ALGORITMA PENCARIAN *STRING* DENGAN ALGORITMA *BRUTE FORCE*, KNUTH-MORRIS-PRATT DAN ALGORITMA DUA ARAH

Dwinanto Cahyo

Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
Jalan Ganesha 10 Bandung
dw_dn_cahyo@yahoo.com

ABSTRAK

Algoritma pencarian *string* (*String Matching*) merupakan salah satu bagian terpenting dalam berbagai proses yang berkaitan dengan data dengan tipe teks. Berbagai perangkat lunak yang digunakan di seluruh dunia saat ini dengan sejumlah sistem operasi berbeda, menggunakan algoritma pencarian string sebagai dasar implementasi. Meskipun terdapat berbagai metode dan bentuk penyimpanan data yang telah dikembangkan hingga saat ini, tetapi tipe data teks masih merupakan bentuk utama pada proses transfer data. Fakta tersebut mendorong perkembangan berbagai algoritma pencarian *string* yang mangkus sehingga memberikan hasil yang akurat dengan waktu singkat.

Hingga saat ini terdapat puluhan algoritma pencarian *string* yang dapat dikelompokkan berdasarkan jenis pola dan metode pencocokan *string* yang digunakan, beberapa contoh algoritma pencarian *string* diantaranya adalah algoritma *brute force* yang merupakan algoritma paling sederhana, algoritma Knuth-Morris-Pratt, algoritma Boyer-Moore, algoritma dua jalur, algoritma Raita, algoritma *quick search* dan algoritma *alpha skip*.

Kata kunci: algoritma pencarian *string*, *brute force*, Knuth-Morris-Pratt, algoritma dua jalur.

1. PENDAHULUAN

Algoritma pencarian *string* banyak digunakan pada proses pencarian data pada suatu kumpulan data, tidak hanya pada bidang teknologi komputer semata, melainkan juga pada berbagai bidang keilmuan yang membutuhkan penyimpanan data dalam jumlah besar untuk kemudian dilakukan pencarian dan pengubahan data sesuai perubahan situasi dan kondisi.

Algoritma pencarian *string* bekerja dengan menghasilkan satu atau lebih pola suatu *string* tertentu dalam sebuah tipe data teks sebagai kumpulan *string*. Dengan demikian, algoritma pencarian *string* membutuhkan minimal dua data masukan, yaitu *string* yang akan dicari dan data teks atau *string* yang lebih panjang sebagai lokasi pencarian.

Algoritma pencarian *string* bekerja dengan memanfaatkan konsep *automata*, yaitu proses dibagi menjadi sejumlah kondisi berbeda sesuai masukan yang telah diproses. Berdasarkan arah pemeriksaan karakter, metode yang digunakan dikelompokkan menjadi:

1. Metode pembacaan berawal dari posisi kiri mengarah ke kanan, salah satu contohnya adalah algoritma Knut-Morris-Pratt. Metode ini tergolong alamiah karena sesuai dengan arah pembacaan biasa dan memiliki karakteristik seperti proses pada *automata*.
2. Metode pembacaan berawal dari posisi kanan mengarah ke kiri, misalnya algoritma Boyer-Moore. Metode ini umumnya menghasilkan algoritma yang sederhana dan dianggap paling mangkus.
3. Metode pencarian dengan aturan tertentu, salah satunya adalah pencarian dua arah yang diawali dengan kemunculan algoritma dua jalur yang dicetuskan oleh Crochemore-Perrin. Metode ini melakukan dua jenis pencarian, yang pertama adalah mencari bagian kanan pola dari kiri ke kanan, dan jika tidak ada ketidakcocokan, pencarian dilanjutkan dengan bagian kiri.
4. Metode yang tidak memiliki suatu pola tertentu, biasanya algoritma ini menggunakan sebagian metode dari algoritma pada tiga kelompok di atas yang dikombinasikan dengan metode lain, contohnya adalah algotima *quick search*.

Pada makalah ini dibahas tiga algoritma pencarian *string*, yaitu algoritma *brute force*, algoritma Knuth-Morris-Pratt dan algoritma dua jalur. Pembahasan mencakup metode pencocokan yang digunakan dan variasi yang mungkin pada bentuk implementasi algoritma pada bahasa pemrograman C, berikut beberapa penamaan variabel dan fungsi atau prosedur standar yang digunakan pada makalah ini :

- *char* input*, data teks sebagai sumber pencarian
- *char* tes*, pola *string* yang akan dicari
- *int len_i*, variabel jumlah karakter pada *input*

- *int len_t*, variabel jumlah karakter pada *tes*
- *int getLenStr(char* str)*, fungsi penghitung jumlah karakter pada sebuah *string*
- *void output(int x)*, prosedur pengembali nilai hasil pencarian
- *int memcmp(char* b1, char* b2, int len)*, fungsi bawaan dari bahasa C pada *library "string.h"* yang membandingkan *byte string b1* dengan *byte string b2* dan keduanya diasumsikan memiliki panjang *len*. Fungsi ini mengembalikan nilai *0* jika keduan *string* sama, atau nilai perbedaan karakter yang berbeda dalam *byte*.

2. ALGORITMA *BRUTE FORCE*

Algoritma *brute force* menggunakan metode pemeriksaan setiap karakter pada teks mulai dari posisi pertama hingga posisi ke (*len_i – len_t*), jika terdapat kecocokan dengan awal pola yang dicari, maka dilakukan pergeseran pada pola satu karakter ke kanan. Kompleksitas waktu pada algoritma *brute force* secara umum adalah sebesar ***O****(len_i*len_t)*, berikut bentuk implementasi algoritma *brute force* pada bahasa C:

```
void BF2(char* input, char* tes) {
  int inp = 0;
  int ts;
  int found = 0;
  int len_i = getLenStr(input);
  int len_t = getLenStr(tes);

/* pencarian */
  while ((inp<=(len_i-len_t)) &&
(!found)) {
    ts = 0;
    while ((ts<len_t) &&
(tes[ts]==input[ts+inp])) {
      ts++;
    }
    if (ts == (len_t)) {
      found = 1;
    } else {
      inp++;
    }
  }

  if (found) {
    output(inp);
  } else {
    output(-1);
  }
}
```

Bentuk implementasi tersebut memberikan nilai posisi pertama yang merupakan awal kesamaan pola *string* pada data teks, dengan indeks pertama dimulai dari *0*. Pemeriksaan dilakukan dengan melakukan pengulangan dari awal karakter teks menggunakan *while* hingga mencapai posisi (*len_i-len_t*) atau telah ditemukan kesamaan pola. Ketika ditemukan kesamaan dengan karakter pertama pada pola, pengulangan kedua dilakukan dengan *while* hingga akhir karakter pola atau tidak ditemukan lagi kesamaan karakter.

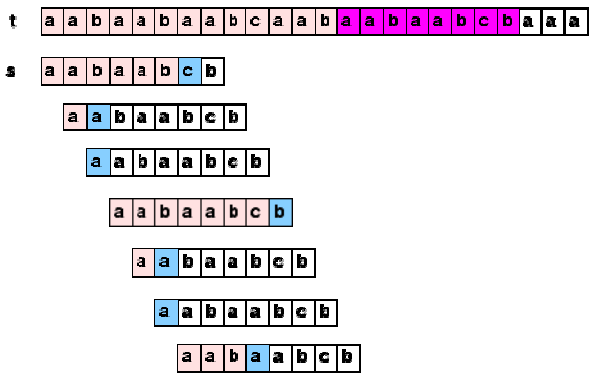
Dengan tetap berpegang pada prinsip dasar algoritma *brute force*, kita dapat mempersingkat penulisan algoritma pencarian *string* tersebut menjadi:

```
void BF(char *input, char *tes) {
  char *inp;
  int len_t = getLenStr(tes);

  /* pencarian */
  for (inp=input; *input!=endStr;
++input) {
    if (memcmp(tes,input,len_t)==0){
      output(input - inp);
    }
  }
}
```

Implementasi di atas mempersingkat pengulangan yang sebelumnya dilakukan dengan dua buah *while* menjadi sebuah *for*. Dengan menggunakan fungsi *memcmp()* dan variabel *inp*, metode yang digunakan seruap dengan sebelumnya, namun kali ini yang dibandingkan adalah keseluruhan pola dengan tetap melakukan pergeseran ke satu posisi pemeriksaan pada data teks ke kanan jika belum ditemukan kesamaan. Perbedaan waktu eksekusi program antara implementasi pertama dengan kedua tidak terlalu besar pada jumlah data teks di bawah 10.000 karakter.

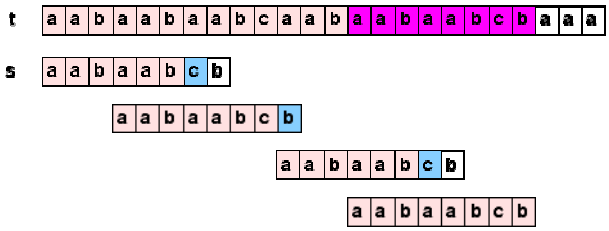
Kekurangan bentuk implementasi tersebut adalah kurang bersifat umum, hal ini disebabkan penggunaan fungsi bawaan *library* bahasa C, yaitu *memcmp()* yang belum tentu tersedia di bahasa pemrograman lain. Serta bentuk pengulangan yang tidak biasa sehingga sedikit sulit dimengerti oleh beberapa orang yang belum memahami bahasa C.



Gambar 2.1 Ilustrasi pencarian *string* dengan algoritma *brute force*

3. ALGORITMA KNUTH-MORRIS-PRATT

Algortima Knutt-Morris-Pratt dirancang dari hasil analisis terhadap algoritma Knutt-Morris dengan penambahan pada bagian prosedur awal atau yang juga dikenal dengan istilah fungsi pinggiran.



Gambar 3.1 Ilustrasi pencarian *string* dengan algoritma Knutt-Morris-Pratt

Dari gambar di atas tampak perbedaan jelas antara algoritma *brute force* dengan Knutt-Morris-Pratt dalam proses pencarian *string*, yaitu algoritma Knutt-Morris-Pratt tidak selalu melakukan pergeseran pola sebanyak satu karakter ke kanan, seperti yang dilakukan algoritma *brute force*.

Karakteristik utama algoritma Knutt-Morris-Pratt adalah dibutuhkan sebuah tabel (array) yang berisi nilai pinggiran dari masing-masing karakter pada *string tes*. Tabel tersebut dibuat sebelum proses pencarian *string* dilakukan dengan sebuah prosedur terpisah. Nilai pinggiran suatu karakter pada sebuah *string* merupakan jumlah pengulangan susunan karakter dari awal *string* hingga karakter tersebut. Pembahasan mendetail mengenai nilai pinggiran berada di luar cakupan makalah ini sehingga tidak akan dibahas lebih lanjut.

Selain itu, secara kompleksitas, algoritma Knutt-Morris-Pratt memiliki kompleksitas waktu yang lebih sedikit dibandingkan dengan algoritma *brute force*, yaitu untuk prosedur penyusunan tabel berisi nilai pinggiran adalah sebesar ***O****(len_i)*., sedangkan untuk prosedur pencarian *string* sebesar ***O****(len_i+len_t)*. Berikut bentuk dasar implementasi dalam bahasa C:

```
/* pembuatan array of nilai tepi */
void setTepi2(char *tesx, int len_ix,
int* arrKmp) {
  arrKmp[1] = 0 ;
  int q = 2;
  int k = 0;
  for (q=2; q<=len_ix; q++) {
   while ((k>0) && (tesx[q]!=tesx[k+1]))
{
    k = arrKmp[k];
   }
   if (tesx[q] == tesx[k+1]) {
    k = k + 1;
   }
   arrKmp[q] = k;
  }
 }
```

```
/* pencarian string */
void finKMP2(char *input, char *tes) {
  int len_t = getLenStr(input);
  int len_i = getLenStr(tes);
  int inp, ts, found;
  int* arrKmp = (int*) malloc (len_t *
sizeof(int));

  setTepi2(tes, len_i, arrKmp);
  ts = 0;
  inp = 1;
  found = 0;
  while ((inp<=len_t) && (!found)) {
   while ((ts>0) &&
(tes[ts+1]!=input[inp])) {
    ts = arrKmp[ts];
   }
   if (tes[ts+1] == input[inp]) {
    ts++;
   }
   if (ts == len_i) {
    found = 1;
   } else {
    inp++;
   }
  }
  if (found) {
   output(inp-len_i);
  } else {
   output(-1);
  }
 }
```

Secara garis besar, algortima Knutt-Morris-Pratt menggunakan tabel berisi nilai pinggiran untuk mengetahui nilai pergeseran pola. Setiap kali ditemui perbedaan karakter antara teks dengan karakter pola pada posisi ke (*x+1*), maka nilai pergeseran pola dihitung melalui pengurangan antara *x* dengan nilai pinggiran terbesar pada tabel dengan indeks [*0..x*], dengan posisi pergeseran dihitung dari posisi awal pola.

x = posisi sebelum ditemukan perbedaan karakter
b(x) = nilai pinggiran terbesar pada indeks [0..x]
i = jumlah pergeseran
i = x – b(x)

Kemudian, perbandingan selanjutnya dilakukan dimulai dari posisi ke *i* dihitung dari awal pola.

Selanjutnya, dengan memberikan sedikit perubahan pada prosedur pencarian *string* dengan algoritma Knutt-Morris-Pratt, kita akan memperoleh bentuk implementasi sebagai berikut:

```
/* pembuatan array of nilai tepi */
void preKmp(char *tes_x, int len_tx,
int* arrKMP) {
  int i, j;
  i = 0;
  j = -1;
  arrKMP[0] = -1;
  while (i < len_tx) {
   while (j > -1 && tes_x[i] !=
tes_x[j]) {
    j = arrKMP[j];
```

```
/* pencarian string */
void finKMP(char *input, char *tes) {
  int len_t = getLenStr(tes);
  int len_i = getLenStr(input);
  int i, j;
  int* arrKmp = (int*) malloc (len_t *
sizeof(int));
  /* pengisian array preKmp */
  preKmp(tes, len_t, arrKmp);

  /* algoritma pencarian */
  i = 0;
  j = 0;
  while (j < len_i) {
   while (i > -1 && tes[i] != input[j])
{
    i = arrKmp[i];
   }
   i++;
   j++;
   if (i >= len_t) {
    output(j-i+1);
    i = arrKmp[i];
   }
  }
 }
```

Implementasi di atas memiliki sejumlah perbedaan metode dalam prosedur pengisian tabel nilai pinggir, kedua implementasi algoritma Knutt-Morris-Pratt tidak menggunakan fungsi atau prosedur dari *library* bahasa C, sehingga memungkinkan untuk diimplementasikan dalam bahasa pemrograman lain. Sama halnya seperti pada algoritma *brute force*, kedua bentuk implementasi tersebut tidak memberikan perbedaan yang besar dalam waktu eksekusi ketika digunakan pada karakter berjumlah di bawah 10.000.

4. ALGORITMA DUA JALUR

Kedua algoritma yang dijelaskan sebelumnya, yaitu algoritma *brute force* dan algoritma Knutt-Morris-Pratt merupakan contoh algoritma pencarian *string* yang menggunakan metode pemeriksaan satu arah, yaitu dari kiri ke kanan. Algoritma dua jalur (*Two Ways Algorythm*) merupakan suatu bentuk algoritma pencarian *string* yang mengkombinasikan metode pemeriksaan dari kiri ke kanan dengan metode dari kanan ke kiri.

Karakteristik algoritma dua jalan adalah bahwa pola *x* yang akan diperiksa dibagi menjadi dua bagian, yaitu *xl* dan *xr*, sehingga $x = xl + xr$. Oleh karena itu, diperlukan sebuah prosedur untuk menghasilkan pembagian pola yang baik dan memberikan efisiensi pada proses pencarian *string*. Sedangkan pada proses pencarian *string* itu sendiri, kedua bagian pola diproses secara bergantian, yaitu diawali dengan pemeriksaan pola bagian kanan (*xr*) dari arah kiri ke kanan, jika tidak ditemukan ketidakcocokan, proses akan dilanjutkan dengan pemeriksaan pola bagian kiri (*xl*) dari arah kanan ke kiri.

Prosedur pembagian pola pada algoritma dua jalan memiliki kompleksitas waktu senilai ***O****(len_i)*, sedangkan proses pencarian *string* memiliki kompleksias waktu ***O****(len_t)*. Dengan total operasi perbandingan pada kasus terburuk adalah sejumlah *2len_i-len_t*.

Berikut implementasi algoritma dua jalan pada bahasa C

```
/* prosedur pembagi pola */
 int maxSuf(char *tesx, int len_tx, int
*p) {
  int ms, j, k;
  char a, b;

  ms = -1;
  j = 0;
  k = *p = 1;
  while (j + k < len_tx) {
   a = tesx[j + k];
   b = tesx[ms + k];
   if (a < b) {
    j += k;
    k = 1;
    *p = j - ms;
   } else if (a == b) {
    if (k != *p) {
     ++k;
    } else {
     j += *p;
     k = 1;
    }
   } else { /* a > b */
    ms = j;
    j = ms + 1;
    k = *p = 1;
   }
  }
  return(ms);
 }
int maxSufTilde(char *tesx, int len_tx,
int *p) {
  int ms, j, k;
  char a, b;

  ms = -1;
  j = 0;
  k = *p = 1;
  while (j + k < len_tx) {
   a = tesx[j + k];
   b = tesx[ms + k];
   if (a > b) {
    j += k;
    k = 1;
    *p = j - ms;
   } else if (a == b) {
    if (k != *p) {
     ++k;
    } else {
     j += *p;
     k = 1;
    }
   } else { /* a < b */
    ms = j;
    j = ms + 1;
    k = *p = 1;
   }
  }
  return(ms);
 }
```

```
/* algoritma dua jalan */
 void TW(char *inp, char *tes) {
  int i, j, ell, memory, p, per, q;
  int m = getLenStr(tes);
  int n = getLenStr(inp);

  /* Persiapan */
  i = maxSuf(tes, m, &p);
  j = maxSufTilde(tes, m, &q);
  if (i > j) {
   ell = i;
   per = p;
  } else {
   ell = j;
   per = q;
  }

  /* Pencarian */
  if (memcmp(tes, tes+per, ell+1)==0) {
   j = 0;
   memory = -1;
   while (j <= n-m) {
    i = max(ell, memory) + 1;
    while (i<m && (tes[i]==inp[i+j])) {
     ++i;
    }
    if (i >= m) {
     i = ell;
     while (i>memory &&
(tes[i]==inp[i+j])) {
      --i;
     }
     if (i <= memory) {
      output(j);
     }
     j += per;
     memory = m - per - 1;
    } else {
     j += (i - ell);
     memory = -1;
    }
   }
  } else {
   per = max(ell+1, m-ell-1) + 1;
   j = 0;
   while (j <= n - m) {
    i = ell + 1;
    while (i<m && (tes[i]==inp[i+j])) {
     ++i;
    }
    if (i >= m) {
     i = ell;
     while (i>=0 && tes[i]==inp[i+j]) {
      --i;
     }
     if (i < 0) {
      output(j);
     }
     j += per;
    } else {
     j += (i - ell);
    }
   }
  }
 }
```

Dengan pembagian *xl* dan *xr*, maka pengulangan antara *xl* dan *xr* yang berupa w akan ada jika dan hanya jika kondisi berikut terpenuhi:

- *w* adalah akhiran *xl* atau *xl* adalah akhiran *w*
- *w* adalah awalan *xr* atau *xr* adalah awalan *w*

Dengan kata lain,*w* akan muncul pada kedua sisi antara *xl* dan *xr*. Jumlah pengulangan minimum yang timbul disebut dengan periode lokal dan dinotasikan dengan *per(xl,xr)*, ketika nilai pengulangan antara kedua hasil pembagian pola adalah sama dengan nilai *per(xl,xr)*, maka kondisi tersebut disebut faktorisasi kritis dari *x* dan nilai tersebut yang dibutuhkan dalam pemrosesan algoritma dua jalan.

Untuk menghitung nilai faktorisasi kritis, kita perlu menghitung maksimal akhiran dari *x* dengan arah dari kiri dan sebaliknya, kemudian baru dipilih nilai maksimum diantara keduanya sebagai faktorisasi kritis.

Penggunaan algoritma dua jalan ini telah diuji cona dan hasilnya ketika menggunakan karakter lebih dari 10.000, diperlukan waktu yang mendekati waktu yang dbutuhkan oleh algoritma Knutt-Morris-Pratt.

5. KESIMPULAN

Algoritma pencarian *string* merupakan salah satu komponen penting dalam pemrosesan *string* dan juga digunakan sebagai dasar dari berbagai aplikasi perangkat lunak khususnya sebagai bentuk dasar transfer data.

Algoritma pencarian *string* bekerja dengan menghasilkan posisi tempat terjadinya pengulangan suatu pola *string* dalam *string* yang lebih panjang.

Beberapa algoritma yang dianggap mangkus dan tergolong sering dipelajari adalah algoritma Knutt-Morris-Pratt dan algoritma dua jalan, masing-masing menggunakan metode pemeriksaan *string* yang berbeda dalam konteks arah. Kemudian, terdapat pula algoritma dasar dan tergolong kurang efektif, yaitu algoritma *brute force*.

REFERENSI

[1] http://www-igm.univ.mlv.fr, diakses pada tanggal 14 Mei 2007 pukul 15.30.

[2] http://en.wikipedia.org/wiki/Algorythm, diakses pada tanggal 19 Mei 2007 pukul 12.30.

[3] Rinaldi Muni. *Diktat Kuliah Strategi Algoritmik*. Program Studi Teknik Informatika ITB: 2007.