dari (i, j, k) adalah (ℓ, 0, 0).



Gambar 1. Pembandingan dengan algoritma Apostolico-Crochemore memperhatikan (i, j, k).

Komputasi (perhitungan) untuk menentukan (i, j, k) berikutnya mempertimbangkan tiga kasus yang bergantung pada nilai i. Ketiga kasus tersebut sebagai berikut.

1. $i = \ell$
   Jika x[i] = y[i+j] maka (i, j, k) berikutnya adalah (i+1, j, k).
   Jika x[i] ≠ y[i+j] maka (i, j, k) berikutnya adalah (ℓ, j+1, max{0, k-1}).
2. $\ell < i < m$
   Jika x[i] = y[i+j] maka (i, j, k) berikutnya adalah (i+1, j, k).
   Jika x[i] ≠ y[i+j] maka ada dua kasus yang muncul yang bergantung pada nilai kmpNext[i] :
   - kmpNext[i] $\leq \ell$ maka (i, j, k) berikutnya (ℓ, i+j-kmpNext[i], max{0, kmpNext[i]}).
   - kmpNext[i] $> \ell$ maka (i, j, k) berikutnua (kmpNext[i], i+j-kmpNext[i], kmpNext[i] ℓ).
3. i = m
   Jika k < ℓ dan x[k] = y[j+k] maka (i, j, k) berikutnya adalah (i, j, k+1).
4. Sebaliknya salah satu dari k < ℓ dan x[k] ≠ y[j+k], atau k = ℓ (jika k = ℓ kemunculan x diberitahukan) pada kedua kasus tersebut perhitungan (i, j, k) berikutnya sama seperti perhitungan pada kasus ℓ < i < m.

Algoritma Apostolico-Crochemore secara keseluruhan dapat dilihat sebagai berikut (dalam bahasa C).

```
void AXAMAC(char *x, int m, char *y, int n) {
   int i, j, k, ell, kmpNext[XSIZE];

   /* Preprocessing */
   preKmp(x, m, kmpNext);
   for (ell = 1; x[ell - 1] == x[ell]; ell++);
   if (ell == m)
      ell = 0;

   /* Searching */
   i = ell;
   j = k = 0;
   while (j <= n - m) {
      while (i < m && x[i] == y[i + j])
         ++i;
      if (i >= m) {
         while (k < ell && x[k] == y[j + k])
            ++k;
         if (k >= ell)
            OUTPUT(j);
      }
      j += (i - kmpNext[i]);
      if (i == ell)
         k = MAX(0, k - 1);
      else
         if (kmpNext[i] <= ell) {
            k = MAX(0, kmpNext[i]);
            i = ell;
         }
         else {
            k = ell;
            i = kmpNext[i];
         }
   }
}
```

3.3 Kompleksitas Algoritma Apostolico-Crochemore

Kompleksitas waktu algoritma Apostolico-Crochemore pada fase proses awal adalah O(m) dengan m adalah

panjang *pattern* yang dicari. Fase pencarian *string* algoritm Apostolico-Crochemore memiliki kompleksitas waktu O(n) dengan n adalah panjang karakter teks. Algoritm Apostolico-Crochemore melakukan perbandingan karakter sebanyak 3n/2 karakter dalam kasus terburuk.

3.4 Contoh Pencarian *String* dengan Algoritma Apostolico-Crochemore

Agar lebih mudah mengerti cara kerja algoritma Apostolico-Crochemore, maka dapat kita lihat contoh sebagai berikut.

Teks y : B C A D C B C A B A B A B D A D A C A B
Pattern x : B C A B A B A B
Dengan algoritma Apostolico-Crochemore, carilah *pattern* x pada teks y tersebut.

Fase Proses Awal

Untuk menyelesaikan permasalahan tersebut dengan algoritma Apostolico-Crochemore, ada dua fase yang harus diselesaikan. Fase pertama yaitu fase proses awal (*preprocessing*). Pada fase ini dihitung fungsi pinggiran pada *pattern* x dengan menggunakan algoritma untuk menghitung fungsi pinggiran. Hasilnya diperoleh tabel kmpNext berikut.

Tabel 2. Fungsi pinggiran *kmpNext* pada *pattern* x = BCABABAB

i	0	1	2	3	4	5	6	7	8
x[i]	B	C	A	B	A	B	A	B	
kmpNext[i]	-1	0	0	-1	1	-1	1	-1	1

ℓ **= 1**

Fase Pencarian *String*

Pada fase ini, saat terjadi ketidakcocokan maka dilakukan pergeseran ke arah kanan sebanyak (i-kmpNext[i]) karakter. Tahapan-tahapan pencarian *string* dapat dilihat sebagai berikut.

Tahap 1
B C A D C B C A B A B A B D A D A C A B
1 2 3
B C A B A B A B
Digeser sebanyak : 3-kmpNext[3] = 3- -1 = 4 karakter

Tahap 2
B C A D C B C A B A B A B D A D A C A B
1
B C A B A B A B
Digeser sebanyak : 1-kmpNext[1] = 1-0 = 1 karakter

Tahap 3
B C A D C B C A B A B A B D A D A C A B
8 1 2 3 4 5 6 7
B C A B A B A B

Pada tahap 3, *pattern* x sudah ditemukan di dalam teks. Jumlah perbandingan yang dilakukan untuk mencari *pattern* dilakukan sebanyak 12 kali.

Jika pencarian *string* terus dilakukan sampai tes habis maka *pattern* digeser sebanyak : 8-kmpNext[8] = 8-1 = 7 karakter.

Tahap 4
B C A D C B C A B A B A B D A D A C A B
1
B C A B A B A B

Pada tahap 4, seluruh teks sudah habis dicari. Maka total perbandingan yang dilakukan seluruhnya sampai teks habis adalah 13 kali.

3.4 Perbandingan Antara Algoritma Apostolico-Crochemore dengan Algoritma Knuth-Morris-Pratt

Jika dibandingkan antara algoritma Apostolico-Crochemore dengan algoritma KMP, maka dapat dilihat persamaan dan perbedaan antara kedua algoritma tersebut.

Persamaan

Kedua algoritma sama-sama memiliki dua fase yaitu fase proses awal (*preprocessing*) dan fase pencarian *string*. Pada fase proses awal dilakukan perhitungan fungsi pinggiran terhadap *pattern* yang ingin dicari sebagai indikasi pergeseran karakter saat terjadi ketidacocokan perbandingan karakter antara *pattern* dengan teks. Kompleksitas waktu fase proses awal adalah O(m). Pencarian *string* sama-sama dilakukan dari kiri ke kanan.

Perbedaan

Pada algoritma Apostolico-Crochemore :
- kompleksitas waktu fase pencarian *string* adalah O(n)
- perbandingan dimulai dari karakter *pattern* yang kedua

Sedangkan pada algoritma KMP :
- kompleksitas waktu fase pencarian *string* adalah O(n+m)
- perbandingan dimulai dari karakter *pattern* yang pertama

4 APLIKASI MASALAH PENCARIAN *STRING*

Aplikasi dari masalah pencarian *string* ini banyak dipakai. Misalnya, pada program untuk pemrosesan kata dan *spreadsheet* terdapat fitur *FIND* untuk mencari lokasi sebuah kata atau nomor pada sebuah dokumen. Program untuk memeriksa *grammar* juga membutuhkan algoritma pencarian *string* untuk memeriksa kalimat.

Dalam basis data, kebanyakan pencarian ditangani dengan indeks. Tetapi jika mencari *string* yang kecil diantara basis data yang besar dibutuhkan algoritma pencarian *string*.

Bahkan bidang biologi molekuler juga membutuhkan algoritma pencarian *string* karena molekul biologi dapat diaproksimasi sebagai urutan nukleotida atau asam amino.

5 KESIMPULAN

Algoritma pencarian *string* digunakan dalam berbagai macam aplikasi di berbagai macam bidang sehingga algoritma pencarian *string* sendiri berkembang pesat. Saat ini algoritma pencarian *string* memiliki banyak variasi. Algoritma yang sudah umum digunakan ada dua yaitu algoritma Knuth-Morris-Pratt (KMP) dan algoritm Boyer-Moore (BM).

Ada salah satu variasi algoritma pencarian *string* yang menyerupai algoritma KMP yaitu algoritma Apostolico-Crochemore. Algoritma ini memiliki kompleksitas waktu untuk fase proses awal sebesar O(m) dan fase pencarian *string* sebesar O(n). Pada kasus terburuk, perbandingan karakter yang dilakukan sebanyak 3n/2 kali. Algoritma ini merupakan perbaikan dari algoritma KMP karena algoritma ini memiliki kompleksitas waktu yang lebih baik dari algoritma KMP.

REFERENSI

[1] Rinaldi Munir, "Diktat Kuliah IF2251 Strategi Algoritmik", Program Studi Teknik Informatika, STEI ITB, 2007.

[2] Wikimedia Foundation, Inc. "String searching algorithm", http://en.wikipedia.org/wiki/String_searching_algorithm, tanggal akses : 15 Mei 2007

[3] Christian.Charras, Thierry.Lecroq, "Apostolico-Crochemore algorithm", http://www-igm.univ-mlv.fr/~lecroq/string/node16.html , tanggal akses : 15 Mei 2007