Implementasi Pencocokan String Tidak Eksak dengan Algoritma Program Dinamis

Samudra Harapan Bekti 13508075
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
if18075@students.if.itb.ac.id

***Abstrak*—Makalah ini memfokuskan pada masalah pencocokan string yang memperbolehkan kesalahan, yang disebut juga pencocokan string tidak eksak/aproksimasi. Tujuan utama masalah ini adalah melakukan pencocokan pola pada teks dimana salah satu atau kedua dari teks tersebut mengalami korupsi. Salah satu contoh aplikasi masalah ini adalah mengembalikan sinyal dari suatu transmisi melalui kanal ber-*noise*, mencari bagian urutan DNA setelah mengalami mutasi, dan pencarian teks dimana terdapat kesalahan ketik atau ejaan.**

Ide utama dari masalah ini adalah menghitung jarak perbedaan antara kedua teks yang dibandingkan. Salah satu model yang sering dipelajari adalah *edit distance*, dimana setiap penghapusan, penyisipan, dan penggantian karakter pada teks memiliki ongkos tersendiri. Dibantu dengan algoritma program dinamis, jarak minimum antara kedua teks dapat dicari beserta langkah-langkah yang perlu ditempuh untuk mentransformasikan teks tersebut menjadi teks yang baru.

***Kata Kunci*—pencocokan string, program dinamis, *edit distance*, tak eksak, koreksi, ejaan.**

I. PENDAHULUAN

1.1 Pencocokan String Tak Eksak

Untuk keperluan ini kita menggunakan s, x, y, z, v, w, untuk merepresentasikan string dan a, b, c, ... untuk merepresentasikan huruf. Untuk sebarang string $s \in \Sigma^*$, kita menulis panjang dari string s sebagai $|s|$. Kita juga menulis s_i sebagai karakter ke-i dari s dengan i merupakan suatu integer $\in \{1..|s|\}$. Sebuah string kosong dinotasikan sebagai ϵ.

Pencocokan string tak eksak memperbolehkan pencocokan terhadap dua teks yang mengandung kesalahan. Secara formal hal ini didefinisikan sebagai berikut.

Σ merupakan suatu alfabet terbatas.
$T \in \Sigma^*$ merupakan teks dengan panjang $n = |T|$.
$P \in \Sigma^*$ merupakan pola dengan panjang $m = |P|$.
$k \in \mathbb{R}$ merupakan batas kesalahan maksimum yang diperbolehkan.
$d : \Sigma^* \times \Sigma^* \longrightarrow \mathbb{R}$ sebagai fungsi jarak.

Masalahnya adalah: diberikan T, P, k, dan d, hasilkan himpunan dari semua posisi teks j sedemikian sehingga terdapat i dimana $d(P, T_{i..j}) \leq k$.

Jarak $d(x, y)$ antara dua buah string x dan y adalah ongkos minimum dari runtutan operasi yang mentransformasikan x menjadi y (dan ∞ jika tidak terdapat urutan yang sesuai). Ongkos dari runtutan operasi adalah jumlah dari ongkos operasi secara individual. Suatu operasi merupakan himpunan operasi terbatas dengan bentuk $\delta(z, w) = t$, dimana z dan w merupakan string yang berbeda dan t merupakan angka real non-negatif. Setelah suatu operasi telah mengubah upa-string z menjadi w, maka tidak ada operasi lagi yang dapat dilakukan pada w.

Beberapa operasi yang banyak digunakan pada pencocokan string tak eksak adalah:

Penyisipan: $\delta(\varepsilon, a)$, contohnya, menyisipkan huruf a.
Penghapusan: $\delta(a, \varepsilon)$, contohnya, menghapus huruf a.
Penukaran atau substitusi: $\delta(a, b)$ untuk $a \neq b$, contohnya, mensubstitusi a dengan b.
Transposisi: $\delta(ab, ba)$ untuk $a \neq b$, contohnya, menukar huruf a dan b yang bersebelahan.

Berikut adalah beberapa metode yang sering digunakan untuk menghitung jarak antar dua string:

- *Levenshtein* atau *edit distance* memperbolehkan operasi penyisipan, penghapusan, dan substitusi. Untuk memudahkan persoalan maka semua operasi tersebut memiliki ongkos sebesar 1. Algoritma ini dapat dikatakan sebagai "jumlah minimum operasi penyisipan, penghapusan, dan substitusi untuk menyamakan dua buah string". Banyak literatur yang menyebutkan kasus ini sebagai "pencocokan string dengan k perbedaan". Jarak yang dihasilkan memenuhi persamaan $0 \leq d(x, y) \leq \max(|x|, |y|)$.
- *Hamming distance* hanya memperbolehkan operasi substitusi. Untuk menyederhanakan permasalahan maka operasi substitusi ini memiliki ongkos sebesar 1. Secara sederhana algoritma ini dapat dikatakan sebagai "pencocokan string dengan k kesalahan. Jarak yang dihasilkan memenuhi persamaan $0 \leq d(x, y) \leq |x|$.
- *Episode distance* hanya memperbolehkan operasi penyisipan, dimana operasi penyisipan tersebut memiliki ongkos 1. Disebut juga *episode*

matching karena algoritma ini memodelkan urutan kejadian dimana semua kejadian tersebut harus terjadi dalam selang waktu yang singkat. Jarak yang dihasilkan tidak simetris sehingga tidak memungkinkan untuk mengubah x menjadi y pada kasus ini. Oleh karena itu, $d(x, y)$ adalah $|y| - |x|$ atau ∞.

- Masalah *Longest Common Subsequence* (LCS) hanya memperbolehkan operasi penyisipan dan penghapusan, dan semuanya memiliki ongkos sebesar 1. LCS mengukur jarak dari pasangan karakter terjauh yang dapat dibuat antara dua buah string sehingga pasangan tersebut terurut sesuai dengan urutan huruf. Jarak yang dihasilkan merupakan jarak dari karakter yang tidak berpasangan, dan memenuhi $0 \leq d(x, y) \leq |x| + +|y|$.

Pada keperluan kali ini hanya akan digunakan *Levenshtein* atau *edit distance* karena algoritma ini paling sering digunakan dan dapat diimplementasikan dengan algoritma program dinamis.

1.2 Program Dinamis

Program Dinamis (*dynamic programming*) adalah metode pemecahan masalah dengan cara menguraikan solusi menjadi sekumpulan langkah (step) atau tahapan (stage) sedemikian sehingga solusi dari peresoalan dapat dipandang dari serangkaian keputusan yang saling berkaitan. Pada penyelesaian persoalan dengan metode ini:

1. Terdapat sejumlah berhingga pilihan yang mungkin,
2. Solusi pada setiap tahap dibangun dari hasil solusi tahap sebelumnya,
3. Kita menggunakan persyaratan optimasi dan kendala untuk membatasi sejumlah pilihan yang harus dipertimbangkan pada suatu tahap.

Cara penyelesaian dengan metode program dinamis mengingatkan kita pada metode greedy, karena metode greedy juga membentuk solusi secara bertahap. Pada metode greedy, kita membuat keputusan pada setiap tahap dengan cara mengambil pilihan yang paling menarik (yang memenuhi ukuran optimasi yang digunakan). Pengambilan keputusan pada setiap tahap didasarkan hanya pada informasi lokal, dan pada setiap tahap itu kita tidak pernah membuat keputusan yang salah. Dengan ini pengambilan keputusan pada setiap langkah greedy tidak pernah mempertimbangkan lebih jauh apakah pilihan tersebut pada langkah-langkah selanjutnya merupakan pilihan yang tepat.

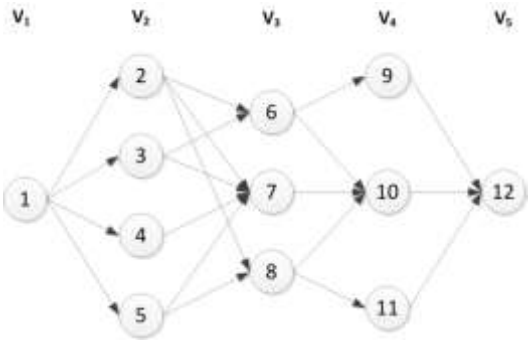
Pada program dinamis, serangkaian keputusan yang optimal dibuat dengan menggunakan Prinsip Optimalitas. Prinsip ini berbunyi: jika solusi total optimal, maka bagian solusi sampai tahap ke-k juga optimal. Prinsip optimalitas berarti bahwa jika kita bekerja dari tahap k ke tahap k + 1, kita dapat menggunakan hasil optimal dari tahap k tanpa harus kembali ke tahap awal. Jika pada seiap tahap kita menghitung ongkos (cost), maka dapat dirumuskan bahwa

Ongkos pada tahap k + 1 = (ongkos yang dihasilkan pada tahap k) + (ongkos dari tahap k ke tahap k + 1).

Dengan prinsip optimalitas ini dijamin bahwa pengambilan keputusan pada suatu tahap adalah keputusan yang benar untuk tahap-tahap selanjutnya. Jadi, perbedaan mendasar antara metode *greedy* dengan metode program dinamis adalah bahwa pada metode *greedy* hanya satu rangkaian keputusan yang pernah dihasilkan, sedangkan pada metode program dinamis lebih dari satu rangkaian keputusan. Hanya rangkaian keputusan yang memenuhi prinsip optimalitas.

Program dinamis diterapkan pada persoalan yang memiliki karakteristik sebagai berikut:

1. Persoalan dapat dibagi menjadi beberapa tahap (*stage*), yang pada setiap tahap hanya diambil satu keputusan.
2. Masing-masing tahap terdiri dari sejumlah status (*state*) yang berhubungan dengan tahap tersebut. Secara umum, status merupakan bermacam kemungkinan masukan yang ada pada tahap tersebut. Gambar berikut memperlihatkan perbedaan antara tahap dan status diberikan pada graf multitahap (*multistage graph*). Tiap simpul di dalam graf tersebut meyatakan status, sedangkana V1, V2, ... menyatakaan tahap.



Gambar 1.2.1: Graf yang menyatakan tahap dan status

3. Hasil dari keputusan yang diambil pada setiap tahap ditransformasikan dari status yang bersangkutan ke status berikutnya pada tahap berikutnya.
4. Ongkos (*cost*) pada suatu tahap meningkat secara teratur (*steadily*) dengan bertambahnya jumlah tahapan.
5. Ongkos pada suatu tahap bergantung pada ongkos tahap-tahap yang sudah berjalan dan ongkos pada tahap tersebut.
6. Keputusan terbaik pada suatu tahap bersifat independen terhadap keputusan yang dilakukan pada tahap sebelumnya.

7. Adanya hubungan rekursif yang mengidentifikasikan keputusan terbaik untuk setiap status pada tahap k memberikan keputusan terbaik untuk setiap status pada tahap k + 1.
8. Prinsip optimalitas berlaku pada persoalan tersebut.

Solusi yang dihasilkan oleh program dinamis dapat lebih dari satu buah.

II. METODE

2.1 Menghitung Edit Distance

Algoritma edit distance yang kali ini digunakan berbasis pada algoritma program dinamis. Untuk menghitung jarak antara kedua string, yakni $d(x, y)$. Sebuah matriks $C_{0..|x|,..|y|}$ akan diisi dimana $C_{i,j}$ merepresentasikan jumlah operasi minimum yang dibutuhkan untuk mencocokkan $x_{1..i}$ dengan $y_{1..j}$. Nilai elemen matriks ini dihitung dengan program dinamis sebagai berikut:

$$C_{i,0} = i$$
$$C_{0,j} = j$$
$$C_{i,j} = \text{if } (x_i = y_j) \text{ then } C_{i-1,j-1}$$
$$\text{else } 1 + \min(C_{i-1,j}, C_{i,j-1}, C_{i-1,j-1})$$

dimana pada akhir komputasi, $C_{|x|,|y|} = d(x, y)$.

Penjelasan algoritma tersebut diatas adalah sebagai berikut. Pertama-tama, $C_{i,0}$ dan $C_{0,j}$ merepresentasikan *edit distance* antara suatu string dengan panjang i atau j dan string kosong. Dengan jelas dibutuhkan penghapusan sebanyak i (dan juga j) pada string yang tidak kosong. Untuk dua string tidak kosong dengan panjang i dan j, kita mengasumsikan secara induktif bahwa semua *edit distance* antara string yang lebih pendek telah dihitung sebelumnya.

Perhatikan karakter terakhir x_i dan y_j. Jika keduanya sama maka kita tidak perlu memperhatikan keduanya dan kita langsung melanjutkan proses konversi $x_{1..i-1}$ menjadi $y_{1..j-1}$. Di sisi lain, jika keduanya tidak sama maka kita harus melakukan suatu operasi. Kita dapat melakukan salah satu diantara ketiga operasi yang diperbolehkan: menghapus x_i dan mengkonversi $x_{1..i-1}$ menjadi $y_{1..j-1}$ dengan cara terbaik, menyisipkan y_j pada akhir $x_{1..i}$ dan mengkonversi $x_{1..i-1}$ menjadi $y_{1..j-1}$ dengan cara terbaik, atau mensubstitusi x_i dengan y_j dan mengkonversi $x_{1..i-1}$ menjadi $y_{1..j-1}$ dengan cara terbaik. Pada semua kasus, ongkos yang diperlukan adalah 1 ditambah dengan ongkos untuk sisa proses keseluruhan (yang sudah dihitung sebelumnya). Perhatikan bahwa penyisipan pada suatu string sama saja dengan menghapus pada string lainnya.

Algoritma program dinamis harus dapat mengisi matriks sedemikian rupa sehingga tetangga atas, kiri, dan kiri atas suatu sel telah terisi terlebih dahulu sebelum menghitung isi sel yang tersebut. Hal ini dapat dicapai dengan cara mengisi sel secara traversal dari kiri ke kanan baris atau atas ke bawah kolom terlebih dahulu, atau menggunakan analisis rekurens dimana matriks diisi secara diagonal (dari kiri atas ke kanan bawah atau dari kanan atas ke kiri bawah).

		s	u	r	g	e	r	y
	0	1	2	3	4	5	6	7
s	1	**0**	1	2	3	4	5	6
u	2	1	**0**	1	2	3	4	5
r	3	2	1	**0**	1	2	3	4
v	4	3	2	1	**1**	2	3	4
e	5	4	3	2	2	**1**	**2**	3
y	6	5	4	3	3	2	2	**2**

Gambar 2.1.1: Matriks yang dihasilkan oleh program dinamis untuk menghitung jarak antara kata “survey” dan “surgery”. Angka yang dicetak tebal menujukkan jalur menuju hasil akhir.

Jika diperhatikan, nilai sel yang bertetangga tidak akan berbeda lebih dari satu satuan, dan nilai sel dari diagonal kiri atas ke kanan bawah tidak berkurang.

Algoritma ini memiliki kompleksitas $O(|x||y|)$ pada kasus terburuk dan rata-rata. Walaupun begitu, ruang yang dibutuhkan hanya $O(\min(|x|, |y|))$ karena jika kita mengisi matriks dari mulai kolom, maka kita hanya perlu menyimpan isi kolom sebelumnya untuk menghitung isi kolom yang baru, begitu seterusnya hingga semua kolom terisi. Oleh karena itu, kita dapat mengisi matriks dengan mengisi kolom atau baris terlebih dahulu sehingga ruang yang dibutuhkan minimum.

Di sisi lain, urutan operasi yang dilakukan untuk mentransformasikan x menjadi y dapat diperoleh dari matriks dengan cara menyusuri jalur (yang merepresentasikan urutan operasi) dari sel $C_{|x|,|y|}$ ke sel $C_{0,0}$ yang sesuai. Jalur yang dihasilkan dapat saja lebih dari satu. Pada kasus ini kita harus menghitung terlebih dahulu semua isi matriks atau minimal suatu daerah disekitar diagonal utama matriks.

2.2 Pencarian Teks

Algoritma diatas dapat dikembangkan lebih jauh sehingga dapat digunakan untuk mencari pola pendek P pada teks panjang T. Algoritma yang digunakan pada dasarnya sama, dengan $x = P$ dan $y = T$ (menggunakan aturan mengisi kolom terlebih dahulu sehingga hanya membutuhkan ruang sebesar $O(m)$. Satu-satunya perbedaan dari algoritma sebelumnya adalah kita harus memperbolehkan posisi dari teks apapun yang berpotensi untuk dijadikan titik awal pencocokan. Hal ini dapat

diperoleh dengan mengubah $C_{0,j} = 0$ untuk semua $j \epsilon 0..n$. Dalam kata lain, pola kosong cocok dengan kesalahan sebesar nol pada posisi apapun (karena pola ini cocok dengan teks upa-stirng dengan panjang nol).

Algoritma ini kemudian menginisialisasi kolom $C_{0,m}$ dengan nilai $C_i = i$, dan memproses teks karakter demi karakter. Setiap bertemu dengan karakter baru T_j, vektor kolom diperbaharui menjadi $C'_{0..m}$. Rumus untuk memperbaharui nilai vektor tersebut adalah

$$C'_i = \text{if } (P_i = T_j) \text{ then } C_{i-i}$$
$$\text{else } 1 + \min(C'_{i-1}, C_i, C_{i-1})$$

Waktu perncarian algoritma ini adalah $O(mn)$ dan penggunaan ruang algoritma ini adalah $O(m)$. Gambar berikut menunjukkan hasil pencocokan string "survey" dengan algoritma diatas

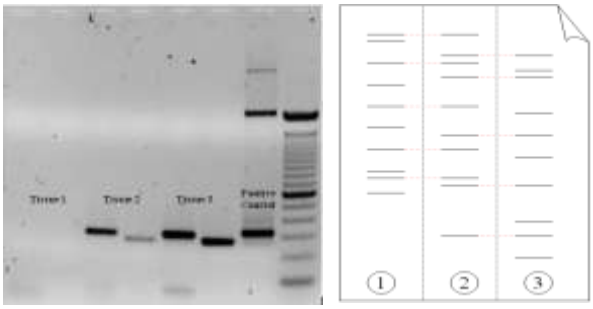
		s	u	r	g	e	r	y
	0	0	0	0	0	0	0	0
s	1	0	1	1	1	1	1	1
u	2	1	0	1	2	2	2	2
r	3	2	1	0	1	2	2	3
v	4	3	2	1	1	2	3	3
e	5	4	3	2	2	1	2	3
y	6	5	4	3	3	**2**	**2**	**2**

Gambar 2.1.2 Hasil algoritma program dinamis untuk mencari string "survey" pada kata "surgery" dengan dua kesalahan. Setiap kolom pada matriks diatas merupakan isi dari vektor C. Angka yang dicetak tebal menunjukkan posisi teks yang cocok.

III. CONTOH APLIKASI PERMASALAHAN

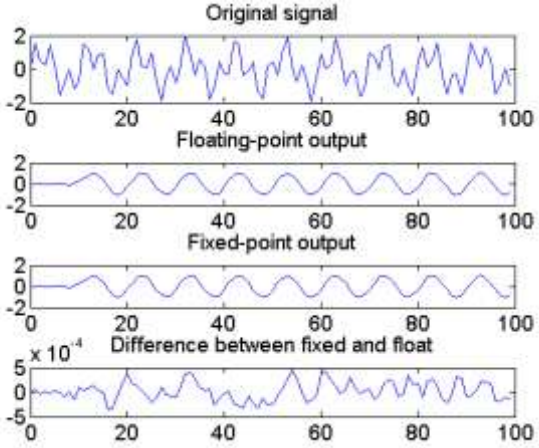
3.1 Biologi Komputasi

DNA dan urutan protein dapat dipandang sebagai teks panjang dengan alfabet yang spesifik (misalnya { A, C, G, T } pada DNA). Urutan tersebut merepresentasikan kode genetik dari makhluk hidup. Banyak riset yang melakukan pencocokan DNA untuk menentukan seberapa mirip DNA suatu makhluk hidup dengan makhluk hidup lainnya. Proses pencocokan ini tidak dapat dilakukan secara eksak karena untuk suatu jenis makhluk hidup, urutan DNA dapat saja berbeda akibat dari mutasi atau faktor genetik lainnya. Tetapi dengan menghitung jarak antara kedua DNA maka dapat diketahui seberapa mirip kedua makhluk hidup tersebut.



(a) (b)

Gambar 3.1.1: (a) Contoh DNA seorang manusia setelah proses *Ethidium Bromide-stained Polymerase Chain Reaction.* (b) Pencocokan pola antara tiga buah DNA yang dihasilkan dari reaksi sebelumnya.



Gambar 3.2.1: Sampel pada sinyal asli yang diterima dan mengandung kesalahan dapat diperbaiki dengan mengambil perbedaan antara komponen penyusunnya dan mencocokannya dengan sinyal benar yang terdekat.

3.3 Pengolahan Teks

Masalah memperbaiki pengejaan kata yang salah merupakan salah satu masalah tertua dalam dunia pencocokan string. Seiring dengan pesatnya perkembangan informasi maka saat ini akan sulit untuk mencari bagian teks secara eksak pada internet karena dapat saja teks yang digunakan sebagai pola pencarian mengandung kesalahan. Pada kasus seperti ini pencocokan string tidak eksak akan diperlukan sehingga kesalahan dalam pengejaan pola pencarian dapat dikoreksi atau diberikan alternatif pengejaan.

Gambar 3.3.1: Contoh layar pengecekan ejaan pada aplikasi Microsoft Word. Beberapa alternatif pengejaan akan ditampilkan ketika aplikasi Microsoft Word tidak mengenali kata yang ditulis.

II. KESIMPULAN

Pencocokan string tak eksak merupakan salah satu permasalahan yang masih sering diteliti hingga saat ini. Pada dasarnya algoritma pencocokan string tak eksak akan menghitung jumlah operasi yang perlu dilakukan pada suatu string agar string tersebut cocok dengan string yang dituju. Berbagai algoritma telah dikembangkan seperti *Levenshtein* atau yang lebih dikenal sebagai *edit distance*, *Hamming distance*, *Episode distance*, atau bahkan algoritma *Longest Common Subsequence* (LCS). Program dinamis dapat digunakan untuk menyelesaikan permasalahan ini dengan menggunakan prinsip optimalitas, dimana setiap tahap dalam algoritma ini akan memberikan hasil yang optimal. Algoritma edit distance yang diimplementasikan dengan program dinamis akan memiliki kompleksitas $O(mn)$ dan penggunaan ruang sebesar $O(m)$.

REFERENSI

Navarro, G. 2000. "A Guided Tour to Approximate String Matching" University of Chile.

Navarro, G. 1998. "A practical index for text retrieval allowing errors", CLEI Electron, Springer-verlag, Berlin.

Sellers, P. 1980. "The theory and computation of evolutionary distances: pattern recognition", ACM 33, 8, 132-142.

Lowrance, R. 1975. "An extension of the string-to-string correction problem" J. ACM 22, 178-183

Munir, R. 2009. "Diktat Kuliah IF 3051 Strategi Algoritma", Program Studi Teknik Informatika, Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung.

http://bix.ucsd.edu/algorithms/presentations/Ch06_EditDist.pdf, waktu akses 24 November 2010, 22:48 PM.

http://www.merriampark.com/ld.htm, waktu akses 26 November 2010, 20:23 PM.

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, 29 April 2010

Samudra Harapan Bekti – 13508075