Desain, Analisis dan Implementasi Algoritma Komputasi String dengan Metode Dynamic Programming dan Meet In the Middle pada Permasalahan Klasik SPOJ 9967 Playing With Words

Dewangga Winasforcepta Winardi, Rully Soelaiman dan F. X. Arunanto
Departemen Teknik Informatika, Fakultas Teknologi Informasi, Institut Teknologi Sepuluh Nopember (ITS)
Jl. Arief Rahman Hakim, Surabaya 60111 Indonesia
e-mail: dewangga13@mhs.if.its.ac.id, rully@is.its.ac.id, anto@if.its.ac.

***Abstrak*—Diberikan dua buah *string* $orig1$ dan $orig2$. Diberikan tiga tahapan proses transformasi untuk menghasilkan *string* $ad1$ dan $ad2$. Tahap pertama adalah *string* $orig1$ diacak urutan karakter-karakternya. Tahap kedua adalah *string* $orig2$ diacak urutan karakter-karakternya. Tahap terakhir adalah salah satu karakter dari *string* $orig1$ atau $orig2$ diganti dengan karakter sebelum atau sesudahnya dalam alfabet. Jarak dua buah *string* didefinisikan sebagai jumlah dari selisih mutlak dari karakter-karakter pada posisi yang sama. Diberikan sebuah bilangan bulat X yang merupakan jarak dari *string* $orig1$ dan *string* $ad1$ dijumlahkan dengan jarak dari *string* $orig2$ dan *string* $ad2$. Tentukan jumlah kemungkinan kombinasi *string* $orig1$ dan $orig2$ jika diberikan *string* $ad1$, $ad2$ dan nilai X.**

Pada penelitian ini akan dirancang penyelesaian masalah yang disampaikan pada paragraf pertama dengan menggunakan pendekatan *dynamic programming* dan teknik *meet in the middle*. Solusi yang dikembangkan berjalan dengan kompleksitas waktu $\mathcal{O}(2^{|S|} * MAX\_DIST^2 * T)$, di mana $|S|$ adalah panjang *string* yang diberikan dan MAX_DIST adalah jarak antar *string* maksimal.

Algoritma yang dirancang dapat menyelesaikan permasalahan yang diberikan dengan benar. Pada kasus terburuk, waktu eksekusi program yang mengimplementasi algoritma yang dirancang tidak melebihi batas waktu eksekusi program yang telah diberikan, yaitu $6,459$ detik. Sehingga dapat disimpulkan algoritma yang dibangun dapat menyelesaikan permasalahan yang diberikan.

***Kata Kunci*—string, divide and conquer, meet in the middle, dynamic programming**

I. PENDAHULUAN

Diberikan dua buah *string* $orig1$ dan $orig2$. Diberikan tiga tahapan proses transformasi untuk menghasilkan *string* $ad1$ dan $ad2$ sebagai berikut:

1) String $orig1$ diacak urutan karakter-karakternya.
2) String $orig2$ diacak urutan karakter-karakternya.
3) Salah satu karakter dari *string* $orig1$ atau $orig2$ diganti dengan karakter sebelum atau sesudahnya dalam alfabet.

Jarak dua buah *string* didefinisikan sebagai jumlah dari selisih mutlak dari karakter-karakter pada posisi yang sama. Diberikan sebuah bilangan bulat X yang merupakan jarak dari *string* $orig1$ dan *string* $ad1$ dijumlahkan dengan jarak dari *string* $orig2$ dan *string* $ad2$. Berapakah jumlah kemungkinan kombinasi *string* $orig1$ dan $orig2$ jika diberikan *string* $ad1$, $ad2$ dan nilai X.

Solusi naif dari permasalahan di atas adalah dengan mengkomputasi semua kemungkinan *string* $orig1$ dan $orig2$ lalu menghitung berapa banyak kombinasi *string* $orig1$ dan $orig2$ yang memiliki $dist(orig1, ad1) + dist(orig2, ad2) = X$. Namun solusi

Tabel 1 Kombinasi *string* $orig1$ dan *string* $orig2$ dari *string* $ad1 = bc$ dan *string* $ad2 = efg$ tanpa operasi *replace* dengan $X = 4$

$orig1$	$dist(orig1, ad1)$	$orig2$	$dist(orig2, ad2)$
bc	0	feg	4
bc	0	gef	4
bc	0	gfe	4
cb	2	egf	2
cb	2	fge	2

dapat dimanfaatkan untuk menyelesaikan setiap submasalah sebelum solusi setiap submasalah digabungkan untuk membentuk jawaban akhir. Tujuan dari penelitian ini adalah untuk menyelesaikan permasalahan, menguji kebenaran dan menguji performa dari algoritma yang dibangun.

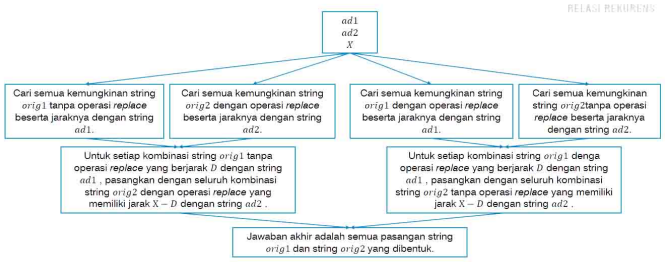
II. METODE PENYELESAIAN

Untuk merancang algoritma yang optimal untuk menyelesaikan permasalahan yang diberikan, dapat dilakukan penyelesaian permasalahan yang lebih sederhana. Awalnya, diasumsikan tidak ada operasi *replace* untuk mentransformasi pesan asli menjadi kalimat iklan. Gambar 1 adalah algoritma untuk menghitung jumlah kemungkinan pesan asli dari pesan iklan yang diberikan. Contohnya ketika kalimat pesan adalah $ad1 = bc$, *string* $ad2 = efg$ dan $X = 4$. Setelah mencari seluruh kombinasi *string* $orig1$ dari *string* $ad1$ dan *string* $orig2$ dari *string* $ad2$ beserta jarak masing-masing, setiap kombinasi *string* $orig1$ dengan jarak D dipasangkan dengan setiap *string* $orig2$ dengan jarak $X - D$ dengan $0 \leq D \leq X$. Tabel 1 menunjukkan bahwa terdapat 5 kombinasi pesan asli dari pesan iklan dengan *string* $ad1 = bc$, *string* $ad2 = efg$ dan $X = 4$.

Tabel 2 Kombinasi *string* $orig1$ dan *string* $orig2$ dari *string* $ad1 = c$ dan *string* $ad2 = n$ dengan sekali operasi *replace* dengan $X = 1$

$orig1$	$dist(orig1, ad1)$	$orig2$	$dist(orig2, ad2)$
c	0	m	1
c	0	o	1
b	1	n	0
d	1	n	0

replace sesuai pada deskripsi permasalahan. Contohnya ketika kalimat pesan adalah $ad1 = c$, *string* $ad2 = n$ dan $X = 1$. Hasil didapatkan dengan dua perhitungan yaitu perhitungan jumlah kombinasi *string* $orig1$ tanpa operasi *replace* dengan jarak D terhadap *string* $ad1$ yang dipasangkan dengan setiap kombinasi *string* $orig2$ dengan sekali operasi *replace* dengan jarak $X - D$ terhadap *string* $ad2$. Berikutnya jawaban ditambahkan dengan hasil perhitungan kedua yaitu perhitungan jumlah kombinasi *string* $orig1$ dengan sekali operasi *replace* dengan jarak D terhadap *string* $ad1$ yang dipasangkan dengan setiap kombinasi *string* $orig2$ tanpa operasi *replace* dengan jarak $X - D$ terhadap *string* $ad2$. Tabel 2 menunjukkan bahwa terdapat 4 kombinasi pesan asli dari pesan iklan dengan *string* $ad1 = c$, *string* $ad2 = n$ dan $X = 1$.



Gambar 2 Algoritma komputasi jumlah kombinasi *string* $orig$ tanpa operasi *replace*

Pada permasalahan tidak diperlukan informasi kombinasi pesan asli yang mungkin, melainkan hanya memerlukan informasi jumlah kemungkinan kombinasi pesan asli yang mungkin. Sehingga algoritma komputasi dapat dioptimasi menjadi lebih efisien. Gambar 3 adalah algoritma yang optimal untuk menghitung jumlah kemungkinan pesan asli dari pesan iklan. Contohnya pada kasus ketika kalimat pesan adalah $ad1 = c$, *string* $ad2 = n$ dan $X = 1$. Algoritma pada Gambar 3 mirip dengan algoritma pada Gambar 2. Hanya saja pada algoritma ini tidak dicari semua kemungkinan *string* $orig$, hanya dihitung kemungkinannya saja. Contohnya pada perhitungan kemungkinan *string* $orig1$ dari *string* $ad1 = c$. Algoritma hanya menghitung jumlah kombinasi *string* $orig1$ tanpa operasi *replace* dengan jarak 0 dan 1 terhadap *string* $ad1$ seperti yang terlihat pada Tabel 3. Hasil akhir perhitungan adalah terdapat 4 kombinasi pesan asli seperti yang terlihat pada Tabel 4.

Tabel 3 Hasil perhitungan jumlah kombinasi *string* $orig1$ tanpa operasi *replace* dengan jarak 0 dan 1 terhadap *string* $ad1$

D	Jumlah kombinasi *string* $orig1$
0	1
1	0

Tabel 4 Kombinasi *string* $orig1$ dan *string* $orig2$ dari *string* $ad1 = c$ dan *string* $ad2 = n$ dengan sekali operasi *replace* dengan $X = 1$

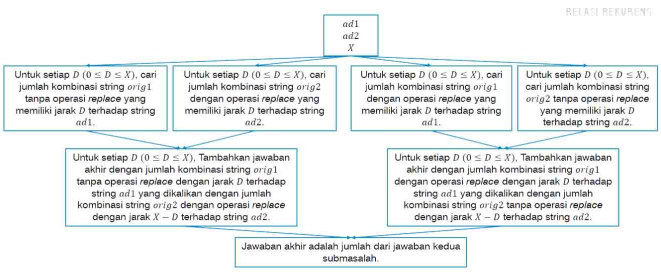
D	Jumlah kombinasi *string* $orig1$	$X - D$	Jumlah kombinasi *string* $orig2$	total
0	1	1	2	2
1	0	0	0	0
0	0	1	1	0
1	2	0	1	2

$$answer = \sum_{dist=0}^{dist=min_{(X,250)}} ((F_{(S_0,2^{|S_0|},bound-dist)} * G_{(S_1,2^{|S_1|},bound-X+dist)}) + (G_{(S_0,2^{|S_0|},bound-dist)} * F_{(S_1,2^{|S_1|},bound-X+dist)})) \quad (1)$$

$$F_{(S,mask,dist)} = \begin{cases} 0, & \text{if } dist > bound \vee (mask = 0 \wedge dist \neq bound) \\ 1, & \text{if } (mask = 0) \wedge (dist = bound) \\ \sum_{i=0}^{i=NSB_{(mask)}} F1_{(S,mask,set\_bit(mask)_i,dist)} & \text{otherwise} \end{cases} \quad (2)$$

$$G_{(S,mask,dist)} = \begin{cases} 0, & \text{if } (dist > bound) \vee mask = bound \\ \sum_{i=0}^{i=NSB_{(mask)}} G1_{(S,mask,set\_bit(mask)_i,dist)} + G2_{(S,mask,set\_bit(mask)_i,dist)} + G3_{(S,mask,set\_bit(mask)_i,dist)} & \text{otherwise} \end{cases} \quad (3)$$

$$F1_{(S,mask,idx,dist)} = \begin{cases} F_{(S,mask-2^{idx},dist-|S_{idx}-S_{curIdx}|)} & \text{if } idx = |S| - 1 \vee duplicate\_rule1_{(S,mask,idx)} = True \\ 0, & \text{otherwise} \end{cases}$$



Gambar 3 Algoritma komputasi jumlah kombinasi *string* $orig$ tanpa operasi *replace*

$$G1_{(S,mask,idx,dist)} = \begin{cases} G_{(S,mask-2^{idx},dist+|S_{idx}-S_{curIdx}|)} & \text{if } idx = |S|-1 \vee duplicate\_rule1_{(S,mask,idx)} = True \\ 0, & \text{otherwise} \end{cases}$$

$$G2_{(S,mask,idx,dist)} = \begin{cases} F_{(S,mask-2^{idx},dist+|S_{idx}+1-S_{curIdx}|)} & \text{if } idx = |S|-1 \vee (duplicate\_rule1_{(S,mask,idx)} = True \wedge duplicate\_rule2_{(S,mask,idx)} = True) \\ 0, & \text{otherwise} \end{cases}$$

$$G3_{(S,mask,idx,dist)} = \begin{cases} F_{(S,mask-2^{idx},dist+|S_{idx}-1-S_{curIdx}|)} & \text{if } (idx = |S|-1 \vee duplicate\_rule1_{(S,mask,idx)} = True) \wedge (idx = 0 \vee duplicate\_rule3_{(S,mask,idx)} = True) \\ 0, & \text{otherwise} \end{cases}$$

$$duplicate\_rule1_{(S,mask,dist)} = \begin{cases} True & \text{if } idx < |S|-1 \wedge (S_{idx} \neq S_{idx+1} \vee ((S_{idx} = S_{idx+1}) \wedge (is\_on_{(mask,idx+1)} = False)) \\ False, & \text{otherwise} \end{cases}$$

$$duplicate\_rule2_{(S,mask,dist)} = \begin{cases} True & \text{if } idx < |S|-1 \wedge (charFirstPos_{(S,S_{idx}+1)} = -1 \vee (charFirstPos_{(S,S_{idx}+1)} \neq -1 \wedge is\_on_{(mask\ charFirstPos_{(S,S_{idx}+1)})} = False \\ False, & \text{otherwise} \end{cases}$$

Tabel 5 Daftar notasi persamaan relasi rekurens

Notasi	Deskripsi
S	String yang akan dicari kemungkinan *string* awalnya.
$mask$	Sebuah bilangan bulat yang bertugas sebagai *bitmask* yang merepresentasikan kondisi karakter mana saja yang sudah diambil pada kondisi (*state*) tersebut.
$dist$	Jarak *string* yang sudah terbentuk pada kondisi tersebut dengan *string* S dari $bound$ atau secara matematis dapat dituliskan dengan $bound - distance(currentString, S)$.
$bound$	Nilai batas jarak maksimal yang bernilai $min(X, 250)$
idx	Index karakter pada *string* S yang akan diambil atau digunakan.
$NSB_{(mask)}$	Mengembalikan jumlah angka 1 pada $mask$ apabila direpresentasikan dalam basis biner
$set\_bit_{(mask)}$	Himpunan index bilangan bernilai satu dari $mask$ apabila direpresentasikan dalam basis biner.
$is\_on_{(mask,idx)}$	Mengembalikan nilai $true$ apabila bilangan pada index idx pada $mask$ dalam basis biner bernilai 1.
$charLastPos_{(S,C)}$	Index terbesar dari karakter C pada *string* S.
$charFirstPos_{(S,C)}$	Index terkecil dari karakter C pada *string* S.
$curIdx$	Angka yang merepresentasikan panjang *string* $orig$ pada *state* tersebut. Nilai $curIdx$ adalah jumlah bit yang bernilai 0 pada $mask$.

$$duplicate\_rule3_{(S,mask,dist)} = \begin{cases} True & \text{if } idx > 0-1 \wedge (charLastPos_{(S,S_{idx}-1)} = -1 \vee (charLastPos_{(S,S_{idx}-1)} \neq -1 \wedge is\_on_{(mask,charLastPos_{(S,S_{idx}-1)})} = True \\ False, & \text{otherwise} \end{cases}$$

$$is\_on_{(mask,idx)} = \begin{cases} True & \text{if } (mask \& 2^{idx}) = 1 \\ False, & \text{otherwise} \end{cases}$$

Waktu Maksimal	1.02 detik
Waktu Minimal	0.98 detik
Waktu Rata-Rata	1.004 detik
Memori Maksimal	19 MB
Memori Minimal	19 MB
Memori Rata-Rata	19 MB

Persamaan 1 adalah persamaan untuk mendapatkan jawaban akhir dengan fungsi $F_{(S,mask,idx)}$ pada Persamaan 2 adalah fungsi untuk menghitung kemungkinan kombinasi *string orig* dari *string* S tanpa operasi *replace* dan fungsi $G_{(S,mask,idx)}$ adalah fungsi untuk menghitung kemungkinan kombinasi *string orig* dari *string* S dengan sekali operasi *replace*. Tabel 5 adalah daftar notasi dari persamaan-persamaan tersebut.

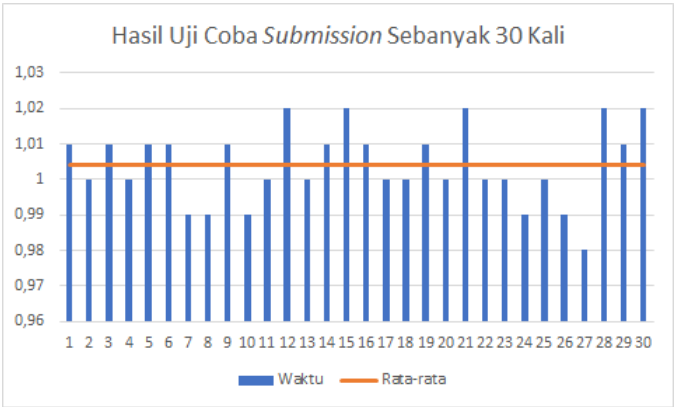
III. UJI COBA DAN ANALISIS

A. Uji coba kebenaran

Uji coba kebenaran dilakukan dengan mengumpulkan berkas kode sumber hasil implementasi ke situs sistem penilaian daring SPOJ kali. Permasalahan yang diselesaikan adalah Playing With Words dengan kode PWORDS. Hasil uji kebenaran dan waktu eksekusi program saat pengumpulan kasus uji pada situs SPOJ ditunjukkan pada Gambar 4.

Gambar 4 Hasil uji kebenaran dengan melakukan *submission* ke situs penilaian daring SPOJ

Berikutnya adalah pengujian performa dari algoritma yang dirancang dan diimplementasi dengan melakukan uji *submission* dengan mengumpulkan berkas kode implementasi dari algoritma yang dibangun sebanyak 30 kali ke situs penilaian daring SPOJ dengan mencatat waktu eksekusi serta memori yang dibutuhkan. Hasil dari pengujian dapat dilihat pada grafik pada Gambar 5 dan Tabel 6.



Gambar 5 Hasil Uji Coba *Submission* ke situs penilaian daring SPOJ sebanyak 30 kali

B. Analisis Kompleksitas

Berdasarkan Persamaan 1 didapatkan algoritma dengan kompleksitas waktu $\mathcal{O}(2^{|S|} * MAX\_DIST^2 * T)$ di mana $|S|$ adalah panjang *string* masukan, MAX_DIST^2 adalah jarak maksimal antar *string* dan T adalah banyaknya kasus uji. Pada umumnya, eksekusi program pada situs penilaian daring SPOJ adalah 1 detik untuk setiap 100.000.000 proses. Pada kasus terburuk, yaitu ketika $|S| = 10$, $MAX\_DIST = 250$ dan $T = 10$, eksekusi program dengan kompleksitas waktu $\mathcal{O}(2^{|S|} * MAX\_DIST^2 * T)$ akan melakukan 640.000.000 proses di mana jika dengan menggunakan standar berupa 100.000.000 proses per detik, program akan membutuhkan waktu eksekusi sebesar $6,4$ detik. Sehingga Algoritma dengan kompleksitas waktu $\mathcal{O}(2^{|S|} * MAX\_DIST^2 * T)$ dapat menyelesaikan permasalahan yang diberikan.

IV. KESIMPULAN

Dari hasil uji coba yang telah dilakukan terhadap perancangan dan implementasi algoritma untuk menyelesaikan permasalahan klasik SPOJ 9967 Playing With Words dapat diambil kesimpulan sebagai berikut:

1) Implementasi algoritma dengan menggunakan pendekatan *dynamic programming* dan teknik *meet in the middle* dapat menyelesaikan permasalahan permasalahan klasik SPOJ 9967 Playing With Words dengan benar.
2) Kompleksitas waktu sebesar $\mathcal{O}(2^{|S|} * MAX\_DIST^2 * T)$ dapat menyelesaikan permasalahan klasik SPOJ 9967 Playing With Words.
3) Waktu yang dibutuhkan program untuk menyelesaikan permasalahan klasik SPOJ 9967 Playing With Words minimum 0.98 detik, maksimum 1.02 detik dan rata-rata 1.004 detik. Memori yang dibutuhkan adalah sebesar 19 MB.

UCAPAN TERIMA KASIH

Penulis mengucapkan puji syukur kehadirat Allah SWT atas segala rahmat dan karunia-Nya sehingga memungkinkan penulis untuk dapat menyelesaikan penelitian ini. Penulis juga mengucapkan terima kasih kepada orang tua dan keluarga penulis, juga kepada Bapak Rully Soelaiman dan Bapak F.X. Arunanto selaku dosen pembimbing penulis dan kepada semua pihak yang telah memberikan dukungan baik secara langsung maupun tidak langsung selama penulis mengerjakan penelitian ini.

DAFTAR PUSTAKA

[1] **Introduction To Java - MFC 158 G.** [Online]. Available: http://www.acsu.buffalo.edu/ fineberg/mfc158/week10lecture.htm. [Accessed 24-May-2017].
[2] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, **Introduction To Algorithm,** 2nd ed. Cambridge, Massachusetts London, England: The MIT Press, 2001.
[3] S. Halim and F. Halim, **Competitive Programming 3**.Singapore, 2013.
[4] E. Elmaghiraby, **Journal of Mathematical Analysis and Applications,** vol. 29, no. 3, pp. 523557, Mar. 1970.