PENGEMBANGAN DFA DALAM ALGORITMA PENCARIAN STRING

Nurio Juliandatu Masido

Sekolah Teknik Elektro dan Informatika ITB
Kampus ITB, Jalan Ganesha no 10. Bandung
e-mail: nuriojm@yahoo.com, if15083@students.if.itb.ac.id

ABSTRAK

Pencarian *string* (Pencocokan string[1]) merupakan salah satu masalah yang paling penting dalam dunia keinformatikaan. Berbagai masalah dalam ilmu komputer memerlukan pencarian dan pencocokan string. Misalnya saja pada compiler-compiler program, dan pada saat diperlukannya fungsi pencarian pada suatu aplikasi untuk mendapatkan informasi secara cepat. *DFA* (*Determined Finite Automata*) telah dikenal sebagai suatu metode yang digunakan untuk menerima suatu pattern atau tidak menerimanya, bagaimana cara mengembangakan DFA dalam algoritma pencarian string akan dijelaskan lebih lanjut di dalam makalah ini.

Kata kunci: *string*, pencocokan, pencarian, DFA.

1. PENDAHULUAN

Pencarian string adalah pokok perhatian baik dari sisi teori maupun dalam praktek di bidang ilmu komputer. Pencocokan string menjadi sangat penting dalam pemrosesan teks yang panjang (jumlah yang sangat besar). Algoritma pencarian string mempunyai peranan penting dalam teori ilmu komputer dengan adanya tantangan berbagai masalah yang dapat diperluas dalam bidang-bidang :

- pencocokan pola (*pattern*)
- pembandingan genomik (biologi molekular)
- mendapatkan informasi
- kompresi data/teks
- pemrosesan string dalam database yang besar

String adalah rangkaian simbol-simbol[3] yang digambarkan dari suatu himpunan yang telah ditentukan yang disebut alfabet/abjad.

Contoh Alfabet:

- Kode ASCII, Unicode
- Abjad biner ({0,1})
- Sistem berbasis pasangan DNA ({A,C,G,T})
- Abjad Latin, Abjad Yunani, Abjad Cina, Hiragana, Katakana, dan lain-lain.

Contoh strings:

- program-program Java/C/ADA, dokumen HTML/XML
- rangkaian DNA
- file gambar/video/suara

Masalah pencocokan string (string matching atau pattern matching) dirumuskan sebagai berikut [1]:

1. teks (*text*), yaitu (*long*) *string* yang panjangnya n karakter
2. *pattern*, yaitu *string* dengan panjang m karakter dimaan (m < n) yang akan di cari di dalam teks.

Contoh 1:

Pattern : rio
Teks : romario
^ target

Kata ’automata’ adalah bentuk jamak dari ‘automaton’ yang berasal dari kata Yunani *automatos*[2]. Dari kata Yunani tersebut kata ”automaton” memiliki arti *self-acting*. Dalam kamus The American Heritage Dictionary, kata *automaton* memiliki beberapa arti berikut:

1. *a robot*
2. *one that behaves in an automatic or mechanical fashion*

DFA sendiri adalah suatu mesin yang biasanya digunakan untuk keperluan yang khusus, seperti pada mesin jaja yang status-status dan pattern-nya telah ditentukan sebelumnya.

Contoh 2 :

Pattern : BEGIN
Teks : BEGIH -> tidak diterima

Dengan DFA yang menerima BEGIN, maka string BEGIH tidak diterima. karena ketika begitu bertemu H status kembali direset ke status awal.

Karena keterbatasan harus adanya status-status dan simbol-simbol yang telah ditentukan sebelumnya ini tampak sulit untuk mengembangkannya sebagai algoritma untuk pencarian string dimana status-status dan bahkan jumlah statusnya belum diketahui sebelumnya, hanya diketahui setelah kode dieksekusi. artinya kita harus dapat membangun status-status tersebut pada saat kode di eksekusi.

2. ALGORITMA DFA BIASA

Untuk memecahkan masalah ini pertama-tama kita akan melihat metoda pemakaian DFA untuk pencarian String yang telah ditentukan sebelumnya.

Diberikan sebuah teks yang panjang dan mesin DFA harus menemukan kata 'BEGIN' pertama yang terdapat di dalam teks tersebut. Maka algoritma-nya dapat berupa seperti algoritma di bawah ini (dalam bahasa Java):

```
public int FindSpecifiedPatternWithDFA(String T){
   int i = -1;
   int status = -1;
   int start = -1;
   while((i<T.length())&&(status!=5)){
        swicth(T[i]){
           'B' : if (status == -1){
                      status = 1;
                 }else{
                      status = -1;
                 }
                 break;
           'E' : if (status == 1){
                      status = 2;
                 }else{
                      status = -1;
                 }
                 break;
            'G' : if (status == 2){
                      status = 3;
                  }else{
                       status = -1;
                  }
                 break;
            'I' : if (status == 3){
                      status = 4;
                    }else{
                      status = -1;
                    }
                 break;
            'N' : if (status == 4){
                         status = 5;
                         start = i-4;
                    }else{
                         status = -1;
                    }
                    break;
                 }
                 i++;
        }
        return start;
  }
```

Algoritma 1. Pencarian BEGIN di dalam teks dengan menggunakan DFA

Dalam algoritma fungsi FindSpecifiedPatternWithDFA di atas, kita membaca satu-satu karakter-karakter di dalam string **T** masukan sampai status tercapai 5 (jumlah karakter 'BEGIN') atau telah mencapai akhir string **T**. Kemudian akhirnya fungsi ini akan mengembalikan indeks(posisi) karakter B (karakter pertama 'BEGIN') apabila *pattern* 'BEGIN' ditemukan di dalam string tadi.

Algoritma di atas masih dapat di optimasi dengan tidak lagi membaca teks apabila panjang sisa teks lebih kecil dari karakter yang diperlukan untuk melengkapi status, yaitu dengan cara menambahkan kondisi ((T.length() - i) >= (5 – Status)) pada *while*.

Dalam algoritma di atas dapat kita lihat bahwa fungsi (selanjutnya penulis sebut mesin DFA) ini mengubah Status setiap kali pembacaan satu karakter dari string T, dan akan berhenti pada saat Status telah sama dengan panjang 'BEGIN' (pola yang dicari) atau telah mencapai ujung string (pola tidak ditemukan).

Apabila kita lihat lebih teliti lagi, algoritma mesin DFA ini memiliki suatu pola yang dapat kita lihat dengan meneliti lebih rinci lagi. 'B' adalah karakter pertama (pada posisi ke-1) pada pola 'BEGIN', dan apabila status masih status awal (-1) maka jika membaca 'B' status akan di ubah ke nilai 1 (seperti posisi B dalam pola 'BEGIN'), jika karakter yang dibaca bukan 'B' maka status akan di ubah kembali ke status awal (-1). 'E' adalah karakter ke-2, bila status adalah status 1 dan membaca 'E' maka status akan di ubah ke status sesuai posisi 'E' di dalam *pattern* 'BEGIN' yaitu status 2, jika bukan 'E' maka status diubah kembali ke status awal . Demikian seterusnya sampai pada saat status bernilai 4, jika membaca 'N' maka status akan di set ke 5, dan start di beri nilai counter (i) dikurangi 4, jika tidak maka start tetap bernilai -1, dan status diubah kembali ke -1. Start menunjukkan posisi karakter pertama 'B' jika *pattern* 'BEGIN' ditemukan, jika *pattern* ini tidak ditemukan maka start akan bernilai -1.

3. PENGEMBANGANALGORITMA DFA

Setelah pembahasan pada bagian 2, tampaklah sebuah cara untuk mengembangkan algoritma tersebut untuk pencarian sembarang pola (*pattern*) dalam sembarang abjad. Pembangkitan status-status dapat dilakukan dengan menggunakan status sebanyak jumlah karakter di dalam *pattern* di tambah satu status (status awal). Dan status terima adalah status bernilai jumlah karakter di dalam pola. Kemudian mengganti pola percabangan *switch* dengan *if* di dalam *while*. Rincinya dapat di lihat seperti pada algoritma di bawah ini :

```
  /**
      Algoritma    di    bawah    ini    merupakan
pengembangan dari algoritma mencari pattern yang
patternnya    telah    ditentukan    sebelumnya,
```

```
dikembangkan dengan adanya pola perubahan status
yang mengikuti kecocokan karakter.
  Dalam contoh berikut, masukan S adalah pola
yang di cari, sedangkan T adalah string tempat
mencari pola S.
  */

public int FindWithDFA(String S,String T){

  int i, j, start, Status;
  i = j = Status = 0;
  start = -1;

// Store the length of both T and S
// so that no need to read it again next time

  int pjPat = S.length();
  int pjText = T.length();

//Create Symbols
  char [] Sym = new char[S.length()];
  for(int i = 0; i<S.length(); i++){
      Sym[i] = S[i];
  }

// Selama masih mungkin terdapat
while((Status != pjPat) && (i < pjText) &&
 ((Status + (pjText-i)) > pjPat)){
   j = 0;
   // Cari status
   while(Status != j){
     j++;
   }
   // Status ditemukan
   // bandingkan karakter dengan simbol
   // pada posisi status berikutnya
   if(T.charAt(i)==Sym[j]){
     // jika sama maka status ditambah
      Status++;
      if(Status == pjPat){
         start = i-pjPat;
      }
   }else{
     // jika tidak sama maka status
     // diubah kembali menjadi 0
     Status = 0;
   }
   i++;
 }// endwhile
 // kembalikan nilai posisi karakter awal string
 // atau -1 jika tidak ditemukan pattern di
 // dalam teks (string)
 return start;
}
```

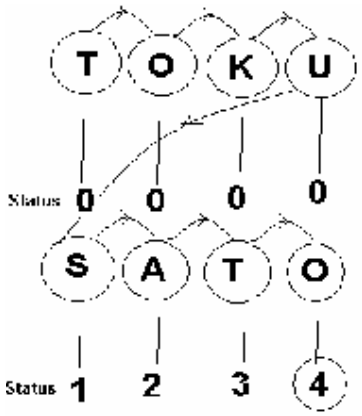
Sederhana sekali, karena status-status dapat diinterpretasikan dengan bilangan integer (bilangan bulat), dalam algoritma tersebut, status-status DFA dianggap adalah posisi kecocokan terakhir simbol-simbol dalam *pattern*. Sedangkan simbol-simbol yang digunakan untuk mencocokkan di-*generate* secara otomatis dengan memakai simbol-simbol yang ada pada *pattern* sendiri.

Dalam algoritma tersebut pencarian dilakukan dengan menggunakan dua buah *counter* (i dan j), dimana salah satunya menunjukkan simbol yang sedang ditunjuk pada T, dan yang lain digunakan untuk membandingkan status dan pemanggilan simbol-simbol yang tadinya telah di-*generate*.

Misalkan contoh *pattern* yang dicari adalah TOKUSATSU maka, simbol-simbol yang digenerate (dalam contoh disimpan di dalam Array Sym), adalah ‘T’, ‘O’, ‘K’, ’U’, ‘S’, ‘A’, ‘T’, ‘S’, dan ‘U’. Status-status adalah himpunan bilangan dari 1-9, ditambah bilangan 0 sebagai status awal. Sama seperti pada mesin DFA biasa pembacaan dilakukan satu-persatu dari T, dan dibaca oleh mesin, kemudian dibandingkan dengan simbol pada posisi ke status di tambah satu, jika sama maka status ditambah satu, jika tidak maka status kembali diubah menjadi 0.

Misalnya dalam string ‘TOKUSATOKUSATSU’ akan dicari pola ‘SATO’. Maka langkah per langkah penyelesaian dengan algoritma ini adalah sebagai berikut :

1. Ciptakan simbol-simbol yang akan di bandingkan dengan simbol-simbol yang terdapat pada pola. Maka simbol-simbol itu adalah : ‘S’, ‘A’, ‘T’, dan ‘O’
2. Baca satu persatu string ‘TOKUSATOKUSAT SU’ untuk mengubah status DFA, jika menemukan simbol yang sama dengan simbol pada array Sym pada posisi status, ubah status dengan status di tambah satu, jika tidak sama, maka ubah status menjadi 0.
3. Ulangi 2 sampai menemukan status yang sama dengan panjang pattern atau telah membaca semua karakter di dalam pattern atau tidak mungkin lagi melengkapi status dengan sisa karakter yang ada pada string.



Gambar 1. Perubahan status DFA selama pembacaan string ‘TOKUSATOKUSATSU’ sampai status sama dengan panjang *pattern* yang di cari (‘SATO’).

Algoritma ini selain memecahkan masalah *generate* status dan simbol-simbol juga memberikan kecepatan yang lebih baik daripada algoritma sebelumnya pada saat pembandingan, karena langsung membandingkan karakter

berikutnya dengan simbol (Sym) yang berada pada posisi status tersebut.

4. KELEBIHAN DAN KELEMAHAN DFA

Setiap hal pasti punya kelebihan dan kekurangan, begitu juga algoritma DFA ini. salah satu kelemahannya di bandingkan algoritma lain seperti Knuth-Morris-Pratt (KMP) adalah masih membaca karakter-karakter (simbol-simbol) yang sebenarnya lebih baik dilewatkan saja karena tidak mengarah ke solusi. Namun Algoritma ini juga memiliki kelebihan, dimana kelebihan DFA di bandingkan dengan algoritma lainnya adalah dalam meminimumkan pembacaan source. Sangat bagus apabila yang akan dibaca berasal dari source yang susah di baca. Karena proses pembacaan dengan menggunakan DFA hanya dilakukan sekali lewat saja (maksimal sebanyak karakter).

5. KESIMPULAN

1. Sebagai salah satu perhatian utama dalam dunia ilmu komputer, algoritma pencocokan atau pencarian string perlu terus dikembangkan. Sebagai contoh **Google**, suatu situs pencarian yang sangat terkenal, hanya dengan memanfaatkan algoritma pencarian string dan penyimpanan data yang baik, menjadi salah satu yang terbaik dalam pencarian data yang diinginkan pengguna.
2. Dengan sedikit pengubahan sebuah mesin DFA biasa akan dapat dimodifikasi menjadi sebuah mesin DFA untuk dapat menerima tidak hanya pola yang telah ditentukan sebelumnya tapi juga pola-pola yang ditentukan pada saat eksekusi program (sebagai masukan program).
3. Keuntungan menggunakan DFA untuk pencocokan string adalah meminimkan pembacaan (input/output) dari luar sistem, karena pembacaan dilakukan maksimal hanya sebanyak jumlah karakter (simbol) di dalam *source* (O(n) = n). Bagus digunakan jika kecepatan perhitungan tidak menjadi masalah, ketika harus membaca *source* yang sulit di jangkau, sehingga harus diminimumkan pembacaan source.

REFERENSI

[1] Munir, Rinaldi, "Diktat Kuliah IF2251 Strategi Algoritmik", 2006, Program Studi Teknik Informatika Sekolah Teknik Elektro dan Informatika Institut Teknologi Bandung,186.

[2] Harlili, "Diktat Kuliah IF2253 Otomata dan Teori Bahasa Formal".

[3] http://blog.platinumsolutions.com/node/140, String Matching and Finite States, di akses 22 Mei 2007 jam 12:25