Analisis Algoritma Pencarian String (*String Matching*)

Eko Gunocipto Hartoyo[1], Yus Gias Vembrina[2], Anggia Ferdina Meilana[3]

Departemen Teknik Informatika, Institut Teknologi Bandung
Jl. Ganesha 10, Bandung

E-mail : if13024@students.if.itb.ac.id[1],
if13042@students.if.itb.ac.id[2], if13043@students.if.itb.ac.id[3]

Abstrak

Permasalahan pencocokan string (*string matching*) merupakan permasalahan yang sangat terkenal dalam dunia informatika. Contoh implementasi dari permasalahan pencocokan string adalah pada pencocokan sebuah string pada Microsoft Word atau editor, atau dalam kasus yang lebih besar lagi, yaitu pencocokan website dengan memasukkan kata-kata kunci sebagaimana yang telah diimplementasikan pada search engine, seperti Yahoo atau Google.
Berbagai cara yang telah diterapkan untuk menyelesaikan permasalahan ini, diantaranya Algoritma Staightforward Matching, dengan menggunakan Finite Automata, Algoritma Knuth-Morris-Pratt, Algoritma Boyer-Moore. Dalam paper yang sederhana ini kami akan mencoba menganalisis masing-masing cara tersebut.

Kata kunci: *Staightforward Matching, Finite Automata, Knuth-Morris-Pratt, Boyer-Moore*

1. Pendahuluan

Masalah utama dalam pencarian string adalah untuk mencari sebuah string yang terdiri dari beberapa karakter (yang biasa disebut pattern) dalam sejumlah besar text. Pencarian string juga bias digunakan untuk mencari pola bit dalam sejumlah besar file binary. Dalam menganalisis algoritma-algoritma pencarian string kami akan mengasumsikan bahwa panjang string (pattern) adalah S dan panjang text tempat pattern tersebut dicari dimisalkan dengan T.

2. Analisis berbagai Algoritma Pencocokan String Sederhana

2.1Analisis Straightforward matching

Algoritma straightforward matching adalah cara brute-force untuk menyelesaikan permasalahan pencarian string. Jalannya algoritma ini adalah sebagai berikut.

Awalnya kita membandingkan karakter pertama dari string dangan karakter pertama dari text. Jika sama maka kita bergerak ke karakter selanjutnya sampai kita telah mencocokan keseluruhan string atau menemukan karakter yang tidak sama Jika kita menemukan karakter yang tidak sama maka kita akan bergerak satu langkah kedepan dan akan memulai lagi mencocokan string dari awal.
Kasus terburuk adalah kita sukses dalam setiap perbandingan kecuali pada karakter terakhir string.
Dalam kasus ini kita melakukan S*(T-S+1) kali perbandingan. Algoritma ini sangat powerfull, namun akan sangat lama dan tidak efektif bila kita harus mencari sebuah string dalam text yang sangat panjang. Bayangkan bila search engine yang ada sekarang menggunakan Algoritma ini, betapa lamanya kita harus menunggu karena informasi yang kita inginkan baru akan muncul setelah search engine membandingkan karakter demi karakter yang ada di seluruh dunia.

2.2 Analisis Finite Automata

Finite Automata digunakan untuk memutuskan apakah sebuah kata atau string adalah kata yang diterima oleh mesin automata tersebut. Sebuah aoutomata memiliki status dan fungsi transisi yang merubah status berdasarkan status saat ini dan karakter yang dibaca saat ini. Ada yang disebut sebagai status akhir dan status ini mungkin lebih dari satu. Bila string itu dijalankan dan berakhir disalah satu status akhir maka string itu diterima atrau termasuk dalam behasa tersebut.

Kita bisa membuat finite automata untuk menerima string yang kita cari dan jika kita menemukan sebuah kata yang berakhir di status akhir finite automata yang kita buat maka kita bisa mengetahiu bahwa kita telah menemukan string yang kita cari.

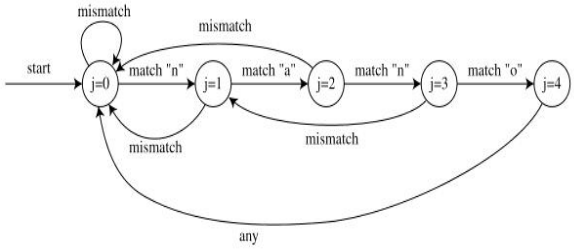
Karena kita melihat setiap karakter yang ada pada text, maka sedikitnya kita telah melakukan T perbandingan, namun keistimewaan finite automata kita bisa menentukan status mana yang akan dituju setelah sebuah status menerima sebuah karakter. Hal ini yang kemudian menjadi dasar pemikiran untuk Algoritma Knuth-Morris-Pratt.

2.3 Analisis Algoritma Knuth-Morris-Pratt

Dalam Algoritma Knuth-Morris-Pratt (KMP), untuk setiap karakter yang dibandingkan kita bisa memutuskan apakah berhasil atau gagal.

Algoritma KMP membangun sebuah mesin automata yang status-statusnya adalah status dari string yang sedang kita cari. Dan setiap status memiliki fungsi berhasil dan gagal. Berhasil artinya status akan bergerak lebih mendekat ke status akhir dan gagal artinya status bisa jadi semakin jauh atau tetap terhadap status akhir. Kita akan mendapatkan sebuah karakter dari text saat kita berhasil dalam membandingkan dan akan mereuse karakter bila kita gagal.

Contohnya :



Perbandingan yang gagal paling banyak adalah sejumlah S-1 kali. Dan yang membuat algoritma ini lebih baik dari brute-force adalah jumlah perbandingan yang lebih sedikit dari algoritma standar brute force. Kompleksitas brute force adalah O(S * T) , sedangkan algoritma KMP memiliki kompleksitas O(S + T).

2.4 Analisis Algoritma Boyer-Moore

Rinaldi[1] mendefinisikan bahwa Algoritma Boyer-Moore merupakan variasi lain dari pencarian string dengan melompat maju sejauh mungkin. Tetapi, algoritma Boyer Moore(BM) berbeda dengan algoritma Knuth-Morris-Pratt (KMP), dimana algoritma Boyer Moore melakukan perbandingan pattern mulai dari kanan sedangkan algoritma Knuth-Morris-Pratt melakukan perbandingan dari kiri.

Jika kita mencocokan dari kanan string atau *pattern* maka ketidakcocokan mungkin membantu kita untuk menggerakkan *pattern* tersebut dengan jarak yang lebih jauh yang artinya akan lebih signifikan dalam mengurangi proses perbandingan. Jadi kita bisa melompati atau tidak melakukan perbandingan-perbandingan karakter yang diprediksikan akan gagal.

Contoh :
Text : there they are
Pattern : they

Dengan satu kali perbandingan karakter ‘r’ dan ‘y’ kita bisa melihat bahwa gerakan dari 4 karakter adalah yang terbaik. Kita juga harus mempertimbangkan karakter-karakter yang telah kita cocokan jadi kita tidak bergeser terlalu sedikit.

Algoritma Boyer Moore ini menggunakan gerakan geser (slide) dan lompat (jump). Garakan geser memberikan informasi berapa banyak *pattern* harus digeser untuk mendapatkan karekter yang cocok. Gerakan Lompat memberikan informasi berapa banyak *pattern* harus digeser untuk mencocokan karakter terakhir yang cocok dengan kemunculan awalnya dengan *pattern*.

Studi telah menunjukan bahwa dengan text bahasa natural dan dengan sebuah *pattern* yang terdiri dari enam atau lebih karakter, maka jumlah perbandingan sekita 0,4T. Seiring dengan meningkatnya panjang *pattern* maka algoritma Boyer Moore ini memiliki perbandingan yang lebih rendah, sekitar 0,25T.

3. Analisis berbagai Kesalahan yang mungkin terjadi pada saat Pencocokan String (String Matching)

Algoritma yang kami analisis diatas belum mampu untuk menangani kesalahan yang terjadi berikut ini :

1. Kemungkinan ada karakter teks yang hilang.
2. Kemungkinan ada karakter yang hilang dari string (*pattern*)
3. Kemungkinan ada karakter didalam string atau teks yang mungkin di tambahkan, diganti, atau dihilangkan.

4. Kesimpulan

Permasalahan Pencarian String (String Macthing) pada dasarnya adalah permasalahan mencari sebuah string (*pattern*) pada sebuah teks.

Algoritma yang kami analisis hanya dapat menangani permasalahan string yang bersifat *exact string matching* sedangkan untuk permasalahan string yang bersifat *inexact string matching* diperlukan algoritma lain yang lebih advance.

5. Daftar Pustaka

[1] Munir Rinaldi, Diktat Kuliah Strategi Algoritmik, 193, 2005.

[2] www.cs.canisius.edu

[3] www.ics.uci.edu