



TUGAS AKHIR - KI141502

**DESAIN DAN ANALISIS ALGORITMA KOMPUTASI STRING  
DENGAN METODE MEET IN THE MIDDLE DAN DYNAMIC  
PROGRAMMING PADA PERMASALAHAN KLASIK SPOJ 9967  
PLAYING WITH WORDS**

**DEWANGGA WINASFORCEPTA WINARDI  
NRP 5113 100 098**

**Dosen Pembimbing 1  
Rully Soelaiman, S.Kom., M.Kom.**

**Dosen Pembimbing 2  
Ir. F.X. Arunanto, M.Sc.**

**JURUSAN TEKNIK INFORMATIKA  
Fakultas Teknologi Informasi  
Institut Teknologi Sepuluh Nopember  
Surabaya, 2017**

*Halaman ini sengaja dikosongkan*



TUGAS AKHIR - KI141502

**DESAIN DAN ANALISIS ALGORITMA KOMPUTASI STRING  
DENGAN METODE MEET IN THE MIDDLE DAN DYNAMIC  
PROGRAMMING PADA PERMASALAHAN KLASIK SPOJ 9967  
PLAYING WITH WORDS**

DEWANGGA WINASFORCEPTA WINARDI  
NRP 5113 100 098

Dosen Pembimbing 1  
Rully Soelaiman, S.Kom., M.Kom.

Dosen Pembimbing 2  
Ir. F.X. Arunanto, M.Sc.

JURUSAN TEKNIK INFORMATIKA  
Fakultas Teknologi Informasi  
Institut Teknologi Sepuluh Nopember  
Surabaya, 2017

*Halaman ini sengaja dikosongkan*



UNDERGRADUATE THESES - KI141502

**ALGORITHM DESIGN AND ANALYSIS FOR STRING COMPUTATION USING MEET IN THE MIDDLE TECHNIQUE AND DYNAMIC PROGRAMMING IN SPOJ CLASSIC PROBLEM 9967 PLAYING WITH WORDS**

DEWANGGA WINASFORCEPTA WINARDI  
NRP 5113 100 098

Supervisor 1  
Rully Soelaiman, S.Kom., M.Kom.

Supervisor 2  
Ir. F.X. Arunanto, M.Sc.

**INFORMATICS DEPARTMENT**  
Faculty of Information Technology  
Institut Teknologi Sepuluh Nopember  
Surabaya, 2017

*Halaman ini sengaja dikosongkan*

**DESAIN DAN ANALISIS ALGORITMA KOMPUTASI  
STRING DENGAN METODE MEET IN THE MIDDLE DAN  
DYNAMIC PROGRAMMING PADA PERMASALAHAN  
KLASIK SPOJ 9967 PLAYING WITH WORDS**

**TUGAS AKHIR**

Diajukan Guna Memenuhi Salah Satu Syarat  
Memperoleh Gelar Sarjana Komputer  
pada  
Bidang Studi Algoritma Pemrograman  
Program Studi S-1 Jurusan Teknik Informatika  
Fakultas Teknologi Informasi  
Institut Teknologi Sepuluh Nopember

Oleh:

**Dewangga Winasforcepta Winardi**  
NRP: 5113 100 098

Disetujui oleh Dosen Pembimbing Tugas Akhir:

Rully Soelaiman, S.Kom., M.Kom.  
NIP: 197002131994021001

.....  
(Pembimbing 1)

Ir. F.X. Arunanto, M.Sc.  
NIP: 195701011983031004

.....  
(Pembimbing 2)

**SURABAYA  
JUNI 2017**

*Halaman ini sengaja dikosongkan*

# **DESAIN DAN ANALISIS ALGORITMA KOMPUTASI STRING DENGAN METODE MEET IN THE MIDDLE DAN DYNAMIC PROGRAMMING PADA PERMASALAHAN KLASIK SPOJ 9967 PLAYING WITH WORDS**

Nama : DEWANGGA WINASFORCEPTA WINARDI  
NRP : 5113 100 098  
Jurusan : Teknik Informatika FTIF - ITS  
Pembimbing I : Rully Soelaiman, S.Kom., M.Kom.  
Pembimbing II : Ir. F.X. Arunanto, M.Sc.

## **Abstrak**

*Diberikan dua buah string orig1 dan orig2. Diberikan tiga tahapan proses enkripsi 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.*

*Meet in the middle adalah sebuah teknik pencarian dengan paradigma divide and conquer yang membagi permasalahan menjadi dua, lalu menyelesaiannya masing-masing, lalu menggabungkannya kembali.*

*Dynamic programming adalah sebuah paradigma untuk mendapatkan nilai optimal dari beberapa kemungkinan jawaban, dimana permasalahan tersebut memiliki submasalah tumpang tindih dan struktur optimal.*

*Pada tugas akhir ini akan dirancang penyelesaian masalah yang disampaikan pada paragraf pertama dengan menggunakan teknik meet in the middle dan pendekatan dynamic programming.*

*Solusi yang dikembangkan berjalan dengan kompleksitas waktu  $\mathcal{O}(2^{|S|} * \text{MAX\_DIST}^2 * T)$ , dimana  $|S|$  adalah panjang string yang diberikan dan MAX\_DIST adalah jarak antar string maksimal.*

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

# **ALGORITHM DESIGN AND ANALYSIS FOR STRING COMPUTATION USING MEET IN THE MIDDLE TECHNIQUE AND DYNAMIC PROGRAMMING IN SPOJ CLASSIC PROBLEM 9967 PLAYING WITH WORDS**

Name : DEWANGGA WINASFORCEPTA WINARDI  
NRP : 5113 100 098  
Major : Informatics Department Faculty of IT - ITS  
Supervisor I : Rully Soelaiman, S.Kom., M.Kom.  
Supervisor II : Ir. F.X. Arunanto, M.Sc.

## **Abstract**

*Given two strings orig1 and orig2. Given three steps to encrypt orig1 and orig2 to become ad1 and ad2. First step is shuffle the letters position of string orig1. The Second step is shuffle the letters position of string orig2. The last step is replace one letter from string orig1 or orig2 with its next or previous letter in the alphabet. Distance between two strings is defined as the sum of absolute difference between letter in same position. Given an integer X which is distance(orig1, ad1) + distance(orig2, ad2). Find the number of possible string orig1 and orig2 from given string ad1, ad2 and integer X.*

*Meet in the middle is a searching technique with divide and conquer paradigm that divide problem into two or more subproblems, and then solve each, and combine all the subproblem's answers to get main answer.*

*Dynamic programming is a paradigm to get optimal solution from some possible answer, which each of subproblem has overlapping subproblems and optimal structure.*

*In this final project the writer will design and analyse an algorithm to solve the problem that stated in the first paragraph using meet in the middle and dynamic programming.*

*The solution will run in  $\mathcal{O}(2^{|S|} * MAX\_DIST^2 * T)$  time complexity where  $|S|$  is the given string ad1 and ad2 and  $MAX\_DIST$  is the maximum distance between two strings.*

**Keywords:** string, divide and conquer, meet in the middle, dynamic programming

## **KATA PENGANTAR**

Puji syukur penulis panjatkan kepada Allah SWT. atas pimpinan, penyertaan, dan karunia-Nya sehingga penulis dapat menyelesaikan Tugas Akhir yang berjudul :

### **DESAIN DAN ANALISIS ALGORITMA KOMPUTASI STRING DENGAN METODE MEET IN THE MIDDLE DAN DYNAMIC PROGRAMMING PADA PERMASALAHAN KLASIK SPOJ 9967 PLAYING WITH WORDS.**

Penelitian Tugas Akhir ini dilakukan untuk memenuhi salah satu syarat meraih gelar Sarjana di Jurusan Teknik Informatika Fakultas Teknologi Informasi Institut Teknologi Sepuluh Nopember.

Dengan selesainya Tugas Akhir ini diharapkan apa yang telah dikerjakan penulis dapat memberikan manfaat bagi perkembangan ilmu pengetahuan terutama di bidang teknologi informasi serta bagi diri penulis sendiri selaku peneliti.

Penulis mengucapkan terima kasih kepada semua pihak yang telah memberikan dukungan baik secara langsung maupun tidak langsung selama penulis mengerjakan Tugas Akhir maupun selama menempuh masa studi antara lain:

- Almh. Mama yang selalu memberi perhatian, kasih sayang serta dukungan atas apapun keputusan dan kegiatan yang dilakukan oleh penulis dan menjadi semangat utama bagi diri penulis baik selama penulis menempuh masa kuliah maupun penggerjaan Tugas Akhir ini.
- Bapak, Ibu dan keluarga penulis yang selalu memberikan perhatian, dorongan dan kasih sayang.
- Bapak Rully Soelaiman, S.Kom., M.Kom. selaku Dosen

Pembimbing yang telah banyak meluangkan waktu untuk memberikan ilmu, nasihat, motivasi, pandangan dan bimbingan kepada penulis baik selama penulis menempuh masa kuliah maupun selama pengerjaan Tugas Akhir ini.

- Bapak Ir. F.X. ARUNANTO, M.Sc. selaku dosen pembimbing yang telah memberikan ilmu, dan masukan kepada penulis.
- Teman-teman, Kakak-kakak dan Adik-adik administrator Laboratorium Pemrograman yang selalu menjadi teman untuk berbagi ilmu.
- Seluruh tenaga pengajar dan karyawan Jurusan Teknik Informatika ITS yang telah memberikan ilmu dan waktunya demi berlangsungnya kegiatan belajar mengajar di Jurusan Teknik Informatika ITS.
- Seluruh teman penulis di Jurusan Teknik Informatika ITS yang telah memberikan dukungan dan semangat kepada penulis selama penulis menyelesaikan Tugas Akhir ini.

Penulis mohon maaf apabila masih ada kekurangan pada Tugas Akhir ini. Penulis juga mengharapkan kritik dan saran yang membangun untuk pembelajaran dan perbaikan di kemudian hari. Semoga melalui Tugas Akhir ini penulis dapat memberikan kontribusi dan manfaat yang sebaik-baiknya.

Surabaya, Juni 2017

Dewangga Winasforcepta Winardi

## DAFTAR ISI

<b>SAMPUL</b> . . . . .	<b>i</b>
<b>LEMBAR PENGESAHAN</b> . . . . .	<b>vii</b>
<b>ABSTRAK</b> . . . . .	<b>ix</b>
<b>ABSTRACT</b> . . . . .	<b>xi</b>
<b>KATA PENGANTAR</b> . . . . .	<b>xiii</b>
<b>DAFTAR ISI</b> . . . . .	<b>xv</b>
<b>DAFTAR TABEL</b> . . . . .	<b>xix</b>
<b>DAFTAR GAMBAR</b> . . . . .	<b>xxxiii</b>
<b>DAFTAR KODE SUMBER</b> . . . . .	<b>xxxv</b>
<b>BAB I PENDAHULUAN</b> . . . . .	<b>1</b>
1.1 Latar Belakang . . . . .	1
1.2 Rumusan Masalah . . . . .	2
1.3 Batasan Masalah . . . . .	2
1.4 Tujuan . . . . .	2
1.5 Manfaat . . . . .	3
1.6 Metodologi . . . . .	3
1.7 Sistematika Penulisan . . . . .	4
<b>BAB II DASAR TEORI</b> . . . . .	<b>5</b>
2.1 Deskripsi Permasalahan . . . . .	5
2.2 Deskripsi Umum . . . . .	6
2.2.1 String . . . . .	6
2.2.2 Rekurens . . . . .	8
2.2.3 <i>Divide and Conquer</i> . . . . .	8
2.2.4 <i>Meet In The Middle</i> . . . . .	8
2.2.5 <i>Dynamic Programming</i> . . . . .	8
2.2.6 <i>State</i> . . . . .	9
2.2.7 <i>Bitmask</i> . . . . .	9
2.3 Analisa Submasalah Optimal . . . . .	9

2.3.1	Membagi Permasalah Menjadi Submasalah yang <i>Independent</i> . . . . .	9
2.3.2	Submasalah Optimal untuk Menghitung Jumlah Kombinasi String <i>Orig</i> dari String <i>Ad</i> Tanpa Operasi <i>Replace</i> dengan Jarak <i>D</i>	15
2.3.3	Submasalah Optimal untuk Menghitung Jumlah Kombinasi String <i>Orig</i> dari String <i>Ad</i> dengan Operasi <i>Replace</i> dengan Jarak <i>D</i>	17
2.4	Pemodelan Relasi Rekurens . . . . .	21
2.4.1	Pemodelan Relasi Rekurens Submasalah Optimal untuk Menghitung Jumlah Kemungkinan String Awal Tanpa Operasi <i>Replace</i> dengan Jarak $X - dist$ . . . . .	26
2.4.2	Pemodelan Relasi Rekurens Submasalah Optimal untuk Menghitung Jumlah Kemungkinan String Awal dengan Sekali Operasi <i>Replace</i> dengan Jarak $X - dist$ . . . . .	27
<b>BAB III DESAIN</b>	. . . . .	<b>33</b>
3.1	Desain Umum Sistem . . . . .	33
3.2	Desain Fungsi Preprocess . . . . .	33
3.3	Desain Fungsi Init . . . . .	34
3.4	Desain Fungsi Solve . . . . .	70
3.4.1	Desain Fungsi F . . . . .	72
3.4.2	Desain Fungsi G . . . . .	75
<b>BAB IV IMPLEMENTASI</b>	. . . . .	<b>87</b>
4.1	Lingkungan Implementasi . . . . .	87
4.2	Rancangan Data . . . . .	87
4.2.1	Data Masukan . . . . .	88
4.2.2	Data Keluaran . . . . .	88
4.3	Implementasi Algoritma . . . . .	88
4.3.1	Header-Header yang Diperlukan . . . . .	88
4.3.2	Variabel Global . . . . .	89
4.3.3	Implementasi Fungsi Main . . . . .	90

4.3.4	Implementasi Fungsi Preprocess . . . . .	90
4.3.5	Implementasi Fungsi ReadInput . . . . .	91
4.3.6	Implementasi Fungsi Init . . . . .	91
4.3.7	Implementasi Fungsi Solve . . . . .	92
4.3.8	Implementasi Fungsi F . . . . .	93
4.3.9	Implementasi Fungsi F1 . . . . .	94
4.3.10	Implementasi Fungsi G . . . . .	94
4.3.11	Implementasi Fungsi G1 . . . . .	95
4.3.12	Implementasi Fungsi G2 . . . . .	96
4.3.13	Implementasi Fungsi G3 . . . . .	96
4.3.14	Implementasi Fungsi Duplicate Rule 1 . . .	97
4.3.15	Implementasi Fungsi Duplicate Rule 2 . . .	98
4.3.16	Implementasi Fungsi Duplicate Rule 3 . . .	98
<b>BAB V</b>	<b>UJI COBA DAN EVALUASI . . . . .</b>	<b>101</b>
5.1	Lingkungan Uji Coba . . . . .	101
5.2	Uji Coba Kebenaran . . . . .	101
5.3	Analisa Kompleksitas Waktu . . . . .	136
<b>BAB VI</b>	<b>KESIMPULAN . . . . .</b>	<b>169</b>
6.1	Kesimpulan . . . . .	169
6.2	Saran . . . . .	169
<b>DAFTAR PUSTAKA . . . . .</b>	<b>171</b>	
<b>BIODATA PENULIS . . . . .</b>	<b>173</b>	

*Halaman ini sengaja dikosongkan*

## DAFTAR TABEL

Tabel 2.1	Kombinasi string <i>orig1</i> dan <i>orig2</i> dengan <i>ad1</i> = <i>c</i> , string <i>ad2</i> = <i>n</i> dan <i>X</i> = 1 . . . . .	6
Tabel 2.2	Kombinasi string <i>orig1</i> dan <i>orig2</i> dengan <i>ad1</i> = <i>bd</i> , string <i>ad2</i> = <i>gj</i> dan <i>X</i> = 5 . . . . .	7
Tabel 2.3	Kombinasi string <i>orig1</i> dengan nilai string <i>ad1</i> = <i>bd</i> tanpa operasi <i>replace</i> . . . . .	13
Tabel 2.4	Kombinasi string <i>orig2</i> dengan nilai string <i>ad2</i> = <i>gj</i> dengan operasi <i>replace</i> . . . . .	13
Tabel 2.5	Kombinasi string <i>orig1</i> dengan nilai string <i>ad1</i> = <i>bd</i> tanpa operasi <i>replace</i> dan string <i>orig2</i> dengan nilai string <i>ad2</i> = <i>gj</i> dengan operasi <i>replace</i> dengan <i>X</i> = 5 . . . . .	14
Tabel 2.6	Kombinasi string <i>orig1</i> dengan nilai string <i>ad1</i> = <i>bd</i> dengan operasi <i>replace</i> . . . . .	14
Tabel 2.7	Kombinasi string <i>orig2</i> dengan nilai string <i>ad2</i> = <i>gj</i> tanpa operasi <i>replace</i> . . . . .	14
Tabel 2.8	Kombinasi string <i>orig1</i> dengan nilai string <i>ad1</i> = <i>bd</i> dengan operasi <i>replace</i> dan string <i>orig2</i> dengan nilai string <i>ad2</i> = <i>gj</i> tanpa ope- rasi <i>replace</i> dengan <i>X</i> = 5 . . . . .	15
Tabel 3.1	Tabel himpunan setBit setelah fungsi prepro- cess dijalankan (1) . . . . .	35
Tabel 3.2	Tabel himpunan setBit setelah fungsi prepro- cess dijalankan (2) . . . . .	36
Tabel 3.3	Tabel himpunan setBit setelah fungsi prepro- cess dijalankan (3) . . . . .	37

Tabel 3.4 Tabel himpunan setBit setelah fungsi preprocess dijalankan (4) . . . . .	38
Tabel 3.5 Tabel himpunan setBit setelah fungsi preprocess dijalankan (5) . . . . .	39
Tabel 3.6 Tabel himpunan setBit setelah fungsi preprocess dijalankan (6) . . . . .	40
Tabel 3.7 Tabel himpunan setBit setelah fungsi preprocess dijalankan (7) . . . . .	41
Tabel 3.8 Tabel himpunan setBit setelah fungsi preprocess dijalankan (8) . . . . .	42
Tabel 3.9 Tabel himpunan setBit setelah fungsi preprocess dijalankan (9) . . . . .	43
Tabel 3.10 Tabel himpunan setBit setelah fungsi preprocess dijalankan (10) . . . . .	44
Tabel 3.11 Tabel himpunan setBit setelah fungsi preprocess dijalankan (11) . . . . .	45
Tabel 3.12 Tabel himpunan setBit setelah fungsi preprocess dijalankan (12) . . . . .	46
Tabel 3.13 Tabel himpunan setBit setelah fungsi preprocess dijalankan (13) . . . . .	47
Tabel 3.14 Tabel himpunan setBit setelah fungsi preprocess dijalankan (14) . . . . .	48
Tabel 3.15 Tabel himpunan setBit setelah fungsi preprocess dijalankan (15) . . . . .	49
Tabel 3.16 Tabel himpunan setBit setelah fungsi preprocess dijalankan (16) . . . . .	50
Tabel 3.17 Tabel himpunan setBit setelah fungsi preprocess dijalankan (17) . . . . .	51
Tabel 3.18 Tabel himpunan setBit setelah fungsi preprocess dijalankan (18) . . . . .	52
Tabel 3.19 Tabel himpunan setBit setelah fungsi preprocess dijalankan (19) . . . . .	53

Tabel 3.20 Tabel himpunan setBit setelah fungsi preprocess dijalankan (20) . . . . .	54
Tabel 3.21 Tabel himpunan setBit setelah fungsi preprocess dijalankan (21) . . . . .	55
Tabel 3.22 Tabel himpunan setBit setelah fungsi preprocess dijalankan (22) . . . . .	56
Tabel 3.23 Tabel himpunan setBit setelah fungsi preprocess dijalankan (23) . . . . .	57
Tabel 3.24 Tabel himpunan setBit setelah fungsi preprocess dijalankan (24) . . . . .	58
Tabel 3.25 Tabel himpunan setBit setelah fungsi preprocess dijalankan (25) . . . . .	59
Tabel 3.26 Tabel himpunan setBit setelah fungsi preprocess dijalankan (26) . . . . .	60
Tabel 3.27 Tabel himpunan setBit setelah fungsi preprocess dijalankan (27) . . . . .	61
Tabel 3.28 Tabel himpunan setBit setelah fungsi preprocess dijalankan (28) . . . . .	62
Tabel 3.29 Tabel himpunan setBit setelah fungsi preprocess dijalankan (29) . . . . .	63
Tabel 3.30 Tabel himpunan setBit setelah fungsi preprocess dijalankan (30) . . . . .	64
Tabel 3.31 Tabel himpunan setBit setelah fungsi preprocess dijalankan (31) . . . . .	65
Tabel 3.32 Tabel himpunan setBit setelah fungsi preprocess dijalankan (32) . . . . .	66
Tabel 3.33 Tabel himpunan setBit setelah fungsi preprocess dijalankan (33) . . . . .	67
Tabel 3.34 Tabel himpunan setBit setelah fungsi preprocess dijalankan (34) . . . . .	68
Tabel 3.35 Tabel himpunan setBit setelah fungsi preprocess dijalankan (35) . . . . .	69

Tabel 3.36 Hasil $charFirstPos_{(S,C)}$ dan $charLastPos_{(S,C)}$ dengan string $S = "inicontoh"$ setelah fungsi init dijalankan . . . . .	71
Tabel 3.37 Simulasi fungsi $F$ dengan $S = "kbenh"$ , $X = 5$ dan $dist = 5$ . . . . .	74
Tabel 3.38 Simulasi fungsi $G$ dengan $S = "kbenh"$ , $X = 5$ dan $dist = 0 (1)$ . . . . .	77
Tabel 3.39 Simulasi fungsi $G$ dengan $S = "kbenh"$ , $X = 5$ dan $dist = 0 (2)$ . . . . .	78
Tabel 3.40 Simulasi fungsi $G$ dengan $S = "kbenh"$ , $X = 5$ dan $dist = 0 (3)$ . . . . .	79
Tabel 3.41 Simulasi fungsi $G$ dengan $S = "kbenh"$ , $X = 5$ dan $dist = 0 (4)$ . . . . .	80
Tabel 3.42 Simulasi fungsi $G$ dengan $S = "kbenh"$ , $X = 5$ dan $dist = 0 (5)$ . . . . .	81
Tabel 3.43 Simulasi fungsi $G$ dengan $S = "kbenh"$ , $X = 5$ dan $dist = 0 (6)$ . . . . .	82
Tabel 5.1 Kecepatan maksimal, minimal dan rata-rata dari hasil uji coba pengumpulan 30 kali pada situs pengujian daring SPOJ . . . . .	104
Tabel 5.2 Simulasi perhitungan jumlah kombinasi string $orig1$ tanpa operasi $replace$ dengan $dist = 0$ pada kasus string $ad1 = c$ , string $ad2 = n$ dan $X = 1$ . . . . .	104
Tabel 5.3 Simulasi perhitungan jumlah kombinasi string $orig2$ dengan operasi $replace$ dengan $dist = 0$ pada kasus string $ad1 = c$ , string $ad2 = n$ dan $X = 1$ . . . . .	105

Tabel 5.4 Simulasi perhitungan jumlah kombinasi string <i>orig1</i> dengan operasi <i>replace</i> dengan <i>dist</i> = 0 pada kasus string <i>ad1</i> = <i>c</i> , string <i>ad2</i> = <i>n</i> dan <i>X</i> = 1 . . . . .	105
Tabel 5.5 Simulasi perhitungan jumlah kombinasi string <i>orig2</i> tanpa operasi <i>replace</i> dengan <i>dist</i> = 0 pada kasus string <i>ad1</i> = <i>c</i> , string <i>ad2</i> = <i>n</i> dan <i>X</i> = 1 . . . . .	105
Tabel 5.6 Simulasi perhitungan jumlah kombinasi string <i>orig1</i> tanpa operasi <i>replace</i> dengan <i>dist</i> = 1 pada kasus string <i>ad1</i> = <i>c</i> , string <i>ad2</i> = <i>n</i> dan <i>X</i> = 1 . . . . .	106
Tabel 5.7 Simulasi perhitungan jumlah kombinasi string <i>orig2</i> dengan operasi <i>replace</i> dengan <i>dist</i> = 1 pada kasus string <i>ad1</i> = <i>c</i> , string <i>ad2</i> = <i>n</i> dan <i>X</i> = 1 . . . . .	106
Tabel 5.8 Simulasi perhitungan jumlah kombinasi string <i>orig1</i> dengan operasi <i>replace</i> dengan <i>dist</i> = 1 pada kasus string <i>ad1</i> = <i>c</i> , string <i>ad2</i> = <i>n</i> dan <i>X</i> = 1 . . . . .	107
Tabel 5.9 Simulasi perhitungan jumlah kombinasi string <i>orig2</i> tanpa operasi <i>replace</i> dengan <i>dist</i> = 1 pada kasus string <i>ad1</i> = <i>c</i> , string <i>ad2</i> = <i>n</i> dan <i>X</i> = 1 . . . . .	107
Tabel 5.10 Simulasi perhitungan jumlah kombinasi string <i>orig1</i> tanpa operasi <i>replace</i> dengan <i>dist</i> = 0 pada kasus string <i>ad1</i> = <i>kbenh</i> , string <i>ad2</i> = <i>kbenh</i> dan <i>X</i> = 5 . . . . .	110
Tabel 5.11 Simulasi perhitungan jumlah kombinasi string <i>orig2</i> dengan operasi <i>replace</i> dengan <i>dist</i> = 0 pada kasus string <i>ad1</i> = <i>kbenh</i> , string <i>ad2</i> = <i>kbenh</i> dan <i>X</i> = 5 (1) . . . . .	111

Tabel 5.12 Simulasi perhitungan jumlah kombinasi string <i>orig2</i> dengan operasi <i>replace</i> dengan <i>dist</i> = 0 pada kasus string <i>ad1</i> = <i>kbenh</i> , string <i>ad2</i> = <i>kbenh</i> dan <i>X</i> = 5 (2) . . . . .	112
Tabel 5.13 Simulasi perhitungan jumlah kombinasi string <i>orig2</i> dengan operasi <i>replace</i> dengan <i>dist</i> = 0 pada kasus string <i>ad1</i> = <i>kbenh</i> , string <i>ad2</i> = <i>kbenh</i> dan <i>X</i> = 5 (3) . . . . .	113
Tabel 5.14 Simulasi perhitungan jumlah kombinasi string <i>orig2</i> dengan operasi <i>replace</i> dengan <i>dist</i> = 0 pada kasus string <i>ad1</i> = <i>kbenh</i> , string <i>ad2</i> = <i>kbenh</i> dan <i>X</i> = 5 (4) . . . . .	114
Tabel 5.15 Simulasi perhitungan jumlah kombinasi string <i>orig2</i> dengan operasi <i>replace</i> dengan <i>dist</i> = 0 pada kasus string <i>ad1</i> = <i>kbenh</i> , string <i>ad2</i> = <i>kbenh</i> dan <i>X</i> = 5 (5) . . . . .	115
Tabel 5.16 Simulasi perhitungan jumlah kombinasi string <i>orig2</i> dengan operasi <i>replace</i> dengan <i>dist</i> = 0 pada kasus string <i>ad1</i> = <i>kbenh</i> , string <i>ad2</i> = <i>kbenh</i> dan <i>X</i> = 5 (6) . . . . .	116
Tabel 5.17 Simulasi perhitungan jumlah kombinasi string <i>orig1</i> dengan operasi <i>replace</i> dengan <i>dist</i> = 0 pada kasus string <i>ad1</i> = <i>kbenh</i> , string <i>ad2</i> = <i>kbenh</i> dan <i>X</i> = 5 (1) . . . . .	117
Tabel 5.18 Simulasi perhitungan jumlah kombinasi string <i>orig1</i> dengan operasi <i>replace</i> dengan <i>dist</i> = 0 pada kasus string <i>ad1</i> = <i>kbenh</i> , string <i>ad2</i> = <i>kbenh</i> dan <i>X</i> = 5 (2) . . . . .	118
Tabel 5.19 Simulasi perhitungan jumlah kombinasi string <i>orig1</i> dengan operasi <i>replace</i> dengan <i>dist</i> = 0 pada kasus string <i>ad1</i> = <i>kbenh</i> , string <i>ad2</i> = <i>kbenh</i> dan <i>X</i> = 5 (3) . . . . .	119

Tabel 5.20 Simulasi perhitungan jumlah kombinasi string <i>orig2</i> tanpa operasi <i>replace</i> dengan <i>dist</i> = 0 pada kasus string <i>ad1</i> = <i>kbenh</i> , string <i>ad2</i> = <i>kbenh</i> dan <i>X</i> = 5 (1) . . . . .	120
Tabel 5.21 Simulasi perhitungan jumlah kombinasi string <i>orig2</i> tanpa operasi <i>replace</i> dengan <i>dist</i> = 0 pada kasus string <i>ad1</i> = <i>kbenh</i> , string <i>ad2</i> = <i>kbenh</i> dan <i>X</i> = 5 (2) . . . . .	121
Tabel 5.22 Simulasi perhitungan jumlah kombinasi string <i>orig1</i> tanpa operasi <i>replace</i> dengan <i>dist</i> = 1 pada kasus string <i>ad1</i> = <i>kbenh</i> , string <i>ad2</i> = <i>kbenh</i> dan <i>X</i> = 5 . . . . .	122
Tabel 5.23 Simulasi perhitungan jumlah kombinasi string <i>orig2</i> dengan operasi <i>replace</i> dengan <i>dist</i> = 1 pada kasus string <i>ad1</i> = <i>kbenh</i> , string <i>ad2</i> = <i>kbenh</i> dan <i>X</i> = 5 (1) . . . . .	123
Tabel 5.24 Simulasi perhitungan jumlah kombinasi string <i>orig2</i> dengan operasi <i>replace</i> dengan <i>dist</i> = 1 pada kasus string <i>ad1</i> = <i>kbenh</i> , string <i>ad2</i> = <i>kbenh</i> dan <i>X</i> = 5 (2) . . . . .	124
Tabel 5.25 Simulasi perhitungan jumlah kombinasi string <i>orig2</i> dengan operasi <i>replace</i> dengan <i>dist</i> = 1 pada kasus string <i>ad1</i> = <i>kbenh</i> , string <i>ad2</i> = <i>kbenh</i> dan <i>X</i> = 5 (3) . . . . .	125
Tabel 5.26 Simulasi perhitungan jumlah kombinasi string <i>orig2</i> dengan operasi <i>replace</i> dengan <i>dist</i> = 1 pada kasus string <i>ad1</i> = <i>kbenh</i> , string <i>ad2</i> = <i>kbenh</i> dan <i>X</i> = 5 (4) . . . . .	126
Tabel 5.27 Simulasi perhitungan jumlah kombinasi string <i>orig2</i> dengan operasi <i>replace</i> dengan <i>dist</i> = 1 pada kasus string <i>ad1</i> = <i>kbenh</i> , string <i>ad2</i> = <i>kbenh</i> dan <i>X</i> = 5 (5) . . . . .	127

Tabel 5.28 Simulasi perhitungan jumlah kombinasi string <i>orig1</i> dengan operasi <i>replace</i> dengan <i>dist</i> = 1 pada kasus string <i>ad1</i> = <i>kbenh</i> , string <i>ad2</i> = <i>kbenh</i> dan <i>X</i> = 5 (1) . . . . .	128
Tabel 5.29 Simulasi perhitungan jumlah kombinasi string <i>orig1</i> dengan operasi <i>replace</i> dengan <i>dist</i> = 1 pada kasus string <i>ad1</i> = <i>kbenh</i> , string <i>ad2</i> = <i>kbenh</i> dan <i>X</i> = 5 (2) . . . . .	129
Tabel 5.30 Simulasi perhitungan jumlah kombinasi string <i>orig1</i> dengan operasi <i>replace</i> dengan <i>dist</i> = 1 pada kasus string <i>ad1</i> = <i>kbenh</i> , string <i>ad2</i> = <i>kbenh</i> dan <i>X</i> = 5 (3) . . . . .	130
Tabel 5.31 Simulasi perhitungan jumlah kombinasi string <i>orig2</i> tanpa operasi <i>replace</i> dengan <i>dist</i> = 1 pada kasus string <i>ad1</i> = <i>kbenh</i> , string <i>ad2</i> = <i>kbenh</i> dan <i>X</i> = 5 . . . . .	130
Tabel 5.32 Simulasi perhitungan jumlah kombinasi string <i>orig1</i> tanpa operasi <i>replace</i> dengan <i>dist</i> = 2 pada kasus string <i>ad1</i> = <i>kbenh</i> , string <i>ad2</i> = <i>kbenh</i> dan <i>X</i> = 5 . . . . .	131
Tabel 5.33 Simulasi perhitungan jumlah kombinasi string <i>orig2</i> dengan operasi <i>replace</i> dengan <i>dist</i> = 2 pada kasus string <i>ad1</i> = <i>kbenh</i> , string <i>ad2</i> = <i>kbenh</i> dan <i>X</i> = 5 (1) . . . . .	132
Tabel 5.34 Simulasi perhitungan jumlah kombinasi string <i>orig2</i> dengan operasi <i>replace</i> dengan <i>dist</i> = 2 pada kasus string <i>ad1</i> = <i>kbenh</i> , string <i>ad2</i> = <i>kbenh</i> dan <i>X</i> = 5 (2) . . . . .	133
Tabel 5.35 Simulasi perhitungan jumlah kombinasi string <i>orig2</i> dengan operasi <i>replace</i> dengan <i>dist</i> = 2 pada kasus string <i>ad1</i> = <i>kbenh</i> , string <i>ad2</i> = <i>kbenh</i> dan <i>X</i> = 5 (3) . . . . .	134

Tabel 5.36 Simulasi perhitungan jumlah kombinasi string <i>orig2</i> dengan operasi <i>replace</i> dengan <i>dist</i> = 2 pada kasus string <i>ad1</i> = <i>kbenh</i> , string <i>ad2</i> = <i>kbenh</i> dan <i>X</i> = 5 (4) . . . . .	135
Tabel 5.37 Simulasi perhitungan jumlah kombinasi string <i>orig2</i> dengan operasi <i>replace</i> dengan <i>dist</i> = 2 pada kasus string <i>ad1</i> = <i>kbenh</i> , string <i>ad2</i> = <i>kbenh</i> dan <i>X</i> = 5 (5) . . . . .	136
Tabel 5.38 Simulasi perhitungan jumlah kombinasi string <i>orig1</i> dengan operasi <i>replace</i> dengan <i>dist</i> = 2 pada kasus string <i>ad1</i> = <i>kbenh</i> , string <i>ad2</i> = <i>kbenh</i> dan <i>X</i> = 5 (1) . . . . .	137
Tabel 5.39 Simulasi perhitungan jumlah kombinasi string <i>orig1</i> dengan operasi <i>replace</i> dengan <i>dist</i> = 2 pada kasus string <i>ad1</i> = <i>kbenh</i> , string <i>ad2</i> = <i>kbenh</i> dan <i>X</i> = 5 (2) . . . . .	138
Tabel 5.40 Simulasi perhitungan jumlah kombinasi string <i>orig1</i> dengan operasi <i>replace</i> dengan <i>dist</i> = 2 pada kasus string <i>ad1</i> = <i>kbenh</i> , string <i>ad2</i> = <i>kbenh</i> dan <i>X</i> = 5 (3) . . . . .	139
Tabel 5.41 Simulasi perhitungan jumlah kombinasi string <i>orig2</i> tanpa operasi <i>replace</i> dengan <i>dist</i> = 2 pada kasus string <i>ad1</i> = <i>kbenh</i> , string <i>ad2</i> = <i>kbenh</i> dan <i>X</i> = 5 . . . . .	140
Tabel 5.42 Simulasi perhitungan jumlah kombinasi string <i>orig1</i> tanpa operasi <i>replace</i> dengan <i>dist</i> = 3 pada kasus string <i>ad1</i> = <i>kbenh</i> , string <i>ad2</i> = <i>kbenh</i> dan <i>X</i> = 5 . . . . .	141
Tabel 5.43 Simulasi perhitungan jumlah kombinasi string <i>orig2</i> dengan operasi <i>replace</i> dengan <i>dist</i> = 3 pada kasus string <i>ad1</i> = <i>kbenh</i> , string <i>ad2</i> = <i>kbenh</i> dan <i>X</i> = 5 (1) . . . . .	142

Tabel 5.44 Simulasi perhitungan jumlah kombinasi string <i>orig2</i> dengan operasi <i>replace</i> dengan <i>dist</i> = 3 pada kasus string <i>ad1</i> = <i>kbenh</i> , string <i>ad2</i> = <i>kbenh</i> dan <i>X</i> = 5 (2) . . . . .	143
Tabel 5.45 Simulasi perhitungan jumlah kombinasi string <i>orig2</i> dengan operasi <i>replace</i> dengan <i>dist</i> = 3 pada kasus string <i>ad1</i> = <i>kbenh</i> , string <i>ad2</i> = <i>kbenh</i> dan <i>X</i> = 5 (3) . . . . .	144
Tabel 5.46 Simulasi perhitungan jumlah kombinasi string <i>orig1</i> dengan operasi <i>replace</i> dengan <i>dist</i> = 3 pada kasus string <i>ad1</i> = <i>kbenh</i> , string <i>ad2</i> = <i>kbenh</i> dan <i>X</i> = 5 (1) . . . . .	145
Tabel 5.47 Simulasi perhitungan jumlah kombinasi string <i>orig1</i> dengan operasi <i>replace</i> dengan <i>dist</i> = 3 pada kasus string <i>ad1</i> = <i>kbenh</i> , string <i>ad2</i> = <i>kbenh</i> dan <i>X</i> = 5 (2) . . . . .	146
Tabel 5.48 Simulasi perhitungan jumlah kombinasi string <i>orig1</i> dengan operasi <i>replace</i> dengan <i>dist</i> = 3 pada kasus string <i>ad1</i> = <i>kbenh</i> , string <i>ad2</i> = <i>kbenh</i> dan <i>X</i> = 5 (3) . . . . .	147
Tabel 5.49 Simulasi perhitungan jumlah kombinasi string <i>orig1</i> dengan operasi <i>replace</i> dengan <i>dist</i> = 3 pada kasus string <i>ad1</i> = <i>kbenh</i> , string <i>ad2</i> = <i>kbenh</i> dan <i>X</i> = 5 (4) . . . . .	148
Tabel 5.50 Simulasi perhitungan jumlah kombinasi string <i>orig1</i> dengan operasi <i>replace</i> dengan <i>dist</i> = 3 pada kasus string <i>ad1</i> = <i>kbenh</i> , string <i>ad2</i> = <i>kbenh</i> dan <i>X</i> = 5 (5) . . . . .	149
Tabel 5.51 Simulasi perhitungan jumlah kombinasi string <i>orig2</i> tanpa operasi <i>replace</i> dengan <i>dist</i> = 3 pada kasus string <i>ad1</i> = <i>kbenh</i> , string <i>ad2</i> = <i>kbenh</i> dan <i>X</i> = 5 . . . . .	149

Tabel 5.52 Simulasi perhitungan jumlah kombinasi string <i>orig1</i> tanpa operasi <i>replace</i> dengan <i>dist</i> = 4 pada kasus string <i>ad1</i> = <i>kbenh</i> , string <i>ad2</i> = <i>kbenh</i> dan <i>X</i> = 5 . . . . .	150
Tabel 5.53 Simulasi perhitungan jumlah kombinasi string <i>orig2</i> dengan operasi <i>replace</i> dengan <i>dist</i> = 4 pada kasus string <i>ad1</i> = <i>kbenh</i> , string <i>ad2</i> = <i>kbenh</i> dan <i>X</i> = 5 (1) . . . . .	151
Tabel 5.54 Simulasi perhitungan jumlah kombinasi string <i>orig2</i> dengan operasi <i>replace</i> dengan <i>dist</i> = 4 pada kasus string <i>ad1</i> = <i>kbenh</i> , string <i>ad2</i> = <i>kbenh</i> dan <i>X</i> = 5 (2) . . . . .	152
Tabel 5.55 Simulasi perhitungan jumlah kombinasi string <i>orig2</i> dengan operasi <i>replace</i> dengan <i>dist</i> = 4 pada kasus string <i>ad1</i> = <i>kbenh</i> , string <i>ad2</i> = <i>kbenh</i> dan <i>X</i> = 5 (3) . . . . .	153
Tabel 5.56 Simulasi perhitungan jumlah kombinasi string <i>orig1</i> dengan operasi <i>replace</i> dengan <i>dist</i> = 4 pada kasus string <i>ad1</i> = <i>kbenh</i> , string <i>ad2</i> = <i>kbenh</i> dan <i>X</i> = 5 (1) . . . . .	154
Tabel 5.57 Simulasi perhitungan jumlah kombinasi string <i>orig1</i> dengan operasi <i>replace</i> dengan <i>dist</i> = 4 pada kasus string <i>ad1</i> = <i>kbenh</i> , string <i>ad2</i> = <i>kbenh</i> dan <i>X</i> = 5 (2) . . . . .	155
Tabel 5.58 Simulasi perhitungan jumlah kombinasi string <i>orig1</i> dengan operasi <i>replace</i> dengan <i>dist</i> = 4 pada kasus string <i>ad1</i> = <i>kbenh</i> , string <i>ad2</i> = <i>kbenh</i> dan <i>X</i> = 5 (3) . . . . .	156
Tabel 5.59 Simulasi perhitungan jumlah kombinasi string <i>orig1</i> dengan operasi <i>replace</i> dengan <i>dist</i> = 4 pada kasus string <i>ad1</i> = <i>kbenh</i> , string <i>ad2</i> = <i>kbenh</i> dan <i>X</i> = 5 (4) . . . . .	157

Tabel 5.60 Simulasi perhitungan jumlah kombinasi string <i>orig1</i> dengan operasi <i>replace</i> dengan <i>dist</i> = 4 pada kasus string <i>ad1</i> = <i>kbenh</i> , string <i>ad2</i> = <i>kbenh</i> dan <i>X</i> = 5 (5) . . . . .	158
Tabel 5.61 Simulasi perhitungan jumlah kombinasi string <i>orig2</i> tanpa operasi <i>replace</i> dengan <i>dist</i> = 4 pada kasus string <i>ad1</i> = <i>kbenh</i> , string <i>ad2</i> = <i>kbenh</i> dan <i>X</i> = 5 . . . . .	158
Tabel 5.62 Simulasi perhitungan jumlah kombinasi string <i>orig1</i> tanpa operasi <i>replace</i> dengan <i>dist</i> = 5 pada kasus string <i>ad1</i> = <i>kbenh</i> , string <i>ad2</i> = <i>kbenh</i> dan <i>X</i> = 5 . . . . .	159
Tabel 5.63 Simulasi perhitungan jumlah kombinasi string <i>orig2</i> dengan operasi <i>replace</i> dengan <i>dist</i> = 5 pada kasus string <i>ad1</i> = <i>kbenh</i> , string <i>ad2</i> = <i>kbenh</i> dan <i>X</i> = 5 (1) . . . . .	160
Tabel 5.64 Simulasi perhitungan jumlah kombinasi string <i>orig2</i> dengan operasi <i>replace</i> dengan <i>dist</i> = 5 pada kasus string <i>ad1</i> = <i>kbenh</i> , string <i>ad2</i> = <i>kbenh</i> dan <i>X</i> = 5 (2) . . . . .	161
Tabel 5.65 Simulasi perhitungan jumlah kombinasi string <i>orig2</i> dengan operasi <i>replace</i> dengan <i>dist</i> = 5 pada kasus string <i>ad1</i> = <i>kbenh</i> , string <i>ad2</i> = <i>kbenh</i> dan <i>X</i> = 5 (3) . . . . .	162
Tabel 5.66 Simulasi perhitungan jumlah kombinasi string <i>orig1</i> dengan operasi <i>replace</i> dengan <i>dist</i> = 5 pada kasus string <i>ad1</i> = <i>kbenh</i> , string <i>ad2</i> = <i>kbenh</i> dan <i>X</i> = 5 (1) . . . . .	163
Tabel 5.67 Simulasi perhitungan jumlah kombinasi string <i>orig1</i> dengan operasi <i>replace</i> dengan <i>dist</i> = 5 pada kasus string <i>ad1</i> = <i>kbenh</i> , string <i>ad2</i> = <i>kbenh</i> dan <i>X</i> = 5 (2) . . . . .	164

Tabel 5.68 Simulasi perhitungan jumlah kombinasi string <i>orig1</i> dengan operasi <i>replace</i> dengan <i>dist</i> = 5 pada kasus string <i>ad1</i> = <i>kbenh</i> , string <i>ad2</i> = <i>kbenh</i> dan <i>X</i> = 5 (3) . . . . .	165
Tabel 5.69 Simulasi perhitungan jumlah kombinasi string <i>orig1</i> dengan operasi <i>replace</i> dengan <i>dist</i> = 5 pada kasus string <i>ad1</i> = <i>kbenh</i> , string <i>ad2</i> = <i>kbenh</i> dan <i>X</i> = 5 (4) . . . . .	166
Tabel 5.70 Simulasi perhitungan jumlah kombinasi string <i>orig1</i> dengan operasi <i>replace</i> dengan <i>dist</i> = 5 pada kasus string <i>ad1</i> = <i>kbenh</i> , string <i>ad2</i> = <i>kbenh</i> dan <i>X</i> = 5 (5) . . . . .	167
Tabel 5.71 Simulasi perhitungan jumlah kombinasi string <i>orig2</i> tanpa operasi <i>replace</i> dengan <i>dist</i> = 5 pada kasus string <i>ad1</i> = <i>kbenh</i> , string <i>ad2</i> = <i>kbenh</i> dan <i>X</i> = 5 . . . . .	168

*Halaman ini sengaja dikosongkan*

## DAFTAR GAMBAR

Gambar 2.1	Ilustrasi umum penyelesaian permasalahan dengan metode <i>meet in the middle</i> tanpa operasi <i>replace</i> . . . . .	10
Gambar 2.2	Ilustrasi umum penyelesaian permasalahan dengan metode <i>meet in the middle</i> dengan operasi <i>replace</i> . . . . .	12
Gambar 2.3	Ilustrasi umum penyelesaian permasalahan dengan metode <i>meet in the middle</i> dengan operasi <i>replace</i> tanpa mempedulikan kombinasi string yang dihasilkan . . . . .	16
Gambar 2.4	Ilustrasi perhitungan jumlah kombinasi string <i>orig</i> dari string <i>ad</i> tanpa operasi <i>replace</i> dengan nilai string $ad = bcd$ dan $D = 4$ . . . . .	18
Gambar 2.5	Contoh kasus tumpang tindih pada perhitungan jumlah kombinasi string <i>orig</i> dari string <i>ad</i> tanpa operasi <i>replace</i> dengan nilai string $ad = bcd$ dan $D = 4$ . . . . .	19
Gambar 2.6	Ilustrasi perhitungan jumlah kombinasi string <i>orig</i> dari string <i>ad</i> tanpa operasi <i>replace</i> dengan nilai string $ad = bcd$ dan $D = 4$ tanpa menghitung kasus yang tumpang tindih . . . . .	20
Gambar 2.7	Ilustrasi perhitungan jumlah kombinasi string <i>orig</i> dari string <i>ad</i> dengan operasi <i>replace</i> dengan nilai string $ad = be$ dan $D = 1$	22

Gambar 2.8 Contoh kasus tumpang tindih pada perhitungan kombinasi string <i>orig</i> dari string <i>ad</i> dengan operasi <i>replace</i> dengan nilai string <i>ad</i> = <i>be</i> dan <i>D</i> = 1 . . . . .	23
Gambar 2.9 Submasalah perhitungan jumlah kombinasi string <i>orig</i> terhadap string <i>ad</i> tanpa operasi <i>replace</i> pada submasalah perhitungan jumlah kombinasi string <i>orig</i> terhadap string <i>ad</i> dengan operasi <i>replace</i> . . . . .	24
Gambar 3.1 Pseudocode Fungsi Main . . . . .	34
Gambar 3.2 Pseudocode Fungsi Preprocess . . . . .	34
Gambar 3.3 Pseudocode Fungsi Init . . . . .	70
Gambar 3.4 Pseudocode Fungsi Solve . . . . .	72
Gambar 3.5 Pseudocode Fungsi F . . . . .	73
Gambar 3.6 Pseudocode Fungsi F1 . . . . .	75
Gambar 3.7 Pseudocode Fungsi duplicate_rule1 . . . . .	75
Gambar 3.8 Pseudocode Fungsi G . . . . .	83
Gambar 3.9 Pseudocode Fungsi G1 . . . . .	83
Gambar 3.10 Pseudocode Fungsi G2 . . . . .	84
Gambar 3.11 Pseudocode Fungsi G3 . . . . .	84
Gambar 3.12 Pseudocode Fungsi duplicate_rule2 . . . . .	84
Gambar 3.13 Pseudocode Fungsi duplicate_rule3 . . . . .	85
Gambar 5.1 Hasil uji coba pada situs penilaian SPOJ . .	101
Gambar 5.2 Grafik hasil uji coba pada situs SPOJ sebanyak 30 kali . . . . .	102
Gambar 5.3 Hasil pengujian sebanyak 30 kali pada situs penilaian daring SPOJ (1) . . . . .	103
Gambar 5.4 Hasil pengujian sebanyak 30 kali pada situs penilaian daring SPOJ (2) . . . . .	103
Gambar 5.5 Hasil pengujian sebanyak 30 kali pada situs penilaian daring SPOJ (3) . . . . .	104

## DAFTAR KODE SUMBER

Kode Sumber 4.3.1	Header yang diperlukan . . . . .	89
Kode Sumber 4.3.2	Variabel global . . . . .	90
Kode Sumber 4.3.3	Fungsi main . . . . .	90
Kode Sumber 4.3.4	Fungsi preprocess . . . . .	91
Kode Sumber 4.3.5	Fungsi readInput . . . . .	91
Kode Sumber 4.3.6	Fungsi init . . . . .	92
Kode Sumber 4.3.7	Fungsi solve . . . . .	93
Kode Sumber 4.3.8	Fungsi F (1) . . . . .	93
Kode Sumber 4.3.9	Fungsi F (2) . . . . .	94
Kode Sumber 4.3.10	Fungsi F1 . . . . .	94
Kode Sumber 4.3.11	Fungsi G . . . . .	95
Kode Sumber 4.3.12	Fungsi G1 (1) . . . . .	95
Kode Sumber 4.3.13	Fungsi G1 (2) . . . . .	96
Kode Sumber 4.3.14	Fungsi G2 . . . . .	96
Kode Sumber 4.3.15	Fungsi G3 . . . . .	97
Kode Sumber 4.3.16	Fungsi duplicate_rule1 (1) . . . . .	97
Kode Sumber 4.3.17	Fungsi duplicate_rule1 (2) . . . . .	98
Kode Sumber 4.3.18	Fungsi duplicate_rule2 . . . . .	98
Kode Sumber 4.3.19	Fungsi duplicate_rule3 . . . . .	99

*Halaman ini sengaja dikosongkan*

## BAB I

# PENDAHULUAN

Pada bab ini akan dijelaskan latar belakang, rumusan masalah, batasan masalah, tujuan, metodologi dan sistematika penulisan Tugas Akhir.

### 1.1 Latar Belakang

Tugas Akhir ini mengacu pada permasalahan klasik SPOJ 9967 *Playing With Words*. Diberikan dua buah string *orig1* dan *orig2*. Diberikan tiga tahapan proses enkripsi 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  $X$ .

Untuk menyelesaikan permasalahan di atas, penulis akan menggunakan pendekatan solusi dengan teknik *meet in the middle* dan *dynamic programming*. Selain dapat menjawab pertanyaan dengan benar, waktu juga menjadi salah satu faktor penting untuk memberikan gambaran tentang performa dari algoritma yang dirancang.

Hasil dari Tugas Akhir ini diharapkan dapat memberikan gambaran

mengenai peforma algoritma dengan teknik *meet in the middle* dan *dynamic programming*.

## 1.2 Rumusan Masalah

Rumusan masalah yang diangkat dalam Tugas Akhir ini adalah sebagai berikut:

1. Penerapan *meet in the middle* dan *dynamic programming* sebagai pendekatan untuk menyelesaikan permasalahan klasik SPOJ 9967 *Playing With Words*.
2. Pengimplementasian desain algoritma yang didasari oleh metode *meet in the middle* dan *dynamic programming* untuk menyelesaikan permasalahan klasik SPOJ 9967 *Playing With Words*.

## 1.3 Batasan Masalah

Permasalahan yang dibahas pada Tugas Akhir ini memiliki beberapa batasan, yaitu sebagai berikut:

1. Implementasi algoritma menggunakan bahasa pemrograman C++.
2. *Dataset* yang digunakan untuk menguji algoritma yang telah dirancang adalah *dataset* pada permasalahan klasik SPOJ 9967 *Playing With Words*.

## 1.4 Tujuan

Tujuan dari Tugas Akhir ini adalah sebagai berikut:

1. Menyelesaikan permasalahan klasik SPOJ 9967 *Playing With Words* dengan algoritma yang dibangun.
2. Mengevaluasi dan menganalisa peforma dari algoritma yang dibangun.

## 1.5 Manfaat

Manfaat dari Tugas Akhir ini adalah sebagai berikut:

1. Mengetahui performa dari algoritma yang dirancang dengan teknik *meet in the middle* dan pendekatan *dynamic programming* untuk menyelesaikan permasalahan klasik SPOJ 9967 *Playing With Words*.

## 1.6 Metodologi

Metodologi yang digunakan dalam pengerjaan Tugas Akhir ini adalah sebagai berikut:

1. Penyusunan proposal Tugas Akhir

Pada tahap ini dilakukan penyusunan proposal Tugas Akhir yang berisi permasalahan dan gagasan solusi yang akan diteliti pada permasalahan klasik SPOJ 9967 *Playing With Words*.

2. Studi literatur

Pada tahap ini dilakukan pencarian informasi dan studi literatur mengenai pengetahuan atau metode yang dapat digunakan dalam penyelesaian masalah. Informasi didapatkan dari materi-materi yang berhubungan dengan algoritma yang digunakan untuk penyelesaian permasalahan ini, materi-materi tersebut didapatkan dari buku, jurnal, maupun internet.

3. Desain

Pada tahap ini dilakukan desain rancangan algoritma yang digunakan dalam solusi untuk pemecahan permasalahan klasik SPOJ 9967 *Playing With Words*.

4. Implementasi perangkat lunak

Pada tahap ini dilakukan implementasi atau realiasi dari rancangan desain algoritma yang telah dibangun pada tahap desain ke dalam bentuk program.

5. Uji coba dan evaluasi

Pada tahap ini dilakukan uji coba kebenaran implementasi

dan uji coba generalisasi. Pengujian kebenaran dilakukan pada sistem penilaian daring SPOJ sesuai dengan masalah yang dikerjakan untuk diuji apakah luaran dari program telah sesuai. Uji coba generalisasi digunakan untuk menguji struktur data generalisasi pada variasi permasalahan lain.

#### 6. Penyusunan buku Tugas Akhir

Pada tahap ini dilakukan penyusunan buku Tugas Akhir yang berisi dokumentasi hasil penggerjaan Tugas Akhir.

### 1.7 Sistematika Penulisan

Berikut adalah sistematika penulisan buku Tugas Akhir ini:

#### 1. BABI: PENDAHULUAN

Bab ini berisi latar belakang, rumusan masalah, batasan masalah, tujuan, metodologi dan sistematika penulisan Tugas Akhir.

#### 2. BAB II: DASAR TEORI

Bab ini berisi dasar teori mengenai permasalahan dan algoritma penyelesaian yang digunakan dalam Tugas Akhir

#### 3. BAB III: DESAIN

Bab ini berisi desain algoritma dan struktur data yang digunakan dalam penyelesaian permasalahan.

#### 4. BAB IV: IMPLEMENTASI

Bab ini berisi implementasi berdasarkan desain algoritma yang telah dilakukan pada tahap desain.

#### 5. BAB V: UJI COBA DAN EVALUASI

Bab ini berisi uji coba dan evaluasi dari hasil implementasi yang telah dilakukan pada tahap implementasi.

#### 6. BAB VI: KESIMPULAN

Bab ini berisi kesimpulan yang didapat dari hasil uji coba yang telah dilakukan.

## BAB II

### DASAR TEORI

Pada bab ini akan dijelaskan mengenai dasar teori yang menjadi dasar penggerjaan Tugas Akhir ini.

#### 2.1 Deskripsi Permasalahan

Amr M. sangat curiga dengan sebuah iklan yang terdiri dari 2 buah string  $ad1$  dan  $ad2$ . Ia menduga kedua string tersebut menyimpan pesan tersembunyi. Setelah menghubungi sumber yang terpercaya, ia menemukan proses untuk menyembunyikan pesan tersebut. Pesan asli yang dibawa selalu berupa dua buah string  $orig1$  dan  $orig2$ . Berikut adalah langkah-langkah transformasi pesan asli menjadi pesan pada iklan:

1. Karakter-karakter pada string  $orig1$  diacak urutannya.
2. Karakter-karakter pada string  $orig2$  diacak urutannya.
3. Salah satu karakter dari string  $orig1$  atau  $orig2$  diganti dengan karakter sebelum atau sesudahnya dalam alfabet.

Langkah - langkah di atas akan menghasilkan string  $ad1$  dan  $ad2$  dari  $orig1$  dan  $orig2$  secara berurutan. Contohnya untuk string  $orig1 = "bcd"$  dan  $orig2 = "wcy"$  dapat menghasilkan string  $ad1 = "dcb"$  dan  $ad2 = "cxy"$  di mana  $"cxy"$  berasal dari  $"wcy"$  yang diacak menjadi  $"cwy"$  dan karakter 'w' digantikan dengan karakter 'x'.

Setelah melakukan riset, Amr menemukan sebuah jarak  $X$ , di mana  $X$  adalah  $jarak(orig1 + ad1)$  ditambah dengan  $jarak(orig2 + ad2)$ . Jarak antara dua string didefinisikan sebagai jumlah dari selisih absolut dari karakter-karakter pada posisi yang sama. Contohnya  $jarak("ab", "cd") = |'a' - 'c'| + |'b' - 'd'| = 4$ . Diberikan

orig1	orig2	X	Validitas
b	n	1	Valid
d	n	1	Valid
c	m	1	Valid
c	o	1	Valid

Tabel 2.1 Kombinasi string  $orig1$  dan  $orig2$  dengan  $ad1 = c$ , string  $ad2 = n$  dan  $X = 1$

string  $ad1$ ,  $ad2$  dan sebuah bilangan bulat  $X$ . Hitung jumlah kemungkinan string  $orig1$  dan  $orig2$  yang mungkin.

Sebagai contoh, tabel 2.1 adalah kombinasi dari string  $orig1$  dan  $orig2$  dengan string  $ad1 = c$ , string  $ad2 = n$  dan  $X = 1$ . Jawaban dari kasus tersebut adalah 4 karena terdapat 4 kombinasi dengan  $X = 1$ . Contoh berikutnya adalah kasus ketika  $ad1 = bd$ , string  $ad2 = gj$  dan  $X = 5$ . Berdasarkan daftar kombinasi string  $orig1$  dan  $orig2$  pada tabel 2.2, terdapat 8 kombinasi string  $orig1$  dan string  $orig2$  yang memenuhi kriteria  $X = 8$  sehingga jawaban akhir dari kasus tersebut adalah 8.

## 2.2 Deskripsi Umum

Pada subbab ini akan dijelaskan mengenai deskripsi-deskripsi umum yang terdapat pada Tugas Akhir ini.

### 2.2.1 String

Pada dunia ilmu komputer, string didefinisikan sebagai sebuah rangkaian karakter. String pada umumnya dipahami sebagai sebuah struktur data dan diimplementasi menggunakan struktur data array[1].

orig1	orig2	X	Validitas
ad	gj	1	Tidak valid
cd	gj	1	Tidak valid
bc	gj	1	Tidak valid
be	gj	1	Tidak valid
bd	fj	1	Tidak valid
bd	hj	1	Tidak valid
bd	gi	1	Tidak valid
bd	gk	1	Tidak valid
ad	jg	7	Tidak valid
cd	jg	7	Tidak valid
bc	jg	7	Tidak valid
be	jg	7	Tidak valid
bd	ig	5	Valid
bd	kg	7	Tidak valid
bd	jf	7	Tidak valid
bd	jh	5	Valid
cb	gj	3	Tidak valid
eb	gj	5	Valid
da	gj	5	Valid
dc	gj	3	Tidak valid
db	fj	5	Valid
db	hj	5	Valid
db	gi	5	Valid
db	gk	5	Valid
cb	jg	9	Tidak valid
eb	jg	11	Tidak valid
da	jg	11	Tidak valid
dc	jg	9	Tidak valid
db	ig	9	Tidak valid
db	kg	11	Tidak valid
db	jf	11	Tidak valid
db	jh	9	Tidak valid

Tabel 2.2 Kombinasi string  $orig1$  dan  $orig2$  dengan  $ad1 = bd$ , string  $ad2 = gj$  dan  $X = 5$

### 2.2.2 Rekurens

Ketika sebuah algoritma mengandung sebuah persamaan rekursif yang memanggil dirinya sendiri, waktu prosesnya dapat dikatakan sebagai rekurens. **Rekurens** adalah sebuah persamaan atau pertidaksamaan yang mendeskripsikan sebuah fungsi dalam hal nilai pada masukan yang lebih kecil[2].

### 2.2.3 *Divide and Conquer*

Dalam ilmu komputer, *divide and conquer* (D&C) adalah paradigma perancangan algoritma yang bekerja dengan memecah permasalahan menjadi dua atau lebih submasalah dengan karakteristik yang sama atau berkaitan hingga cukup sederhana untuk diselesaikan secara langsung. Solusi dari masing-masing submasalah akan dikombinasikan untuk mendapatkan solusi dari permasalahan utama. Pada umumnya *divide and conquer* (D&C) merujuk pada aplikasi algoritma yang mereduksi setiap permasalahan menjadi hanya satu submasalah[3].

### 2.2.4 *Meet In The Middle*

Dalam dunia pemrograman komputer, meet in the middle adalah sebuah teknik pencarian dua arah dengan membagi dua permasalahan, lalu menyelesaiakannya secara terpisah, lalu menggabungkan keduanya untuk mendapatkan hasil yang diinginkan[3].

### 2.2.5 *Dynamic Programming*

Dalam dunia ilmu komputer, *dynamic programming* adalah sebuah metode penyelesaian masalah yang memecah sebuah permasalahan yang rumit menjadi submasalah-submasalah yang lebih sederhana. *dynamic programming* bersifat efektif ketika submasalah dari permasalahan yang diberikan mungkin berasal dari lebih dari satu pilihan. Teknik kunci dari *dynamic programming* adalah menyimp-

an solusi untuk setiap submasalah untuk digunakan jika submasalah tersebut muncul kembali[2].

### **2.2.6 State**

*State* atau *state variable* adalah himpunan variabel parameter dari sebuah submasalah dari permasalahan yang diberikan[4].

### **2.2.7 Bitmask**

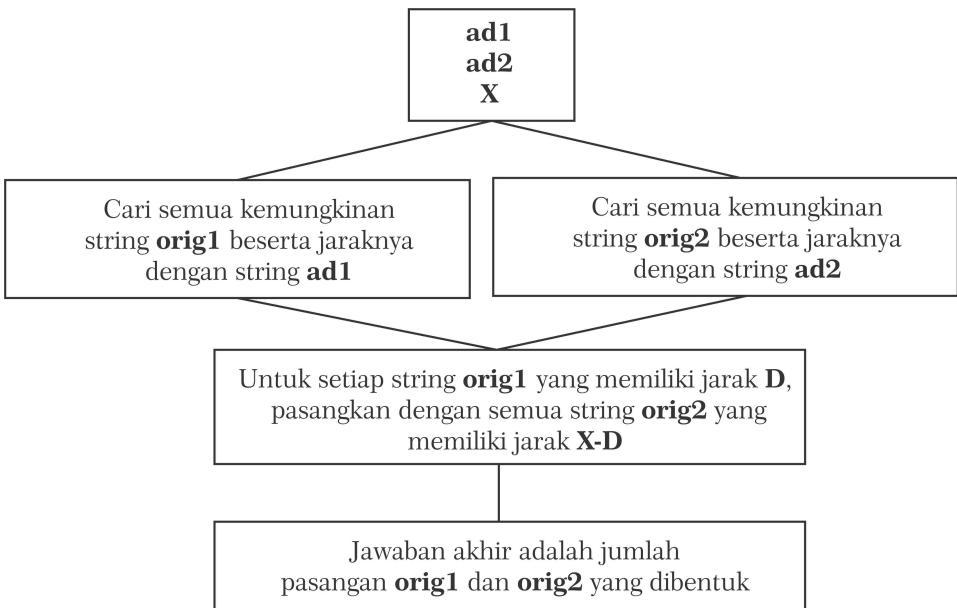
Bitmask adalah sebuah bilangan bulat yang disimpan dan direpresentasikan sebagai himpunan dari nilai *boolean*. Salah satu contoh pemanfaatan teknik *bitmasking* adalah penggunaan bitmask sebagai salah satu index pada tabel memo pada teknik *dynamic programming*[3].

## **2.3 Analisa Submasalah Optimal**

Pada subbab ini akan dijelaskan mengenai submasalah-submasalah yang jawabannya dapat membangun jawaban akhir. Tujuan utama dari permasalahan yang diberikan adalah untuk mencari jumlah kemungkinan string *orig1* dan *orig2* dari string *ad1* dan *ad2* yang memiliki jarak  $dist(orig1, ad1) + dist(orig2, ad2) = X$  yang berikutnya disebut dengan jawaban akhir.

### **2.3.1 Membagi Permasalah Menjadi Submasalah yang *Independent***

Pada permasalahan klasik SPOJ 9967 *Playing With Words*, jawaban akhir merupakan banyak kombinasi string *orig1* dan string *orig2* yang mungkin. Dapat dilihat bahwa perhitungan jumlah kombinasi string *orig1* dari string *ad1* dan perhitungan jumlah kombinasi string *orig2* dari string *ad2* tidak memiliki keterkaitan satu sama lain. Artinya adalah kita dapat menghitung jumlah kombinasi string *orig1* dari string *ad1* tanpa mempedulikan kondisi string



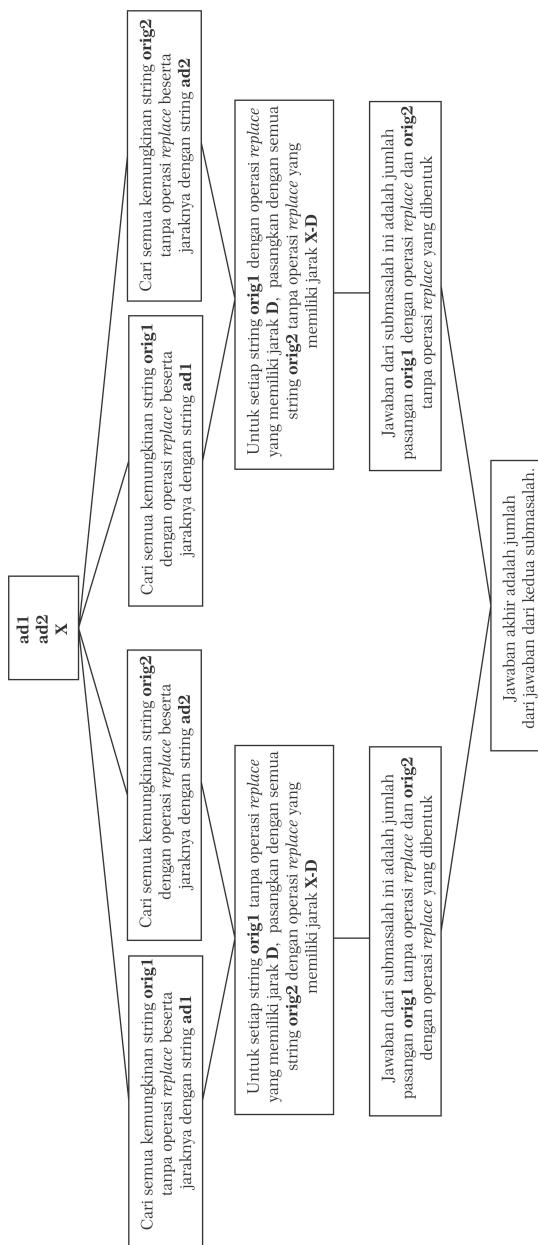
Gambar 2.1 Ilustrasi umum penyelesaian permasalahan dengan metode *meet in the middle* tanpa operasi *replace*

*orig2*. Begitu juga sebaliknya pada perhitungan jumlah kombinasi string *orig2*. Dengan kata lain perhitungan jumlah kombinasi string *orig1* dan *orig2* dapat dilakukan secara terpisah. Teknik tersebut disebut dengan *meet in the middle*.

Apabila operasi *replace* pada permasalahan klasik SPOJ 9967 *Playing With Words* diabaikan, maka jawaban akhir dari dapat dihitung dengan alur secara umum pada gambar 2.1. Karena operasi *replace* diabaikan, maka jawaban akhir dapat dibentuk dengan mencari seluruh kemungkinan pasangan string *orig1* yang memiliki jarak *D* terhadap string *ad1* dengan string *orig2* yang memiliki jarak *X - D* terhadap string *ad2*.

Ketika terdapat operasi *replace*, dimana operasi *replace* adalah operasi dimana salah satu karakter pada string *orig1* atau *orig2* diganti dengan karakter sebelumnya atau sesudahnya secara alfabetis, maka perlu dilakukan penyesuaian pada proses perhitungan jawaban. Selain perhitungan kombinasi string *orig1* dan string *orig2*, perlu juga dilakukan perhitungan untuk kombinasi string *orig1* dan string *orig2* setelah dilakukan satu kali operasi *replace*. Sehingga jawaban akhir dari permasalahan klasik SPOJ 9967 *Playing With Words* adalah total dari jumlah kemungkinan pasangan string *orig1* tanpa operasi *replace* yang memiliki jarak  $D$  terhadap string *ad1* dengan string *orig2* dengan operasi *replace* yang memiliki jarak  $X - D$  terhadap string *ad2* dan jumlah kemungkinan pasangan string *orig1* dengan operasi *replace* yang memiliki jarak  $D$  terhadap string *ad1* dengan string *orig2* tanpa operasi *replace* yang memiliki jarak  $X - D$  terhadap string *ad2*. Gambar 2.2 adalah ilustrasi umum penyelesaian permasalahan klasik SPOJ 9967 *Playing With Words* dengan metode *meet in the middle*.

Sebagai contoh kasus ketika string *orig1* = *bd*, string *orig2* = *gj* dan  $X = 5$ . Langkah pertama untuk menyelesaikan kasus ini adalah dengan mencari semua kemungkinan kombinasi string *orig1* tanpa operasi *replace* dan semua kemungkinan string *orig2* dengan operasi *replace*. Tabel 2.3 menunjukkan seluruh kombinasi string *orig1* tanpa operasi *replace* beserta jarak masing-masing dengan string *ad1* dan tabel 2.4 menunjukkan seluruh kombinasi string *orig2* dengan operasi *replace* beserta jarak masing-masing dengan string *ad2*. Berikutnya adalah mencari seluruh pasangan kombinasi string *orig1* tanpa operasi *replace* dengan jarak terhadap string *ad1* sebesar  $D$  dan kombinasi string *orig2* dengan operasi *replace* dengan jarak terhadap string *ad2* sebesar  $X - D$  yang hasilnya dapat dilihat pada tabel 2.5. Berikutnya hal yang sama juga dilakukan untuk kombinasi string *orig1* dengan operasi *replace* dan string *orig2* tanpa operasi *replace*. Tabel 2.6 menunjukkan seluruh kombinasi string *orig1* dengan operasi *replace* beserta jarak masing-masing



Gambar 2.2 Ilustrasi umum penyelesaian permasalahan dengan metode *meet in the middle* dengan operasi *replace*

<i>orig1</i>	Jarak dengan <i>ad1</i>
<i>bd</i>	0
<i>db</i>	4

Tabel 2.3 Kombinasi string *orig1* dengan nilai string *ad1* = *bd* tanpa operasi *replace*

<i>orig2</i>	Jarak dengan <i>ad2</i>
<i>fj</i>	1
<i>hj</i>	1
<i>gi</i>	1
<i>gk</i>	1
<i>ig</i>	5
<i>kg</i>	7
<i>jf</i>	7
<i>jh</i>	5

Tabel 2.4 Kombinasi string *orig2* dengan nilai string *ad2* = *gj* dengan operasi *replace*

dengan string *ad1* dan tabel 2.7 menunjukkan seluruh kombinasi string *orig2* tanpa operasi *replace* beserta jarak masing-masing dengan string *ad2*. Berikutnya adalah mencari seluruh pasangan kombinasi string *orig1* dengan operasi *replace* dengan jarak terhadap string *ad1* sebesar  $D$  dan kombinasi string *orig2* dengan operasi *replace* tanpa jarak terhadap string *ad2* sebesar  $X - D$  yang hasilnya dapat dilihat pada tabel 2.8. Jawaban akhir dari kasus ketika string *orig1* = *bd*, string *orig2* = *gj* dan  $X = 5$  adalah jumlah dari kombinasi pasangan string *orig1* dan string *orig2* pada tabel 2.5 dan tabel 2.8.

Pada deskripsi permasalahan klasik SPOJ 9967 *Playing With Words* jawaban akhir adalah banyak kemungkinan string *orig1* dan *orig2* tanpa perlu menyertakan daftar string *orig1* dan *orig2* yang mung-

<i>orig1</i>	Jarak dengan <i>ad1</i>	<i>orig2</i>	Jarak dengan <i>ad2</i>
<i>bd</i>	0	<i>ig</i>	5
<i>bd</i>	0	<i>jh</i>	5
<i>db</i>	4	<i>fj</i>	1
<i>db</i>	4	<i>hj</i>	1
<i>db</i>	4	<i>gi</i>	1
<i>db</i>	4	<i>gk</i>	1

Tabel 2.5 Kombinasi string *orig1* dengan nilai string *ad1* = *bd* tanpa operasi *replace* dan string *orig2* dengan nilai string *ad2* = *gj* dengan operasi *replace* dengan  $X = 5$

<i>orig1</i>	Jarak dengan <i>ad1</i>
<i>ad</i>	1
<i>cd</i>	1
<i>bc</i>	1
<i>be</i>	1
<i>cb</i>	3
<i>eb</i>	5
<i>da</i>	5
<i>dc</i>	3

Tabel 2.6 Kombinasi string *orig1* dengan nilai string *ad1* = *bd* dengan operasi *replace*

<i>orig2</i>	Jarak dengan <i>ad2</i>
<i>gj</i>	0
<i>jk</i>	6

Tabel 2.7 Kombinasi string *orig2* dengan nilai string *ad2* = *gj* tanpa operasi *replace*

$orig1$	Jarak dengan $ad1$	$orig2$	Jarak dengan $ad2$
$eb$	5	$gj$	0
$da$	5	$gj$	0

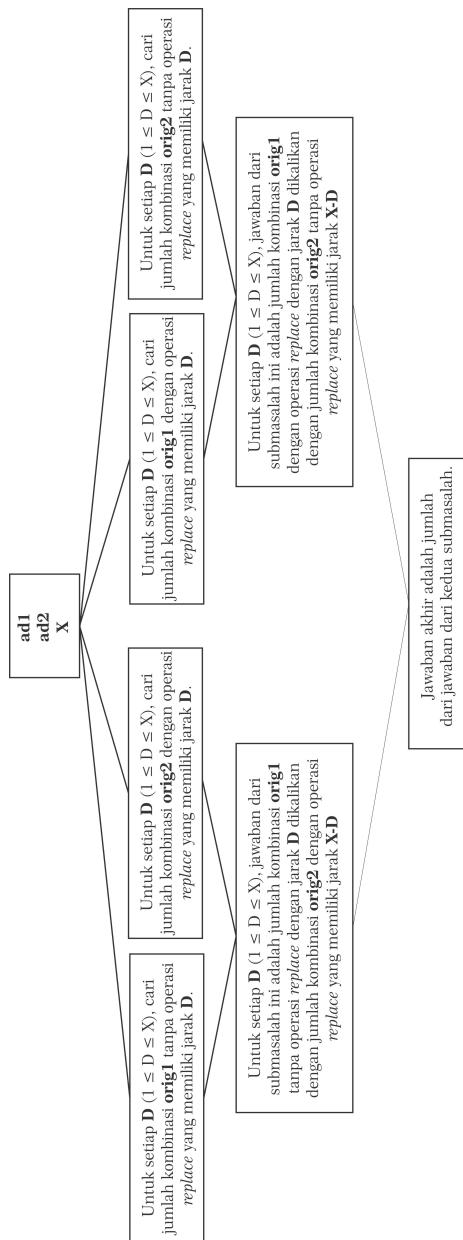
Tabel 2.8 Kombinasi string  $orig1$  dengan nilai string  $ad1 = bd$  dengan operasi *replace* dan string  $orig2$  dengan nilai string  $ad2 = gj$  tanpa operasi *replace* dengan  $X = 5$

kin. Maka dari itu, proses perhitungan jawaban akhir dapat disederhanakan agar algoritma yang dibangun lebih optimal. Seperti yang terlihat pada ilustrasi umum penyelesaian pada gambar 2.3, proses pencarian kombinasi string  $orig1$  dan  $orig2$  dapat diganti dengan hanya menghitung jumlah kemungkinan kombinasi string  $orig1$  dengan atau tanpa operasi *replace* dengan jarak terhadap string  $ad1$  sebesar  $D$  dan jumlah kemungkinan kombinasi string  $orig2$  dengan atau tanpa operasi *replace* dengan jarak terhadap string  $ad2$  sebesar  $D$  dengan  $1 \leq D \leq X$ .

### 2.3.2 Submasalah Optimal untuk Menghitung Jumlah Kombinasi String $Orig$ dari String $Ad$ Tanpa Operasi *Replace* dengan Jarak $D$

Ilustrasi pada gambar 2.3 menunjukkan bahwa untuk mendapatkan jawaban akhir dari permasalahan klasik SPOJ 9967 *Playing With Words* salah satu langkah yang harus dilakukan adalah menghitung jumlah kombinasi string  $orig$  dari string  $ad$  tanpa operasi *replace* dengan jarak  $D$ . Terdapat dua kali perhitungan untuk proses ini yaitu perhitungan untuk mencari jumlah kombinasi string  $orig1$  dan string  $orig2$  tanpa operasi *replace* dengan jarak  $D$ .

Perhitungan jumlah kombinasi string  $orig$  tanpa operasi *replace* dapat dilakukan dengan memanfaatkan teknik *bitmasking*. Gambar 2.4 adalah ilustrasi perhitungan kombinasi string  $orig$  dari string  $ad$  dengan nilai string  $ad = bcd$  dan  $D = 4$  tanpa operasi *repla-*



Gambar 2.3 Ilustrasi umum penyelesaian permasalahan dengan metode *meet in the middle* dengan operasi *replace* tanpa mempedulikan kombinasi string yang dihasilkan

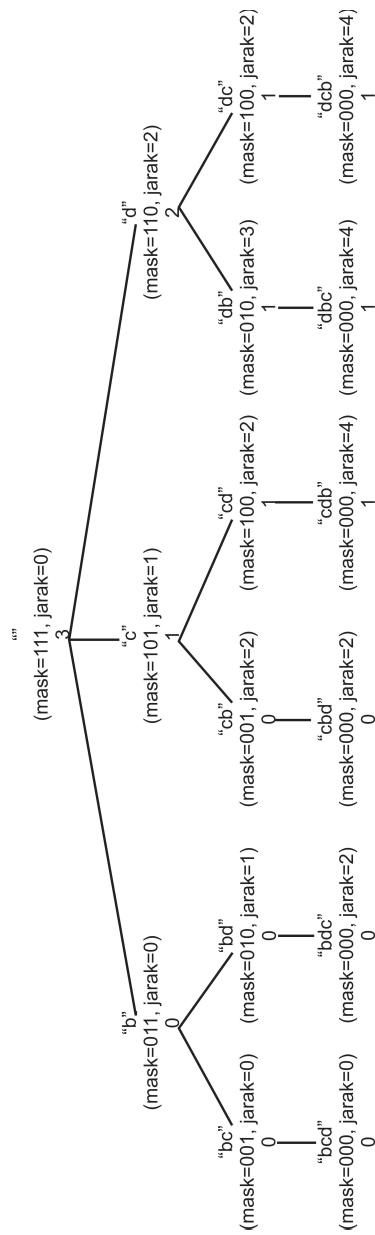
ce. Nilai dari suatu *state* adalah jumlah dari nilai seluruh *state* yang merupakan *child* dari *state* tersebut dengan kasus dasar pada *state* dengan *mask* = 0, apabila *jarak* = *D* maka *state* tersebut akan bernilai 1, jika tidak maka akan bernilai 0.

Namun, seperti yang dapat dilihat pada gambar 2.5, terdapat beberapa kasus dimana pada suatu *state*, *state* dengan nilai tersebut bersifat tumpang tindih dengan suatu *state* lain. Pada gambar 2.5, *state* yang saling tumpang tindih ditandai dengan tulisan berwarna merah. Dengan adanya kasus *state* yang tumpang tindih tersebut dapat dimanfaatkan untuk melakukan optimasi pada algoritma yang dirancang dengan tidak melakukan perhitungan ulang pada *state* yang sudah pernah muncul sebelumnya. Sehingga proses perhitungan yang sebelumnya seperti dengan ilustrasi pada gambar 2.4 dapat disederhanakan menjadi seperti yang terdapat pada gambar 2.6. Teknik tersebut dikenal dengan teknik *dynamic programming*.

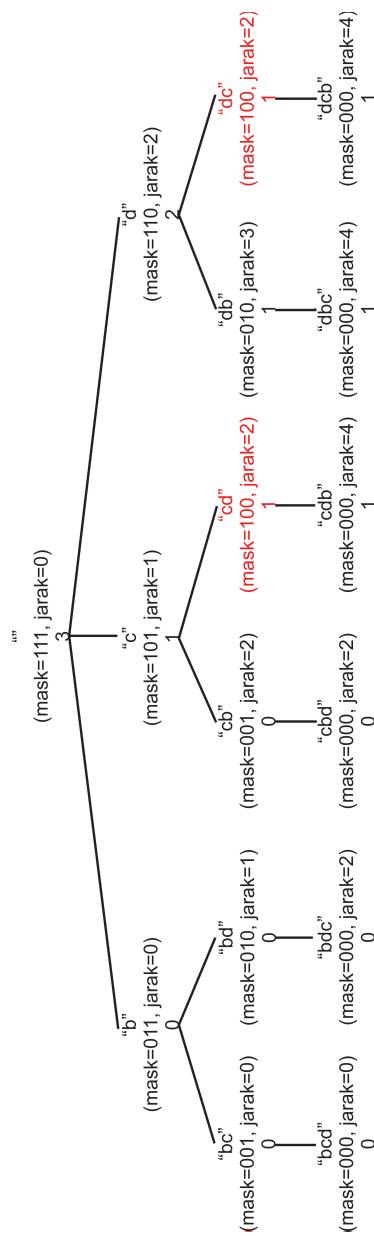
### 2.3.3 Submasalah Optimal untuk Menghitung Jumlah Kombinasi String *Orig* dari String *Ad* dengan Operasi *Replace* dengan Jarak *D*

Ilustrasi pada gambar 2.3 menunjukkan bahwa untuk mendapatkan jawaban akhir dari permasalahan klasik SPOJ 9967 *Playing With Words* salah satu langkah yang harus dilakukan adalah menghitung jumlah kombinasi string *orig* dari string *ad* dengan operasi *replace* dengan jarak *D*. Terdapat dua kali perhitungan untuk proses ini yaitu perhitungan untuk mencari jumlah kombinasi string *orig1* dan string *orig2* tanpa operasi *replace* dengan jarak *D*.

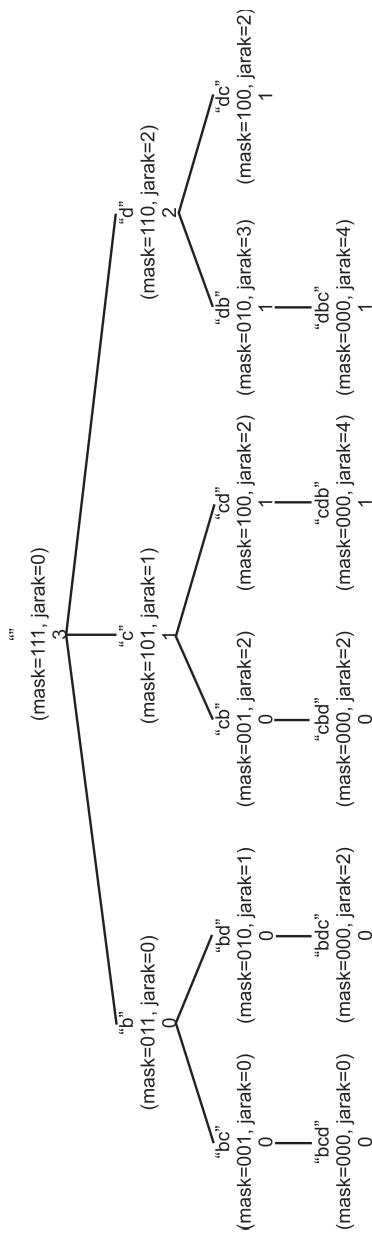
Cara untuk mendapatkan jawaban dari submasalah ini hampir sama dengan cara mencari jawaban pada submasalah perhitungan jumlah kombinasi string *orig* dari string *ad* tanpa operasi *replace*. Gambar 2.7 merupakan ilustrasi dari penyelesaian submasalah perhitungan jumlah kombinasi string *orig* dari string *ad* dengan operasi *replace* dengan nilai *ad* = *be* dan *D* = 1. Sedikit berbeda dengan penye-



Gambar 2.4 Ilustrasi perhitungan jumlah kombinasi string *orig* dari string *ad* tanpa operasi *replace* dengan nilai string *ad* = *bcd* dan *D* = 4



Gambar 2.5 Contoh kasus tumpang tindih pada perhitungan kombinasi string *orig* dari string *ad* tanpa operasi *replace* dengan nilai string *ad* = *bcd* dan *D* = 4



Gambar 2.6 Ilustrasi perhitungan jumlah kombinasi string *orig* dari string *ad* tanpa operasi *replace* dengan nilai string *ad* = *bcd* dan *D* = 4 tanpa menghitung kasus yang tumpang tindih

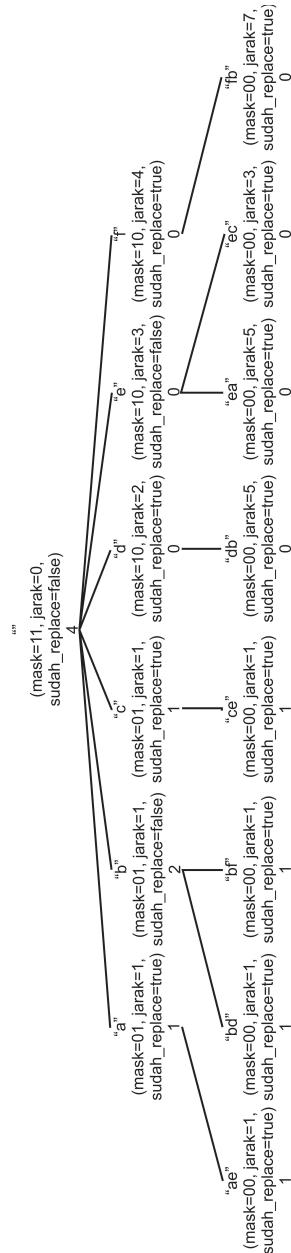
lesaian submasalah perhitungan jumlah kombinasi string *orig* dari string *ad* tanpa operasi *replace*, pada penyelesaian submasalah ini terdapat satu parameter lagi pada setiap *state*, yaitu penanda bahwa pada *state* tersebut sudah pernah melakukan operasi *replace* atau belum.

Sama seperti pada penyelesaian submasalah perhitungan jumlah kombinasi string *orig* dari string *ad* tanpa operasi *replace*, pada submasalah perhitungan jumlah kombinasi string *orig* dari string *ad* dengan operasi *replace* juga memiliki *state* yang saling tumpang tindih seperti yang terlihat pada gambar 2.8 sehingga dapat dilakukan optimasi menggunakan teknik *dynamic programming* untuk meningkatkan efisiensi algoritma yang dibangun. Pada gambar 2.9 dapat dilihat bahwa terdapat beberapa *state* yang ternyata memiliki kondisi yang mampu diselesaikan dengan metode yang sama dengan metode penyelesaian submasalah perhitungan jumlah kombinasi string *orig* dari string *ad* tanpa operasi *replace*. Sehingga penyelesaian submasalah perhitungan jumlah kombinasi string *orig* dari string *ad* dengan operasi *replace* dapat disederhanakan dengan memanfaatkan penyelesaian submasalah perhitungan jumlah kombinasi string *orig* dari string *ad* tanpa operasi *replace*.

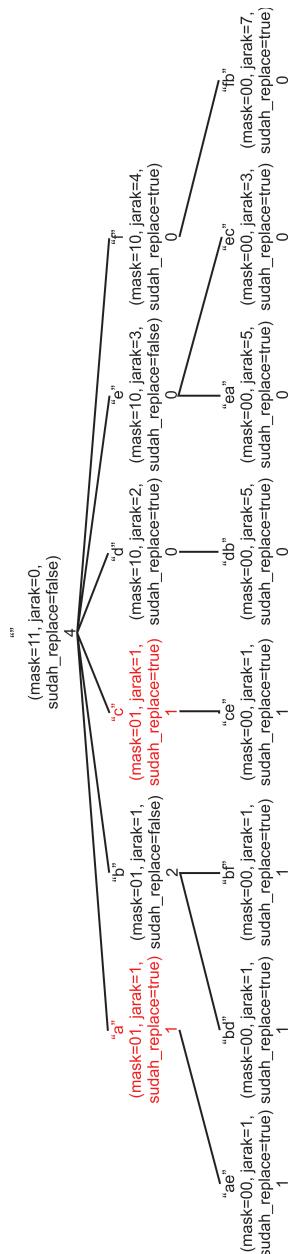
## 2.4 Pemodelan Relasi Rekurens

$$\begin{aligned} \text{answer} = \sum_{dist=0}^{\min(X, 250)} & ((F_{(S_0, 2^{|S_0|}, bound-dist)} \\ & G_{(S_1, 2^{|S_1|}, bound-X+dist)}) * \\ & F_{(S_1, 2^{|S_1|}, bound-X+dist)}) + (G_{(S_0, 2^{|S_0|}, bound-dist)} \\ & F_{(S_1, 2^{|S_1|}, bound-X+dist)}) \end{aligned} \quad (2.4.1)$$

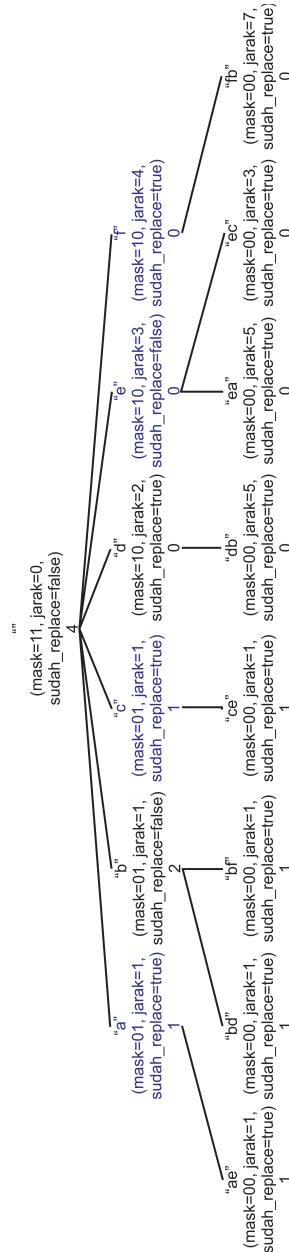
Pada subbab ini akan dijelaskan tentang relasi rekurens berdasarkan analisis pada subbab 2.3. Pada subbagian 2.3.1, dijelaskan bahwa permasalahan dapat dipecah menjadi dua submasalah yang dapat diselesaikan tanpa bergantung satu sama lain demgam memecahkan permasalahan berdasarkan masing-masing string *ad*. Karena terdapat



Gambar 2.7 Ilustrasi perhitungan jumlah kombinasi string *orig* dari string *ad* dengan operasi *replace* dengan nilai string *ad* = *be* dan *D* = 1



Gambar 2.8 Contoh kasus tumpang tindih pada perhitungan kombinasi string *orig* dari string *ad* dengan operasi *replace* dengan nilai string *ad* = *be* dan *D* = 1



Gambar 2.9 Submasalah perhitungan jumlah kombinasi string *orig* terhadap string *ad* tanpa operasi *replace* pada submasalah perhitungan jumlah kombinasi string *orig* terhadap string *ad* dengan operasi *replace*

sebuah operasi *replace* yang dilakukan, maka untuk menyelesaikan masing-masing submasalah harus dilakukan dua jenis perhitungan, yaitu operasi perhitungan jumlah kemungkinan string *orig* tanpa operasi *replace* dan operasi perhitungan jumlah kemungkinan string *orig* dengan operasi *replace*. Kedua operasi tersebut didefinisikan dalam bentuk fungsi sebagai berikut:

1.  $F_{(S,mask,dist)}$ , yaitu fungsi untuk menghitung jumlah kemungkinan string awal dari string *S* tanpa operasi *replace* dimana *S* adalah string awal yang akan dihitung, *mask* adalah nilai *bitmask* dan *dist* adalah jarak string awal dengan string yang dibentuk pada *state* tersebut.
2.  $G_{(S,mask,dist)}$ , yaitu fungsi untuk menghitung jumlah kemungkinan string awal dari string *S* dengan sekali operasi *replace* dimana *S* adalah string awal yang akan dihitung, *mask* adalah nilai *bitmask* dan *dist* adalah jarak string awal dengan string yang dibentuk pada *state* tersebut.

Jawaban permasalahan klasik SPOJ 9967 *Playing With Words* dapat dihitung dengan memanfaatkan kedua fungsi di atas. Persamaan 2.4.1 merupakan persamaan untuk menghitung jawaban utama dari permasalahan klasik SPOJ 9967 *Playing With Words* dimana  $S_0$  adalah string *ad1*,  $S_1$  adalah string *ad2*,  $X$  adalah jumlah jarak(*ad1*, *orig1*) dengan jarak(*ad2*, *orig2*) dan  $bound = \min(250, X)$ .

### 2.4.1 Pemodelan Relasi Rekurens Submasalah Optimal untuk Menghitung Jumlah Kemungkinan String Awal Tanpa Operasi *Replace* dengan Jarak $X - dist$

$$F_{(S,mask,dist)} = \begin{cases} 0, & \text{if } dist > bound, \\ & \text{or } (mask = 0 \text{ and } dist \neq bound) \\ 1, & \text{if } mask = 0 \text{ and } dist = bound \\ \sum_{i=0}^{i=NSB_{(mask)}} F1_{(S,mask,\text{set\_bit}(mask)_i,dist)}, & \text{otherwise} \end{cases} \quad (2.4.2)$$

$$F1_{(S,mask,idx,dist)} = \begin{cases} F_{(S,mask-2^{idx},dist+|S_{idx}-S_{curIdx}|)}, & \text{idx } = |S| - 1 \text{ or } \\ & \text{duplicate\_rule1}_{(S,mask,idx)} = \\ 0, & \text{otherwise} \end{cases} \quad (2.4.3)$$

$$\text{duplicate\_rule1}_{(S,mask,idx)} = \begin{cases} True, & \text{if } idx < |S| - 1 \text{ and } \\ & (S_{idx} \neq S_{idx+1} \\ & \text{or } ((S_{idx} = S_{idx+1}) \text{ and } \\ & (is\_off_{(mask,idx+1)}))) \\ False, & \text{otherwise} \end{cases} \quad (2.4.4)$$

Pada persamaan 2.4.1 terdapat fungsi  $F_{(S,mask,dist)}$  yang merupakan fungsi untuk menghitung jumlah kemungkinan string *orig* dari string *S* tanpa operasi *replace* dengan jarak  $X - dist$ . Nilai dari fungsi  $F_{(S,mask,dist)}$  adalah hasil penjumlahan seluruh *state* yang

berhubungan, yaitu  $state F_{(S,mask-2^{idx},dist+|S_{idx}-S_{curIdx}|)}$  dimana  $curIdx$  adalah jumlah bit tidak menyala pada  $mask$  dan  $idx$  adalah  $set\_bit(mask)_i$  untuk setiap  $i$  dimana  $0 \leq i \leq NSB_{mask}$  dengan  $set\_bit(mask)$  adalah Himpunan index bit menyala pada  $mask$  dan  $NSB_{mask}$  adalah jumlah bit menyala pada  $mask$ . Tidak semua  $state F_{(S,mask-2^{idx},dist+|S_{idx}-S_{curIdx}|)}$  dijumlahkan untuk mendapatkan nilai dari fungsi  $F_{(S,mask,d)}$ . Hanya  $state$  yang valid yang nilainya dijumlahkan untuk membentuk nilai dari fungsi  $F_{(S,mask,d)}$ . Persamaan 2.4.3 adalah persamaan rekurens untuk menentukan apakah  $state F_{(S,mask-2^{idx},dist+|S_{idx}-S_{curIdx}|)}$  merupakan  $state$  yang valid dari sebuah  $state F_{(S,mask,d)}$  dimana  $is\_odd_{(mask, idx)}$  bernilai  $True$  jika  $mask \& 2^{idx} = 0$ . Persamaan 2.4.2 adalah relasi rekurens dari submasalah perhitungan jumlah kemungkinan string  $orig$  dari string  $S$  tanpa operasi  $replace$  dengan jarak  $X - dist$ .

## 2.4.2 Pemodelan Relasi Rekurens Submasalah Optimal untuk Menghitung Jumlah Kemungkinan String Awal dengan Sekali Operasi *Replace* dengan Jarak $X - dist$

$$G_{(S,mask,dist)} = \begin{cases} 0, & dist > bound \text{ or} \\ & mask = bound \\ \sum_{i=0}^{i=NSB_{(mask)}} (G1_{(S,mask,} \\ & set\_bit_{(mask)_i,dist)}) \\ + G2_{(S,mask,} \\ & set\_bit_{(mask)_i,dist)}) \\ + G3_{(S,mask,} \\ & set\_bit_{(mask)_i,dist)}), & \text{otherwise} \end{cases} \quad (2.4.5)$$

Pada persamaan 2.4.1 terdapat fungsi  $G_{(S,mask,dist)}$  yang merupakan fungsi untuk menghitung jumlah kemungkinan string  $orig$  dari string  $S$  dengan sekali operasi  $replace$  dengan jarak  $X - dist$ . Sama

halnya dengan fungsi  $F_{(S,mask,dist)}$ , nilai dari fungsi  $G_{(S,mask,dist)}$  adalah hasil penjumlahan dari seluruh *state* yang berhubungan dan valid. Terdapat tiga kasus *state* yang mungkin, yaitu:

1. *State*  $G_{(S,mask-2^{idx},dist+|S_{idx}-S_{curIdx}|)}$  dengan kasus ketika mengambil karakter posisi  $idx$  pada string  $S$  sebagai karakter posisi  $curIdx$  pada string  $orig$  tanpa melakukan *replace*.
2. *State*  $F_{(S,mask-2^{idx},dist+|S_{idx}+1-S_{curIdx}|)}$  dengan kasus ketika mengambil karakter posisi  $idx$  pada string  $S$  sebagai karakter posisi  $curIdx$  pada string  $orig$  dengan melakukan *replace* dengan karakter setelahnya secara alfabetis.
3. *State*  $F_{(S,mask-2^{idx},dist+|S_{idx}-1-S_{curIdx}|)}$  dengan kasus ketika mengambil karakter posisi  $idx$  pada string  $S$  sebagai karakter posisi  $curIdx$  pada string  $orig$  dengan melakukan *replace* dengan karakter sebelumnya secara alfabetis.

Masing - masing jenis *state* yang berhubungan langsung dengan *state*  $G_{(S,mask,dist)}$  memiliki syarat tersendiri untuk menjadi sebuah *state* yang valid. Berikut adalah syarat dari masing - masing jenis *state* yang dapat dibentuk dari *state*  $G_{(S,mask,dist)}$ :

1. Persamaan 2.4.6 adalah persamaan yang menentukan apakah *state*  $G_{(S,mask-2^{idx},dist+|S_{idx}-S_{curIdx}|)}$  merupakan sebuah *state* yang valid dari *state*  $G_{(S,mask,dist)}$ .
2. Persamaan 2.4.7 adalah persamaan yang menentukan apakah *state*  $F_{(S,mask-2^{idx},dist+|S_{idx}+1-S_{curIdx}|)}$  merupakan sebuah *state* yang valid dari *state*  $G_{(S,mask,dist)}$  dengan  $charFirstPos_{(S,C)}$  adalah posisi pertama karakter  $C$  pada string  $S$ . Apabila karakter  $C$  tidak ada pada string  $S$  maka akan bernilai  $-1$ .
3. Persamaan 2.4.8 adalah persamaan yang menentukan apakah *state*  $F_{(S,mask-2^{idx},dist+|S_{idx}-1-S_{curIdx}|)}$  merupakan sebuah *state* yang valid dari *state*  $G_{(S,mask,dist)}$  dengan  $charLastPos_{(S,C)}$  adalah posisi terakhir karakter  $C$  pada string  $S$ . Apabila karakter  $C$  tidak ada pada string  $S$  maka

an akan bernilai  $-1$ .

$$G1_{(S,mask,idx,dist)} = \begin{cases} G_{(S,mask \\ -2^{idx},dist \\ +|S_{idx} \\ - S_{curIdx}|)}, & idx = |S| - 1 \text{ or} \\ & duplicate\_rule1_{(S,mask,idx)} = \\ & True \\ 0, & \text{otherwise} \end{cases} \quad (2.4.6)$$

$$G2_{(S,mask,idx,dist)} = \begin{cases} F_{(S,mask \\ -2^{idx},dist \\ +|S_{idx}+1 \\ - S_{curIdx}|)}, & idx = |S| - 1 \text{ or} \\ & (duplicate\_rule1_{(S,mask,idx)} = \\ & True \text{ and} \\ & duplicate\_rule2_{(S,mask,idx)} = \\ & True) \\ 0, & \text{otherwise} \end{cases} \quad (2.4.7)$$

$$G3_{(S,mask,idx,dist)} = \begin{cases} F_{(S,mask \\ -2^{idx},dist \\ +|S_{idx}-1 \\ -S_{currIdx}|)}, & (idx = |S| - 1 \text{ or} \\ & \text{duplicate\_rule1}_{(S,mask,idx)} = \\ & \text{True}) \text{ and } (idx = 0 \text{ or} \\ & \text{duplicate\_rule3}_{(S,mask,idx)} = \\ & \text{True}) \\ 0, & \text{otherwise} \end{cases} \quad (2.4.8)$$

$$\text{duplicate\_rule2}_{(S,mask,idx)} = \begin{cases} \text{True}, & \text{if } idx < |S| - 1 \text{ and} \\ & (\text{charFirstPos}_{(S,S_{idx}+1)} = \\ & -1 \quad \quad \quad \text{or} \\ & (\text{charFirstPos}_{(S,S_{idx}+1)} \neq \\ & -1 \quad \quad \quad \text{and} \\ & \text{is\_off}_{(mask,} \\ & \text{charFirstPos}_{(S,S_{idx}+1)}) \\ \text{False}, & \text{otherwise} \end{cases} \quad (2.4.9)$$

$$duplicate\_rule3(S, mask, idx) = \begin{cases} True, & \text{if } idx > 0 - 1 \text{ and} \\ & (charLastPos(S, S_{idx-1}) = -1 \text{ or} \\ & (charLastPos(S, S_{idx-1}) \neq -1 \text{ and} \\ & is\_on_{(mask,)} \\ & charFirstPos(S, S_{idx+1})) \\ False, & \text{otherwise} \end{cases} \quad (2.4.10)$$

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

*Halaman ini sengaja dikosongkan*

## BAB III

## DESAIN

Pada bab ini akan dibahas tentang desain algoritma untuk menyelesaikan permasalahan klasik SPOJ 9967 *Playing With Words*.

### 3.1 Desain Umum Sistem

Pada subbab ini akan dijelaskan mengenai gambaran secara umum dari algoritma yang dirancang.

Program akan diawali dengan melakukan *preprocess* lalu dilanjutkan dengan menerima masukan berupa banyak data uji. Untuk setiap data uji berupa sebuah baris yang terdiri dari tiga data masukan yang dipisahkan oleh sebuah spasi, yaitu string  $ad1$ , string  $ad2$  dan bilangan bulat  $X$ . String  $ad1$  dan  $ad2$  adalah string hasil enkripsi susai dengan deskripsi permasalahan klasik SPOJ 9967 *Playing With Words* dan  $X = dist(ad1, orig1) + dist(ad2, orig2)$  dimana  $dist(st1, st2)$  adalah total jarak absolut masing - masing karakter  $st1$  dan  $st2$  pada posisi yang sama. Setelah menerima masukan, maka masukan tersebut diolah dan hasilnya ditampilkan di layar. Secara garis besar seperti yang terlihat pada Gambar 3.1.

### 3.2 Desain Fungsi Preprocess

Fungsi *preprocess* merupakan fungsi yang bertujuan agar algoritma yang menyelesaikan permasalahan dapat berjalan dengan benar dan efisien. Pada fungsi ini akan dilakukan perhitungan daftar bit yang bernilai 1 pada setiap bilangan bulat dengan konstanta rentang bilangan yang telah ditentukan. Konstanta rentang bilangan yang digunakan adalah 0 hingga  $2^{10}$  dimana 10 merupakan panjang maksimal string  $ad1$  dan  $ad2$  yang mungkin. Tabel 3.1 sampai tabel

```

Main()
1 preprocess()
2  $TC = \text{Input}()$ 
3 for  $T = 0$  to  $TC - 1$ 
4     readInput()
5     init()
6     solveProblem()
7     writeOutput()

```

Gambar 3.1 Pseudocode Fungsi Main

```

preprocess()
1 for  $num = 0$  to  $2^{10} - 1$ 
2     for  $bitPos = 0$  to  $10 - 1$ 
3         if  $\text{isBitOn}(num, bitPos)$ 
4              $setBit_{(powerNum)}.\text{push}(bitPos)$ 

```

Gambar 3.2 Pseudocode Fungsi Preprocess

3.35 adalah nilai dari himpunan  $setBit$  setalah fungsi preprocess dijalankan. Gambar 3.2 adalah pseudocode untuk fungsi *preprocess*.

### 3.3 Desain Fungsi Init

Fungsi init merupakan fungsi yang bertujuan untuk melakukan initialisasi nilai awal dan perhitungan data - data yang diperlukan untuk menyelesaikan permasalahan klasik SPOJ 9967 *Playing With Words* untuk setiap kasus uji.

Karena algoritma yang dibangun menggunakan pendekatan para-

<i>Num</i>	<i>index</i>	0	1	2	3	4	5	6	7	8	9
0	-	-	-	-	-	-	-	-	-	-	-
1	0	-	-	-	-	-	-	-	-	-	-
2	1	-	-	-	-	-	-	-	-	-	-
3	0	1	-	-	-	-	-	-	-	-	-
4	2	-	-	-	-	-	-	-	-	-	-
5	0	2	-	-	-	-	-	-	-	-	-
6	1	2	-	-	-	-	-	-	-	-	-
7	0	1	2	-	-	-	-	-	-	-	-
8	3	-	-	-	-	-	-	-	-	-	-
9	0	3	-	-	-	-	-	-	-	-	-
10	1	3	-	-	-	-	-	-	-	-	-
11	0	1	3	-	-	-	-	-	-	-	-
12	2	3	-	-	-	-	-	-	-	-	-
13	0	2	3	-	-	-	-	-	-	-	-
14	1	2	3	-	-	-	-	-	-	-	-
15	0	1	2	3	-	-	-	-	-	-	-
16	4	-	-	-	-	-	-	-	-	-	-
17	0	4	-	-	-	-	-	-	-	-	-
18	1	4	-	-	-	-	-	-	-	-	-
19	0	1	4	-	-	-	-	-	-	-	-
20	2	4	-	-	-	-	-	-	-	-	-
21	0	2	4	-	-	-	-	-	-	-	-
22	1	2	4	-	-	-	-	-	-	-	-
23	0	1	2	4	-	-	-	-	-	-	-
24	3	4	-	-	-	-	-	-	-	-	-
25	0	3	4	-	-	-	-	-	-	-	-
26	1	3	4	-	-	-	-	-	-	-	-
27	0	1	3	4	-	-	-	-	-	-	-
28	2	3	4	-	-	-	-	-	-	-	-
29	0	2	3	4	-	-	-	-	-	-	-

Tabel 3.1 Tabel himpunan setBit setelah fungsi preprocess dijalankan (1)

<i>Num</i>	<i>index</i>	0	1	2	3	4	5	6	7	8	9
30		1	2	3	4	-	-	-	-	-	-
31		0	1	2	3	4	-	-	-	-	-
32		5	-	-	-	-	-	-	-	-	-
33		0	5	-	-	-	-	-	-	-	-
34		1	5	-	-	-	-	-	-	-	-
35		0	1	5	-	-	-	-	-	-	-
36		2	5	-	-	-	-	-	-	-	-
37		0	2	5	-	-	-	-	-	-	-
38		1	2	5	-	-	-	-	-	-	-
39		0	1	2	5	-	-	-	-	-	-
40		3	5	-	-	-	-	-	-	-	-
41		0	3	5	-	-	-	-	-	-	-
42		1	3	5	-	-	-	-	-	-	-
43		0	1	3	5	-	-	-	-	-	-
44		2	3	5	-	-	-	-	-	-	-
45		0	2	3	5	-	-	-	-	-	-
46		1	2	3	5	-	-	-	-	-	-
47		0	1	2	3	5	-	-	-	-	-
48		4	5	-	-	-	-	-	-	-	-
49		0	4	5	-	-	-	-	-	-	-
50		1	4	5	-	-	-	-	-	-	-
51		0	1	4	5	-	-	-	-	-	-
52		2	4	5	-	-	-	-	-	-	-
53		0	2	4	5	-	-	-	-	-	-
54		1	2	4	5	-	-	-	-	-	-
55		0	1	2	4	5	-	-	-	-	-
56		3	4	5	-	-	-	-	-	-	-
57		0	3	4	5	-	-	-	-	-	-
58		1	3	4	5	-	-	-	-	-	-
59		0	1	3	4	5	-	-	-	-	-

Tabel 3.2 Tabel himpunan setBit setelah fungsi preprocess dijalankan (2)

<i>index</i> <i>Num</i>	0	1	2	3	4	5	6	7	8	9
60	2	3	4	5	-	-	-	-	-	-
61	0	2	3	4	5	-	-	-	-	-
62	1	2	3	4	5	-	-	-	-	-
63	0	1	2	3	4	5	-	-	-	-
64	6	-	-	-	-	-	-	-	-	-
65	0	6	-	-	-	-	-	-	-	-
66	1	6	-	-	-	-	-	-	-	-
67	0	1	6	-	-	-	-	-	-	-
68	2	6	-	-	-	-	-	-	-	-
69	0	2	6	-	-	-	-	-	-	-
70	1	2	6	-	-	-	-	-	-	-
71	0	1	2	6	-	-	-	-	-	-
72	3	6	-	-	-	-	-	-	-	-
73	0	3	6	-	-	-	-	-	-	-
74	1	3	6	-	-	-	-	-	-	-
75	0	1	3	6	-	-	-	-	-	-
76	2	3	6	-	-	-	-	-	-	-
77	0	2	3	6	-	-	-	-	-	-
78	1	2	3	6	-	-	-	-	-	-
79	0	1	2	3	6	-	-	-	-	-
80	4	6	-	-	-	-	-	-	-	-
81	0	4	6	-	-	-	-	-	-	-
82	1	4	6	-	-	-	-	-	-	-
83	0	1	4	6	-	-	-	-	-	-
84	2	4	6	-	-	-	-	-	-	-
85	0	2	4	6	-	-	-	-	-	-
86	1	2	4	6	-	-	-	-	-	-
87	0	1	2	4	6	-	-	-	-	-
88	3	4	6	-	-	-	-	-	-	-
89	0	3	4	6	-	-	-	-	-	-

Tabel 3.3 Tabel himpunan setBit setelah fungsi preprocess dijalankan (3)

<i>Num</i>	<i>index</i>	0	1	2	3	4	5	6	7	8	9
90		1	3	4	6	-	-	-	-	-	-
91		0	1	3	4	6	-	-	-	-	-
92		2	3	4	6	-	-	-	-	-	-
93		0	2	3	4	6	-	-	-	-	-
94		1	2	3	4	6	-	-	-	-	-
95		0	1	2	3	4	6	-	-	-	-
96		5	6	-	-	-	-	-	-	-	-
97		0	5	6	-	-	-	-	-	-	-
98		1	5	6	-	-	-	-	-	-	-
99		0	1	5	6	-	-	-	-	-	-
100		2	5	6	-	-	-	-	-	-	-
101		0	2	5	6	-	-	-	-	-	-
102		1	2	5	6	-	-	-	-	-	-
103		0	1	2	5	6	-	-	-	-	-
104		3	5	6	-	-	-	-	-	-	-
105		0	3	5	6	-	-	-	-	-	-
106		1	3	5	6	-	-	-	-	-	-
107		0	1	3	5	6	-	-	-	-	-
108		2	3	5	6	-	-	-	-	-	-
109		0	2	3	5	6	-	-	-	-	-
110		1	2	3	5	6	-	-	-	-	-
111		0	1	2	3	5	6	-	-	-	-
112		4	5	6	-	-	-	-	-	-	-
113		0	4	5	6	-	-	-	-	-	-
114		1	4	5	6	-	-	-	-	-	-
115		0	1	4	5	6	-	-	-	-	-
116		2	4	5	6	-	-	-	-	-	-
117		0	2	4	5	6	-	-	-	-	-
118		1	2	4	5	6	-	-	-	-	-
119		0	1	2	4	5	6	-	-	-	-

Tabel 3.4 Tabel himpunan setBit setelah fungsi preprocess dijalankan (4)

<i>Num</i>	<i>index</i>	0	1	2	3	4	5	6	7	8	9
120		3	4	5	6	-	-	-	-	-	-
121		0	3	4	5	6	-	-	-	-	-
122		1	3	4	5	6	-	-	-	-	-
123		0	1	3	4	5	6	-	-	-	-
124		2	3	4	5	6	-	-	-	-	-
125		0	2	3	4	5	6	-	-	-	-
126		1	2	3	4	5	6	-	-	-	-
127		0	1	2	3	4	5	6	-	-	-
128		7	-	-	-	-	-	-	-	-	-
129		0	7	-	-	-	-	-	-	-	-
130		1	7	-	-	-	-	-	-	-	-
131		0	1	7	-	-	-	-	-	-	-
132		2	7	-	-	-	-	-	-	-	-
133		0	2	7	-	-	-	-	-	-	-
134		1	2	7	-	-	-	-	-	-	-
135		0	1	2	7	-	-	-	-	-	-
136		3	7	-	-	-	-	-	-	-	-
137		0	3	7	-	-	-	-	-	-	-
138		1	3	7	-	-	-	-	-	-	-
139		0	1	3	7	-	-	-	-	-	-
140		2	3	7	-	-	-	-	-	-	-
141		0	2	3	7	-	-	-	-	-	-
142		1	2	3	7	-	-	-	-	-	-
143		0	1	2	3	7	-	-	-	-	-
144		4	7	-	-	-	-	-	-	-	-
145		0	4	7	-	-	-	-	-	-	-
146		1	4	7	-	-	-	-	-	-	-
147		0	1	4	7	-	-	-	-	-	-
148		2	4	7	-	-	-	-	-	-	-
149		0	2	4	7	-	-	-	-	-	-

Tabel 3.5 Tabel himpunan setBit setelah fungsi preprocess dijalankan (5)

<i>Num</i>	<i>index</i>	0	1	2	3	4	5	6	7	8	9
150		1	2	4	7	-	-	-	-	-	-
151		0	1	2	4	7	-	-	-	-	-
152		3	4	7	-	-	-	-	-	-	-
153		0	3	4	7	-	-	-	-	-	-
154		1	3	4	7	-	-	-	-	-	-
155		0	1	3	4	7	-	-	-	-	-
156		2	3	4	7	-	-	-	-	-	-
157		0	2	3	4	7	-	-	-	-	-
158		1	2	3	4	7	-	-	-	-	-
159		0	1	2	3	4	7	-	-	-	-
160		5	7	-	-	-	-	-	-	-	-
161		0	5	7	-	-	-	-	-	-	-
162		1	5	7	-	-	-	-	-	-	-
163		0	1	5	7	-	-	-	-	-	-
164		2	5	7	-	-	-	-	-	-	-
165		0	2	5	7	-	-	-	-	-	-
166		1	2	5	7	-	-	-	-	-	-
167		0	1	2	5	7	-	-	-	-	-
168		3	5	7	-	-	-	-	-	-	-
169		0	3	5	7	-	-	-	-	-	-
170		1	3	5	7	-	-	-	-	-	-
171		0	1	3	5	7	-	-	-	-	-
172		2	3	5	7	-	-	-	-	-	-
173		0	2	3	5	7	-	-	-	-	-
174		1	2	3	5	7	-	-	-	-	-
175		0	1	2	3	5	7	-	-	-	-
176		4	5	7	-	-	-	-	-	-	-
177		0	4	5	7	-	-	-	-	-	-
178		1	4	5	7	-	-	-	-	-	-
179		0	1	4	5	7	-	-	-	-	-

Tabel 3.6 Tabel himpunan setBit setelah fungsi preprocess dijalankan (6)

<i>Num</i>	<i>index</i>	0	1	2	3	4	5	6	7	8	9
180		2	4	5	7	-	-	-	-	-	-
181		0	2	4	5	7	-	-	-	-	-
182		1	2	4	5	7	-	-	-	-	-
183		0	1	2	4	5	7	-	-	-	-
184		3	4	5	7	-	-	-	-	-	-
185		0	3	4	5	7	-	-	-	-	-
186		1	3	4	5	7	-	-	-	-	-
187		0	1	3	4	5	7	-	-	-	-
188		2	3	4	5	7	-	-	-	-	-
189		0	2	3	4	5	7	-	-	-	-
190		1	2	3	4	5	7	-	-	-	-
191		0	1	2	3	4	5	7	-	-	-
192		6	7	-	-	-	-	-	-	-	-
193		0	6	7	-	-	-	-	-	-	-
194		1	6	7	-	-	-	-	-	-	-
195		0	1	6	7	-	-	-	-	-	-
196		2	6	7	-	-	-	-	-	-	-
197		0	2	6	7	-	-	-	-	-	-
198		1	2	6	7	-	-	-	-	-	-
199		0	1	2	6	7	-	-	-	-	-
200		3	6	7	-	-	-	-	-	-	-
201		0	3	6	7	-	-	-	-	-	-
202		1	3	6	7	-	-	-	-	-	-
203		0	1	3	6	7	-	-	-	-	-
204		2	3	6	7	-	-	-	-	-	-
205		0	2	3	6	7	-	-	-	-	-
206		1	2	3	6	7	-	-	-	-	-
207		0	1	2	3	6	7	-	-	-	-
208		4	6	7	-	-	-	-	-	-	-
209		0	4	6	7	-	-	-	-	-	-

Tabel 3.7 Tabel himpunan setBit setelah fungsi preprocess dijalankan (7)

<i>Num</i>	<i>index</i>	0	1	2	3	4	5	6	7	8	9
210		1	4	6	7	-	-	-	-	-	-
211		0	1	4	6	7	-	-	-	-	-
212		2	4	6	7	-	-	-	-	-	-
213		0	2	4	6	7	-	-	-	-	-
214		1	2	4	6	7	-	-	-	-	-
215		0	1	2	4	6	7	-	-	-	-
216		3	4	6	7	-	-	-	-	-	-
217		0	3	4	6	7	-	-	-	-	-
218		1	3	4	6	7	-	-	-	-	-
219		0	1	3	4	6	7	-	-	-	-
220		2	3	4	6	7	-	-	-	-	-
221		0	2	3	4	6	7	-	-	-	-
222		1	2	3	4	6	7	-	-	-	-
223		0	1	2	3	4	6	7	-	-	-
224		5	6	7	-	-	-	-	-	-	-
225		0	5	6	7	-	-	-	-	-	-
226		1	5	6	7	-	-	-	-	-	-
227		0	1	5	6	7	-	-	-	-	-
228		2	5	6	7	-	-	-	-	-	-
229		0	2	5	6	7	-	-	-	-	-
230		1	2	5	6	7	-	-	-	-	-
231		0	1	2	5	6	7	-	-	-	-
232		3	5	6	7	-	-	-	-	-	-
233		0	3	5	6	7	-	-	-	-	-
234		1	3	5	6	7	-	-	-	-	-
235		0	1	3	5	6	7	-	-	-	-
236		2	3	5	6	7	-	-	-	-	-
237		0	2	3	5	6	7	-	-	-	-
238		1	2	3	5	6	7	-	-	-	-
239		0	1	2	3	5	6	7	-	-	-

Tabel 3.8 Tabel himpunan setBit setelah fungsi preprocess dijalankan (8)

<i>Num</i>	<i>index</i>	0	1	2	3	4	5	6	7	8	9
240		4	5	6	7	-	-	-	-	-	-
241		0	4	5	6	7	-	-	-	-	-
242		1	4	5	6	7	-	-	-	-	-
243		0	1	4	5	6	7	-	-	-	-
244		2	4	5	6	7	-	-	-	-	-
245		0	2	4	5	6	7	-	-	-	-
246		1	2	4	5	6	7	-	-	-	-
247		0	1	2	4	5	6	7	-	-	-
248		3	4	5	6	7	-	-	-	-	-
249		0	3	4	5	6	7	-	-	-	-
250		1	3	4	5	6	7	-	-	-	-
251		0	1	3	4	5	6	7	-	-	-
252		2	3	4	5	6	7	-	-	-	-
253		0	2	3	4	5	6	7	-	-	-
254		1	2	3	4	5	6	7	-	-	-
255		0	1	2	3	4	5	6	7	-	-
256		8	-	-	-	-	-	-	-	-	-
257		0	8	-	-	-	-	-	-	-	-
258		1	8	-	-	-	-	-	-	-	-
259		0	1	8	-	-	-	-	-	-	-
260		2	8	-	-	-	-	-	-	-	-
261		0	2	8	-	-	-	-	-	-	-
262		1	2	8	-	-	-	-	-	-	-
263		0	1	2	8	-	-	-	-	-	-
264		3	8	-	-	-	-	-	-	-	-
265		0	3	8	-	-	-	-	-	-	-
266		1	3	8	-	-	-	-	-	-	-
267		0	1	3	8	-	-	-	-	-	-
268		2	3	8	-	-	-	-	-	-	-
269		0	2	3	8	-	-	-	-	-	-

Tabel 3.9 Tabel himpunan setBit setelah fungsi preprocess dijalankan (9)

<i>Num</i>	<i>index</i>	0	1	2	3	4	5	6	7	8	9
270		1	2	3	8	-	-	-	-	-	-
271		0	1	2	3	8	-	-	-	-	-
272		4	8	-	-	-	-	-	-	-	-
273		0	4	8	-	-	-	-	-	-	-
274		1	4	8	-	-	-	-	-	-	-
275		0	1	4	8	-	-	-	-	-	-
276		2	4	8	-	-	-	-	-	-	-
277		0	2	4	8	-	-	-	-	-	-
278		1	2	4	8	-	-	-	-	-	-
279		0	1	2	4	8	-	-	-	-	-
280		3	4	8	-	-	-	-	-	-	-
281		0	3	4	8	-	-	-	-	-	-
282		1	3	4	8	-	-	-	-	-	-
283		0	1	3	4	8	-	-	-	-	-
284		2	3	4	8	-	-	-	-	-	-
285		0	2	3	4	8	-	-	-	-	-
286		1	2	3	4	8	-	-	-	-	-
287		0	1	2	3	4	8	-	-	-	-
288		5	8	-	-	-	-	-	-	-	-
289		0	5	8	-	-	-	-	-	-	-
290		1	5	8	-	-	-	-	-	-	-
291		0	1	5	8	-	-	-	-	-	-
292		2	5	8	-	-	-	-	-	-	-
293		0	2	5	8	-	-	-	-	-	-
294		1	2	5	8	-	-	-	-	-	-
295		0	1	2	5	8	-	-	-	-	-
296		3	5	8	-	-	-	-	-	-	-
297		0	3	5	8	-	-	-	-	-	-
298		1	3	5	8	-	-	-	-	-	-
299		0	1	3	5	8	-	-	-	-	-

Tabel 3.10 Tabel himpunan setBit setelah fungsi preprocess dijalankan (10)

<i>Num</i>	<i>index</i>	0	1	2	3	4	5	6	7	8	9
300		2	3	5	8	-	-	-	-	-	-
301		0	2	3	5	8	-	-	-	-	-
302		1	2	3	5	8	-	-	-	-	-
303		0	1	2	3	5	8	-	-	-	-
304		4	5	8	-	-	-	-	-	-	-
305		0	4	5	8	-	-	-	-	-	-
306		1	4	5	8	-	-	-	-	-	-
307		0	1	4	5	8	-	-	-	-	-
308		2	4	5	8	-	-	-	-	-	-
309		0	2	4	5	8	-	-	-	-	-
310		1	2	4	5	8	-	-	-	-	-
311		0	1	2	4	5	8	-	-	-	-
312		3	4	5	8	-	-	-	-	-	-
313		0	3	4	5	8	-	-	-	-	-
314		1	3	4	5	8	-	-	-	-	-
315		0	1	3	4	5	8	-	-	-	-
316		2	3	4	5	8	-	-	-	-	-
317		0	2	3	4	5	8	-	-	-	-
318		1	2	3	4	5	8	-	-	-	-
319		0	1	2	3	4	5	8	-	-	-
320		6	8	-	-	-	-	-	-	-	-
321		0	6	8	-	-	-	-	-	-	-
322		1	6	8	-	-	-	-	-	-	-
323		0	1	6	8	-	-	-	-	-	-
324		2	6	8	-	-	-	-	-	-	-
325		0	2	6	8	-	-	-	-	-	-
326		1	2	6	8	-	-	-	-	-	-
327		0	1	2	6	8	-	-	-	-	-
328		3	6	8	-	-	-	-	-	-	-
329		0	3	6	8	-	-	-	-	-	-

Tabel 3.11 Tabel himpunan setBit setelah fungsi preprocess dijalankan (11)

<i>Num</i>	<i>index</i>	0	1	2	3	4	5	6	7	8	9
330		1	3	6	8	-	-	-	-	-	-
331		0	1	3	6	8	-	-	-	-	-
332		2	3	6	8	-	-	-	-	-	-
333		0	2	3	6	8	-	-	-	-	-
334		1	2	3	6	8	-	-	-	-	-
335		0	1	2	3	6	8	-	-	-	-
336		4	6	8	-	-	-	-	-	-	-
337		0	4	6	8	-	-	-	-	-	-
338		1	4	6	8	-	-	-	-	-	-
339		0	1	4	6	8	-	-	-	-	-
340		2	4	6	8	-	-	-	-	-	-
341		0	2	4	6	8	-	-	-	-	-
342		1	2	4	6	8	-	-	-	-	-
343		0	1	2	4	6	8	-	-	-	-
344		3	4	6	8	-	-	-	-	-	-
345		0	3	4	6	8	-	-	-	-	-
346		1	3	4	6	8	-	-	-	-	-
347		0	1	3	4	6	8	-	-	-	-
348		2	3	4	6	8	-	-	-	-	-
349		0	2	3	4	6	8	-	-	-	-
350		1	2	3	4	6	8	-	-	-	-
351		0	1	2	3	4	6	8	-	-	-
352		5	6	8	-	-	-	-	-	-	-
353		0	5	6	8	-	-	-	-	-	-
354		1	5	6	8	-	-	-	-	-	-
355		0	1	5	6	8	-	-	-	-	-
356		2	5	6	8	-	-	-	-	-	-
357		0	2	5	6	8	-	-	-	-	-
358		1	2	5	6	8	-	-	-	-	-
359		0	1	2	5	6	8	-	-	-	-

Tabel 3.12 Tabel himpunan setBit setelah fungsi preprocess dijalankan (12)

<i>Num</i>	<i>index</i>	0	1	2	3	4	5	6	7	8	9
360		3	5	6	8	-	-	-	-	-	-
361		0	3	5	6	8	-	-	-	-	-
362		1	3	5	6	8	-	-	-	-	-
363		0	1	3	5	6	8	-	-	-	-
364		2	3	5	6	8	-	-	-	-	-
365		0	2	3	5	6	8	-	-	-	-
366		1	2	3	5	6	8	-	-	-	-
367		0	1	2	3	5	6	8	-	-	-
368		4	5	6	8	-	-	-	-	-	-
369		0	4	5	6	8	-	-	-	-	-
370		1	4	5	6	8	-	-	-	-	-
371		0	1	4	5	6	8	-	-	-	-
372		2	4	5	6	8	-	-	-	-	-
373		0	2	4	5	6	8	-	-	-	-
374		1	2	4	5	6	8	-	-	-	-
375		0	1	2	4	5	6	8	-	-	-
376		3	4	5	6	8	-	-	-	-	-
377		0	3	4	5	6	8	-	-	-	-
378		1	3	4	5	6	8	-	-	-	-
379		0	1	3	4	5	6	8	-	-	-
380		2	3	4	5	6	8	-	-	-	-
381		0	2	3	4	5	6	8	-	-	-
382		1	2	3	4	5	6	8	-	-	-
383		0	1	2	3	4	5	6	8	-	-
384		7	8	-	-	-	-	-	-	-	-
385		0	7	8	-	-	-	-	-	-	-
386		1	7	8	-	-	-	-	-	-	-
387		0	1	7	8	-	-	-	-	-	-
388		2	7	8	-	-	-	-	-	-	-
389		0	2	7	8	-	-	-	-	-	-

Tabel 3.13 Tabel himpunan setBit setelah fungsi preprocess dijalankan (13)

<i>Num</i>	<i>index</i>	0	1	2	3	4	5	6	7	8	9
390		1	2	7	8	-	-	-	-	-	-
391		0	1	2	7	8	-	-	-	-	-
392		3	7	8	-	-	-	-	-	-	-
393		0	3	7	8	-	-	-	-	-	-
394		1	3	7	8	-	-	-	-	-	-
395		0	1	3	7	8	-	-	-	-	-
396		2	3	7	8	-	-	-	-	-	-
397		0	2	3	7	8	-	-	-	-	-
398		1	2	3	7	8	-	-	-	-	-
399		0	1	2	3	7	8	-	-	-	-
400		4	7	8	-	-	-	-	-	-	-
401		0	4	7	8	-	-	-	-	-	-
402		1	4	7	8	-	-	-	-	-	-
403		0	1	4	7	8	-	-	-	-	-
404		2	4	7	8	-	-	-	-	-	-
405		0	2	4	7	8	-	-	-	-	-
406		1	2	4	7	8	-	-	-	-	-
407		0	1	2	4	7	8	-	-	-	-
408		3	4	7	8	-	-	-	-	-	-
409		0	3	4	7	8	-	-	-	-	-
410		1	3	4	7	8	-	-	-	-	-
411		0	1	3	4	7	8	-	-	-	-
412		2	3	4	7	8	-	-	-	-	-
413		0	2	3	4	7	8	-	-	-	-
414		1	2	3	4	7	8	-	-	-	-
415		0	1	2	3	4	7	8	-	-	-
416		5	7	8	-	-	-	-	-	-	-
417		0	5	7	8	-	-	-	-	-	-
418		1	5	7	8	-	-	-	-	-	-
419		0	1	5	7	8	-	-	-	-	-

Tabel 3.14 Tabel himpunan setBit setelah fungsi preprocess dijalankan (14)

<i>Num</i>	<i>index</i>	0	1	2	3	4	5	6	7	8	9
420		2	5	7	8	-	-	-	-	-	-
421		0	2	5	7	8	-	-	-	-	-
422		1	2	5	7	8	-	-	-	-	-
423		0	1	2	5	7	8	-	-	-	-
424		3	5	7	8	-	-	-	-	-	-
425		0	3	5	7	8	-	-	-	-	-
426		1	3	5	7	8	-	-	-	-	-
427		0	1	3	5	7	8	-	-	-	-
428		2	3	5	7	8	-	-	-	-	-
429		0	2	3	5	7	8	-	-	-	-
430		1	2	3	5	7	8	-	-	-	-
431		0	1	2	3	5	7	8	-	-	-
432		4	5	7	8	-	-	-	-	-	-
433		0	4	5	7	8	-	-	-	-	-
434		1	4	5	7	8	-	-	-	-	-
435		0	1	4	5	7	8	-	-	-	-
436		2	4	5	7	8	-	-	-	-	-
437		0	2	4	5	7	8	-	-	-	-
438		1	2	4	5	7	8	-	-	-	-
439		0	1	2	4	5	7	8	-	-	-
440		3	4	5	7	8	-	-	-	-	-
441		0	3	4	5	7	8	-	-	-	-
442		1	3	4	5	7	8	-	-	-	-
443		0	1	3	4	5	7	8	-	-	-
444		2	3	4	5	7	8	-	-	-	-
445		0	2	3	4	5	7	8	-	-	-
446		1	2	3	4	5	7	8	-	-	-
447		0	1	2	3	4	5	7	8	-	-
448		6	7	8	-	-	-	-	-	-	-
449		0	6	7	8	-	-	-	-	-	-

Tabel 3.15 Tabel himpunan setBit setelah fungsi preprocess dijalankan (15)

<i>Num</i>	<i>index</i>	0	1	2	3	4	5	6	7	8	9
450		1	6	7	8	-	-	-	-	-	-
451		0	1	6	7	8	-	-	-	-	-
452		2	6	7	8	-	-	-	-	-	-
453		0	2	6	7	8	-	-	-	-	-
454		1	2	6	7	8	-	-	-	-	-
455		0	1	2	6	7	8	-	-	-	-
456		3	6	7	8	-	-	-	-	-	-
457		0	3	6	7	8	-	-	-	-	-
458		1	3	6	7	8	-	-	-	-	-
459		0	1	3	6	7	8	-	-	-	-
460		2	3	6	7	8	-	-	-	-	-
461		0	2	3	6	7	8	-	-	-	-
462		1	2	3	6	7	8	-	-	-	-
463		0	1	2	3	6	7	8	-	-	-
464		4	6	7	8	-	-	-	-	-	-
465		0	4	6	7	8	-	-	-	-	-
466		1	4	6	7	8	-	-	-	-	-
467		0	1	4	6	7	8	-	-	-	-
468		2	4	6	7	8	-	-	-	-	-
469		0	2	4	6	7	8	-	-	-	-
470		1	2	4	6	7	8	-	-	-	-
471		0	1	2	4	6	7	8	-	-	-
472		3	4	6	7	8	-	-	-	-	-
473		0	3	4	6	7	8	-	-	-	-
474		1	3	4	6	7	8	-	-	-	-
475		0	1	3	4	6	7	8	-	-	-
476		2	3	4	6	7	8	-	-	-	-
477		0	2	3	4	6	7	8	-	-	-
478		1	2	3	4	6	7	8	-	-	-
479		0	1	2	3	4	6	7	8	-	-

Tabel 3.16 Tabel himpunan setBit setelah fungsi preprocess dijalankan (16)

<i>Num</i>	<i>index</i>	0	1	2	3	4	5	6	7	8	9
480		5	6	7	8	-	-	-	-	-	-
481		0	5	6	7	8	-	-	-	-	-
482		1	5	6	7	8	-	-	-	-	-
483		0	1	5	6	7	8	-	-	-	-
484		2	5	6	7	8	-	-	-	-	-
485		0	2	5	6	7	8	-	-	-	-
486		1	2	5	6	7	8	-	-	-	-
487		0	1	2	5	6	7	8	-	-	-
488		3	5	6	7	8	-	-	-	-	-
489		0	3	5	6	7	8	-	-	-	-
490		1	3	5	6	7	8	-	-	-	-
491		0	1	3	5	6	7	8	-	-	-
492		2	3	5	6	7	8	-	-	-	-
493		0	2	3	5	6	7	8	-	-	-
494		1	2	3	5	6	7	8	-	-	-
495		0	1	2	3	5	6	7	8	-	-
496		4	5	6	7	8	-	-	-	-	-
497		0	4	5	6	7	8	-	-	-	-
498		1	4	5	6	7	8	-	-	-	-
499		0	1	4	5	6	7	8	-	-	-
500		2	4	5	6	7	8	-	-	-	-
501		0	2	4	5	6	7	8	-	-	-
502		1	2	4	5	6	7	8	-	-	-
503		0	1	2	4	5	6	7	8	-	-
504		3	4	5	6	7	8	-	-	-	-
505		0	3	4	5	6	7	8	-	-	-
506		1	3	4	5	6	7	8	-	-	-
507		0	1	3	4	5	6	7	8	-	-
508		2	3	4	5	6	7	8	-	-	-
509		0	2	3	4	5	6	7	8	-	-

Tabel 3.17 Tabel himpunan setBit setelah fungsi preprocess dijalankan (17)

<i>index</i> <i>Num</i>	0	1	2	3	4	5	6	7	8	9
510	1	2	3	4	5	6	7	8	-	-
511	0	1	2	3	4	5	6	7	8	-
512	9	-	-	-	-	-	-	-	-	-
513	0	9	-	-	-	-	-	-	-	-
514	1	9	-	-	-	-	-	-	-	-
515	0	1	9	-	-	-	-	-	-	-
516	2	9	-	-	-	-	-	-	-	-
517	0	2	9	-	-	-	-	-	-	-
518	1	2	9	-	-	-	-	-	-	-
519	0	1	2	9	-	-	-	-	-	-
520	3	9	-	-	-	-	-	-	-	-
521	0	3	9	-	-	-	-	-	-	-
522	1	3	9	-	-	-	-	-	-	-
523	0	1	3	9	-	-	-	-	-	-
524	2	3	9	-	-	-	-	-	-	-
525	0	2	3	9	-	-	-	-	-	-
526	1	2	3	9	-	-	-	-	-	-
527	0	1	2	3	9	-	-	-	-	-
528	4	9	-	-	-	-	-	-	-	-
529	0	4	9	-	-	-	-	-	-	-
530	1	4	9	-	-	-	-	-	-	-
531	0	1	4	9	-	-	-	-	-	-
532	2	4	9	-	-	-	-	-	-	-
533	0	2	4	9	-	-	-	-	-	-
534	1	2	4	9	-	-	-	-	-	-
535	0	1	2	4	9	-	-	-	-	-
536	3	4	9	-	-	-	-	-	-	-
537	0	3	4	9	-	-	-	-	-	-
538	1	3	4	9	-	-	-	-	-	-
539	0	1	3	4	9	-	-	-	-	-

Tabel 3.18 Tabel himpunan setBit setelah fungsi preprocess dijalankan (18)

<i>Num</i>	<i>index</i>	0	1	2	3	4	5	6	7	8	9
540		2	3	4	9	-	-	-	-	-	-
541		0	2	3	4	9	-	-	-	-	-
542		1	2	3	4	9	-	-	-	-	-
543		0	1	2	3	4	9	-	-	-	-
544		5	9	-	-	-	-	-	-	-	-
545		0	5	9	-	-	-	-	-	-	-
546		1	5	9	-	-	-	-	-	-	-
547		0	1	5	9	-	-	-	-	-	-
548		2	5	9	-	-	-	-	-	-	-
549		0	2	5	9	-	-	-	-	-	-
550		1	2	5	9	-	-	-	-	-	-
551		0	1	2	5	9	-	-	-	-	-
552		3	5	9	-	-	-	-	-	-	-
553		0	3	5	9	-	-	-	-	-	-
554		1	3	5	9	-	-	-	-	-	-
555		0	1	3	5	9	-	-	-	-	-
556		2	3	5	9	-	-	-	-	-	-
557		0	2	3	5	9	-	-	-	-	-
558		1	2	3	5	9	-	-	-	-	-
559		0	1	2	3	5	9	-	-	-	-
560		4	5	9	-	-	-	-	-	-	-
561		0	4	5	9	-	-	-	-	-	-
562		1	4	5	9	-	-	-	-	-	-
563		0	1	4	5	9	-	-	-	-	-
564		2	4	5	9	-	-	-	-	-	-
565		0	2	4	5	9	-	-	-	-	-
566		1	2	4	5	9	-	-	-	-	-
567		0	1	2	4	5	9	-	-	-	-
568		3	4	5	9	-	-	-	-	-	-
569		0	3	4	5	9	-	-	-	-	-

Tabel 3.19 Tabel himpunan setBit setelah fungsi preprocess dijalankan (19)

<i>Num</i>	<i>index</i>	0	1	2	3	4	5	6	7	8	9
570		1	3	4	5	9	-	-	-	-	-
571		0	1	3	4	5	9	-	-	-	-
572		2	3	4	5	9	-	-	-	-	-
573		0	2	3	4	5	9	-	-	-	-
574		1	2	3	4	5	9	-	-	-	-
575		0	1	2	3	4	5	9	-	-	-
576		6	9	-	-	-	-	-	-	-	-
577		0	6	9	-	-	-	-	-	-	-
578		1	6	9	-	-	-	-	-	-	-
579		0	1	6	9	-	-	-	-	-	-
580		2	6	9	-	-	-	-	-	-	-
581		0	2	6	9	-	-	-	-	-	-
582		1	2	6	9	-	-	-	-	-	-
583		0	1	2	6	9	-	-	-	-	-
584		3	6	9	-	-	-	-	-	-	-
585		0	3	6	9	-	-	-	-	-	-
586		1	3	6	9	-	-	-	-	-	-
587		0	1	3	6	9	-	-	-	-	-
588		2	3	6	9	-	-	-	-	-	-
589		0	2	3	6	9	-	-	-	-	-
590		1	2	3	6	9	-	-	-	-	-
591		0	1	2	3	6	9	-	-	-	-
592		4	6	9	-	-	-	-	-	-	-
593		0	4	6	9	-	-	-	-	-	-
594		1	4	6	9	-	-	-	-	-	-
595		0	1	4	6	9	-	-	-	-	-
596		2	4	6	9	-	-	-	-	-	-
597		0	2	4	6	9	-	-	-	-	-
598		1	2	4	6	9	-	-	-	-	-
599		0	1	2	4	6	9	-	-	-	-

Tabel 3.20 Tabel himpunan setBit setelah fungsi preprocess dijalankan (20)

<i>Num</i>	<i>index</i>	0	1	2	3	4	5	6	7	8	9
600		3	4	6	9	-	-	-	-	-	-
601		0	3	4	6	9	-	-	-	-	-
602		1	3	4	6	9	-	-	-	-	-
603		0	1	3	4	6	9	-	-	-	-
604		2	3	4	6	9	-	-	-	-	-
605		0	2	3	4	6	9	-	-	-	-
606		1	2	3	4	6	9	-	-	-	-
607		0	1	2	3	4	6	9	-	-	-
608		5	6	9	-	-	-	-	-	-	-
609		0	5	6	9	-	-	-	-	-	-
610		1	5	6	9	-	-	-	-	-	-
611		0	1	5	6	9	-	-	-	-	-
612		2	5	6	9	-	-	-	-	-	-
613		0	2	5	6	9	-	-	-	-	-
614		1	2	5	6	9	-	-	-	-	-
615		0	1	2	5	6	9	-	-	-	-
616		3	5	6	9	-	-	-	-	-	-
617		0	3	5	6	9	-	-	-	-	-
618		1	3	5	6	9	-	-	-	-	-
619		0	1	3	5	6	9	-	-	-	-
620		2	3	5	6	9	-	-	-	-	-
621		0	2	3	5	6	9	-	-	-	-
622		1	2	3	5	6	9	-	-	-	-
623		0	1	2	3	5	6	9	-	-	-
624		4	5	6	9	-	-	-	-	-	-
625		0	4	5	6	9	-	-	-	-	-
626		1	4	5	6	9	-	-	-	-	-
627		0	1	4	5	6	9	-	-	-	-
628		2	4	5	6	9	-	-	-	-	-
629		0	2	4	5	6	9	-	-	-	-

Tabel 3.21 Tabel himpunan setBit setelah fungsi preprocess dijalankan (21)

<i>Num</i>	<i>index</i>	0	1	2	3	4	5	6	7	8	9
630		1	2	4	5	6	9	-	-	-	-
631		0	1	2	4	5	6	9	-	-	-
632		3	4	5	6	9	-	-	-	-	-
633		0	3	4	5	6	9	-	-	-	-
634		1	3	4	5	6	9	-	-	-	-
635		0	1	3	4	5	6	9	-	-	-
636		2	3	4	5	6	9	-	-	-	-
637		0	2	3	4	5	6	9	-	-	-
638		1	2	3	4	5	6	9	-	-	-
639		0	1	2	3	4	5	6	9	-	-
640		7	9	-	-	-	-	-	-	-	-
641		0	7	9	-	-	-	-	-	-	-
642		1	7	9	-	-	-	-	-	-	-
643		0	1	7	9	-	-	-	-	-	-
644		2	7	9	-	-	-	-	-	-	-
645		0	2	7	9	-	-	-	-	-	-
646		1	2	7	9	-	-	-	-	-	-
647		0	1	2	7	9	-	-	-	-	-
648		3	7	9	-	-	-	-	-	-	-
649		0	3	7	9	-	-	-	-	-	-
650		1	3	7	9	-	-	-	-	-	-
651		0	1	3	7	9	-	-	-	-	-
652		2	3	7	9	-	-	-	-	-	-
653		0	2	3	7	9	-	-	-	-	-
654		1	2	3	7	9	-	-	-	-	-
655		0	1	2	3	7	9	-	-	-	-
656		4	7	9	-	-	-	-	-	-	-
657		0	4	7	9	-	-	-	-	-	-
658		1	4	7	9	-	-	-	-	-	-
659		0	1	4	7	9	-	-	-	-	-

Tabel 3.22 Tabel himpunan setBit setelah fungsi preprocess dijalankan (22)

<i>Num</i>	<i>index</i>	0	1	2	3	4	5	6	7	8	9
660		2	4	7	9	-	-	-	-	-	-
661		0	2	4	7	9	-	-	-	-	-
662		1	2	4	7	9	-	-	-	-	-
663		0	1	2	4	7	9	-	-	-	-
664		3	4	7	9	-	-	-	-	-	-
665		0	3	4	7	9	-	-	-	-	-
666		1	3	4	7	9	-	-	-	-	-
667		0	1	3	4	7	9	-	-	-	-
668		2	3	4	7	9	-	-	-	-	-
669		0	2	3	4	7	9	-	-	-	-
670		1	2	3	4	7	9	-	-	-	-
671		0	1	2	3	4	7	9	-	-	-
672		5	7	9	-	-	-	-	-	-	-
673		0	5	7	9	-	-	-	-	-	-
674		1	5	7	9	-	-	-	-	-	-
675		0	1	5	7	9	-	-	-	-	-
676		2	5	7	9	-	-	-	-	-	-
677		0	2	5	7	9	-	-	-	-	-
678		1	2	5	7	9	-	-	-	-	-
679		0	1	2	5	7	9	-	-	-	-
680		3	5	7	9	-	-	-	-	-	-
681		0	3	5	7	9	-	-	-	-	-
682		1	3	5	7	9	-	-	-	-	-
683		0	1	3	5	7	9	-	-	-	-
684		2	3	5	7	9	-	-	-	-	-
685		0	2	3	5	7	9	-	-	-	-
686		1	2	3	5	7	9	-	-	-	-
687		0	1	2	3	5	7	9	-	-	-
688		4	5	7	9	-	-	-	-	-	-
689		0	4	5	7	9	-	-	-	-	-

Tabel 3.23 Tabel himpunan setBit setelah fungsi preprocess dijalankan (23)

<i>index</i> <i>Num</i>	0	1	2	3	4	5	6	7	8	9
690	1	4	5	7	9	-	-	-	-	-
691	0	1	4	5	7	9	-	-	-	-
692	2	4	5	7	9	-	-	-	-	-
693	0	2	4	5	7	9	-	-	-	-
694	1	2	4	5	7	9	-	-	-	-
695	0	1	2	4	5	7	9	-	-	-
696	3	4	5	7	9	-	-	-	-	-
697	0	3	4	5	7	9	-	-	-	-
698	1	3	4	5	7	9	-	-	-	-
699	0	1	3	4	5	7	9	-	-	-
700	2	3	4	5	7	9	-	-	-	-
701	0	2	3	4	5	7	9	-	-	-
702	1	2	3	4	5	7	9	-	-	-
703	0	1	2	3	4	5	7	9	-	-
704	6	7	9	-	-	-	-	-	-	-
705	0	6	7	9	-	-	-	-	-	-
706	1	6	7	9	-	-	-	-	-	-
707	0	1	6	7	9	-	-	-	-	-
708	2	6	7	9	-	-	-	-	-	-
709	0	2	6	7	9	-	-	-	-	-
710	1	2	6	7	9	-	-	-	-	-
711	0	1	2	6	7	9	-	-	-	-
712	3	6	7	9	-	-	-	-	-	-
713	0	3	6	7	9	-	-	-	-	-
714	1	3	6	7	9	-	-	-	-	-
715	0	1	3	6	7	9	-	-	-	-
716	2	3	6	7	9	-	-	-	-	-
717	0	2	3	6	7	9	-	-	-	-
718	1	2	3	6	7	9	-	-	-	-
719	0	1	2	3	6	7	9	-	-	-

Tabel 3.24 Tabel himpunan setBit setelah fungsi preprocess dijalankan (24)

<i>index</i> <i>Num</i>	0	1	2	3	4	5	6	7	8	9
720	4	6	7	9	-	-	-	-	-	-
721	0	4	6	7	9	-	-	-	-	-
722	1	4	6	7	9	-	-	-	-	-
723	0	1	4	6	7	9	-	-	-	-
724	2	4	6	7	9	-	-	-	-	-
725	0	2	4	6	7	9	-	-	-	-
726	1	2	4	6	7	9	-	-	-	-
727	0	1	2	4	6	7	9	-	-	-
728	3	4	6	7	9	-	-	-	-	-
729	0	3	4	6	7	9	-	-	-	-
730	1	3	4	6	7	9	-	-	-	-
731	0	1	3	4	6	7	9	-	-	-
732	2	3	4	6	7	9	-	-	-	-
733	0	2	3	4	6	7	9	-	-	-
734	1	2	3	4	6	7	9	-	-	-
735	0	1	2	3	4	6	7	9	-	-
736	5	6	7	9	-	-	-	-	-	-
737	0	5	6	7	9	-	-	-	-	-
738	1	5	6	7	9	-	-	-	-	-
739	0	1	5	6	7	9	-	-	-	-
740	2	5	6	7	9	-	-	-	-	-
741	0	2	5	6	7	9	-	-	-	-
742	1	2	5	6	7	9	-	-	-	-
743	0	1	2	5	6	7	9	-	-	-
744	3	5	6	7	9	-	-	-	-	-
745	0	3	5	6	7	9	-	-	-	-
746	1	3	5	6	7	9	-	-	-	-
747	0	1	3	5	6	7	9	-	-	-
748	2	3	5	6	7	9	-	-	-	-
749	0	2	3	5	6	7	9	-	-	-

Tabel 3.25 Tabel himpunan setBit setelah fungsi preprocess dijalankan (25)

<i>index</i> <i>Num</i>	0	1	2	3	4	5	6	7	8	9
750	1	2	3	5	6	7	9	-	-	-
751	0	1	2	3	5	6	7	9	-	-
752	4	5	6	7	9	-	-	-	-	-
753	0	4	5	6	7	9	-	-	-	-
754	1	4	5	6	7	9	-	-	-	-
755	0	1	4	5	6	7	9	-	-	-
756	2	4	5	6	7	9	-	-	-	-
757	0	2	4	5	6	7	9	-	-	-
758	1	2	4	5	6	7	9	-	-	-
759	0	1	2	4	5	6	7	9	-	-
760	3	4	5	6	7	9	-	-	-	-
761	0	3	4	5	6	7	9	-	-	-
762	1	3	4	5	6	7	9	-	-	-
763	0	1	3	4	5	6	7	9	-	-
764	2	3	4	5	6	7	9	-	-	-
765	0	2	3	4	5	6	7	9	-	-
766	1	2	3	4	5	6	7	9	-	-
767	0	1	2	3	4	5	6	7	9	-
768	8	9	-	-	-	-	-	-	-	-
769	0	8	9	-	-	-	-	-	-	-
770	1	8	9	-	-	-	-	-	-	-
771	0	1	8	9	-	-	-	-	-	-
772	2	8	9	-	-	-	-	-	-	-
773	0	2	8	9	-	-	-	-	-	-
774	1	2	8	9	-	-	-	-	-	-
775	0	1	2	8	9	-	-	-	-	-
776	3	8	9	-	-	-	-	-	-	-
777	0	3	8	9	-	-	-	-	-	-
778	1	3	8	9	-	-	-	-	-	-
779	0	1	3	8	9	-	-	-	-	-

Tabel 3.26 Tabel himpunan setBit setelah fungsi preprocess dijalankan (26)

<i>Num</i>	<i>index</i>	0	1	2	3	4	5	6	7	8	9
780		2	3	8	9	-	-	-	-	-	-
781		0	2	3	8	9	-	-	-	-	-
782		1	2	3	8	9	-	-	-	-	-
783		0	1	2	3	8	9	-	-	-	-
784		4	8	9	-	-	-	-	-	-	-
785		0	4	8	9	-	-	-	-	-	-
786		1	4	8	9	-	-	-	-	-	-
787		0	1	4	8	9	-	-	-	-	-
788		2	4	8	9	-	-	-	-	-	-
789		0	2	4	8	9	-	-	-	-	-
790		1	2	4	8	9	-	-	-	-	-
791		0	1	2	4	8	9	-	-	-	-
792		3	4	8	9	-	-	-	-	-	-
793		0	3	4	8	9	-	-	-	-	-
794		1	3	4	8	9	-	-	-	-	-
795		0	1	3	4	8	9	-	-	-	-
796		2	3	4	8	9	-	-	-	-	-
797		0	2	3	4	8	9	-	-	-	-
798		1	2	3	4	8	9	-	-	-	-
799		0	1	2	3	4	8	9	-	-	-
800		5	8	9	-	-	-	-	-	-	-
801		0	5	8	9	-	-	-	-	-	-
802		1	5	8	9	-	-	-	-	-	-
803		0	1	5	8	9	-	-	-	-	-
804		2	5	8	9	-	-	-	-	-	-
805		0	2	5	8	9	-	-	-	-	-
806		1	2	5	8	9	-	-	-	-	-
807		0	1	2	5	8	9	-	-	-	-
808		3	5	8	9	-	-	-	-	-	-
809		0	3	5	8	9	-	-	-	-	-

Tabel 3.27 Tabel himpunan setBit setelah fungsi preprocess dijalankan (27)

<i>Num</i>	<i>index</i>	0	1	2	3	4	5	6	7	8	9
810		1	3	5	8	9	-	-	-	-	-
811		0	1	3	5	8	9	-	-	-	-
812		2	3	5	8	9	-	-	-	-	-
813		0	2	3	5	8	9	-	-	-	-
814		1	2	3	5	8	9	-	-	-	-
815		0	1	2	3	5	8	9	-	-	-
816		4	5	8	9	-	-	-	-	-	-
817		0	4	5	8	9	-	-	-	-	-
818		1	4	5	8	9	-	-	-	-	-
819		0	1	4	5	8	9	-	-	-	-
820		2	4	5	8	9	-	-	-	-	-
821		0	2	4	5	8	9	-	-	-	-
822		1	2	4	5	8	9	-	-	-	-
823		0	1	2	4	5	8	9	-	-	-
824		3	4	5	8	9	-	-	-	-	-
825		0	3	4	5	8	9	-	-	-	-
826		1	3	4	5	8	9	-	-	-	-
827		0	1	3	4	5	8	9	-	-	-
828		2	3	4	5	8	9	-	-	-	-
829		0	2	3	4	5	8	9	-	-	-
830		1	2	3	4	5	8	9	-	-	-
831		0	1	2	3	4	5	8	9	-	-
832		6	8	9	-	-	-	-	-	-	-
833		0	6	8	9	-	-	-	-	-	-
834		1	6	8	9	-	-	-	-	-	-
835		0	1	6	8	9	-	-	-	-	-
836		2	6	8	9	-	-	-	-	-	-
837		0	2	6	8	9	-	-	-	-	-
838		1	2	6	8	9	-	-	-	-	-
839		0	1	2	6	8	9	-	-	-	-

Tabel 3.28 Tabel himpunan setBit setelah fungsi preprocess dijalankan (28)

<i>Num</i>	<i>index</i>	0	1	2	3	4	5	6	7	8	9
840		3	6	8	9	-	-	-	-	-	-
841		0	3	6	8	9	-	-	-	-	-
842		1	3	6	8	9	-	-	-	-	-
843		0	1	3	6	8	9	-	-	-	-
844		2	3	6	8	9	-	-	-	-	-
845		0	2	3	6	8	9	-	-	-	-
846		1	2	3	6	8	9	-	-	-	-
847		0	1	2	3	6	8	9	-	-	-
848		4	6	8	9	-	-	-	-	-	-
849		0	4	6	8	9	-	-	-	-	-
850		1	4	6	8	9	-	-	-	-	-
851		0	1	4	6	8	9	-	-	-	-
852		2	4	6	8	9	-	-	-	-	-
853		0	2	4	6	8	9	-	-	-	-
854		1	2	4	6	8	9	-	-	-	-
855		0	1	2	4	6	8	9	-	-	-
856		3	4	6	8	9	-	-	-	-	-
857		0	3	4	6	8	9	-	-	-	-
858		1	3	4	6	8	9	-	-	-	-
859		0	1	3	4	6	8	9	-	-	-
860		2	3	4	6	8	9	-	-	-	-
861		0	2	3	4	6	8	9	-	-	-
862		1	2	3	4	6	8	9	-	-	-
863		0	1	2	3	4	6	8	9	-	-
864		5	6	8	9	-	-	-	-	-	-
865		0	5	6	8	9	-	-	-	-	-
866		1	5	6	8	9	-	-	-	-	-
867		0	1	5	6	8	9	-	-	-	-
868		2	5	6	8	9	-	-	-	-	-
869		0	2	5	6	8	9	-	-	-	-

Tabel 3.29 Tabel himpunan setBit setelah fungsi preprocess dijalankan (29)

<i>Num</i>	<i>index</i>	0	1	2	3	4	5	6	7	8	9
870		1	2	5	6	8	9	-	-	-	-
871		0	1	2	5	6	8	9	-	-	-
872		3	5	6	8	9	-	-	-	-	-
873		0	3	5	6	8	9	-	-	-	-
874		1	3	5	6	8	9	-	-	-	-
875		0	1	3	5	6	8	9	-	-	-
876		2	3	5	6	8	9	-	-	-	-
877		0	2	3	5	6	8	9	-	-	-
878		1	2	3	5	6	8	9	-	-	-
879		0	1	2	3	5	6	8	9	-	-
880		4	5	6	8	9	-	-	-	-	-
881		0	4	5	6	8	9	-	-	-	-
882		1	4	5	6	8	9	-	-	-	-
883		0	1	4	5	6	8	9	-	-	-
884		2	4	5	6	8	9	-	-	-	-
885		0	2	4	5	6	8	9	-	-	-
886		1	2	4	5	6	8	9	-	-	-
887		0	1	2	4	5	6	8	9	-	-
888		3	4	5	6	8	9	-	-	-	-
889		0	3	4	5	6	8	9	-	-	-
890		1	3	4	5	6	8	9	-	-	-
891		0	1	3	4	5	6	8	9	-	-
892		2	3	4	5	6	8	9	-	-	-
893		0	2	3	4	5	6	8	9	-	-
894		1	2	3	4	5	6	8	9	-	-
895		0	1	2	3	4	5	6	8	9	-
896		7	8	9	-	-	-	-	-	-	-
897		0	7	8	9	-	-	-	-	-	-
898		1	7	8	9	-	-	-	-	-	-
899		0	1	7	8	9	-	-	-	-	-

Tabel 3.30 Tabel himpunan setBit setelah fungsi preprocess dijalankan (30)

<i>index</i> <i>Num</i>	0	1	2	3	4	5	6	7	8	9
900	2	7	8	9	-	-	-	-	-	-
901	0	2	7	8	9	-	-	-	-	-
902	1	2	7	8	9	-	-	-	-	-
903	0	1	2	7	8	9	-	-	-	-
904	3	7	8	9	-	-	-	-	-	-
905	0	3	7	8	9	-	-	-	-	-
906	1	3	7	8	9	-	-	-	-	-
907	0	1	3	7	8	9	-	-	-	-
908	2	3	7	8	9	-	-	-	-	-
909	0	2	3	7	8	9	-	-	-	-
910	1	2	3	7	8	9	-	-	-	-
911	0	1	2	3	7	8	9	-	-	-
912	4	7	8	9	-	-	-	-	-	-
913	0	4	7	8	9	-	-	-	-	-
914	1	4	7	8	9	-	-	-	-	-
915	0	1	4	7	8	9	-	-	-	-
916	2	4	7	8	9	-	-	-	-	-
917	0	2	4	7	8	9	-	-	-	-
918	1	2	4	7	8	9	-	-	-	-
919	0	1	2	4	7	8	9	-	-	-
920	3	4	7	8	9	-	-	-	-	-
921	0	3	4	7	8	9	-	-	-	-
922	1	3	4	7	8	9	-	-	-	-
923	0	1	3	4	7	8	9	-	-	-
924	2	3	4	7	8	9	-	-	-	-
925	0	2	3	4	7	8	9	-	-	-
926	1	2	3	4	7	8	9	-	-	-
927	0	1	2	3	4	7	8	9	-	-
928	5	7	8	9	-	-	-	-	-	-
929	0	5	7	8	9	-	-	-	-	-

Tabel 3.31 Tabel himpunan setBit setelah fungsi preprocess dijalankan (31)

<i>index</i> <i>Num</i>	0	1	2	3	4	5	6	7	8	9
930	1	5	7	8	9	-	-	-	-	-
931	0	1	5	7	8	9	-	-	-	-
932	2	5	7	8	9	-	-	-	-	-
933	0	2	5	7	8	9	-	-	-	-
934	1	2	5	7	8	9	-	-	-	-
935	0	1	2	5	7	8	9	-	-	-
936	3	5	7	8	9	-	-	-	-	-
937	0	3	5	7	8	9	-	-	-	-
938	1	3	5	7	8	9	-	-	-	-
939	0	1	3	5	7	8	9	-	-	-
940	2	3	5	7	8	9	-	-	-	-
941	0	2	3	5	7	8	9	-	-	-
942	1	2	3	5	7	8	9	-	-	-
943	0	1	2	3	5	7	8	9	-	-
944	4	5	7	8	9	-	-	-	-	-
945	0	4	5	7	8	9	-	-	-	-
946	1	4	5	7	8	9	-	-	-	-
947	0	1	4	5	7	8	9	-	-	-
948	2	4	5	7	8	9	-	-	-	-
949	0	2	4	5	7	8	9	-	-	-
950	1	2	4	5	7	8	9	-	-	-
951	0	1	2	4	5	7	8	9	-	-
952	3	4	5	7	8	9	-	-	-	-
953	0	3	4	5	7	8	9	-	-	-
954	1	3	4	5	7	8	9	-	-	-
955	0	1	3	4	5	7	8	9	-	-
956	2	3	4	5	7	8	9	-	-	-
957	0	2	3	4	5	7	8	9	-	-
958	1	2	3	4	5	7	8	9	-	-
959	0	1	2	3	4	5	7	8	9	-

Tabel 3.32 Tabel himpunan setBit setelah fungsi preprocess dijalankan (32)

<i>Num</i>	<i>index</i>	0	1	2	3	4	5	6	7	8	9
960		6	7	8	9	-	-	-	-	-	-
961		0	6	7	8	9	-	-	-	-	-
962		1	6	7	8	9	-	-	-	-	-
963		0	1	6	7	8	9	-	-	-	-
964		2	6	7	8	9	-	-	-	-	-
965		0	2	6	7	8	9	-	-	-	-
966		1	2	6	7	8	9	-	-	-	-
967		0	1	2	6	7	8	9	-	-	-
968		3	6	7	8	9	-	-	-	-	-
969		0	3	6	7	8	9	-	-	-	-
970		1	3	6	7	8	9	-	-	-	-
971		0	1	3	6	7	8	9	-	-	-
972		2	3	6	7	8	9	-	-	-	-
973		0	2	3	6	7	8	9	-	-	-
974		1	2	3	6	7	8	9	-	-	-
975		0	1	2	3	6	7	8	9	-	-
976		4	6	7	8	9	-	-	-	-	-
977		0	4	6	7	8	9	-	-	-	-
978		1	4	6	7	8	9	-	-	-	-
979		0	1	4	6	7	8	9	-	-	-
980		2	4	6	7	8	9	-	-	-	-
981		0	2	4	6	7	8	9	-	-	-
982		1	2	4	6	7	8	9	-	-	-
983		0	1	2	4	6	7	8	9	-	-
984		3	4	6	7	8	9	-	-	-	-
985		0	3	4	6	7	8	9	-	-	-
986		1	3	4	6	7	8	9	-	-	-
987		0	1	3	4	6	7	8	9	-	-
988		2	3	4	6	7	8	9	-	-	-
989		0	2	3	4	6	7	8	9	-	-

Tabel 3.33 Tabel himpunan setBit setelah fungsi preprocess dijalankan (33)

<i>index</i> <i>Num</i>	0	1	2	3	4	5	6	7	8	9
990	1	2	3	4	6	7	8	9	-	-
991	0	1	2	3	4	6	7	8	9	-
992	5	6	7	8	9	-	-	-	-	-
993	0	5	6	7	8	9	-	-	-	-
994	1	5	6	7	8	9	-	-	-	-
995	0	1	5	6	7	8	9	-	-	-
996	2	5	6	7	8	9	-	-	-	-
997	0	2	5	6	7	8	9	-	-	-
998	1	2	5	6	7	8	9	-	-	-
999	0	1	2	5	6	7	8	9	-	-
1000	3	5	6	7	8	9	-	-	-	-
1001	0	3	5	6	7	8	9	-	-	-
1002	1	3	5	6	7	8	9	-	-	-
1003	0	1	3	5	6	7	8	9	-	-
1004	2	3	5	6	7	8	9	-	-	-
1005	0	2	3	5	6	7	8	9	-	-
1006	1	2	3	5	6	7	8	9	-	-
1007	0	1	2	3	5	6	7	8	9	-
1008	4	5	6	7	8	9	-	-	-	-
1009	0	4	5	6	7	8	9	-	-	-
1010	1	4	5	6	7	8	9	-	-	-
1011	0	1	4	5	6	7	8	9	-	-
1012	2	4	5	6	7	8	9	-	-	-
1013	0	2	4	5	6	7	8	9	-	-
1014	1	2	4	5	6	7	8	9	-	-
1015	0	1	2	4	5	6	7	8	9	-
1016	3	4	5	6	7	8	9	-	-	-
1017	0	3	4	5	6	7	8	9	-	-
1018	1	3	4	5	6	7	8	9	-	-
1019	0	1	3	4	5	6	7	8	9	-

Tabel 3.34 Tabel himpunan setBit setelah fungsi preprocess dijalankan (34)

<i>Num</i>	<i>index</i>	0	1	2	3	4	5	6	7	8	9
1020		2	3	4	5	6	7	8	9	-	-
1021		0	2	3	4	5	6	7	8	9	-
1022		1	2	3	4	5	6	7	8	9	-
1023		0	1	2	3	4	5	6	7	8	9

Tabel 3.35 Tabel himpunan setBit setelah fungsi preprocess dijalankan (35)

digma *dynamic programming* yang menggunakan teknik memoisasi, maka algoritma yang dibangun harus melakukan inisialisasi nilai untuk setiap memo yang digunakan. Terdapat dua variabel memo yang digunakan pada algoritma yang dibangun, yaitu  $\text{memo } F_{(idx,mask,dist)}$  untuk mencatat hasil perhitungan fungsi  $F_{(S,mask,dist)}$  dan  $\text{memo } G_{(idx,mask,dist)}$  untuk mencatat hasil perhitungan fungsi  $G_{(S,mask,d)}$ .

Untuk mempermudah menyelesaikan permasalahan klasik SPOJ 9967 *Playing With Words*, algoritma yang dibangun membutuhkan string masukan *ad1* dan *ad2* dalam keadaan yang sudah terurut *ascending* secara alfabetis.

Pada bagian berikutnya adalah perhitungan data - data yang dibutuhkan untuk perhitungan jawaban akhir dari permasalahan klasik SPOJ 9967 *Playing With Words*. Data - data yang diperlukan adalah sebagai berikut:

1.  $\text{maxMask}_{(S)}$  yaitu nilai maksimal *mask* untuk string *S*. Nilai maksimal *mask* dari string *S* adalah  $2^{|S|} - 1$ .
2.  $\text{charFirstPos}_{(S,C)}$  yaitu posisi pertama karakter *C* pada string *S*.
3.  $\text{charLastPos}_{(S,C)}$  yaitu posisi terakhir karakter *C* pada string *S*.

```

init()
1  memoF = ∅
2  memoG = ∅
3  charFirstPos = ∅
4  charLastPos = ∅
5  maxMask(ad1) = 2|ad1| - 1
6  sort(ad1)
7  for i = 0 to |ad1| - 1
8    charLastPos(ad1,ad1i) = i
9    if charFirstPos(ad1,ad1i) = ∅
10      charFirstPos(ad1,ad1i) = i
11  maxMask(ad2) = 2|ad2| - 1
12  sort(ad2)
13 for i = 0 to |ad2| - 1
14    charLastPos(ad2,ad2i) = i
15    if charFirstPos(ad2,ad2i) = ∅
16      charFirstPos(ad2,ad2i) = i

```

Gambar 3.3 Pseudocode Fungsi Init

Sebagai contoh, ketika string masukan  $S = "inicontoh"$ , maka string  $S$  setelah diurutkan secara alfabetis akan menjadi  $"chiinnoot"$ . Tabel 3.36 adalah nilai dari  $charFirstPos_{(S,C)}$  dan  $charLastPos_{(S,C)}$ . Nilai dari  $maxMask_{(S)}$  adalah 511. Gambar 3.3 adalah pseudocode dari fungsi init.

### 3.4 Desain Fungsi Solve

Fungsi solve adalah fungsi yang bertujuan untuk menyelesaikan permasalahan sesuai dengan deskripsi permasalahan untuk setiap input yang diberikan. Setalah melalui proses *preprocessing*, memba-

$C$	$charFirstPos_{(S,C)}$	$charLastPos_{(S,C)}$
$a$	$\emptyset$	$\emptyset$
$b$	$\emptyset$	$\emptyset$
$c$	0	0
$d$	$\emptyset$	$\emptyset$
$e$	$\emptyset$	$\emptyset$
$f$	$\emptyset$	$\emptyset$
$g$	$\emptyset$	$\emptyset$
$h$	1	1
$i$	2	3
$j$	$\emptyset$	$\emptyset$
$k$	$\emptyset$	$\emptyset$
$l$	$\emptyset$	$\emptyset$
$m$	$\emptyset$	$\emptyset$
$n$	4	5
$o$	6	7
$p$	$\emptyset$	$\emptyset$
$q$	$\emptyset$	$\emptyset$
$r$	$\emptyset$	$\emptyset$
$s$	$\emptyset$	$\emptyset$
$t$	8	8
$u$	$\emptyset$	$\emptyset$
$v$	$\emptyset$	$\emptyset$
$w$	$\emptyset$	$\emptyset$
$x$	$\emptyset$	$\emptyset$
$y$	$\emptyset$	$\emptyset$
$z$	$\emptyset$	$\emptyset$

Tabel 3.36 Hasil  $charFirstPos_{(S,C)}$  dan  $charLastPos_{(S,C)}$  dengan string  $S = "inicontoh"$  setelah fungsi init dijalankan

```

solve(ad1, ad2, X)
1  ret = 0
2  bound = min(X, 250)
3  for dist = 0 to min(250, X)
4      rem = X - dist
5      if rem > 250
6          continue
7      if rem < 0
8          break
9      ret = ret + F(ad1,maxMaskad1,bound-dist)
10     + G(ad2,maxMaskad2,bound-rem)
10     ret = ret + G(ad1,maxMaskad1,bound-dist)
11     + F(ad2,maxMaskad2,bound-rem)
11  return ret

```

Gambar 3.4 Pseudocode Fungsi Solve

ca masukan dan inisialisasi, masukan akan diolah untuk menghasilkan jawaban dari permasalahan klasik SPOJ 9967 *Playing With Words*. Algoritma fungsi solve yang dibangun akan didasari oleh persamaan - persamaan yang terdapat pada subbab 2.4.

Fungsi solve merupakan fungsi yang mengimplementasi persamaan 2.4.1. Fungsi solve sendiri akan membutuhkan beberapa fungsi - fungsi lain untuk membantu. Fungsi - fungsi tersebut antara lain fungsi  $F_{(S,mask,dist)}$  dan fungsi  $G_{(S,mask,dist)}$ . Gambar 3.4 adalah pseudocode dari fungsi solve.

### 3.4.1 Desain Fungsi F

Pada pseudocode pada gambar 3.4, terdapat perhitungan dengan menggunakan fungsi  $F_{(S,mask,dist)}$  pada baris 9 dan 10. Dimana se-

perti yang telah dijelaskan pada bagian 2.4.1, fungsi  $F_{(S,mask,dist)}$  adalah fungsi untuk menghitung jumlah kemungkinan string *orig* dari string  $S$  tanpa operasi *replace* dengan jarak  $dist$ . Nilai dari fungsi  $F_{(S,mask,dist)}$  adalah hasil penjumlahan seluruh *state* yang berhubungan, yaitu *state*  $F_{(S,mask-2^{idx},dist+|S_{idx}-S_{curIdx}|)}$  dimana  $curIdx$  adalah jumlah bit tidak menyala pada *mask* dan  $idx$  adalah  $set\_bit(mask)_i$  untuk setiap  $i$  dimana  $0 \leq i \leq NSB_{mask}$  dengan  $set\_bit(mask)$  adalah Himpunan index bit menyala pada *mask* dan  $NSB_{mask}$  adalah jumlah bit menyala pada *mask*. Algoritma pada fungsi  $F_{(S,mask,dist)}$  akan didasari oleh persamaan 2.4.2. Gambar 3.5 adalah pseudocode dari fungsi  $F_{(S,mask,dist)}$ .

```

F( $S, mask, dist$ )
1 if  $dist > bound \vee (mask = 0 \wedge dist \neq bound)$ 
2   return 0
3 if  $mask = 0 \wedge dist = bound$ 
4   return 1
5 if  $memoF_{(S,mask,dist)} \neq \emptyset$ 
6   return  $memoF_{(S,mask,dist)}$ 
7  $numberOfSetBit = size_{(setBit_{(mask)})}$ 
8  $retVal = 0$ 
9 for  $i = 0$  to  $numberOfSetBit - 1$ 
10   if  $setBit_{(mask)i} \geq length_{(S)}$ 
11     break
12    $ret = ret + F1_{(S,mask, setBit_{(mask)i}, dist)}$ 
13 return  $memoF_{(S,mask,dist)} = ret$ 

```

Gambar 3.5 Pseudocode Fungsi F

Karena tidak semua *state* yang terhubung dengan *state*  $F_{(S,mask,dist)}$  valid, maka diperlukan sebuah fungsi

Fungsi	Perhitungan Nilai	Nilai
$F_{(behkn,31,5)}$	$F_{(behkn,30,5)} + F_{(behkn,29,8)} + F_{(behkn,27,11)} + F_{(behkn,23,14)} + F_{(behkn,15,17)}$	1
$F_{(behkn,15,17)}$	<i>base case</i>	0
$F_{(behkn,23,14)}$	<i>base case</i>	0
$F_{(behkn,27,11)}$	<i>base case</i>	0
$F_{(behkn,29,8)}$	<i>base case</i>	0
$F_{(behkn,30,5)}$	$F_{(behkn,28,5)} + F_{(behkn,26,8)} + F_{(behkn,22,11)} + F_{(behkn,14,14)}$	1
$F_{(behkn,14,14)}$	<i>base case</i>	0
$F_{(behkn,22,11)}$	<i>base case</i>	0
$F_{(behkn,26,8)}$	<i>base case</i>	0
$F_{(behkn,28,5)}$	$F_{(behkn,24,5)} + F_{(behkn,20,8)} + F_{(behkn,12,11)}$	1
$F_{(behkn,12,11)}$	<i>base case</i>	0
$F_{(behkn,20,8)}$	<i>base case</i>	0
$F_{(behkn,24,5)}$	$F_{(behkn,16,5)} + F_{(behkn,8,8)}$	1
$F_{(behkn,8,8)}$	<i>base case</i>	0
$F_{(behkn,16,5)}$	$F_{(behkn,0,5)}$	1
$F_{(behkn,0,5)}$	<i>base case</i>	1

Tabel 3.37 Simulasi fungsi  $F$  dengan  $S = "kbenh"$ ,  $X = 5$  dan  $dist = 5$

$F1_{(S,mask,idx,dist)}$  untuk menentukan valid atau tidaknya sebuah state  $F_{(S,mask-2^{idx},dist+|S_{idx}-S_{curIdx}|)}$  yang terbentuk dari state  $F_{(S,mask,dist)}$ . Perancangan algoritma fungsi  $F1_{(S,mask,idx,dist)}$  akan didasari oleh persamaan 2.4.3. Gambar 3.6 adalah pseudocode dari fungsi  $F1_{(S,mask,idx,dist)}$  dan gambar 3.7 adalah pseudocode dari fungsi  $duplicate\_rule1(S,mask,idx)$ . Tabel 3.37 adalah simulasi fungsi  $F$  dengan  $S = "kbenh"$ ,  $X = 5$  dan  $dist = 5$ .

```

F1( $S, mask, idx, dist$ )
1    $curIdx = \text{length}_{(S)} - \text{size}_{(\text{setBit}_{(mask)})}$ 
2   if  $idx = \text{length}_{(S)} - 1 \vee \text{duplicateRule1}_{(S,mask,idx)}$ 
3     return  $F_{(S,mask-2^{idx},dist+|S_{idx}-S_{curIdx}|)}$ 
```

Gambar 3.6 Pseudocode Fungsi F1

```

duplicate_rule1( $S, mask, idx$ )
1   return  $idx < \text{length}_{(S)} - 1 \wedge (S_{idx} \neq S_{idx+1} \vee$ 
       $(S_{idx} = S_{idx+1} \wedge \neg \text{isBitOn}_{(mask, idx+1)}))$ 
```

Gambar 3.7 Pseudocode Fungsi  $duplicate\_rule1$

### 3.4.2 Desain Fungsi G

Pada pseudocode pada gambar 3.4, terdapat perhitungan dengan menggunakan fungsi  $G_{(S,mask,dist)}$  pada baris 9 dan 10. Dimana seperti yang telah dijelaskan pada bagian 2.4.1, fungsi  $G_{(S,mask,dist)}$  merupakan fungsi untuk menghitung jumlah kemungkinan string  $orig$  dari string  $S$  dengan sekali operasi *replace* dengan jarak  $X - dist$ . Nilai dari fungsi  $G_{(S,mask,dist)}$  adalah hasil penjumlahan

lahan dari seluruh *state* yang berhubungan dengan dengan *state*  $G_{(S,mask,dist)}$  yang valid. Gambar 3.8 adalah pseudocode dari fungsi  $G_{(S,mask,dist)}$ . Terdapat tiga kasus *state* yang mungkin, yaitu:

1. *State*  $G_{(S,mask-2^{idx},dist+|S_{idx}-S_{curIdx}|)}$  dengan kasus ketika mengambil karakter posisi  $idx$  pada string  $S$  sebagai karakter posisi  $curIdx$  pada string  $orig$  tanpa melakukan *replace*. Fungsi  $G1_{(S,mask,idx,dist)}$  adalah fungsi yang melakukan validasi terhadap *state* jenis pertama. Gambar ?? adalah pseudocode dari fungsi  $G1_{(S,mask,idx,dist)}$ .
2. *State*  $F_{(S,mask-2^{idx},dist+|S_{idx}+1-S_{curIdx}|)}$  dengan kasus ketika mengambil karakter posisi  $idx$  pada string  $S$  sebagai karakter posisi  $curIdx$  pada string  $orig$  dengan melakukan *replace* dengan karakter setelahnya secara alfabetis. Fungsi  $G2_{(S,mask,idx,dist)}$  adalah fungsi yang melakukan validasi terhadap *state* jenis kedua. Gambar ?? adalah pseudocode dari fungsi  $G2_{(S,mask,idx,dist)}$  dan gambar 3.12 adalah pseudocode dari fungsi  $duplicate\_rule2(S, mask, idx)$ .
3. *State*  $F_{(S,mask-2^{idx},dist+|S_{idx}-1-S_{curIdx}|)}$  dengan kasus ketika mengambil karakter posisi  $idx$  pada string  $S$  sebagai karakter posisi  $curIdx$  pada string  $orig$  dengan melakukan *replace* dengan karakter sebelumnya secara alfabetis. Fungsi  $G3_{(S,mask,idx,dist)}$  adalah fungsi yang melakukan validasi terhadap *state* jenis ketiga. Gambar ?? adalah pseudocode dari fungsi  $G3_{(S,mask,idx,dist)}$  dan gambar 3.13 adalah pseudocode dari fungsi  $duplicate\_rule3(S, mask, idx)$ .

Tabel 3.38 sampai dengan 3.43 adalah simulasi fungsi  $G$  dengan  $S = "kbenh"$ ,  $X = 5$  dan  $dist = 0$ .

Fungsi	Perhitungan Nilai	Nilai
$G_{(behkn,31,0)}$	$G_{(behkn,30,0)} + F_{(behkn,30,1)} + F_{(behkn,30,1)} + G_{(behkn,29,3)} + F_{(behkn,29,4)} + F_{(behkn,29,2)} + G_{(behkn,27,6)} + F_{(behkn,27,7)} + F_{(behkn,27,5)} + G_{(behkn,23,9)} + F_{(behkn,23,10)} + F_{(behkn,23,8)} + G_{(behkn,15,12)} + F_{(behkn,15,13)} + F_{(behkn,15,11)}$	8
$F_{(behkn,15,11)}$	base case	0
$F_{(behkn,15,13)}$	base case	0
$G_{(behkn,15,12)}$	base case	0
$F_{(behkn,23,8)}$	base case	0
$F_{(behkn,23,10)}$	base case	0
$G_{(behkn,23,9)}$	base case	0
$F_{(behkn,27,5)}$	$F_{(behkn,26,8)} + F_{(behkn,25,5)} + F_{(behkn,19,11)} + F_{(behkn,11,14)}$	0
$F_{(behkn,11,14)}$	base case	0
$F_{(behkn,19,11)}$	base case	0
$F_{(behkn,25,5)}$	$memoF_{(behkn,25,5)}$	0
$F_{(behkn,26,8)}$	base case	0
$F_{(behkn,27,7)}$	base case	0
$G_{(behkn,27,6)}$	base case	0
$F_{(behkn,29,2)}$	$F_{(behkn,28,5)} + F_{(behkn,25,5)} + F_{(behkn,21,8)} + F_{(behkn,13,11)}$	1
$F_{(behkn,13,11)}$	base case	0
$F_{(behkn,21,8)}$	base case	0
$F_{(behkn,25,5)}$	$memoF_{(behkn,25,5)}$	0
$F_{(behkn,28,5)}$	$memoF_{(behkn,28,5)}$	1
$F_{(behkn,29,4)}$	$F_{(behkn,28,7)} + F_{(behkn,25,7)} + F_{(behkn,21,10)} + F_{(behkn,13,13)}$	0

Tabel 3.38 Simulasi fungsi  $G$  dengan  $S = "kbenh"$ ,  $X = 5$  dan  $dist = 0$  (1)

Fungsi	Perhitungan Nilai	Nilai
$F_{(behkn,13,13)}$	<i>base case</i>	0
$F_{(behkn,21,10)}$	<i>base case</i>	0
$F_{(behkn,25,7)}$	<i>base case</i>	0
$F_{(behkn,28,7)}$	<i>base case</i>	0
$G_{(behkn,29,3)}$	$G_{(behkn,28,6)} + F_{(behkn,28,5)} +$ $F_{(behkn,28,7)} + G_{(behkn,25,6)} +$ $F_{(behkn,25,7)} + F_{(behkn,25,5)} +$ $G_{(behkn,21,9)} + F_{(behkn,21,10)} +$ $F_{(behkn,21,8)} + G_{(behkn,13,12)} +$ $F_{(behkn,13,13)} + F_{(behkn,13,11)}$	1
$F_{(behkn,13,11)}$	<i>base case</i>	0
$F_{(behkn,13,13)}$	<i>base case</i>	0
$G_{(behkn,13,12)}$	<i>base case</i>	0
$F_{(behkn,21,8)}$	<i>base case</i>	0
$F_{(behkn,21,10)}$	<i>base case</i>	0
$G_{(behkn,21,9)}$	<i>base case</i>	0
$F_{(behkn,25,5)}$	$F_{(behkn,24,11)} + F_{(behkn,17,8)} +$ $F_{(behkn,9,11)}$	0
$F_{(behkn,9,11)}$	<i>base case</i>	0
$F_{(behkn,17,8)}$	<i>base case</i>	0
$F_{(behkn,24,11)}$	<i>base case</i>	0
$F_{(behkn,25,7)}$	<i>base case</i>	0
$G_{(behkn,25,6)}$	<i>base case</i>	0
$F_{(behkn,28,7)}$	<i>base case</i>	0
$F_{(behkn,28,5)}$	$F_{(behkn,24,5)} + F_{(behkn,20,8)} +$ $F_{(behkn,12,11)}$	1
$F_{(behkn,12,11)}$	<i>base case</i>	0
$F_{(behkn,20,8)}$	<i>base case</i>	0

Tabel 3.39 Simulasi fungsi  $G$  dengan  $S = "kbenh"$ ,  $X = 5$  dan  $dist = 0$   
(2)

Fungsi	Perhitungan Nilai	Nilai
$F_{(behkn,24,5)}$	$memoF_{(behkn,24,5)}$	1
$G_{(behkn,28,6)}$	<i>base case</i>	0
$F_{(behkn,30,1)}$	$memoF_{(behkn,30,1)}$	0
$F_{(behkn,30,1)}$	$F_{(behkn,28,1)} + F_{(behkn,26,4)} + F_{(behkn,22,7)} + F_{(behkn,14,10)}$	0
$F_{(behkn,14,10)}$	<i>base case</i>	0
$F_{(behkn,22,7)}$	<i>base case</i>	0
$F_{(behkn,26,4)}$	$memoF_{(behkn,26,4)}$	0
$F_{(behkn,28,1)}$	$memoF_{(behkn,28,1)}$	0
$G_{(behkn,30,0)}$	$G_{(behkn,28,0)} + F_{(behkn,28,1)} + F_{(behkn,28,1)} + G_{(behkn,26,3)} + F_{(behkn,26,4)} + F_{(behkn,26,2)} + G_{(behkn,22,6)} + F_{(behkn,22,7)} + F_{(behkn,22,5)} + G_{(behkn,14,9)} + F_{(behkn,14,10)} + F_{(behkn,14,8)}$	6
$F_{(behkn,14,8)}$	<i>base case</i>	0
$F_{(behkn,14,10)}$	<i>base case</i>	0
$G_{(behkn,14,9)}$	<i>base case</i>	0
$F_{(behkn,22,5)}$	$F_{(behkn,20,8)} + F_{(behkn,18,5)} + F_{(behkn,6,11)}$	0
$F_{(behkn,6,11)}$	<i>base case</i>	0
$F_{(behkn,18,5)}$	$memoF_{(behkn,18,5)}$	0
$F_{(behkn,20,8)}$	<i>base case</i>	0
$F_{(behkn,22,7)}$	<i>base case</i>	0
$G_{(behkn,22,6)}$	<i>base case</i>	0
$F_{(behkn,26,2)}$	$F_{(behkn,24,5)} + F_{(behkn,18,5)} + F_{(behkn,10,8)}$	1
$F_{(behkn,10,8)}$	<i>base case</i>	0
$F_{(behkn,18,5)}$	$memoF_{(behkn,18,5)}$	0

Tabel 3.40 Simulasi fungsi  $G$  dengan  $S = "kbenh"$ ,  $X = 5$  dan  $dist = 0$  (3)

Fungsi	Perhitungan Nilai	Nilai
$F_{(behkn,24,5)}$	$memoF_{(behkn,24,5)}$	1
$F_{(behkn,26,4)}$	$F_{(behkn,24,7)} + F_{(behkn,18,7)} + F_{(behkn,10,10)}$	0
$F_{(behkn,10,10)}$	<i>base case</i>	0
$F_{(behkn,18,7)}$	<i>base case</i>	0
$F_{(behkn,24,7)}$	<i>base case</i>	0
$G_{(behkn,26,3)}$	$G_{(behkn,24,6)} + F_{(behkn,24,5)} + F_{(behkn,24,7)} + G_{(behkn,18,6)} + F_{(behkn,18,7)} + F_{(behkn,18,5)} + G_{(behkn,10,9)} + F_{(behkn,10,10)} + F_{(behkn,10,8)}$	1
$F_{(behkn,10,8)}$	<i>base case</i>	0
$F_{(behkn,10,10)}$	<i>base case</i>	0
$G_{(behkn,10,9)}$	<i>base case</i>	0
$F_{(behkn,18,5)}$	$F_{(behkn,16,11)} + F_{(behkn,2,8)}$	0
$F_{(behkn,2,8)}$	<i>base case</i>	0
$F_{(behkn,16,11)}$	<i>base case</i>	0
$F_{(behkn,18,7)}$	<i>base case</i>	0
$G_{(behkn,18,6)}$	<i>base case</i>	0
$F_{(behkn,24,7)}$	<i>base case</i>	0
$F_{(behkn,24,5)}$	$F_{(behkn,16,5)} + F_{(behkn,8,8)}$	1
$F_{(behkn,8,8)}$	<i>base case</i>	0
$F_{(behkn,16,5)}$	$memoF_{(behkn,16,5)}$	1
$G_{(behkn,24,6)}$	<i>base case</i>	0
$F_{(behkn,28,1)}$	$memoF_{(behkn,28,1)}$	0
$F_{(behkn,28,1)}$	$F_{(behkn,24,1)} + F_{(behkn,20,4)} + F_{(behkn,12,7)}$	0

Tabel 3.41 Simulasi fungsi  $G$  dengan  $S = "kbenh"$ ,  $X = 5$  dan  $dist = 0$   
(4)

Fungsi	Perhitungan Nilai	Nilai
$F_{(behkn,12,7)}$	$base\ case$	0
$F_{(behkn,20,4)}$	$memoF_{(behkn,20,4)}$	0
$F_{(behkn,24,1)}$	$memoF_{(behkn,24,1)}$	0
$G_{(behkn,28,0)}$	$G_{(behkn,24,0)} + F_{(behkn,24,1)} + F_{(behkn,24,1)} + G_{(behkn,20,3)} + F_{(behkn,20,4)} + F_{(behkn,20,2)} + G_{(behkn,12,6)} + F_{(behkn,12,7)} + F_{(behkn,12,5)}$	4
$F_{(behkn,12,5)}$	$F_{(behkn,8,8)} + F_{(behkn,4,5)}$	0
$F_{(behkn,4,5)}$	$memoF_{(behkn,4,5)}$	0
$F_{(behkn,8,8)}$	$base\ case$	0
$F_{(behkn,12,7)}$	$base\ case$	0
$G_{(behkn,12,6)}$	$base\ case$	0
$F_{(behkn,20,2)}$	$F_{(behkn,16,5)} + F_{(behkn,4,5)}$	1
$F_{(behkn,4,5)}$	$memoF_{(behkn,4,5)}$	0
$F_{(behkn,16,5)}$	$memoF_{(behkn,16,5)}$	1
$F_{(behkn,20,4)}$	$F_{(behkn,16,7)} + F_{(behkn,4,7)}$	0
$F_{(behkn,4,7)}$	$base\ case$	0
$F_{(behkn,16,7)}$	$base\ case$	0
$G_{(behkn,20,3)}$	$G_{(behkn,16,6)} + F_{(behkn,16,5)} + F_{(behkn,16,7)} + G_{(behkn,4,6)} + F_{(behkn,4,7)} + F_{(behkn,4,5)}$	1
$F_{(behkn,4,5)}$	$F_{(behkn,0,11)}$	0
$F_{(behkn,0,11)}$	$base\ case$	0
$F_{(behkn,4,7)}$	$base\ case$	0
$G_{(behkn,4,6)}$	$base\ case$	0
$F_{(behkn,16,7)}$	$base\ case$	0
$F_{(behkn,16,5)}$	$F_{(behkn,0,5)}$	1

Tabel 3.42 Simulasi fungsi  $G$  dengan  $S = "kbenh"$ ,  $X = 5$  dan  $dist = 0$  (5)

Fungsi	Perhitungan Nilai	Nilai
$F_{(behkn,0,5)}$	<i>base case</i>	1
$G_{(behkn,16,6)}$	<i>base case</i>	0
$F_{(behkn,24,1)}$	$memoF_{(behkn,24,1)}$	0
$F_{(behkn,24,1)}$	$F_{(behkn,16,1)} + F_{(behkn,8,4)}$	0
$F_{(behkn,8,4)}$	$memoF_{(behkn,8,4)}$	0
$F_{(behkn,16,1)}$	$memoF_{(behkn,16,1)}$	0
$G_{(behkn,24,0)}$	$G_{(behkn,16,0)} + F_{(behkn,16,1)} + F_{(behkn,16,1)} + G_{(behkn,8,3)} + F_{(behkn,8,4)} + F_{(behkn,8,2)}$	2
$F_{(behkn,8,2)}$	$F_{(behkn,0,5)}$	1
$F_{(behkn,0,5)}$	<i>base case</i>	1
$F_{(behkn,8,4)}$	$F_{(behkn,0,7)}$	0
$F_{(behkn,0,7)}$	<i>base case</i>	0
$G_{(behkn,8,3)}$	$G_{(behkn,0,6)} + F_{(behkn,0,5)} + F_{(behkn,0,7)}$	1
$F_{(behkn,0,7)}$	<i>base case</i>	0
$F_{(behkn,0,5)}$	<i>base case</i>	1
$G_{(behkn,0,6)}$	<i>base case</i>	0
$F_{(behkn,16,1)}$	$memoF_{(behkn,16,1)}$	0
$F_{(behkn,16,1)}$	$F_{(behkn,0,1)}$	0
$F_{(behkn,0,1)}$	<i>base case</i>	0
$G_{(behkn,16,0)}$	$G_{(behkn,0,0)} + F_{(behkn,0,1)} + F_{(behkn,0,1)}$	0
$F_{(behkn,0,1)}$	<i>base case</i>	0
$F_{(behkn,0,1)}$	<i>base case</i>	0
$G_{(behkn,0,0)}$	<i>base case</i>	0

Tabel 3.43 Simulasi fungsi  $G$  dengan  $S = "kbenh"$ ,  $X = 5$  dan  $dist = 0$  (6)

```

G( $S, mask, dist$ )
1 if  $dist > bound \vee mask = 0$ 
2   return 0
3 if  $memoG_{(S,mask,dist)} \neq \emptyset$ 
4   return  $memoG_{(S,mask,dist)}$ 
5  $numberOfSetBit = \text{size}_{(setBit}_{(mask))}$ 
6  $retVal = 0$ 
7 for  $i = 0$  to  $numberOfSetBit - 1$ 
8   if  $setBit_{(mask)i} \geq length_{(S)}$ 
9     break
10   $ret = ret + G1_{(S,mask, setBit_{(mask)i}, dist)}$ 
11   $ret = ret + G2_{(S,mask, setBit_{(mask)i}, dist)}$ 
12   $ret = ret + G3_{(S,mask, setBit_{(mask)i}, dist)}$ 
13 return  $memoF_{(S,mask,dist)} = ret$ 

```

Gambar 3.8 Pseudocode Fungsi G

```

G1( $S, mask, idx, dist$ )
1  $curIdx = length_{(S)} - \text{size}_{(setBit}_{(mask))}$ 
2 if  $idx = length_{(S)} - 1 \vee \text{duplicateRule1}_{(S,mask,idx)}$ 
3   return  $G_{(S,mask-2^{idx},dist+|S_{idx}-S_{curIdx}|)}$ 

```

Gambar 3.9 Pseudocode Fungsi G1

```

G2( $S, mask, idx, dist$ )
1    $curIdx = \text{length}_{(S)} - \text{size}_{(\text{setBit}_{(mask)})}$ 
2   if  $idx = \text{length}_{(S)} - 1 \vee (\text{duplicateRule2}_{(S, mask, idx)}) \wedge$ 
       $(\text{duplicateRule1}_{(S, mask, idx)})$ 
3     return  $F_{(S, mask - 2^{idx}, dist + |(S_{idx+1}) - S_{curIdx}|)}$ 

```

Gambar 3.10 Pseudocode Fungsi G2

```

G3( $S, mask, idx, dist$ )
1    $curIdx = \text{length}_{(S)} - \text{size}_{(\text{setBit}_{(mask)})}$ 
2   if  $(idx = \text{length}_{(S)} - 1 \vee \text{duplicateRule1}_{(S, mask, idx)}) \wedge$ 
       $(idx = 0 \vee \text{duplicateRule3}_{(S, mask, idx)})$ 
3     return  $F_{(S, mask - 2^{idx}, dist + |(S_{idx-1}) - S_{curIdx}|)}$ 

```

Gambar 3.11 Pseudocode Fungsi G3

```

duplicate_rule2( $S, mask, idx$ )
1   return  $idx < \text{length}_{(S)} - 1 \wedge (\text{charFirstPos}_{(S, S_{idx+1})}$ 
     $= \emptyset \vee (\text{charFirstPos}_{(S, S_{idx+1})} \neq \emptyset$ 
     $\wedge \neg \text{isBitOn}_{(mask, \text{charFirstPos}_{(S, S_{idx+1})})}))$ 

```

Gambar 3.12 Pseudocode Fungsi `duplicate_rule2`

```
duplicate_rule3( $S, mask, idx$ )
1   return  $idx > 0 \wedge (charLastPos_{(S, S_{idx}-1)}$ 
       $= \emptyset \vee (charLastPos_{(S, S_{idx}-1)} \neq \emptyset$ 
       $\wedge isBitOn_{(mask, charLastPos_{(S, S_{idx}-1)})}))$ 
```

Gambar 3.13 Pseudocode Fungsi duplicate\_rule3

*Halaman ini sengaja dikosongkan*

## **BAB IV**

### **IMPLEMENTASI**

Pada bab ini dijelaskan mengenai implementasi dari desain algoritma penyelesaian permasalahan klasik SPOJ 9967 *Playing With Words*.

#### **4.1 Lingkungan Implementasi**

Lingkungan implementasi dalam pembuatan Tugas Akhir ini meliputi perangkat keras dan perangkat lunak yang digunakan untuk melakukan proses pendekatan algoritma dynamic programming untuk permasalahan klasik SPOJ 9967 *Playing With Words* adalah sebagai berikut:

1. Perangkat Keras.
  - Processor Intel(R) Core(TM) i5-4210U CPU @ 1.70GHz.
  - Memory 8 GB.
2. Perangkat Lunak.
  - Sistem operasi Linux Mint 17.1 Rebecca 64 bit.
  - Text editor vim
  - Compiler g++ versi 4.8.4.

#### **4.2 Rancangan Data**

Pada subbab ini dijelaskan mengenai desain data masukan yang diperlukan untuk melakukan proses algoritma, dan data keluaran yang dihasilkan oleh program.

### 4.2.1 Data Masukan

Data masukan adalah data yang akan diproses oleh program sebagai masukan menggunakan algoritma yang telah dirancang dalam tugas akhir ini.

Data masukan berupa berkas teks yang berisi data dengan format yang telah ditentukan pada deskripsi permasalahan klasik SPOJ 9967 *Playing With Words*. Pada masing - masing berkas data masukan, baris pertama berupa sebuah bilangan bulat yang merepresentasikan jumlah kasus uji yang ada pada berkas tersebut. Untuk setiap kasus uji, masukan berupa sebuah baris masukan yang terdiri dari dua buah string  $ad1$  dan  $ad2$ , yang merepresentasikan string hasil transformasi dari string  $orig1$  dan  $orig2$  secara berturut-turut, diikuti oleh sebuah bilangan bulat  $X$  yang merepresentasikan  $dist(ad1, orig1) + dist(ad2, orig2)$ .

### 4.2.2 Data Keluaran

Data keluaran yang dihasilkan oleh program hanya berupa satu nilai, yaitu jumlah kemungkinan string  $orig1$  dan  $orig2$  yang mungkin membentuk string  $ad1$  dan  $ad2$ .

## 4.3 Implementasi Algoritma

Pada subbab ini akan dijelaskan tentang implementasi proses algoritma secara keseluruhan berdasarkan desain yang telah dijelaskan pada bab III.

### 4.3.1 Header-Header yang Diperlukan

Implementasi algoritma dengan teknik *meet in the middle* dan *dynamic programming* untuk menyelesaikan permasalahan klasik SPOJ 9967 *Playing With Words* membutuhkan lima buah header yaitu `stdio`, `algorithm`, `vector`, `string` dan `cstring`, seperti yang terlihat pada

kode sumber 4.3.1.

```
1 #include <cstdio>
2 #include <algorithm>
3 #include <vector>
4 #include <string>
5 #include <cstring>
```

Kode Sumber 4.3.1 Header yang diperlukan

Header cstdio berisi modul untuk menerima masukan dan memberikan keluaran. Header vector berisi struktur data yang digunakan untuk menyimpan data himpunan index bit menyalah dari sebuah bilangan bulat. Header algorithm berisi modul yang memiliki fungsi - fungsi yang sangat berguna dalam membantu mengimplementasi algoritma yang telah dibangun. Contohnya adalah fungsi *max* dan *sort*. Header string berisi modul untuk menyimpan data berupa text. Header cstring berisi modul yang memiliki fungsi-fungsi untuk melakukan pemrosesan string. Contoh fungsi yang membantu mengimplementasikan algoritma yang dibangun adalah fungsi *memset*.

### 4.3.2 Variabel Global

Variabel global digunakan untuk memudahkan dalam mengakses data yang digunakan lintas fungsi. Kode sumber implementasi variabel global dapat dilihat pada kode sumber 4.3.2.

```

1  using namespace std;
2
3  vector<int> set_bit[(1 << 11)];
4  string S[2];
5  int charLastPos[2][50];
6  int charFirstPos[2][50];
7  int X, bound;
8  int memoF[2][(1 << 10) + 2][250 + 2];
9  int memoG[2][(1 << 10) + 2][250 + 2];
10 int maxMask[2];

```

Kode Sumber 4.3.2 Variabel global

### 4.3.3 Implementasi Fungsi Main

Fungsi main adalah implementasi algoritma yang dirancang pada gambar 3.1. Implementasi fungsi main dapat dilihat pada kode sumber 4.3.3.

```

1  int main() {
2      preprocess();
3      int t;
4      scanf("%d", &t);
5      for (int tc=1; tc<=t; tc++) {
6          readInput();
7          init();
8          long long ans = solveProblem();
9          writeOutput(tc, ans);
10     }
11     return 0;
12 }

```

Kode Sumber 4.3.3 Fungsi main

### 4.3.4 Implementasi Fungsi Preprocess

Fungsi preprocess adalah implementasi dari hasil perancangan pada pseudocode 3.2. Implementasi dari fungsi preprocess dapat dilihat

pada kode sumber 4.3.4.

```

1 void preprocess() {
2     for (int i = 0; i < (1 << 10); i++) {
3         for (int j = 0; j < 10; j++) {
4             if (isBitOn(i, j))
5                 ↪ set_bit[i].
6                 ↪ push_back(j);
7 }

```

Kode Sumber 4.3.4 Fungsi preprocess

### 4.3.5 Implementasi Fungsi ReadInput

Fungsi readInput akan membaca masukan dari berkas uji untuk setiap kasus ujinya. Pada awalnya, fungsi akan membaca masukan string *ad1*, lalu dilanjutkan dengan membaca string *ad2* dan diakhiri dengan membaca sebuah bilangan bulat *X*. Implementasi fungsi readInput dapat dilihat pada kode sumber 4.3.5.

```

1 void readInput() {
2     char dummySt[30];
3     scanf("%s", dummySt);
4     S[0] = dummySt;
5     scanf("%s", dummySt);
6     S[1] = dummySt;
7     scanf("%d", &X);
8 }

```

Kode Sumber 4.3.5 Fungsi readInput

### 4.3.6 Implementasi Fungsi Init

Implementasi dari fungsi init dapat dilihat pada kode sumber 4.3.6.

```

1 void init() {
2     memset(charLastPos, -1, sizeof
3             ↪ charLastPos);
4     memset(charFirstPos, -1, sizeof
5             ↪ charFirstPos);
6     memset(memoF, -1, sizeof memoF);
7     memset(memoG, -1, sizeof memoG);
8     for (int idx = 0; idx < 2; idx++) {
9         sort(S[idx].begin(), S[idx].end()
10            ↪ );
11         maxMask[idx] = (1 << S[idx].
12             ↪ length()) - 1;
13         for (int i = 0; i < S[idx].length
14             ↪ (); i++) {
15             ↪
16             ↪ charLastPos[idx][S[idx][i]
17             ↪ ] - 'a'] = i;
18             ↪
19             ↪ if (charFirstPos[idx][S[
20                 ↪ idx][i] - 'a' ==
21                 ↪ -1) {
22                 ↪
23                 ↪ charFirstPos[idx]
24                     ↪ ][S[idx][i]
25                     ↪ - 'a'] = i
26                     ↪ ;
27             }
28         }
29     }
30     bound = min(X, 250);
31 }
```

Kode Sumber 4.3.6 Fungsi init

### 4.3.7 Implementasi Fungsi Solve

Fungsi solve adalah implementasi dari desain algoritma pada gambar 3.4 dimana algoritma tersebut adalah hasil perancangan berdasarkan persamaan 2.4.1. Implementasi dari fungsi solve dapat dilihat pada kode sumber 4.3.7.

```

1 long long solveProblem() {
2     long long ret = 0;
3     for (int dist = 0; dist <= min(250, X);
4         ↪ dist++) {
5         int rem = X - dist;
6         if (rem > 250) continue;
7         if (rem < 0) break;
8         ret += F(0, maxMask[0], bound -
9             ↪ dist) * G(1, maxMask[1],
10            ↪ bound - rem);
11        }
10    return ret;
11 }
```

Kode Sumber 4.3.7 Fungsi solve

### 4.3.8 Implementasi Fungsi F

terdapat fungsi F yang telah dirancang algoritmanya pada subbab 3.4.1. Implementasi dari fungsi F dapat dilihat pada kode sumber 4.3.8 dan 4.3.9.

```

1     for (int i=0; i<NSB; i++) {
2         if (set_bit[mask][i] >= S[idx].
3             ↪ length()) break;
4         ret += F1(idx, mask, set_bit[mask
5             ↪ ][i], dist);
6     }
5     return memoF[idx][mask][dist] = ret;
6 }
```

Kode Sumber 4.3.8 Fungsi F (1)

```

1           if (set_bit[mask][i] >= S[idx].
2               ↪ length()) break;
3           ret += F1(idx, mask, set_bit[mask
4               ↪ ][i], dist);
5       }
6   return memoF[idx][mask][dist] = ret;
7 }
```

Kode Sumber 4.3.9 Fungsi F (2)

### 4.3.9 Implementasi Fungsi F1

Fungsi F1 adalah implementasi dari perancangan pada pseudocode 3.6 yang dirancang berdasarkan persamaan 2.4.3. Implementasi dari fungsi F1 dapat dilihat pada kode sumber 4.3.10.

```

1 long long F1(int idx, int mask, int charIdx, int
2     ↪ dist) {
3     int curIdx = S[idx].length() -
4         ↪ __builtin_popcount(mask);
5     if (charIdx == S[idx].length() - 1 ||
6         ↪ duplicate_rule1(idx, mask, charIdx)
7         ↪ ) {
8         return F(idx, mask - (1 <<
9             ↪ charIdx), dist + abs(S[idx
10                ↪ ][charIdx] - S[idx][curIdx
11                ↪ ]));
12     }
13     return 0;
14 }
```

Kode Sumber 4.3.10 Fungsi F1

### 4.3.10 Implementasi Fungsi G

Implementasi dari fungsi G dapat dilihat pada kode sumber 4.3.11. Algoritma pada fungsi F menggunakan pendekatan *dynamic pro-*

gramming.

```

1 long long G(int idx, int mask, int dist) {
2     if (dist > bound || mask == 0) return 0;
3     if (memoG[idx][mask][dist] != -1) return
4         ↪ memoG[idx][mask][dist];
5     int NSB = set_bit[mask].size();
6     long long ret = 0;
7     for (int i=0; i<NSB; i++) {
8         if (set_bit[mask][i] >= S[idx].
9             ↪ length()) break;
10        ret += G1(idx, mask, set_bit[mask
11            ↪ ][i], dist);
12        ret += G2(idx, mask, set_bit[mask
13            ↪ ][i], dist);
13    }
12    return memoG[idx][mask][dist] = ret;
13 }
```

Kode Sumber 4.3.11 Fungsi G

#### 4.3.11 Implementasi Fungsi G1

Fungsi G1 adalah implementasi dari perancangan pada pseudocode 3.9 yang dirancang berdasarkan persamaan 2.4.6. Implementasi dari fungsi G1 dapat dilihat pada kode sumber 4.3.12 dan 4.3.13.

```

1 long long G1(int idx, int mask, int charIdx, int
2     ↪ dist) {
3     int curIdx = S[idx].length() -
4         ↪ __builtin_popcount(mask);
5     if (charIdx == S[idx].length() - 1 ||
6         ↪ duplicate_rule1(idx, mask, charIdx)
7         ↪ ) {
```

Kode Sumber 4.3.12 Fungsi G1 (1)

```

1           return G(idx, mask - (1 <<
2                         ↪ charIdx), dist + abs(S[idx]
2                         ↪ ][charIdx] - S[idx][curIdx
2                         ↪ ]));
3     }
4 }
```

Kode Sumber 4.3.13 Fungsi G1 (2)

### 4.3.12 Implementasi Fungsi G2

Fungsi G2 adalah implementasi dari perancangan pada pseudocode 3.10 yang dirancang berdasarkan persamaan 2.4.7. Implementasi dari fungsi G2 dapat dilihat pada kode sumber 4.3.14.

```

1 long long G2(int idx, int mask, int charIdx, int
2   ↪ dist) {
2     int curIdx = S[idx].length() -
2       ↪ __builtin_popcount(mask);
3     if (charIdx == S[idx].length() - 1 || (
3       ↪ duplicate_rule2(idx, mask, charIdx)
3       ↪ && duplicate_rule1(idx, mask,
3       ↪ charIdx))) {
4       return F(idx, mask - (1 <<
4         ↪ charIdx), dist + abs((S[idx]
4           ↪ ][charIdx] + 1) - S[idx][
4             ↪ curIdx]));
5     }
6   return 0;
7 }
```

Kode Sumber 4.3.14 Fungsi G2

### 4.3.13 Implementasi Fungsi G3

Fungsi G3 adalah implementasi dari perancangan pada pseudocode 3.11 yang dirancang berdasarkan persamaan 2.4.8. Implementasi

dari fungsi G3 dapat dilihat pada kode sumber 4.3.15.

```

1 long long G3(int idx, int mask, int charIdx, int
   ↪ dist) {
2     int curIdx = S[idx].length() -
      ↪ __builtin_popcount(mask);
3     if ((charIdx == S[idx].length() - 1 ||
      ↪ duplicate_rule1(idx, mask, charIdx)
      ↪ ) && (charIdx == 0 ||
      ↪ duplicate_rule3(idx, mask, charIdx)
      ↪ ) ) {
4         return F(idx, mask - (1 <<
           ↪ charIdx), dist + abs((S[idx]
           ↪ ][charIdx] - 1) - S[idx][
           ↪ curIdx]));
5     }
6     return 0;
7 }
```

Kode Sumber 4.3.15 Fungsi G3

#### 4.3.14 Implementasi Fungsi Duplicate Rule 1

Fungsi `duplicate_rule1` adalah implementasi dari perancangan pada pseudocode 3.7 yang dirancang berdasarkan persamaan 2.4.4. Implementasi dari fungsi `duplicate_rule1` dapat dilihat pada kode sumber 4.3.16 and 4.3.17.

```

1 bool duplicate_rule1(int idx, int mask, int
   ↪ charIdx) {
2     return (charIdx < S[idx].length() - 1
   ↪ && (S[idx][charIdx] != S[idx][
   ↪ charIdx + 1])
```

Kode Sumber 4.3.16 Fungsi `duplicate_rule1` (1)

```

1          || (S[idx][charIdx] == S[idx][
2              ↪ charIdx + 1]
2      && !isBitOn(mask, charIdx + 1)))
2      ↪ ;
3  }

```

Kode Sumber 4.3.17 Fungsi duplicate\_rule1 (2)

#### 4.3.15 Implementasi Fungsi Duplicate Rule 2

Fungsi duplicate\_rule2 adalah implementasi dari perancangan pada pseudocode 3.12 yang dirancang berdasarkan persamaan 2.4.9. Implementasi dari fungsi duplicate\_rule2 dapat dilihat pada kode sumber 4.3.18.

```

1 bool duplicate_rule2(int idx, int mask, int
2     ↪ charIdx) {
2     return (charIdx < S[idx].length() - 1
3             && (charFirstPos[idx][(S[idx][
4                 ↪ charIdx] + 1) - 'a'] == -1
4             || (charFirstPos[idx][(S[idx][
5                 ↪ charIdx]) + 1 - 'a'] != -1
5             && !isBitOn(mask, charFirstPos[
5                 ↪ idx][S[idx][charIdx] + 1 -
5                 ↪ 'a']])));
6 }

```

Kode Sumber 4.3.18 Fungsi duplicate\_rule2

#### 4.3.16 Implementasi Fungsi Duplicate Rule 3

Fungsi duplicate\_rule3 adalah implementasi dari perancangan pada pseudocode 3.13 yang dirancang berdasarkan persamaan 2.4.10. Implementasi dari fungsi duplicate\_rule1 dapat dilihat pada kode sumber 4.3.19.

```
1 bool duplicate_rule3(int idx, int mask, int
2   ↪ charIdx) {
3     return (charIdx > 0
4       && (charLastPos[idx][(S[idx] [
5         ↪ charIdx] - 1) - 'a']] == -1
6       || (charLastPos[idx][(S[idx] [
7         ↪ charIdx] - 1) - 'a']] != -1
8       && isBitOn(mask, charLastPos[idx]
9         ↪ ][(S[idx][charIdx] - 1) - 'a']));
10 }
```

Kode Sumber 4.3.19 Fungsi `duplicate_rule3`

*Halaman ini sengaja dikosongkan*

## **BAB V**

### **UJI COBA DAN EVALUASI**

Pada bab ini dijelaskan tentang uji coba dan evaluasi dari implementasi yang telah dilakukan pada tugas akhir ini.

#### **5.1 Lingkungan Uji Coba**

Linkungan uji coba yang digunakan adalah salah satu sistem yang digunakan situs penilaian daring SPOJ, yaitu kluster *Cube* dengan spesifikasi sebagai berikut:

1. Perangkat Keras.
  - Processor Intel(R) Pentium G860 CPU @ 3GHz.
  - Memory 1536 MB.
2. Perangkat Lunak.
  - Compiler gcc versi 6.3.

#### **5.2 Uji Coba Kebenaran**

Uji coba kebenaran dilakukan dengan mengirimkan kode sumber program ke dalam situs penilaian daring SPOJ dan melakukan hasil uji coba kasus sederhana dengan langkah-langkah sesuai dengan algoritma yang telah dirancang dengan keluaran sistem. Permasalahan yang diselesaikan adalah permasalahan klasik SPOJ 9967 *Playing With Words*. Hasil uji coba dengan waktu terbaik pada situs SPOJ ditunjukkan pada gambar 5.1.

19552011	■ 301F-06-04 16:03:31	Playing with Words	accepted with 100.00%	0.98	19M	CPP14-CLANG
----------	--------------------------	--------------------	--------------------------	------	-----	-------------

Gambar 5.1 Hasil uji coba pada situs penilaian SPOJ



Gambar 5.2 Grafik hasil uji coba pada situs SPOJ sebanyak 30 kali

Selain itu, dilakukan pengujian sebanyak 30 kali pada situs penilaian daring SPOJ untuk melihat variasi waktu dan memori yang dibutuhkan program. Hasil uji coba sebanyak 30 kali dapat dilihat pada gambar 5.2, 5.3, 5.4 dan 5.5.

Dari hasil uji coba pada gambar 5.2, 5.3, 5.4 dan 5.5 dapat kita tarik beberapa informasi seperti yang tertera pada tabel 5.1.

Selanjutnya akan dilakukan uji coba menggunakan kasus uji yang diberikan pada deskripsi permasalahan klasik SPOJ 9967 *Playing With Words*. Pada deskripsi permasalahan, terdapat dua kasus. Kasus pertama yaitu kasus dimana nilai dari string  $ad1 = c$ , string  $ad2 = n$  dan  $X = 1$ .

Sesuai dengan algoritma yang telah dirancang, berdasarkan persamaan 2.4.1 yang sudah ditransformasi kedalam bentuk pseudocode pada gambar 2.4.1, algoritma akan melakukan iterasi variabel  $dist$  dari 0 hingga  $X$ .

19552032		2017-06-04 18:44:54	Playing with Words	<b>accepted</b> edit - isdone it	1.01	19M	CPP14-CLANG
19552031		2017-06-04 18:44:58	Playing with Words	<b>accepted</b> edit - isdone it	1.00	19M	CPP14-CLANG
19552030		2017-06-04 18:44:58	Playing with Words	<b>accepted</b> edit - isdone it	1.01	19M	CPP14-CLANG
19552028		2017-06-04 18:44:58	Playing with Words	<b>accepted</b> edit - isdone it	0.99	19M	CPP14-CLANG
19552029		2017-06-04 18:44:58	Playing with Words	<b>accepted</b> edit - isdone it	1.01	19M	CPP14-CLANG
19552024		2017-06-04 18:44:58	Playing with Words	<b>accepted</b> edit - isdone it	1.00	19M	CPP14-CLANG
19552023		2017-06-04 18:44:58	Playing with Words	<b>accepted</b> edit - isdone it	1.02	19M	CPP14-CLANG
19552022		2017-06-04 18:44:58	Playing with Words	<b>accepted</b> edit - isdone it	1.00	19M	CPP14-CLANG
19552020		2017-06-04 18:44:58	Playing with Words	<b>accepted</b> edit - isdone it	1.02	19M	CPP14-CLANG
19552019		2017-06-04 18:44:58	Playing with Words	<b>accepted</b> edit - isdone it	1.00	19M	CPP14-CLANG
19552017		2017-06-04 18:44:58	Playing with Words	<b>accepted</b> edit - isdone it	1.01	19M	CPP14-CLANG
19552016		2017-06-04 18:42:58	Playing with Words	<b>accepted</b> edit - isdone it	1.01	19M	CPP14-CLANG
19552015		2017-06-04 18:42:58	Playing with Words	<b>accepted</b> edit - isdone it	0.99	19M	CPP14-CLANG
19552013		2017-06-04 18:42:58	Playing with Words	<b>accepted</b> edit - isdone it	1.00	19M	CPP14-CLANG
19552012		2017-06-04 18:42:58	Playing with Words	<b>accepted</b> edit - isdone it	1.00	19M	CPP14-CLANG

Gambar 5.3 Hasil pengujian sebanyak 30 kali pada situs penilaian daring SPOJ (1)

19552011		2017-06-04 18:43:51	Playing with Words	<b>accepted</b> edit - isdone it	0.98	19M	CPP14-CLANG
19552010		2017-06-04 18:43:58	Playing with Words	<b>accepted</b> edit - isdone it	1.01	19M	CPP14-CLANG
19552009		2017-06-04 18:43:58	Playing with Words	<b>accepted</b> edit - isdone it	1.00	19M	CPP14-CLANG
19552008		2017-06-04 18:43:58	Playing with Words	<b>accepted</b> edit - isdone it	1.00	19M	CPP14-CLANG
19552005		2017-06-04 18:43:58	Playing with Words	<b>accepted</b> edit - isdone it	1.00	19M	CPP14-CLANG

Gambar 5.4 Hasil pengujian sebanyak 30 kali pada situs penilaian daring SPOJ (2)

L9551998	<input checked="" type="checkbox"/> 2017-06-04 18:42:30	Playing with Words	<b>accepted</b> juli - irwanita	1.00	19M	CPP14-CLANG
L9551997	<input checked="" type="checkbox"/> 2017-06-04 18:42:35	Playing with Words	<b>accepted</b> juli - irwanita	1.00	19M	CPP14-CLANG
L9551996	<input checked="" type="checkbox"/> 2017-06-04 18:42:30	Playing with Words	<b>accepted</b> juli - irwanita	1.02	19M	CPP14-CLANG
L9551993	<input checked="" type="checkbox"/> 2017-06-04 18:42:18	Playing with Words	<b>accepted</b> juli - irwanita	1.01	19M	CPP14-CLANG
L9551991	<input checked="" type="checkbox"/> 2017-06-04 18:42:07	Playing with Words	<b>accepted</b> juli - irwanita	1.00	19M	CPP14-CLANG
L9551989	<input checked="" type="checkbox"/> 2017-06-04 18:41:38	Playing with Words	<b>accepted</b> juli - irwanita	1.02	19M	CPP14-CLANG
L9551988	<input checked="" type="checkbox"/> 2017-06-04 18:41:33	Playing with Words	<b>accepted</b> juli - irwanita	0.99	19M	CPP14-CLANG
L9551986	<input checked="" type="checkbox"/> 2017-06-04 18:41:46	Playing with Words	<b>accepted</b> juli - irwanita	1.00	19M	CPP14-CLANG
L9551985	<input checked="" type="checkbox"/> 2017-06-04 18:41:35	Playing with Words	<b>accepted</b> juli - irwanita	1.00	19M	CPP14-CLANG
L9551982	<input checked="" type="checkbox"/> 2017-06-04 18:41:18	Playing with Words	<b>accepted</b> juli - irwanita	0.99	19M	CPP14-CLANG

Gambar 5.5 Hasil pengujian sebanyak 30 kali pada situs penilaian daring SPOJ (3)

Waktu Maksimal	1.08 detik
Waktu Minimal	0.98 detik
Waktu Rata-Rata	1.01 detik
Memori Maksimal	19 MB
Memori Minimal	19 MB
Memori Rata-Rata	19 MB

Tabel 5.1 Kecepatan maksimal, minimal dan rata-rata dari hasil uji coba pengumpulan 30 kali pada situs pengujian daring SPOJ

Fungsi	Perhitungan Nilai	Nilai
$F_{(c,0,1)}$	<i>base case</i>	1
$F_{(c,1,1)}$	$F_{(c,0,1)}$	1

Tabel 5.2 Simulasi perhitungan jumlah kombinasi string  $orig1$  tanpa operasi *replace* dengan  $dist = 0$  pada kasus string  $ad1 = c$ , string  $ad2 = n$  dan  $X = 1$

Fungsi	Perhitungan Nilai	Nilai
$G_{(n,0,0)}$	<i>base case</i>	0
$F_{(n,0,1)}$	<i>base case</i>	1
$F_{(n,0,1)}$	<i>base case</i>	1
$G_{(n,1,0)}$	$G_{(n,0,0)} + F_{(n,0,1)} + F_{(n,0,1)}$	2

Tabel 5.3 Simulasi perhitungan jumlah kombinasi string  $orig2$  dengan operasi *replace* dengan  $dist = 0$  pada kasus string  $ad1 = c$ , string  $ad2 = n$  dan  $X = 1$

Fungsi	Perhitungan Nilai	Nilai
$G_{(c,0,1)}$	<i>base case</i>	0
$F_{(c,0,2)}$	<i>base case</i>	0
$F_{(c,0,2)}$	<i>base case</i>	0
$G_{(c,1,1)}$	$G_{(c,0,1)} + F_{(c,0,2)} + F_{(c,0,2)}$	0

Tabel 5.4 Simulasi perhitungan jumlah kombinasi string  $orig1$  dengan operasi *replace* dengan  $dist = 0$  pada kasus string  $ad1 = c$ , string  $ad2 = n$  dan  $X = 1$

Fungsi	Perhitungan Nilai	Nilai
$F_{(n,0,0)}$	<i>base case</i>	0
$F_{(n,1,0)}$	$F_{(n,0,0)}$	0

Tabel 5.5 Simulasi perhitungan jumlah kombinasi string  $orig2$  tanpa operasi *replace* dengan  $dist = 0$  pada kasus string  $ad1 = c$ , string  $ad2 = n$  dan  $X = 1$

Fungsi	Perhitungan Nilai	Nilai
$F_{(c,0,0)}$	<i>base case</i>	0
$F_{(c,1,0)}$	$F_{(c,0,0)}$	0

Tabel 5.6 Simulasi perhitungan jumlah kombinasi string  $orig1$  tanpa operasi *replace* dengan  $dist = 1$  pada kasus string  $ad1 = c$ , string  $ad2 = n$  dan  $X = 1$

Fungsi	Perhitungan Nilai	Nilai
$G_{(n,0,1)}$	<i>base case</i>	0
$F_{(n,0,2)}$	<i>base case</i>	0
$F_{(n,0,2)}$	<i>base case</i>	0
$G_{(n,1,1)}$	$G_{(n,0,1)} + F_{(n,0,2)} + F_{(n,0,2)}$	0

Tabel 5.7 Simulasi perhitungan jumlah kombinasi string  $orig2$  dengan operasi *replace* dengan  $dist = 1$  pada kasus string  $ad1 = c$ , string  $ad2 = n$  dan  $X = 1$

Pada iterasi  $dist = 0$ , nilai jawaban akhir akan ditambahkan dengan  $F_{(c,1,1)}$ , yang merupakan jumlah kombinasi string  $orig1$  tanpa operasi *replace* dengan jarak 0 terhadap string  $ad1$ , yang berdasarkan ilustrasi pada tabel 5.2, bernilai 1 yang dikalikan dengan  $G_{(n,1,0)}$ , yang merupakan jumlah kombinasi string  $orig2$  dengan operasi *replace* yang memiliki jarak 1 terhadap string  $ad2$ , yang berdasarkan ilustrasi pada tabel 5.3, bernilai 2. Lalu nilai jawaban akhir akan ditambahkan dengan  $G_{(c,1,1)}$ , yang merupakan jumlah kombinasi string  $orig1$  dengan operasi *replace* dengan jarak 0 terhadap string  $ad1$ , yang berdasarkan ilustrasi pada tabel 5.4, bernilai 0 yang dikalikan dengan  $F_{(n,1,0)}$ , yang merupakan jumlah kombinasi string  $orig2$  tanpa operasi *replace* yang memiliki jarak 1 terhadap string  $ad2$ , yang berdasarkan ilustrasi pada tabel 5.5, bernilai 0. Sehingga nilai jawaban dari iterasi 0 adalah 2 dan nilai jawaban akhir hingga pada iterasi 0 adalah 2.

Fungsi	Perhitungan Nilai	Nilai
$G_{(c,0,0)}$	<i>base case</i>	0
$F_{(c,0,1)}$	<i>base case</i>	1
$F_{(c,0,1)}$	<i>base case</i>	1
$G_{(c,1,0)}$	$G_{(c,0,0)} + F_{(c,0,1)} + F_{(c,0,1)}$	2

Tabel 5.8 Simulasi perhitungan jumlah kombinasi string  $orig1$  dengan operasi *replace* dengan  $dist = 1$  pada kasus string  $ad1 = c$ , string  $ad2 = n$  dan  $X = 1$

Fungsi	Perhitungan Nilai	Nilai
$F_{(n,0,1)}$	<i>base case</i>	1
$F_{(n,1,1)}$	$F_{(n,0,1)}$	1

Tabel 5.9 Simulasi perhitungan jumlah kombinasi string  $orig2$  tanpa operasi *replace* dengan  $dist = 1$  pada kasus string  $ad1 = c$ , string  $ad2 = n$  dan  $X = 1$

Berikutnya, pada iterasi  $dist = 1$ , nilai jawaban akhir akan ditambahkan dengan  $F_{(c,1,0)}$ , yang merupakan jumlah kombinasi string  $orig1$  tanpa operasi *replace* dengan jarak 1 terhadap string  $ad1$ , yang berdasarkan ilustrasi pada tabel 5.6, bernilai 0 yang dikalikan dengan  $G_{(n,1,1)}$ , yang merupakan jumlah kombinasi string  $orig2$  dengan operasi *replace* yang memiliki jarak 0 terhadap string  $ad2$ , yang berdasarkan ilustrasi pada tabel 5.7, bernilai 0. Lalu nilai jawaban akhir akan ditambahkan dengan  $G_{(c,1,0)}$ , yang merupakan jumlah kombinasi string  $orig1$  dengan operasi *replace* dengan jarak 1 terhadap string  $ad1$ , yang berdasarkan ilustrasi pada tabel 5.8, bernilai 2 yang dikalikan dengan  $F_{(n,1,1)}$ , yang merupakan jumlah kombinasi string  $orig2$  tanpa operasi *replace* yang memiliki jarak 0 terhadap string  $ad2$ , yang berdasarkan ilustrasi pada tabel 5.9, bernilai 1. Sehingga nilai jawaban dari iterasi 1 adalah 2 dan nilai jawaban akhir pada kasus dimana string  $ad1 = c$ , string  $ad2 = n$  dan  $X = 1$  adalah 4.

Kasus berikutnya adalah kasus dimana string  $ad1 = kbenh$ , string  $ad2 = kbenh$  dan  $X = 5$ . Proses yang dilakukan sama seperti pada kasus sebelumnya, yaitu dimulai dengan mengiterasi variable  $dist$  dari 0 hingga  $X$ .

Pada iterasi  $dist = 0$ , nilai jawaban akhir akan ditambahkan dengan  $F_{(behkn,31,5)}$ , yang merupakan jumlah kombinasi string  $orig1$  tanpa operasi *replace* dengan jarak 0 terhadap string  $ad1$ , yang berdasarkan ilustrasi pada tabel 5.10, bernilai 1 yang dikalikan dengan  $G_{(behkn,31,0)}$ , yang merupakan jumlah kombinasi string  $orig2$  dengan operasi *replace* yang memiliki jarak 5 terhadap string  $ad2$ , yang berdasarkan ilustrasi pada tabel 5.11 sampai dengan tabel 5.16, bernilai 8. Lalu nilai jawaban akhir akan ditambahkan dengan  $G_{(behkn,31,5)}$ , yang merupakan jumlah kombinasi string  $orig1$  dengan operasi *replace* dengan jarak 0 terhadap string  $ad1$ , yang berdasarkan ilustrasi pada tabel 5.17 sampai dengan tabel 5.19, bernilai 0 yang dikalikan dengan  $F_{(behkn,31,0)}$ , yang merupakan jumlah kombinasi string  $orig2$  tanpa operasi *replace* yang memiliki jarak 5 terhadap string  $ad2$ , yang berdasarkan ilustrasi pada tabel 5.20 sampai dengan tabel 5.21, bernilai 0. Sehingga nilai jawaban dari iterasi 0 adalah 8 dan nilai jawaban akhir hingga pada iterasi 0 adalah 8.

Berikutnya, pada iterasi  $dist = 1$ , nilai jawaban akhir akan Berikutnya, pada iterasi  $dist = 1$ , nilai jawaban akhir akan ditambahkan dengan  $F_{(behkn,31,4)}$ , yang merupakan jumlah kombinasi string  $orig1$  tanpa operasi *replace* dengan jarak 1 terhadap string  $ad1$ , yang berdasarkan ilustrasi pada tabel 5.22, bernilai 0 yang dikalikan dengan  $G_{(behkn,31,1)}$ , yang merupakan jumlah kombinasi string  $orig2$  dengan operasi *replace* yang memiliki jarak 4 terhadap string  $ad2$ , yang berdasarkan ilustrasi pada tabel 5.23 sampai dengan tabel 5.27, bernilai 0. Lalu nilai jawaban akhir akan ditambahkan dengan  $G_{(behkn,31,4)}$ , yang merupakan jumlah kombinasi string  $orig1$  dengan operasi *replace* dengan jarak 1 terhadap string  $ad1$ , yang ber-

dasarkan ilustrasi pada tabel 5.28 sampai dengan tabel 5.30, bernilai 10 yang dikalikan dengan  $F_{(behkn,31,1)}$ , yang merupakan jumlah kombinasi string *orig2* tanpa operasi *replace* yang memiliki jarak 4 terhadap string *ad2*, yang berdasarkan ilustrasi pada tabel 5.31 sampai dengan tabel 5.31, bernilai 0. Sehingga nilai jawaban dari iterasi 1 adalah 0 dan nilai jawaban akhir hingga pada iterasi 1 adalah 8.

Berikutnya, pada iterasi  $dist = 2$ , nilai jawaban akhir akan ditambahkan dengan  $F_{(behkn,31,3)}$ , yang merupakan jumlah kombinasi string *orig1* tanpa operasi *replace* dengan jarak 2 terhadap string *ad1*, yang berdasarkan ilustrasi pada tabel 5.32, bernilai 0 yang dikalikan dengan  $G_{(behkn,31,2)}$ , yang merupakan jumlah kombinasi string *orig2* dengan operasi *replace* yang memiliki jarak 3 terhadap string *ad2*, yang berdasarkan ilustrasi pada tabel 5.33 sampai dengan tabel 5.37, bernilai 0. Lalu nilai jawaban akhir akan ditambahkan dengan  $G_{(behkn,31,3)}$ , yang merupakan jumlah kombinasi string *orig1* dengan operasi *replace* dengan jarak 2 terhadap string *ad1*, yang berdasarkan ilustrasi pada tabel 5.38 sampai dengan tabel 5.40, bernilai 0 yang dikalikan dengan  $F_{(behkn,31,2)}$ , yang merupakan jumlah kombinasi string *orig2* tanpa operasi *replace* yang memiliki jarak 3 terhadap string *ad2*, yang berdasarkan ilustrasi pada tabel 5.41 sampai dengan tabel 5.41, bernilai 0. Sehingga nilai jawaban dari iterasi 2 adalah 0 dan nilai jawaban akhir hingga pada iterasi 2 adalah 8.

Berikutnya, pada iterasi  $dist = 3$ , nilai jawaban akhir akan ditambahkan dengan  $F_{(behkn,31,2)}$ , yang merupakan jumlah kombinasi string *orig1* tanpa operasi *replace* dengan jarak 3 terhadap string *ad1*, yang berdasarkan ilustrasi pada tabel 5.42, bernilai 0 yang dikalikan dengan  $G_{(behkn,31,3)}$ , yang merupakan jumlah kombinasi string *orig2* dengan operasi *replace* yang memiliki jarak 2 terhadap string *ad2*, yang berdasarkan ilustrasi pada tabel 5.43 sampai dengan tabel 5.45, bernilai 0. Lalu nilai jawaban akhir akan ditam-

Fungsi	Perhitungan Nilai	Nilai
$F_{(behkn,0,5)}$	<i>base case</i>	1
$F_{(behkn,16,5)}$	$F_{(behkn,0,5)}$	1
$F_{(behkn,8,8)}$	<i>base case</i>	0
$F_{(behkn,24,5)}$	$F_{(behkn,16,5)} + F_{(behkn,8,8)}$	1
$F_{(behkn,20,8)}$	<i>base case</i>	0
$F_{(behkn,12,11)}$	<i>base case</i>	0
$F_{(behkn,28,5)}$	$F_{(behkn,24,5)} + F_{(behkn,20,8)} + F_{(behkn,12,11)}$	1
$F_{(behkn,26,8)}$	<i>base case</i>	0
$F_{(behkn,22,11)}$	<i>base case</i>	0
$F_{(behkn,14,14)}$	<i>base case</i>	0
$F_{(behkn,30,5)}$	$F_{(behkn,28,5)} + F_{(behkn,26,8)} + F_{(behkn,22,11)} + F_{(behkn,14,14)}$	1
$F_{(behkn,29,8)}$	<i>base case</i>	0
$F_{(behkn,27,11)}$	<i>base case</i>	0
$F_{(behkn,23,14)}$	<i>base case</i>	0
$F_{(behkn,15,17)}$	<i>base case</i>	0
$F_{(behkn,31,5)}$	$F_{(behkn,30,5)} + F_{(behkn,29,8)} + F_{(behkn,27,11)} + F_{(behkn,23,14)} + F_{(behkn,15,17)}$	1

Tabel 5.10 Simulasi perhitungan jumlah kombinasi string *orig1* tanpa operasi *replace* dengan *dist* = 0 pada kasus string *ad1* = *kbenh*, string *ad2* = *kbenh* dan *X* = 5

Fungsi	Perhitungan Nilai	Nilai
$G_{(behkn,0,0)}$	<i>base case</i>	0
$F_{(behkn,0,1)}$	<i>base case</i>	0
$F_{(behkn,0,1)}$	<i>base case</i>	0
$G_{(behkn,16,0)}$	$G_{(behkn,0,0)} + F_{(behkn,0,1)} + F_{(behkn,0,1)}$	0
$F_{(behkn,0,1)}$	<i>base case</i>	0
$F_{(behkn,16,1)}$	$F_{(behkn,0,1)}$	0
$F_{(behkn,16,1)}$	<i>memo</i> $F_{(behkn,16,1)}$	0
$G_{(behkn,0,6)}$	<i>base case</i>	0
$F_{(behkn,0,5)}$	<i>base case</i>	1
$F_{(behkn,0,7)}$	<i>base case</i>	0
$G_{(behkn,8,3)}$	$G_{(behkn,0,6)} + F_{(behkn,0,5)} + F_{(behkn,0,7)}$	1
$F_{(behkn,0,7)}$	<i>base case</i>	0
$F_{(behkn,8,4)}$	$F_{(behkn,0,7)}$	0
$F_{(behkn,0,5)}$	<i>base case</i>	1
$F_{(behkn,8,2)}$	$F_{(behkn,0,5)}$	1
$G_{(behkn,24,0)}$	$G_{(behkn,16,0)} + F_{(behkn,16,1)} + F_{(behkn,16,1)} + G_{(behkn,8,3)} + F_{(behkn,8,4)} + F_{(behkn,8,2)}$	2
$F_{(behkn,16,1)}$	<i>memo</i> $F_{(behkn,16,1)}$	0
$F_{(behkn,8,4)}$	<i>memo</i> $F_{(behkn,8,4)}$	0
$F_{(behkn,24,1)}$	$F_{(behkn,16,1)} + F_{(behkn,8,4)}$	0
$F_{(behkn,24,1)}$	<i>memo</i> $F_{(behkn,24,1)}$	0
$G_{(behkn,16,6)}$	<i>base case</i>	0
$F_{(behkn,0,5)}$	<i>base case</i>	1
$F_{(behkn,16,5)}$	$F_{(behkn,0,5)}$	1
$F_{(behkn,16,7)}$	<i>base case</i>	0

Tabel 5.11 Simulasi perhitungan jumlah kombinasi string *orig2* dengan operasi *replace* dengan *dist* = 0 pada kasus string *ad1* = *kbenh*, string *ad2* = *kbenh* dan *X* = 5 (1)

Fungsi	Perhitungan Nilai	Nilai
$G_{(behkn,4,6)}$	<i>base case</i>	0
$F_{(behkn,4,7)}$	<i>base case</i>	0
$F_{(behkn,0,11)}$	<i>base case</i>	0
$F_{(behkn,4,5)}$	$F_{(behkn,0,11)}$	0
$G_{(behkn,20,3)}$	$G_{(behkn,16,6)} + F_{(behkn,16,5)} + F_{(behkn,16,7)} + G_{(behkn,4,6)} + F_{(behkn,4,7)} + F_{(behkn,4,5)}$	1
$F_{(behkn,16,7)}$	<i>base case</i>	0
$F_{(behkn,4,7)}$	<i>base case</i>	0
$F_{(behkn,20,4)}$	$F_{(behkn,16,7)} + F_{(behkn,4,7)}$	0
$F_{(behkn,16,5)}$	<i>memo</i> $F_{(behkn,16,5)}$	1
$F_{(behkn,4,5)}$	<i>memo</i> $F_{(behkn,4,5)}$	0
$F_{(behkn,20,2)}$	$F_{(behkn,16,5)} + F_{(behkn,4,5)}$	1
$G_{(behkn,12,6)}$	<i>base case</i>	0
$F_{(behkn,12,7)}$	<i>base case</i>	0
$F_{(behkn,8,8)}$	<i>base case</i>	0
$F_{(behkn,4,5)}$	<i>memo</i> $F_{(behkn,4,5)}$	0
$F_{(behkn,12,5)}$	$F_{(behkn,8,8)} + F_{(behkn,4,5)}$	0
$G_{(behkn,28,0)}$	$G_{(behkn,24,0)} + F_{(behkn,24,1)} + F_{(behkn,24,1)} + G_{(behkn,20,3)} + F_{(behkn,20,4)} + F_{(behkn,20,2)} + G_{(behkn,12,6)} + F_{(behkn,12,7)} + F_{(behkn,12,5)}$	4
$F_{(behkn,24,1)}$	<i>memo</i> $F_{(behkn,24,1)}$	0
$F_{(behkn,20,4)}$	<i>memo</i> $F_{(behkn,20,4)}$	0
$F_{(behkn,12,7)}$	<i>base case</i>	0
$F_{(behkn,28,1)}$	$F_{(behkn,24,1)} + F_{(behkn,20,4)} + F_{(behkn,12,7)}$	0

Tabel 5.12 Simulasi perhitungan jumlah kombinasi string *orig2* dengan operasi *replace* dengan *dist* = 0 pada kasus string *ad1* = *kbenh*, string *ad2* = *kbenh* dan *X* = 5 (2)

Fungsi	Perhitungan Nilai	Nilai
$F_{(behkn,28,1)}$	$memoF_{(behkn,28,1)}$	0
$G_{(behkn,24,6)}$	<i>base case</i>	0
$F_{(behkn,16,5)}$	$memoF_{(behkn,16,5)}$	1
$F_{(behkn,8,8)}$	<i>base case</i>	0
$F_{(behkn,24,5)}$	$F_{(behkn,16,5)} + F_{(behkn,8,8)}$	1
$F_{(behkn,24,7)}$	<i>base case</i>	0
$G_{(behkn,18,6)}$	<i>base case</i>	0
$F_{(behkn,18,7)}$	<i>base case</i>	0
$F_{(behkn,16,11)}$	<i>base case</i>	0
$F_{(behkn,2,8)}$	<i>base case</i>	0
$F_{(behkn,18,5)}$	$F_{(behkn,16,11)} + F_{(behkn,2,8)}$	0
$G_{(behkn,10,9)}$	<i>base case</i>	0
$F_{(behkn,10,10)}$	<i>base case</i>	0
$F_{(behkn,10,8)}$	<i>base case</i>	0
$G_{(behkn,26,3)}$	$G_{(behkn,24,6)} + F_{(behkn,24,5)} +$ $F_{(behkn,24,7)} + G_{(behkn,18,6)} +$ $F_{(behkn,18,7)} + F_{(behkn,18,5)} +$ $G_{(behkn,10,9)} + F_{(behkn,10,10)} +$ $F_{(behkn,10,8)}$	1
$F_{(behkn,24,7)}$	<i>base case</i>	0
$F_{(behkn,18,7)}$	<i>base case</i>	0
$F_{(behkn,10,10)}$	<i>base case</i>	0
$F_{(behkn,26,4)}$	$F_{(behkn,24,7)} + F_{(behkn,18,7)} +$ $F_{(behkn,10,10)}$	0
$F_{(behkn,24,5)}$	$memoF_{(behkn,24,5)}$	1
$F_{(behkn,18,5)}$	$memoF_{(behkn,18,5)}$	0
$F_{(behkn,10,8)}$	<i>base case</i>	0
$F_{(behkn,26,2)}$	$F_{(behkn,24,5)} + F_{(behkn,18,5)} +$ $F_{(behkn,10,8)}$	1

Tabel 5.13 Simulasi perhitungan jumlah kombinasi string *orig2* dengan operasi *replace* dengan *dist* = 0 pada kasus string *ad1* = *kbenh*, string *ad2* = *kbenh* dan *X* = 5 (3)

Fungsi	Perhitungan Nilai	Nilai
$G_{(behkn,22,6)}$	<i>base case</i>	0
$F_{(behkn,22,7)}$	<i>base case</i>	0
$F_{(behkn,20,8)}$	<i>base case</i>	0
$F_{(behkn,18,5)}$	$memoF_{(behkn,18,5)}$	0
$F_{(behkn,6,11)}$	<i>base case</i>	0
$F_{(behkn,22,5)}$	$F_{(behkn,20,8)} + F_{(behkn,18,5)} + F_{(behkn,6,11)}$	0
$G_{(behkn,14,9)}$	<i>base case</i>	0
$F_{(behkn,14,10)}$	<i>base case</i>	0
$F_{(behkn,14,8)}$	<i>base case</i>	0
$G_{(behkn,30,0)}$	$G_{(behkn,28,0)} + F_{(behkn,28,1)} + F_{(behkn,28,1)} + G_{(behkn,26,3)} + F_{(behkn,26,4)} + F_{(behkn,26,2)} + G_{(behkn,22,6)} + F_{(behkn,22,7)} + F_{(behkn,22,5)} + G_{(behkn,14,9)} + F_{(behkn,14,10)} + F_{(behkn,14,8)}$	6
$F_{(behkn,28,1)}$	$memoF_{(behkn,28,1)}$	0
$F_{(behkn,26,4)}$	$memoF_{(behkn,26,4)}$	0
$F_{(behkn,22,7)}$	<i>base case</i>	0
$F_{(behkn,14,10)}$	<i>base case</i>	0
$F_{(behkn,30,1)}$	$F_{(behkn,28,1)} + F_{(behkn,26,4)} + F_{(behkn,22,7)} + F_{(behkn,14,10)}$	0
$F_{(behkn,30,1)}$	$memoF_{(behkn,30,1)}$	0
$G_{(behkn,28,6)}$	<i>base case</i>	0
$F_{(behkn,24,5)}$	$memoF_{(behkn,24,5)}$	1
$F_{(behkn,20,8)}$	<i>base case</i>	0
$F_{(behkn,12,11)}$	<i>base case</i>	0
$F_{(behkn,28,5)}$	$F_{(behkn,24,5)} + F_{(behkn,20,8)} + F_{(behkn,12,11)}$	1

Tabel 5.14 Simulasi perhitungan jumlah kombinasi string *orig2* dengan operasi *replace* dengan *dist* = 0 pada kasus string *ad1* = *kbenh*, string *ad2* = *kbenh* dan *X* = 5 (4)

Fungsi	Perhitungan Nilai	Nilai
$F_{(behkn,28,7)}$	<i>base case</i>	0
$G_{(behkn,25,6)}$	<i>base case</i>	0
$F_{(behkn,25,7)}$	<i>base case</i>	0
$F_{(behkn,24,11)}$	<i>base case</i>	0
$F_{(behkn,17,8)}$	<i>base case</i>	0
$F_{(behkn,9,11)}$	<i>base case</i>	0
$F_{(behkn,25,5)}$	$F_{(behkn,24,11)} + F_{(behkn,17,8)} + F_{(behkn,9,11)}$	0
$G_{(behkn,21,9)}$	<i>base case</i>	0
$F_{(behkn,21,10)}$	<i>base case</i>	0
$F_{(behkn,21,8)}$	<i>base case</i>	0
$G_{(behkn,13,12)}$	<i>base case</i>	0
$F_{(behkn,13,13)}$	<i>base case</i>	0
$F_{(behkn,13,11)}$	<i>base case</i>	0
$G_{(behkn,29,3)}$	$G_{(behkn,28,6)} + F_{(behkn,28,5)} + F_{(behkn,28,7)} + G_{(behkn,25,6)} + F_{(behkn,25,7)} + F_{(behkn,25,5)} + G_{(behkn,21,9)} + F_{(behkn,21,10)} + F_{(behkn,21,8)} + G_{(behkn,13,12)} + F_{(behkn,13,13)} + F_{(behkn,13,11)}$	1
$F_{(behkn,28,7)}$	<i>base case</i>	0
$F_{(behkn,25,7)}$	<i>base case</i>	0
$F_{(behkn,21,10)}$	<i>base case</i>	0
$F_{(behkn,13,13)}$	<i>base case</i>	0
$F_{(behkn,29,4)}$	$F_{(behkn,28,7)} + F_{(behkn,25,7)} + F_{(behkn,21,10)} + F_{(behkn,13,13)}$	0
$F_{(behkn,28,5)}$	<i>memo</i> $F_{(behkn,28,5)}$	1
$F_{(behkn,25,5)}$	<i>memo</i> $F_{(behkn,25,5)}$	0
$F_{(behkn,21,8)}$	<i>base case</i>	0
$F_{(behkn,13,11)}$	<i>base case</i>	0

Tabel 5.15 Simulasi perhitungan jumlah kombinasi string *orig2* dengan operasi *replace* dengan *dist* = 0 pada kasus string *ad1* = *kbenh*, string *ad2* = *kbenh* dan *X* = 5 (5)

Fungsi	Perhitungan Nilai	Nilai
$F_{(behkn,27,7)}$	<i>base case</i>	0
$F_{(behkn,26,8)}$	<i>base case</i>	0
$F_{(behkn,25,5)}$	<i>memo</i> $F_{(behkn,25,5)}$	0
$F_{(behkn,19,11)}$	<i>base case</i>	0
$F_{(behkn,11,14)}$	<i>base case</i>	0
$F_{(behkn,27,5)}$	$F_{(behkn,26,8)} + F_{(behkn,25,5)} + F_{(behkn,19,11)} + F_{(behkn,11,14)}$	0
$G_{(behkn,23,9)}$	<i>base case</i>	0
$F_{(behkn,23,10)}$	<i>base case</i>	0
$F_{(behkn,23,8)}$	<i>base case</i>	0
$G_{(behkn,15,12)}$	<i>base case</i>	0
$F_{(behkn,15,13)}$	<i>base case</i>	0
$F_{(behkn,15,11)}$	<i>base case</i>	0
$G_{(behkn,31,0)}$	$G_{(behkn,30,0)} + F_{(behkn,30,1)} + F_{(behkn,30,1)} + G_{(behkn,29,3)} + F_{(behkn,29,4)} + F_{(behkn,29,2)} + G_{(behkn,27,6)} + F_{(behkn,27,7)} + F_{(behkn,27,5)} + G_{(behkn,23,9)} + F_{(behkn,23,10)} + F_{(behkn,23,8)} + G_{(behkn,15,12)} + F_{(behkn,15,13)} + F_{(behkn,15,11)}$	8

Tabel 5.16 Simulasi perhitungan jumlah kombinasi string *orig2* dengan operasi *replace* dengan *dist* = 0 pada kasus string *ad1* = *kbenh*, string *ad2* = *kbenh* dan *X* = 5 (6)

Fungsi	Perhitungan Nilai	Nilai
$G_{(behkn,0,5)}$	<i>base case</i>	0
$F_{(behkn,0,6)}$	<i>base case</i>	0
$F_{(behkn,0,6)}$	<i>base case</i>	0
$G_{(behkn,16,5)}$	$G_{(behkn,0,5)} + F_{(behkn,0,6)} + F_{(behkn,0,6)}$	0
$F_{(behkn,16,6)}$	<i>base case</i>	0
$F_{(behkn,16,6)}$	<i>base case</i>	0
$G_{(behkn,8,8)}$	<i>base case</i>	0
$F_{(behkn,8,9)}$	<i>base case</i>	0
$F_{(behkn,8,7)}$	<i>base case</i>	0
$G_{(behkn,24,5)}$	$G_{(behkn,16,5)} + F_{(behkn,16,6)} + F_{(behkn,16,6)} + G_{(behkn,8,8)} + F_{(behkn,8,9)} + F_{(behkn,8,7)}$	0
$F_{(behkn,24,6)}$	<i>base case</i>	0
$F_{(behkn,24,6)}$	<i>base case</i>	0
$G_{(behkn,20,8)}$	<i>base case</i>	0
$F_{(behkn,20,9)}$	<i>base case</i>	0
$F_{(behkn,20,7)}$	<i>base case</i>	0
$G_{(behkn,12,11)}$	<i>base case</i>	0
$F_{(behkn,12,12)}$	<i>base case</i>	0
$F_{(behkn,12,10)}$	<i>base case</i>	0
$G_{(behkn,28,5)}$	$G_{(behkn,24,5)} + F_{(behkn,24,6)} + F_{(behkn,24,6)} + G_{(behkn,20,8)} + F_{(behkn,20,9)} + F_{(behkn,20,7)} + G_{(behkn,12,11)} + F_{(behkn,12,12)} + F_{(behkn,12,10)}$	0
$F_{(behkn,28,6)}$	<i>base case</i>	0
$F_{(behkn,28,6)}$	<i>base case</i>	0

Tabel 5.17 Simulasi perhitungan jumlah kombinasi string *orig1* dengan operasi *replace* dengan *dist* = 0 pada kasus string *ad1* = *kbenh*, string *ad2* = *kbenh* dan *X* = 5 (1)

Fungsi	Perhitungan Nilai	Nilai
$G_{(behkn,26,8)}$	<i>base case</i>	0
$F_{(behkn,26,9)}$	<i>base case</i>	0
$F_{(behkn,26,7)}$	<i>base case</i>	0
$G_{(behkn,22,11)}$	<i>base case</i>	0
$F_{(behkn,22,12)}$	<i>base case</i>	0
$F_{(behkn,22,10)}$	<i>base case</i>	0
$G_{(behkn,14,14)}$	<i>base case</i>	0
$F_{(behkn,14,15)}$	<i>base case</i>	0
$F_{(behkn,14,13)}$	<i>base case</i>	0
$G_{(behkn,30,5)}$	$G_{(behkn,28,5)} + F_{(behkn,28,6)} +$ $F_{(behkn,28,6)} + G_{(behkn,26,8)} +$ $F_{(behkn,26,9)} + F_{(behkn,26,7)} +$ $G_{(behkn,22,11)} + F_{(behkn,22,12)} +$ $F_{(behkn,22,10)} + G_{(behkn,14,14)} +$ $F_{(behkn,14,15)} + F_{(behkn,14,13)}$	0
$F_{(behkn,30,6)}$	<i>base case</i>	0
$F_{(behkn,30,6)}$	<i>base case</i>	0
$G_{(behkn,29,8)}$	<i>base case</i>	0
$F_{(behkn,29,9)}$	<i>base case</i>	0
$F_{(behkn,29,7)}$	<i>base case</i>	0
$G_{(behkn,27,11)}$	<i>base case</i>	0
$F_{(behkn,27,12)}$	<i>base case</i>	0
$F_{(behkn,27,10)}$	<i>base case</i>	0
$G_{(behkn,23,14)}$	<i>base case</i>	0
$F_{(behkn,23,15)}$	<i>base case</i>	0
$F_{(behkn,23,13)}$	<i>base case</i>	0
$G_{(behkn,15,17)}$	<i>base case</i>	0

Tabel 5.18 Simulasi perhitungan jumlah kombinasi string *orig1* dengan operasi *replace* dengan *dist* = 0 pada kasus string *ad1* = *kbenh*, string *ad2* = *kbenh* dan *X* = 5 (2)

Fungsi	Perhitungan Nilai	Nilai
$F_{(behkn,15,18)}$	<i>base case</i>	0
$F_{(behkn,15,16)}$	<i>base case</i>	0
$G_{(behkn,31,5)}$	$G_{(behkn,30,5)} + F_{(behkn,30,6)} +$ $F_{(behkn,30,6)} + G_{(behkn,29,8)} +$ $F_{(behkn,29,9)} + F_{(behkn,29,7)} +$ $G_{(behkn,27,11)} + F_{(behkn,27,12)} +$ $F_{(behkn,27,10)} + G_{(behkn,23,14)} +$ $F_{(behkn,23,15)} + F_{(behkn,23,13)} +$ $G_{(behkn,15,17)} + F_{(behkn,15,18)} +$ $F_{(behkn,15,16)}$	0

Tabel 5.19 Simulasi perhitungan jumlah kombinasi string *orig1* dengan operasi *replace* dengan *dist* = 0 pada kasus string *ad1* = *kbenh*, string *ad2* = *kbenh* dan *X* = 5 (3)

bahkan dengan  $G_{(behkn,31,2)}$ , yang merupakan jumlah kombinasi string *orig1* dengan operasi *replace* dengan jarak 3 terhadap string *ad1*, yang berdasarkan ilustrasi pada tabel 5.46 sampai dengan tabel 5.50, bernilai 0 yang dikalikan dengan  $F_{(behkn,31,3)}$ , yang merupakan jumlah kombinasi string *orig2* tanpa operasi *replace* yang memiliki jarak 2 terhadap string *ad2*, yang berdasarkan ilustrasi pada tabel 5.51 sampai dengan tabel 5.51, bernilai 0. Sehingga nilai jawaban dari iterasi 3 adalah 0 dan nilai jawaban akhir hingga pada iterasi 3 adalah 8.

Berikutnya, pada iterasi *dist* = 4, nilai jawaban akhir akan ditambahkan dengan  $F_{(behkn,31,1)}$ , yang merupakan jumlah kombinasi string *orig1* tanpa operasi *replace* dengan jarak 4 terhadap string *ad1*, yang berdasarkan ilustrasi pada tabel 5.52, bernilai 0 yang dikalikan dengan  $G_{(behkn,31,4)}$ , yang merupakan jumlah kombinasi string *orig2* dengan operasi *replace* yang memiliki jarak 1 terhadap string *ad2*, yang berdasarkan ilustrasi pada tabel 5.53 sampai dengan tabel 5.55, bernilai 10. Lalu nilai jawaban akhir akan ditam-

Fungsi	Perhitungan Nilai	Nilai
$F_{(behkn,0,0)}$	<i>base case</i>	0
$F_{(behkn,16,0)}$	$F_{(behkn,0,0)}$	0
$F_{(behkn,0,6)}$	<i>base case</i>	0
$F_{(behkn,8,3)}$	$F_{(behkn,0,6)}$	0
$F_{(behkn,24,0)}$	$F_{(behkn,16,0)} + F_{(behkn,8,3)}$	0
$F_{(behkn,16,6)}$	<i>base case</i>	0
$F_{(behkn,4,6)}$	<i>base case</i>	0
$F_{(behkn,20,3)}$	$F_{(behkn,16,6)} + F_{(behkn,4,6)}$	0
$F_{(behkn,12,6)}$	<i>base case</i>	0
$F_{(behkn,28,0)}$	$F_{(behkn,24,0)} + F_{(behkn,20,3)} + F_{(behkn,12,6)}$	0
$F_{(behkn,24,6)}$	<i>base case</i>	0
$F_{(behkn,18,6)}$	<i>base case</i>	0
$F_{(behkn,10,9)}$	<i>base case</i>	0
$F_{(behkn,26,3)}$	$F_{(behkn,24,6)} + F_{(behkn,18,6)} + F_{(behkn,10,9)}$	0
$F_{(behkn,22,6)}$	<i>base case</i>	0
$F_{(behkn,14,9)}$	<i>base case</i>	0
$F_{(behkn,30,0)}$	$F_{(behkn,28,0)} + F_{(behkn,26,3)} + F_{(behkn,22,6)} + F_{(behkn,14,9)}$	0
$F_{(behkn,28,6)}$	<i>base case</i>	0
$F_{(behkn,25,6)}$	<i>base case</i>	0
$F_{(behkn,21,9)}$	<i>base case</i>	0
$F_{(behkn,13,12)}$	<i>base case</i>	0
$F_{(behkn,29,3)}$	$F_{(behkn,28,6)} + F_{(behkn,25,6)} + F_{(behkn,21,9)} + F_{(behkn,13,12)}$	0

Tabel 5.20 Simulasi perhitungan jumlah kombinasi string *orig2* tanpa operasi *replace* dengan *dist* = 0 pada kasus string *ad1* = *kbenh*, string *ad2* = *kbenh* dan *X* = 5 (1)

Fungsi	Perhitungan Nilai	Nilai
$F_{(behkn,27,6)}$	<i>base case</i>	0
$F_{(behkn,23,9)}$	<i>base case</i>	0
$F_{(behkn,15,12)}$	<i>base case</i>	0
$F_{(behkn,31,0)}$	$F_{(behkn,30,0)} + F_{(behkn,29,3)} + F_{(behkn,27,6)} + F_{(behkn,23,9)} + F_{(behkn,15,12)}$	0

Tabel 5.21 Simulasi perhitungan jumlah kombinasi string *orig2* tanpa operasi *replace* dengan *dist* = 0 pada kasus string *ad1* = *kbenh*, string *ad2* = *kbenh* dan *X* = 5 (2)

bahkan dengan  $G_{(behkn,31,1)}$ , yang merupakan jumlah kombinasi string *orig1* dengan operasi *replace* dengan jarak 4 terhadap string *ad1*, yang berdasarkan ilustrasi pada tabel 5.56 sampai dengan tabel 5.60, bernilai 0 yang dikalikan dengan  $F_{(behkn,31,4)}$ , yang merupakan jumlah kombinasi string *orig2* tanpa operasi *replace* yang memiliki jarak 1 terhadap string *ad2*, yang berdasarkan ilustrasi pada tabel 5.61 sampai dengan tabel 5.61, bernilai 0. Sehingga nilai jawaban dari iterasi 4 adalah 0 dan nilai jawaban akhir hingga pada iterasi 4 adalah 8.

Berikutnya, pada iterasi *dist* = 5, nilai jawaban akhir akan ditambahkan dengan  $F_{(behkn,31,0)}$ , yang merupakan jumlah kombinasi string *orig1* tanpa operasi *replace* dengan jarak 5 terhadap string *ad1*, yang berdasarkan ilustrasi pada tabel 5.62, bernilai 0 yang dikalikan dengan  $G_{(behkn,31,5)}$ , yang merupakan jumlah kombinasi string *orig2* dengan operasi *replace* yang memiliki jarak 0 terhadap string *ad2*, yang berdasarkan ilustrasi pada tabel 5.63 sampai dengan tabel 5.65, bernilai 0. Lalu nilai jawaban akhir akan ditambahkan dengan  $G_{(behkn,31,0)}$ , yang merupakan jumlah kombinasi string *orig1* dengan operasi *replace* dengan jarak 5 terhadap string *ad1*, yang berdasarkan ilustrasi pada tabel 5.66 sampai dengan tabel 5.70, bernilai 8 yang dikalikan dengan  $F_{(behkn,31,5)}$ , yang me-

Fungsi	Perhitungan Nilai	Nilai
$F_{(behkn,0,4)}$	<i>base case</i>	0
$F_{(behkn,16,4)}$	$F_{(behkn,0,4)}$	0
$F_{(behkn,8,7)}$	<i>base case</i>	0
$F_{(behkn,24,4)}$	$F_{(behkn,16,4)} + F_{(behkn,8,7)}$	0
$F_{(behkn,20,7)}$	<i>base case</i>	0
$F_{(behkn,12,10)}$	<i>base case</i>	0
$F_{(behkn,28,4)}$	$F_{(behkn,24,4)} + F_{(behkn,20,7)} + F_{(behkn,12,10)}$	0
$F_{(behkn,26,7)}$	<i>base case</i>	0
$F_{(behkn,22,10)}$	<i>base case</i>	0
$F_{(behkn,14,13)}$	<i>base case</i>	0
$F_{(behkn,30,4)}$	$F_{(behkn,28,4)} + F_{(behkn,26,7)} + F_{(behkn,22,10)} + F_{(behkn,14,13)}$	0
$F_{(behkn,29,7)}$	<i>base case</i>	0
$F_{(behkn,27,10)}$	<i>base case</i>	0
$F_{(behkn,23,13)}$	<i>base case</i>	0
$F_{(behkn,15,16)}$	<i>base case</i>	0
$F_{(behkn,31,4)}$	$F_{(behkn,30,4)} + F_{(behkn,29,7)} + F_{(behkn,27,10)} + F_{(behkn,23,13)} + F_{(behkn,15,16)}$	0

Tabel 5.22 Simulasi perhitungan jumlah kombinasi string *orig1* tanpa operasi *replace* dengan *dist* = 1 pada kasus string *ad1* = *kbenh*, string *ad2* = *kbenh* dan *X* = 5

Fungsi	Perhitungan Nilai	Nilai
$G_{(behkn,0,1)}$	<i>base case</i>	0
$F_{(behkn,0,2)}$	<i>base case</i>	0
$F_{(behkn,0,2)}$	<i>base case</i>	0
$G_{(behkn,16,1)}$	$G_{(behkn,0,1)} + F_{(behkn,0,2)} + F_{(behkn,0,2)}$	0
$F_{(behkn,0,2)}$	<i>base case</i>	0
$F_{(behkn,16,2)}$	$F_{(behkn,0,2)}$	0
$F_{(behkn,16,2)}$	<i>memo</i> $F_{(behkn,16,2)}$	0
$G_{(behkn,0,7)}$	<i>base case</i>	0
$F_{(behkn,0,6)}$	<i>base case</i>	0
$F_{(behkn,0,8)}$	<i>base case</i>	0
$G_{(behkn,8,4)}$	$G_{(behkn,0,7)} + F_{(behkn,0,6)} + F_{(behkn,0,8)}$	0
$F_{(behkn,0,8)}$	<i>base case</i>	0
$F_{(behkn,8,5)}$	$F_{(behkn,0,8)}$	0
$F_{(behkn,8,3)}$	<i>memo</i> $F_{(behkn,8,3)}$	0
$G_{(behkn,24,1)}$	$G_{(behkn,16,1)} + F_{(behkn,16,2)} + F_{(behkn,16,2)} + G_{(behkn,8,4)} + F_{(behkn,8,5)} + F_{(behkn,8,3)}$	0
$F_{(behkn,16,2)}$	<i>memo</i> $F_{(behkn,16,2)}$	0
$F_{(behkn,8,5)}$	<i>memo</i> $F_{(behkn,8,5)}$	0
$F_{(behkn,24,2)}$	$F_{(behkn,16,2)} + F_{(behkn,8,5)}$	0
$F_{(behkn,24,2)}$	<i>memo</i> $F_{(behkn,24,2)}$	0
$G_{(behkn,16,7)}$	<i>base case</i>	0
$F_{(behkn,16,6)}$	<i>base case</i>	0
$F_{(behkn,16,8)}$	<i>base case</i>	0

Tabel 5.23 Simulasi perhitungan jumlah kombinasi string *orig2* dengan operasi *replace* dengan *dist* = 1 pada kasus string *ad1* = *kbenh*, string *ad2* = *kbenh* dan *X* = 5 (1)

Fungsi	Perhitungan Nilai	Nilai
$G_{(behkn,4,7)}$	<i>base case</i>	0
$F_{(behkn,4,8)}$	<i>base case</i>	0
$F_{(behkn,4,6)}$	<i>base case</i>	0
$G_{(behkn,20,4)}$	$G_{(behkn,16,7)} + F_{(behkn,16,6)} + F_{(behkn,16,8)} + G_{(behkn,4,7)} + F_{(behkn,4,8)} + F_{(behkn,4,6)}$	0
$F_{(behkn,16,8)}$	<i>base case</i>	0
$F_{(behkn,4,8)}$	<i>base case</i>	0
$F_{(behkn,20,5)}$	$F_{(behkn,16,8)} + F_{(behkn,4,8)}$	0
$F_{(behkn,20,3)}$	$memoF_{(behkn,20,3)}$	0
$G_{(behkn,12,7)}$	<i>base case</i>	0
$F_{(behkn,12,8)}$	<i>base case</i>	0
$F_{(behkn,12,6)}$	<i>base case</i>	0
$G_{(behkn,28,1)}$	$G_{(behkn,24,1)} + F_{(behkn,24,2)} + F_{(behkn,24,2)} + G_{(behkn,20,4)} + F_{(behkn,20,5)} + F_{(behkn,20,3)} + G_{(behkn,12,7)} + F_{(behkn,12,8)} + F_{(behkn,12,6)}$	0
$F_{(behkn,24,2)}$	$memoF_{(behkn,24,2)}$	0
$F_{(behkn,20,5)}$	$memoF_{(behkn,20,5)}$	0
$F_{(behkn,12,8)}$	<i>base case</i>	0
$F_{(behkn,28,2)}$	$F_{(behkn,24,2)} + F_{(behkn,20,5)} + F_{(behkn,12,8)}$	0
$F_{(behkn,28,2)}$	$memoF_{(behkn,28,2)}$	0
$G_{(behkn,24,7)}$	<i>base case</i>	0
$F_{(behkn,24,6)}$	<i>base case</i>	0
$F_{(behkn,24,8)}$	<i>base case</i>	0
$G_{(behkn,18,7)}$	<i>base case</i>	0

Tabel 5.24 Simulasi perhitungan jumlah kombinasi string  $orig2$  dengan operasi  $replace$  dengan  $dist = 1$  pada kasus string  $ad1 = kbenh$ , string  $ad2 = kbenh$  dan  $X = 5$  (2)

Fungsi	Perhitungan Nilai	Nilai
$F_{(behkn,18,8)}$	<i>base case</i>	0
$F_{(behkn,18,6)}$	<i>base case</i>	0
$G_{(behkn,10,10)}$	<i>base case</i>	0
$F_{(behkn,10,11)}$	<i>base case</i>	0
$F_{(behkn,10,9)}$	<i>base case</i>	0
$G_{(behkn,26,4)}$	$G_{(behkn,24,7)} + F_{(behkn,24,6)} +$ $F_{(behkn,24,8)} + G_{(behkn,18,7)} +$ $F_{(behkn,18,8)} + F_{(behkn,18,6)} +$ $G_{(behkn,10,10)} + F_{(behkn,10,11)} +$ $F_{(behkn,10,9)}$	0
$F_{(behkn,24,8)}$	<i>base case</i>	0
$F_{(behkn,18,8)}$	<i>base case</i>	0
$F_{(behkn,10,11)}$	<i>base case</i>	0
$F_{(behkn,26,5)}$	$F_{(behkn,24,8)} + F_{(behkn,18,8)} +$ $F_{(behkn,10,11)}$	0
$F_{(behkn,26,3)}$	<i>memo</i> $F_{(behkn,26,3)}$	0
$G_{(behkn,22,7)}$	<i>base case</i>	0
$F_{(behkn,22,8)}$	<i>base case</i>	0
$F_{(behkn,22,6)}$	<i>base case</i>	0
$G_{(behkn,14,10)}$	<i>base case</i>	0
$F_{(behkn,14,11)}$	<i>base case</i>	0
$F_{(behkn,14,9)}$	<i>base case</i>	0
$G_{(behkn,30,1)}$	$G_{(behkn,28,1)} + F_{(behkn,28,2)} +$ $F_{(behkn,28,2)} + G_{(behkn,26,4)} +$ $F_{(behkn,26,5)} + F_{(behkn,26,3)} +$ $G_{(behkn,22,7)} + F_{(behkn,22,8)} +$ $F_{(behkn,22,6)} + G_{(behkn,14,10)} +$ $F_{(behkn,14,11)} + F_{(behkn,14,9)}$	0

Tabel 5.25 Simulasi perhitungan jumlah kombinasi string *orig2* dengan operasi *replace* dengan *dist* = 1 pada kasus string *ad1* = *kbenh*, string *ad2* = *kbenh* dan *X* = 5 (3)

Fungsi	Perhitungan Nilai	Nilai
$F_{(behkn,28,2)}$	$memoF_{(behkn,28,2)}$	0
$F_{(behkn,26,5)}$	$memoF_{(behkn,26,5)}$	0
$F_{(behkn,22,8)}$	<i>base case</i>	0
$F_{(behkn,14,11)}$	<i>base case</i>	0
$F_{(behkn,30,2)}$	$F_{(behkn,28,2)} + F_{(behkn,26,5)} + F_{(behkn,22,8)} + F_{(behkn,14,11)}$	0
$F_{(behkn,30,2)}$	$memoF_{(behkn,30,2)}$	0
$G_{(behkn,28,7)}$	<i>base case</i>	0
$F_{(behkn,28,6)}$	<i>base case</i>	0
$F_{(behkn,28,8)}$	<i>base case</i>	0
$G_{(behkn,25,7)}$	<i>base case</i>	0
$F_{(behkn,25,8)}$	<i>base case</i>	0
$F_{(behkn,25,6)}$	<i>base case</i>	0
$G_{(behkn,21,10)}$	<i>base case</i>	0
$F_{(behkn,21,11)}$	<i>base case</i>	0
$F_{(behkn,21,9)}$	<i>base case</i>	0
$G_{(behkn,13,13)}$	<i>base case</i>	0
$F_{(behkn,13,14)}$	<i>base case</i>	0
$F_{(behkn,13,12)}$	<i>base case</i>	0
$G_{(behkn,29,4)}$	$G_{(behkn,28,7)} + F_{(behkn,28,6)} + F_{(behkn,28,8)} + G_{(behkn,25,7)} + F_{(behkn,25,8)} + F_{(behkn,25,6)} + G_{(behkn,21,10)} + F_{(behkn,21,11)} + F_{(behkn,21,9)} + G_{(behkn,13,13)} + F_{(behkn,13,14)} + F_{(behkn,13,12)}$	0
$F_{(behkn,28,8)}$	<i>base case</i>	0
$F_{(behkn,25,8)}$	<i>base case</i>	0
$F_{(behkn,21,11)}$	<i>base case</i>	0

Tabel 5.26 Simulasi perhitungan jumlah kombinasi string *orig2* dengan operasi *replace* dengan *dist* = 1 pada kasus string *ad1* = *kbenh*, string *ad2* = *kbenh* dan *X* = 5 (4)

Fungsi	Perhitungan Nilai	Nilai
$F_{(behkn,13,14)}$	<i>base case</i>	0
$F_{(behkn,29,5)}$	$F_{(behkn,28,8)} + F_{(behkn,25,8)} + F_{(behkn,21,11)} + F_{(behkn,13,14)}$	0
$F_{(behkn,29,3)}$	<i>memo</i> $F_{(behkn,29,3)}$	0
$G_{(behkn,27,7)}$	<i>base case</i>	0
$F_{(behkn,27,8)}$	<i>base case</i>	0
$F_{(behkn,27,6)}$	<i>base case</i>	0
$G_{(behkn,23,10)}$	<i>base case</i>	0
$F_{(behkn,23,11)}$	<i>base case</i>	0
$F_{(behkn,23,9)}$	<i>base case</i>	0
$G_{(behkn,15,13)}$	<i>base case</i>	0
$F_{(behkn,15,14)}$	<i>base case</i>	0
$F_{(behkn,15,12)}$	<i>base case</i>	0
$G_{(behkn,31,1)}$	$G_{(behkn,30,1)} + F_{(behkn,30,2)} + F_{(behkn,30,2)} + G_{(behkn,29,4)} + F_{(behkn,29,5)} + F_{(behkn,29,3)} + G_{(behkn,27,7)} + F_{(behkn,27,8)} + F_{(behkn,27,6)} + G_{(behkn,23,10)} + F_{(behkn,23,11)} + F_{(behkn,23,9)} + G_{(behkn,15,13)} + F_{(behkn,15,14)} + F_{(behkn,15,12)}$	0

Tabel 5.27 Simulasi perhitungan jumlah kombinasi string *orig2* dengan operasi *replace* dengan *dist* = 1 pada kasus string *ad1* = *kbenh*, string *ad2* = *kbenh* dan *X* = 5 (5)

Fungsi	Perhitungan Nilai	Nilai
$G_{(behkn,0,4)}$	<i>base case</i>	0
$F_{(behkn,0,5)}$	<i>base case</i>	1
$F_{(behkn,0,5)}$	<i>base case</i>	1
$G_{(behkn,16,4)}$	$G_{(behkn,0,4)} + F_{(behkn,0,5)} + F_{(behkn,0,5)}$	2
$F_{(behkn,16,5)}$	$memoF_{(behkn,16,5)}$	1
$F_{(behkn,16,5)}$	$memoF_{(behkn,16,5)}$	1
$G_{(behkn,8,7)}$	<i>base case</i>	0
$F_{(behkn,8,8)}$	<i>base case</i>	0
$F_{(behkn,8,6)}$	<i>base case</i>	0
$G_{(behkn,24,4)}$	$G_{(behkn,16,4)} + F_{(behkn,16,5)} + F_{(behkn,16,5)} + G_{(behkn,8,7)} + F_{(behkn,8,8)} + F_{(behkn,8,6)}$	4
$F_{(behkn,24,5)}$	$memoF_{(behkn,24,5)}$	1
$F_{(behkn,24,5)}$	$memoF_{(behkn,24,5)}$	1
$G_{(behkn,20,7)}$	<i>base case</i>	0
$F_{(behkn,20,8)}$	<i>base case</i>	0
$F_{(behkn,20,6)}$	<i>base case</i>	0
$G_{(behkn,12,10)}$	<i>base case</i>	0
$F_{(behkn,12,11)}$	<i>base case</i>	0
$F_{(behkn,12,9)}$	<i>base case</i>	0
$G_{(behkn,28,4)}$	$G_{(behkn,24,4)} + F_{(behkn,24,5)} + F_{(behkn,24,5)} + G_{(behkn,20,7)} + F_{(behkn,20,8)} + F_{(behkn,20,6)} + G_{(behkn,12,10)} + F_{(behkn,12,11)} + F_{(behkn,12,9)}$	6
$F_{(behkn,28,5)}$	$memoF_{(behkn,28,5)}$	1
$F_{(behkn,28,5)}$	$memoF_{(behkn,28,5)}$	1

Tabel 5.28 Simulasi perhitungan jumlah kombinasi string *orig1* dengan operasi *replace* dengan *dist* = 1 pada kasus string *ad1* = *kbenh*, string *ad2* = *kbenh* dan *X* = 5 (1)

Fungsi	Perhitungan Nilai	Nilai
$G_{(behkn,26,7)}$	<i>base case</i>	0
$F_{(behkn,26,8)}$	<i>base case</i>	0
$F_{(behkn,26,6)}$	<i>base case</i>	0
$G_{(behkn,22,10)}$	<i>base case</i>	0
$F_{(behkn,22,11)}$	<i>base case</i>	0
$F_{(behkn,22,9)}$	<i>base case</i>	0
$G_{(behkn,14,13)}$	<i>base case</i>	0
$F_{(behkn,14,14)}$	<i>base case</i>	0
$F_{(behkn,14,12)}$	<i>base case</i>	0
$G_{(behkn,30,4)}$	$G_{(behkn,28,4)} + F_{(behkn,28,5)} +$ $F_{(behkn,28,5)} + G_{(behkn,26,7)} +$ $F_{(behkn,26,8)} + F_{(behkn,26,6)} +$ $G_{(behkn,22,10)} + F_{(behkn,22,11)} +$ $F_{(behkn,22,9)} + G_{(behkn,14,13)} +$ $F_{(behkn,14,14)} + F_{(behkn,14,12)}$	8
$F_{(behkn,30,5)}$	<i>memoF</i> $_{(behkn,30,5)}$	1
$F_{(behkn,30,5)}$	<i>memoF</i> $_{(behkn,30,5)}$	1
$G_{(behkn,29,7)}$	<i>base case</i>	0
$F_{(behkn,29,8)}$	<i>base case</i>	0
$F_{(behkn,29,6)}$	<i>base case</i>	0
$G_{(behkn,27,10)}$	<i>base case</i>	0
$F_{(behkn,27,11)}$	<i>base case</i>	0
$F_{(behkn,27,9)}$	<i>base case</i>	0
$G_{(behkn,23,13)}$	<i>base case</i>	0
$F_{(behkn,23,14)}$	<i>base case</i>	0
$F_{(behkn,23,12)}$	<i>base case</i>	0
$G_{(behkn,15,16)}$	<i>base case</i>	0
$F_{(behkn,15,17)}$	<i>base case</i>	0

Tabel 5.29 Simulasi perhitungan jumlah kombinasi string *orig1* dengan operasi *replace* dengan *dist* = 1 pada kasus string *ad1* = *kbenh*, string *ad2* = *kbenh* dan *X* = 5 (2)

Fungsi	Perhitungan Nilai	Nilai
$F_{(behkn,15,15)}$	<i>base case</i>	0
$G_{(behkn,31,4)}$	$G_{(behkn,30,4)} + F_{(behkn,30,5)} +$ $F_{(behkn,30,5)} + G_{(behkn,29,7)} +$ $F_{(behkn,29,8)} + F_{(behkn,29,6)} +$ $G_{(behkn,27,10)} + F_{(behkn,27,11)} +$ $F_{(behkn,27,9)} + G_{(behkn,23,13)} +$ $F_{(behkn,23,14)} + F_{(behkn,23,12)} +$ $G_{(behkn,15,16)} + F_{(behkn,15,17)} +$ $F_{(behkn,15,15)}$	10

Tabel 5.30 Simulasi perhitungan jumlah kombinasi string  $orig1$  dengan operasi *replace* dengan  $dist = 1$  pada kasus string  $ad1 = kbenh$ , string  $ad2 = kbenh$  dan  $X = 5$  (3)

Fungsi	Perhitungan Nilai	Nilai
$F_{(behkn,30,1)}$	$memoF_{(behkn,30,1)}$	0
$F_{(behkn,29,4)}$	$memoF_{(behkn,29,4)}$	0
$F_{(behkn,27,7)}$	<i>base case</i>	0
$F_{(behkn,23,10)}$	<i>base case</i>	0
$F_{(behkn,15,13)}$	<i>base case</i>	0
$F_{(behkn,31,1)}$	$F_{(behkn,30,1)} + F_{(behkn,29,4)} +$ $F_{(behkn,27,7)} + F_{(behkn,23,10)} +$ $F_{(behkn,15,13)}$	0

Tabel 5.31 Simulasi perhitungan jumlah kombinasi string  $orig2$  tanpa operasi *replace* dengan  $dist = 1$  pada kasus string  $ad1 = kbenh$ , string  $ad2 = kbenh$  dan  $X = 5$

Fungsi	Perhitungan Nilai	Nilai
$F_{(behkn,0,3)}$	<i>base case</i>	0
$F_{(behkn,16,3)}$	$F_{(behkn,0,3)}$	0
$F_{(behkn,8,6)}$	<i>base case</i>	0
$F_{(behkn,24,3)}$	$F_{(behkn,16,3)} + F_{(behkn,8,6)}$	0
$F_{(behkn,20,6)}$	<i>base case</i>	0
$F_{(behkn,12,9)}$	<i>base case</i>	0
$F_{(behkn,28,3)}$	$F_{(behkn,24,3)} + F_{(behkn,20,6)} + F_{(behkn,12,9)}$	0
$F_{(behkn,26,6)}$	<i>base case</i>	0
$F_{(behkn,22,9)}$	<i>base case</i>	0
$F_{(behkn,14,12)}$	<i>base case</i>	0
$F_{(behkn,30,3)}$	$F_{(behkn,28,3)} + F_{(behkn,26,6)} + F_{(behkn,22,9)} + F_{(behkn,14,12)}$	0
$F_{(behkn,29,6)}$	<i>base case</i>	0
$F_{(behkn,27,9)}$	<i>base case</i>	0
$F_{(behkn,23,12)}$	<i>base case</i>	0
$F_{(behkn,15,15)}$	<i>base case</i>	0
$F_{(behkn,31,3)}$	$F_{(behkn,30,3)} + F_{(behkn,29,6)} + F_{(behkn,27,9)} + F_{(behkn,23,12)} + F_{(behkn,15,15)}$	0

Tabel 5.32 Simulasi perhitungan jumlah kombinasi string *orig1* tanpa operasi *replace* dengan *dist* = 2 pada kasus string *ad1* = *kbenh*, string *ad2* = *kbenh* dan *X* = 5

Fungsi	Perhitungan Nilai	Nilai
$G_{(behkn,0,2)}$	<i>base case</i>	0
$F_{(behkn,0,3)}$	<i>base case</i>	0
$F_{(behkn,0,3)}$	<i>base case</i>	0
$G_{(behkn,16,2)}$	$G_{(behkn,0,2)} + F_{(behkn,0,3)} + F_{(behkn,0,3)}$	0
$F_{(behkn,0,3)}$	<i>base case</i>	0
$F_{(behkn,16,3)}$	$F_{(behkn,0,3)}$	0
$F_{(behkn,16,3)}$	<i>memoF</i> $_{(behkn,16,3)}$	0
$G_{(behkn,0,8)}$	<i>base case</i>	0
$F_{(behkn,0,7)}$	<i>base case</i>	0
$F_{(behkn,0,9)}$	<i>base case</i>	0
$G_{(behkn,8,5)}$	$G_{(behkn,0,8)} + F_{(behkn,0,7)} + F_{(behkn,0,9)}$	0
$F_{(behkn,8,6)}$	<i>base case</i>	0
$F_{(behkn,8,4)}$	<i>memoF</i> $_{(behkn,8,4)}$	0
$G_{(behkn,24,2)}$	$G_{(behkn,16,2)} + F_{(behkn,16,3)} + F_{(behkn,16,3)} + G_{(behkn,8,5)} + F_{(behkn,8,6)} + F_{(behkn,8,4)}$	0
$F_{(behkn,16,3)}$	<i>memoF</i> $_{(behkn,16,3)}$	0
$F_{(behkn,8,6)}$	<i>base case</i>	0
$F_{(behkn,24,3)}$	$F_{(behkn,16,3)} + F_{(behkn,8,6)}$	0
$F_{(behkn,24,3)}$	<i>memoF</i> $_{(behkn,24,3)}$	0
$G_{(behkn,16,8)}$	<i>base case</i>	0
$F_{(behkn,16,7)}$	<i>base case</i>	0
$F_{(behkn,16,9)}$	<i>base case</i>	0
$G_{(behkn,4,8)}$	<i>base case</i>	0

Tabel 5.33 Simulasi perhitungan jumlah kombinasi string *orig2* dengan operasi *replace* dengan *dist* = 2 pada kasus string *ad1* = *kbenh*, string *ad2* = *kbenh* dan *X* = 5 (1)

Fungsi	Perhitungan Nilai	Nilai
$F_{(behkn,4,9)}$	<i>base case</i>	0
$F_{(behkn,4,7)}$	<i>base case</i>	0
$G_{(behkn,20,5)}$	$G_{(behkn,16,8)} + F_{(behkn,16,7)} + F_{(behkn,16,9)} + G_{(behkn,4,8)} + F_{(behkn,4,9)} + F_{(behkn,4,7)}$	0
$F_{(behkn,20,6)}$	<i>base case</i>	0
$F_{(behkn,20,4)}$	$memoF_{(behkn,20,4)}$	0
$G_{(behkn,12,8)}$	<i>base case</i>	0
$F_{(behkn,12,9)}$	<i>base case</i>	0
$F_{(behkn,12,7)}$	<i>base case</i>	0
$G_{(behkn,28,2)}$	$G_{(behkn,24,2)} + F_{(behkn,24,3)} + F_{(behkn,24,3)} + G_{(behkn,20,5)} + F_{(behkn,20,6)} + F_{(behkn,20,4)} + G_{(behkn,12,8)} + F_{(behkn,12,9)} + F_{(behkn,12,7)}$	0
$F_{(behkn,24,3)}$	$memoF_{(behkn,24,3)}$	0
$F_{(behkn,20,6)}$	<i>base case</i>	0
$F_{(behkn,12,9)}$	<i>base case</i>	0
$F_{(behkn,28,3)}$	$F_{(behkn,24,3)} + F_{(behkn,20,6)} + F_{(behkn,12,9)}$	0
$F_{(behkn,28,3)}$	$memoF_{(behkn,28,3)}$	0
$G_{(behkn,24,8)}$	<i>base case</i>	0
$F_{(behkn,24,7)}$	<i>base case</i>	0
$F_{(behkn,24,9)}$	<i>base case</i>	0
$G_{(behkn,18,8)}$	<i>base case</i>	0
$F_{(behkn,18,9)}$	<i>base case</i>	0
$F_{(behkn,18,7)}$	<i>base case</i>	0

Tabel 5.34 Simulasi perhitungan jumlah kombinasi string *orig2* dengan operasi *replace* dengan *dist* = 2 pada kasus string *ad1* = *kbenh*, string *ad2* = *kbenh* dan *X* = 5 (2)

Fungsi	Perhitungan Nilai	Nilai
$G_{(behkn,10,11)}$	<i>base case</i>	0
$F_{(behkn,10,12)}$	<i>base case</i>	0
$F_{(behkn,10,10)}$	<i>base case</i>	0
$G_{(behkn,26,5)}$	$G_{(behkn,24,8)} + F_{(behkn,24,7)} + F_{(behkn,24,9)} + G_{(behkn,18,8)} + F_{(behkn,18,9)} + F_{(behkn,18,7)} + G_{(behkn,10,11)} + F_{(behkn,10,12)} + F_{(behkn,10,10)}$	0
$F_{(behkn,26,6)}$	<i>base case</i>	0
$F_{(behkn,26,4)}$	$memoF_{(behkn,26,4)}$	0
$G_{(behkn,22,8)}$	<i>base case</i>	0
$F_{(behkn,22,9)}$	<i>base case</i>	0
$F_{(behkn,22,7)}$	<i>base case</i>	0
$G_{(behkn,14,11)}$	<i>base case</i>	0
$F_{(behkn,14,12)}$	<i>base case</i>	0
$F_{(behkn,14,10)}$	<i>base case</i>	0
$G_{(behkn,30,2)}$	$G_{(behkn,28,2)} + F_{(behkn,28,3)} + F_{(behkn,28,3)} + G_{(behkn,26,5)} + F_{(behkn,26,6)} + F_{(behkn,26,4)} + G_{(behkn,22,8)} + F_{(behkn,22,9)} + F_{(behkn,22,7)} + G_{(behkn,14,11)} + F_{(behkn,14,12)} + F_{(behkn,14,10)}$	0
$F_{(behkn,28,3)}$	$memoF_{(behkn,28,3)}$	0
$F_{(behkn,26,6)}$	<i>base case</i>	0
$F_{(behkn,22,9)}$	<i>base case</i>	0
$F_{(behkn,14,12)}$	<i>base case</i>	0
$F_{(behkn,30,3)}$	$F_{(behkn,28,3)} + F_{(behkn,26,6)} + F_{(behkn,22,9)} + F_{(behkn,14,12)}$	0

Tabel 5.35 Simulasi perhitungan jumlah kombinasi string *orig2* dengan operasi *replace* dengan *dist* = 2 pada kasus string *ad1* = *kbenh*, string *ad2* = *kbenh* dan *X* = 5 (3)

Fungsi	Perhitungan Nilai	Nilai
$F_{(behkn,30,3)}$	$memoF_{(behkn,30,3)}$	0
$G_{(behkn,28,8)}$	<i>base case</i>	0
$F_{(behkn,28,7)}$	<i>base case</i>	0
$F_{(behkn,28,9)}$	<i>base case</i>	0
$G_{(behkn,25,8)}$	<i>base case</i>	0
$F_{(behkn,25,9)}$	<i>base case</i>	0
$F_{(behkn,25,7)}$	<i>base case</i>	0
$G_{(behkn,21,11)}$	<i>base case</i>	0
$F_{(behkn,21,12)}$	<i>base case</i>	0
$F_{(behkn,21,10)}$	<i>base case</i>	0
$G_{(behkn,13,14)}$	<i>base case</i>	0
$F_{(behkn,13,15)}$	<i>base case</i>	0
$F_{(behkn,13,13)}$	<i>base case</i>	0
$G_{(behkn,29,5)}$	$G_{(behkn,28,8)} + F_{(behkn,28,7)} +$ $F_{(behkn,28,9)} + G_{(behkn,25,8)} +$ $F_{(behkn,25,9)} + F_{(behkn,25,7)} +$ $G_{(behkn,21,11)} + F_{(behkn,21,12)} +$ $F_{(behkn,21,10)} + G_{(behkn,13,14)} +$ $F_{(behkn,13,15)} + F_{(behkn,13,13)}$	0
$F_{(behkn,29,6)}$	<i>base case</i>	0
$F_{(behkn,29,4)}$	$memoF_{(behkn,29,4)}$	0
$G_{(behkn,27,8)}$	<i>base case</i>	0
$F_{(behkn,27,9)}$	<i>base case</i>	0
$F_{(behkn,27,7)}$	<i>base case</i>	0
$G_{(behkn,23,11)}$	<i>base case</i>	0
$F_{(behkn,23,12)}$	<i>base case</i>	0
$F_{(behkn,23,10)}$	<i>base case</i>	0
$G_{(behkn,15,14)}$	<i>base case</i>	0

Tabel 5.36 Simulasi perhitungan jumlah kombinasi string *orig2* dengan operasi *replace* dengan *dist* = 2 pada kasus string *ad1* = *kbenh*, string *ad2* = *kbenh* dan *X* = 5 (4)

Fungsi	Perhitungan Nilai	Nilai
$F_{(behkn,15,15)}$	<i>base case</i>	0
$F_{(behkn,15,13)}$	<i>base case</i>	0
$G_{(behkn,31,2)}$	$G_{(behkn,30,2)} + F_{(behkn,30,3)} +$ $F_{(behkn,30,3)} + G_{(behkn,29,5)} +$ $F_{(behkn,29,6)} + F_{(behkn,29,4)} +$ $G_{(behkn,27,8)} + F_{(behkn,27,9)} +$ $F_{(behkn,27,7)} + G_{(behkn,23,11)} +$ $F_{(behkn,23,12)} + F_{(behkn,23,10)} +$ $G_{(behkn,15,14)} + F_{(behkn,15,15)} +$ $F_{(behkn,15,13)}$	0

Tabel 5.37 Simulasi perhitungan jumlah kombinasi string *orig2* dengan operasi *replace* dengan *dist* = 2 pada kasus string *ad1* = *kbenh*, string *ad2* = *kbenh* dan *X* = 5 (5)

rupakan jumlah kombinasi string *orig2* tanpa operasi *replace* yang memiliki jarak 0 terhadap string *ad2*, yang berdasarkan ilustrasi pada tabel 5.71 sampai dengan tabel 5.71, bernilai 1. Sehingga nilai jawaban dari iterasi 5 adalah 8 dan nilai jawaban akhir untuk kasus dimana string *ad1* = *kbenh*, string *ad2* = *kbenh* dan *X* = 5 adalah 16.

### 5.3 Analisa Kompleksitas Waktu

Pada pseudocode pada gambar 3.1, terdapat fungsi preprocess. Dengan menggunakan *master theorem*, kompleksitas waktu dari fungsi preprocess adalah  $\mathcal{O}(2^{|S|} * |S|)$  dimana *S* adalah string masukan.

Berikutnya untuk setiap kasus uji terdapat empat fungsi utama. Dengan menggunakan *master theorem*, fungsi readInput memiliki kompleksitas  $\mathcal{O}(1)$ . Berdasarkan pseudocode fungsi init() pada gambar 3.3, fungsi init dapat dipecah menjadi dua bagian

Fungsi	Perhitungan Nilai	Nilai
$G_{(behkn,0,3)}$	$base\ case$	0
$F_{(behkn,0,4)}$	$base\ case$	0
$F_{(behkn,0,4)}$	$base\ case$	0
$G_{(behkn,16,3)}$	$G_{(behkn,0,3)} + F_{(behkn,0,4)} + F_{(behkn,0,4)}$	0
$F_{(behkn,16,4)}$	$memoF_{(behkn,16,4)}$	0
$F_{(behkn,16,4)}$	$memoF_{(behkn,16,4)}$	0
$G_{(behkn,8,6)}$	$base\ case$	0
$F_{(behkn,8,7)}$	$base\ case$	0
$F_{(behkn,0,8)}$	$base\ case$	0
$F_{(behkn,8,5)}$	$F_{(behkn,0,8)}$	0
$G_{(behkn,24,3)}$	$G_{(behkn,16,3)} + F_{(behkn,16,4)} + F_{(behkn,16,4)} + G_{(behkn,8,6)} + F_{(behkn,8,7)} + F_{(behkn,8,5)}$	0
$F_{(behkn,24,4)}$	$memoF_{(behkn,24,4)}$	0
$F_{(behkn,24,4)}$	$memoF_{(behkn,24,4)}$	0
$G_{(behkn,20,6)}$	$base\ case$	0
$F_{(behkn,20,7)}$	$base\ case$	0
$F_{(behkn,16,8)}$	$base\ case$	0
$F_{(behkn,4,8)}$	$base\ case$	0
$F_{(behkn,20,5)}$	$F_{(behkn,16,8)} + F_{(behkn,4,8)}$	0
$G_{(behkn,12,9)}$	$base\ case$	0
$F_{(behkn,12,10)}$	$base\ case$	0
$F_{(behkn,12,8)}$	$base\ case$	0
$G_{(behkn,28,3)}$	$G_{(behkn,24,3)} + F_{(behkn,24,4)} + F_{(behkn,24,4)} + G_{(behkn,20,6)} + F_{(behkn,20,7)} + F_{(behkn,20,5)} + G_{(behkn,12,9)} + F_{(behkn,12,10)} + F_{(behkn,12,8)}$	0

Tabel 5.38 Simulasi perhitungan jumlah kombinasi string *orig1* dengan operasi *replace* dengan *dist* = 2 pada kasus string *ad1* = *kbenh*, string *ad2* = *kbenh* dan *X* = 5 (1)

Fungsi	Perhitungan Nilai	Nilai
$F_{(behkn,28,4)}$	$memoF_{(behkn,28,4)}$	0
$F_{(behkn,28,4)}$	$memoF_{(behkn,28,4)}$	0
$G_{(behkn,26,6)}$	<i>base case</i>	0
$F_{(behkn,26,7)}$	<i>base case</i>	0
$F_{(behkn,24,8)}$	<i>base case</i>	0
$F_{(behkn,18,8)}$	<i>base case</i>	0
$F_{(behkn,10,11)}$	<i>base case</i>	0
$F_{(behkn,26,5)}$	$F_{(behkn,24,8)} + F_{(behkn,18,8)} + F_{(behkn,10,11)}$	0
$G_{(behkn,22,9)}$	<i>base case</i>	0
$F_{(behkn,22,10)}$	<i>base case</i>	0
$F_{(behkn,22,8)}$	<i>base case</i>	0
$G_{(behkn,14,12)}$	<i>base case</i>	0
$F_{(behkn,14,13)}$	<i>base case</i>	0
$F_{(behkn,14,11)}$	<i>base case</i>	0
$G_{(behkn,30,3)}$	$G_{(behkn,28,3)} + F_{(behkn,28,4)} + F_{(behkn,28,4)} + G_{(behkn,26,6)} + F_{(behkn,26,7)} + F_{(behkn,26,5)} + G_{(behkn,22,9)} + F_{(behkn,22,10)} + F_{(behkn,22,8)} + G_{(behkn,14,12)} + F_{(behkn,14,13)} + F_{(behkn,14,11)}$	0
$F_{(behkn,30,4)}$	$memoF_{(behkn,30,4)}$	0
$F_{(behkn,30,4)}$	$memoF_{(behkn,30,4)}$	0
$G_{(behkn,29,6)}$	<i>base case</i>	0
$F_{(behkn,29,7)}$	<i>base case</i>	0
$F_{(behkn,28,8)}$	<i>base case</i>	0
$F_{(behkn,25,8)}$	<i>base case</i>	0

Tabel 5.39 Simulasi perhitungan jumlah kombinasi string *orig1* dengan operasi *replace* dengan *dist* = 2 pada kasus string *ad1* = *kbenh*, string *ad2* = *kbenh* dan *X* = 5 (2)

Fungsi	Perhitungan Nilai	Nilai
$F_{(behkn,21,11)}$	<i>base case</i>	0
$F_{(behkn,13,14)}$	<i>base case</i>	0
$F_{(behkn,29,5)}$	$F_{(behkn,28,8)} + F_{(behkn,25,8)} + F_{(behkn,21,11)} + F_{(behkn,13,14)}$	0
$G_{(behkn,27,9)}$	<i>base case</i>	0
$F_{(behkn,27,10)}$	<i>base case</i>	0
$F_{(behkn,27,8)}$	<i>base case</i>	0
$G_{(behkn,23,12)}$	<i>base case</i>	0
$F_{(behkn,23,13)}$	<i>base case</i>	0
$F_{(behkn,23,11)}$	<i>base case</i>	0
$G_{(behkn,15,15)}$	<i>base case</i>	0
$F_{(behkn,15,16)}$	<i>base case</i>	0
$F_{(behkn,15,14)}$	<i>base case</i>	0
$G_{(behkn,31,3)}$	$G_{(behkn,30,3)} + F_{(behkn,30,4)} + F_{(behkn,30,4)} + G_{(behkn,29,6)} + F_{(behkn,29,7)} + F_{(behkn,29,5)} + G_{(behkn,27,9)} + F_{(behkn,27,10)} + F_{(behkn,27,8)} + G_{(behkn,23,12)} + F_{(behkn,23,13)} + F_{(behkn,23,11)} + G_{(behkn,15,15)} + F_{(behkn,15,16)} + F_{(behkn,15,14)}$	0

Tabel 5.40 Simulasi perhitungan jumlah kombinasi string *orig1* dengan operasi *replace* dengan *dist* = 2 pada kasus string *ad1* = *kbenh*, string *ad2* = *kbenh* dan *X* = 5 (3)

Fungsi	Perhitungan Nilai	Nilai
$F_{(behkn,30,2)}$	$memoF_{(behkn,30,2)}$	0
$F_{(behkn,29,5)}$	$memoF_{(behkn,29,5)}$	0
$F_{(behkn,27,8)}$	<i>base case</i>	0
$F_{(behkn,23,11)}$	<i>base case</i>	0
$F_{(behkn,15,14)}$	<i>base case</i>	0
$F_{(behkn,31,2)}$	$F_{(behkn,30,2)} + F_{(behkn,29,5)} + F_{(behkn,27,8)} + F_{(behkn,23,11)} + F_{(behkn,15,14)}$	0

Tabel 5.41 Simulasi perhitungan jumlah kombinasi string  $orig2$  tanpa operasi *replace* dengan  $dist = 2$  pada kasus string  $ad1 = kbenh$ , string  $ad2 = kbenh$  dan  $X = 5$

an utama, yaitu inisialisasi  $memoF$  dan  $memoG$  dan inisialisasi  $charFirstPos$  dan  $charLastPos$ . Inisialisasi  $memoF$  dan  $memoG$  masing-masing memiliki kompleksitas waktu  $\mathcal{O}(2 * 2^{|S|} * MAX\_DIST)$  dimana  $MAX\_DIST$  adalah jarak maksimum antar string yang mungkin. Sedangkan inisialisasi  $charFirstPos$  dan  $charLastPos$  memiliki kompleksitas waktu  $\mathcal{O}(|S| \log |S| + |S|)$  atau dapat disederhanakan menjadi  $\mathcal{O}(|S| \log |S|)$ . Sehingga fungsi ini memiliki kompleksitas  $\mathcal{O}(2 * 2^{|S|} * MAX\_DIST + |S| \log |S|)$  dan dapat disederhanakan menjadi  $\mathcal{O}(2^{|S|} * MAX\_DIST)$ .

Fungsi berikutnya adalah fungsi *solve*. Pada fungsi *solve* terdapat sebuah iterasi sebanyak  $\min(MAX\_DIST, X)$  dimana pada kasus terburuk,  $\min(MAX\_DIST, X)$  bisa mencapai  $MAX\_DIST$ . Di dalam iterasi tersebut, fungsi *solve* memanggil fungsi *F* dan fungsi *G* masing-masing sebanyak dua kali. Dengan menggunakan *master theorem* memiliki kompleksitas waktu  $\mathcal{O}(2 * 2^{|S|} * MAX\_DIST)$  atau dapat disederhanakan menjadi  $\mathcal{O}(2^{|S|} * MAX\_DIST)$ . Sehingga kompleksitas dari fungsi *solve* secara keseluruhan adalah  $\mathcal{O}(2^{|S|} * MAX\_DIST^2)$ .

Fungsi	Perhitungan Nilai	Nilai
$F_{(behkn,0,2)}$	<i>base case</i>	0
$F_{(behkn,16,2)}$	$F_{(behkn,0,2)}$	0
$F_{(behkn,8,5)}$	$memoF_{(behkn,8,5)}$	0
$F_{(behkn,24,2)}$	$F_{(behkn,16,2)} + F_{(behkn,8,5)}$	0
$F_{(behkn,20,5)}$	$memoF_{(behkn,20,5)}$	0
$F_{(behkn,12,8)}$	<i>base case</i>	0
$F_{(behkn,28,2)}$	$F_{(behkn,24,2)} + F_{(behkn,20,5)} + F_{(behkn,12,8)}$	0
$F_{(behkn,26,5)}$	$memoF_{(behkn,26,5)}$	0
$F_{(behkn,22,8)}$	<i>base case</i>	0
$F_{(behkn,14,11)}$	<i>base case</i>	0
$F_{(behkn,30,2)}$	$F_{(behkn,28,2)} + F_{(behkn,26,5)} + F_{(behkn,22,8)} + F_{(behkn,14,11)}$	0
$F_{(behkn,29,5)}$	$memoF_{(behkn,29,5)}$	0
$F_{(behkn,27,8)}$	<i>base case</i>	0
$F_{(behkn,23,11)}$	<i>base case</i>	0
$F_{(behkn,15,14)}$	<i>base case</i>	0
$F_{(behkn,31,2)}$	$F_{(behkn,30,2)} + F_{(behkn,29,5)} + F_{(behkn,27,8)} + F_{(behkn,23,11)} + F_{(behkn,15,14)}$	0

Tabel 5.42 Simulasi perhitungan jumlah kombinasi string *orig1* tanpa operasi *replace* dengan *dist* = 3 pada kasus string *ad1* = *kbenh*, string *ad2* = *kbenh* dan *X* = 5

Fungsi	Perhitungan Nilai	Nilai
$G_{(behkn,0,3)}$	<i>base case</i>	0
$F_{(behkn,0,4)}$	<i>base case</i>	0
$F_{(behkn,0,4)}$	<i>base case</i>	0
$G_{(behkn,16,3)}$	$G_{(behkn,0,3)} + F_{(behkn,0,4)} + F_{(behkn,0,4)}$	0
$F_{(behkn,0,4)}$	<i>base case</i>	0
$F_{(behkn,16,4)}$	$F_{(behkn,0,4)}$	0
$F_{(behkn,16,4)}$	<i>memoF</i> $_{(behkn,16,4)}$	0
$G_{(behkn,8,6)}$	<i>base case</i>	0
$F_{(behkn,8,7)}$	<i>base case</i>	0
$F_{(behkn,8,5)}$	<i>memoF</i> $_{(behkn,8,5)}$	0
$G_{(behkn,24,3)}$	$G_{(behkn,16,3)} + F_{(behkn,16,4)} + F_{(behkn,16,4)} + G_{(behkn,8,6)} + F_{(behkn,8,7)} + F_{(behkn,8,5)}$	0
$F_{(behkn,16,4)}$	<i>memoF</i> $_{(behkn,16,4)}$	0
$F_{(behkn,8,7)}$	<i>base case</i>	0
$F_{(behkn,24,4)}$	$F_{(behkn,16,4)} + F_{(behkn,8,7)}$	0
$F_{(behkn,24,4)}$	<i>memoF</i> $_{(behkn,24,4)}$	0
$G_{(behkn,20,6)}$	<i>base case</i>	0
$F_{(behkn,20,7)}$	<i>base case</i>	0
$F_{(behkn,20,5)}$	<i>memoF</i> $_{(behkn,20,5)}$	0
$G_{(behkn,12,9)}$	<i>base case</i>	0
$F_{(behkn,12,10)}$	<i>base case</i>	0
$F_{(behkn,12,8)}$	<i>base case</i>	0

Tabel 5.43 Simulasi perhitungan jumlah kombinasi string *orig2* dengan operasi *replace* dengan *dist* = 3 pada kasus string *ad1* = *kbenh*, string *ad2* = *kbenh* dan *X* = 5 (1)

Fungsi	Perhitungan Nilai	Nilai
$G_{(behkn,28,3)}$	$G_{(behkn,24,3)} + F_{(behkn,24,4)} + F_{(behkn,24,4)} + G_{(behkn,20,6)} + F_{(behkn,20,7)} + F_{(behkn,20,5)} + G_{(behkn,12,9)} + F_{(behkn,12,10)} + F_{(behkn,12,8)}$	0
$F_{(behkn,24,4)}$	$memoF_{(behkn,24,4)}$	0
$F_{(behkn,20,7)}$	<i>base case</i>	0
$F_{(behkn,12,10)}$	<i>base case</i>	0
$F_{(behkn,28,4)}$	$F_{(behkn,24,4)} + F_{(behkn,20,7)} + F_{(behkn,12,10)}$	0
$F_{(behkn,28,4)}$	$memoF_{(behkn,28,4)}$	0
$G_{(behkn,26,6)}$	<i>base case</i>	0
$F_{(behkn,26,7)}$	<i>base case</i>	0
$F_{(behkn,26,5)}$	$memoF_{(behkn,26,5)}$	0
$G_{(behkn,22,9)}$	<i>base case</i>	0
$F_{(behkn,22,10)}$	<i>base case</i>	0
$F_{(behkn,22,8)}$	<i>base case</i>	0
$G_{(behkn,14,12)}$	<i>base case</i>	0
$F_{(behkn,14,13)}$	<i>base case</i>	0
$F_{(behkn,14,11)}$	<i>base case</i>	0
$G_{(behkn,30,3)}$	$G_{(behkn,28,3)} + F_{(behkn,28,4)} + F_{(behkn,28,4)} + G_{(behkn,26,6)} + F_{(behkn,26,7)} + F_{(behkn,26,5)} + G_{(behkn,22,9)} + F_{(behkn,22,10)} + F_{(behkn,22,8)} + G_{(behkn,14,12)} + F_{(behkn,14,13)} + F_{(behkn,14,11)}$	0
$F_{(behkn,28,4)}$	$memoF_{(behkn,28,4)}$	0
$F_{(behkn,26,7)}$	<i>base case</i>	0

Tabel 5.44 Simulasi perhitungan jumlah kombinasi string *orig2* dengan operasi *replace* dengan *dist* = 3 pada kasus string *ad1* = *kbenh*, string *ad2* = *kbenh* dan *X* = 5 (2)

Fungsi	Perhitungan Nilai	Nilai
$F_{(behkn,22,10)}$	<i>base case</i>	0
$F_{(behkn,14,13)}$	<i>base case</i>	0
$F_{(behkn,30,4)}$	$F_{(behkn,28,4)} + F_{(behkn,26,7)} + F_{(behkn,22,10)} + F_{(behkn,14,13)}$	0
$F_{(behkn,30,4)}$	<i>memoF</i> $_{(behkn,30,4)}$	0
$G_{(behkn,29,6)}$	<i>base case</i>	0
$F_{(behkn,29,7)}$	<i>base case</i>	0
$F_{(behkn,29,5)}$	<i>memoF</i> $_{(behkn,29,5)}$	0
$G_{(behkn,27,9)}$	<i>base case</i>	0
$F_{(behkn,27,10)}$	<i>base case</i>	0
$F_{(behkn,27,8)}$	<i>base case</i>	0
$G_{(behkn,23,12)}$	<i>base case</i>	0
$F_{(behkn,23,13)}$	<i>base case</i>	0
$F_{(behkn,23,11)}$	<i>base case</i>	0
$G_{(behkn,15,15)}$	<i>base case</i>	0
$F_{(behkn,15,16)}$	<i>base case</i>	0
$F_{(behkn,15,14)}$	<i>base case</i>	0
$G_{(behkn,31,3)}$	$G_{(behkn,30,3)} + F_{(behkn,30,4)} + F_{(behkn,30,4)} + G_{(behkn,29,6)} + F_{(behkn,29,7)} + F_{(behkn,29,5)} + G_{(behkn,27,9)} + F_{(behkn,27,10)} + F_{(behkn,27,8)} + G_{(behkn,23,12)} + F_{(behkn,23,13)} + F_{(behkn,23,11)} + G_{(behkn,15,15)} + F_{(behkn,15,16)} + F_{(behkn,15,14)}$	0

Tabel 5.45 Simulasi perhitungan jumlah kombinasi string *orig2* dengan operasi *replace* dengan *dist* = 3 pada kasus string *ad1* = *kbenh*, string *ad2* = *kbenh* dan *X* = 5 (3)

Fungsi	Perhitungan Nilai	Nilai
$G_{(behkn,0,2)}$	<i>base case</i>	0
$F_{(behkn,0,3)}$	<i>base case</i>	0
$F_{(behkn,0,3)}$	<i>base case</i>	0
$G_{(behkn,16,2)}$	$G_{(behkn,0,2)} + F_{(behkn,0,3)} + F_{(behkn,0,3)}$	0
$F_{(behkn,16,3)}$	<i>memo</i> $F_{(behkn,16,3)}$	0
$F_{(behkn,16,3)}$	<i>memo</i> $F_{(behkn,16,3)}$	0
$G_{(behkn,0,8)}$	<i>base case</i>	0
$F_{(behkn,0,7)}$	<i>base case</i>	0
$F_{(behkn,0,9)}$	<i>base case</i>	0
$G_{(behkn,8,5)}$	$G_{(behkn,0,8)} + F_{(behkn,0,7)} + F_{(behkn,0,9)}$	0
$F_{(behkn,8,6)}$	<i>base case</i>	0
$F_{(behkn,0,7)}$	<i>base case</i>	0
$F_{(behkn,8,4)}$	$F_{(behkn,0,7)}$	0
$G_{(behkn,24,2)}$	$G_{(behkn,16,2)} + F_{(behkn,16,3)} + F_{(behkn,16,3)} + G_{(behkn,8,5)} + F_{(behkn,8,6)} + F_{(behkn,8,4)}$	0
$F_{(behkn,24,3)}$	<i>memo</i> $F_{(behkn,24,3)}$	0
$F_{(behkn,24,3)}$	<i>memo</i> $F_{(behkn,24,3)}$	0
$G_{(behkn,16,8)}$	<i>base case</i>	0
$F_{(behkn,16,7)}$	<i>base case</i>	0
$F_{(behkn,16,9)}$	<i>base case</i>	0
$G_{(behkn,4,8)}$	<i>base case</i>	0
$F_{(behkn,4,9)}$	<i>base case</i>	0
$F_{(behkn,4,7)}$	<i>base case</i>	0

Tabel 5.46 Simulasi perhitungan jumlah kombinasi string *orig1* dengan operasi *replace* dengan *dist* = 3 pada kasus string *ad1* = *kbenh*, string *ad2* = *kbenh* dan *X* = 5 (1)

Fungsi	Perhitungan Nilai	Nilai
$G_{(behkn,20,5)}$	$G_{(behkn,16,8)} + F_{(behkn,16,7)} + F_{(behkn,16,9)} + G_{(behkn,4,8)} + F_{(behkn,4,9)} + F_{(behkn,4,7)}$	0
$F_{(behkn,20,6)}$	base case	0
$F_{(behkn,16,7)}$	base case	0
$F_{(behkn,4,7)}$	base case	0
$F_{(behkn,20,4)}$	$F_{(behkn,16,7)} + F_{(behkn,4,7)}$	0
$G_{(behkn,12,8)}$	base case	0
$F_{(behkn,12,9)}$	base case	0
$F_{(behkn,12,7)}$	base case	0
$G_{(behkn,28,2)}$	$G_{(behkn,24,2)} + F_{(behkn,24,3)} + F_{(behkn,24,3)} + G_{(behkn,20,5)} + F_{(behkn,20,6)} + F_{(behkn,20,4)} + G_{(behkn,12,8)} + F_{(behkn,12,9)} + F_{(behkn,12,7)}$	0
$F_{(behkn,28,3)}$	$memoF_{(behkn,28,3)}$	0
$F_{(behkn,28,3)}$	$memoF_{(behkn,28,3)}$	0
$G_{(behkn,24,8)}$	base case	0
$F_{(behkn,24,7)}$	base case	0
$F_{(behkn,24,9)}$	base case	0
$G_{(behkn,18,8)}$	base case	0
$F_{(behkn,18,9)}$	base case	0
$F_{(behkn,18,7)}$	base case	0
$G_{(behkn,10,11)}$	base case	0
$F_{(behkn,10,12)}$	base case	0
$F_{(behkn,10,10)}$	base case	0

Tabel 5.47 Simulasi perhitungan jumlah kombinasi string *orig1* dengan operasi *replace* dengan *dist* = 3 pada kasus string *ad1* = *kbenh*, string *ad2* = *kbenh* dan *X* = 5 (2)

Fungsi	Perhitungan Nilai	Nilai
$G_{(behkn,26,5)}$	$G_{(behkn,24,8)} + F_{(behkn,24,7)} + F_{(behkn,24,9)} + G_{(behkn,18,8)} + F_{(behkn,18,9)} + F_{(behkn,18,7)} + G_{(behkn,10,11)} + F_{(behkn,10,12)} + F_{(behkn,10,10)}$	0
$F_{(behkn,26,6)}$	base case	0
$F_{(behkn,24,7)}$	base case	0
$F_{(behkn,18,7)}$	base case	0
$F_{(behkn,10,10)}$	base case	0
$F_{(behkn,26,4)}$	$F_{(behkn,24,7)} + F_{(behkn,18,7)} + F_{(behkn,10,10)}$	0
$G_{(behkn,22,8)}$	base case	0
$F_{(behkn,22,9)}$	base case	0
$F_{(behkn,22,7)}$	base case	0
$G_{(behkn,14,11)}$	base case	0
$F_{(behkn,14,12)}$	base case	0
$F_{(behkn,14,10)}$	base case	0
$G_{(behkn,30,2)}$	$G_{(behkn,28,2)} + F_{(behkn,28,3)} + F_{(behkn,28,3)} + G_{(behkn,26,5)} + F_{(behkn,26,6)} + F_{(behkn,26,4)} + G_{(behkn,22,8)} + F_{(behkn,22,9)} + F_{(behkn,22,7)} + G_{(behkn,14,11)} + F_{(behkn,14,12)} + F_{(behkn,14,10)}$	0
$F_{(behkn,30,3)}$	$memoF_{(behkn,30,3)}$	0
$F_{(behkn,30,3)}$	$memoF_{(behkn,30,3)}$	0
$G_{(behkn,28,8)}$	base case	0
$F_{(behkn,28,7)}$	base case	0
$F_{(behkn,28,9)}$	base case	0

Tabel 5.48 Simulasi perhitungan jumlah kombinasi string *orig1* dengan operasi *replace* dengan *dist* = 3 pada kasus string *ad1* = *kbenh*, string *ad2* = *kbenh* dan *X* = 5 (3)

Fungsi	Perhitungan Nilai	Nilai
$G_{(behkn,25,8)}$	<i>base case</i>	0
$F_{(behkn,25,9)}$	<i>base case</i>	0
$F_{(behkn,25,7)}$	<i>base case</i>	0
$G_{(behkn,21,11)}$	<i>base case</i>	0
$F_{(behkn,21,12)}$	<i>base case</i>	0
$F_{(behkn,21,10)}$	<i>base case</i>	0
$G_{(behkn,13,14)}$	<i>base case</i>	0
$F_{(behkn,13,15)}$	<i>base case</i>	0
$F_{(behkn,13,13)}$	<i>base case</i>	0
$G_{(behkn,29,5)}$	$G_{(behkn,28,8)} + F_{(behkn,28,7)} +$ $F_{(behkn,28,9)} + G_{(behkn,25,8)} +$ $F_{(behkn,25,9)} + F_{(behkn,25,7)} +$ $G_{(behkn,21,11)} + F_{(behkn,21,12)} +$ $F_{(behkn,21,10)} + G_{(behkn,13,14)} +$ $F_{(behkn,13,15)} + F_{(behkn,13,13)}$	0
$F_{(behkn,29,6)}$	<i>base case</i>	0
$F_{(behkn,28,7)}$	<i>base case</i>	0
$F_{(behkn,25,7)}$	<i>base case</i>	0
$F_{(behkn,21,10)}$	<i>base case</i>	0
$F_{(behkn,13,13)}$	<i>base case</i>	0
$F_{(behkn,29,4)}$	$F_{(behkn,28,7)} + F_{(behkn,25,7)} +$ $F_{(behkn,21,10)} + F_{(behkn,13,13)}$	0
$G_{(behkn,27,8)}$	<i>base case</i>	0
$F_{(behkn,27,9)}$	<i>base case</i>	0
$F_{(behkn,27,7)}$	<i>base case</i>	0
$G_{(behkn,23,11)}$	<i>base case</i>	0
$F_{(behkn,23,12)}$	<i>base case</i>	0

Tabel 5.49 Simulasi perhitungan jumlah kombinasi string *orig1* dengan operasi *replace* dengan *dist* = 3 pada kasus string *ad1* = *kbenh*, string *ad2* = *kbenh* dan *X* = 5 (4)

Fungsi	Perhitungan Nilai	Nilai
$F_{(behkn,23,10)}$	<i>base case</i>	0
$G_{(behkn,15,14)}$	<i>base case</i>	0
$F_{(behkn,15,15)}$	<i>base case</i>	0
$F_{(behkn,15,13)}$	<i>base case</i>	0
$G_{(behkn,31,2)}$	$G_{(behkn,30,2)} + F_{(behkn,30,3)} +$ $F_{(behkn,30,3)} + G_{(behkn,29,5)} +$ $F_{(behkn,29,6)} + F_{(behkn,29,4)} +$ $G_{(behkn,27,8)} + F_{(behkn,27,9)} +$ $F_{(behkn,27,7)} + G_{(behkn,23,11)} +$ $F_{(behkn,23,12)} + F_{(behkn,23,10)} +$ $G_{(behkn,15,14)} + F_{(behkn,15,15)} +$ $F_{(behkn,15,13)}$	0

Tabel 5.50 Simulasi perhitungan jumlah kombinasi string *orig1* dengan operasi *replace* dengan *dist* = 3 pada kasus string *ad1* = *kbenh*, string *ad2* = *kbenh* dan *X* = 5 (5)

Fungsi	Perhitungan Nilai	Nilai
$F_{(behkn,30,3)}$	<i>memo</i> $F_{(behkn,30,3)}$	0
$F_{(behkn,29,6)}$	<i>base case</i>	0
$F_{(behkn,27,9)}$	<i>base case</i>	0
$F_{(behkn,23,12)}$	<i>base case</i>	0
$F_{(behkn,15,15)}$	<i>base case</i>	0
$F_{(behkn,31,3)}$	$F_{(behkn,30,3)} + F_{(behkn,29,6)} +$ $F_{(behkn,27,9)} + F_{(behkn,23,12)} +$ $F_{(behkn,15,15)}$	0

Tabel 5.51 Simulasi perhitungan jumlah kombinasi string *orig2* tanpa operasi *replace* dengan *dist* = 3 pada kasus string *ad1* = *kbenh*, string *ad2* = *kbenh* dan *X* = 5

Fungsi	Perhitungan Nilai	Nilai
$F_{(behkn,0,1)}$	<i>base case</i>	0
$F_{(behkn,16,1)}$	$F_{(behkn,0,1)}$	0
$F_{(behkn,8,4)}$	<i>memo</i> $F_{(behkn,8,4)}$	0
$F_{(behkn,24,1)}$	$F_{(behkn,16,1)} + F_{(behkn,8,4)}$	0
$F_{(behkn,20,4)}$	<i>memo</i> $F_{(behkn,20,4)}$	0
$F_{(behkn,12,7)}$	<i>base case</i>	0
$F_{(behkn,28,1)}$	$F_{(behkn,24,1)} + F_{(behkn,20,4)} + F_{(behkn,12,7)}$	0
$F_{(behkn,26,4)}$	<i>memo</i> $F_{(behkn,26,4)}$	0
$F_{(behkn,22,7)}$	<i>base case</i>	0
$F_{(behkn,14,10)}$	<i>base case</i>	0
$F_{(behkn,30,1)}$	$F_{(behkn,28,1)} + F_{(behkn,26,4)} + F_{(behkn,22,7)} + F_{(behkn,14,10)}$	0
$F_{(behkn,29,4)}$	<i>memo</i> $F_{(behkn,29,4)}$	0
$F_{(behkn,27,7)}$	<i>base case</i>	0
$F_{(behkn,23,10)}$	<i>base case</i>	0
$F_{(behkn,15,13)}$	<i>base case</i>	0
$F_{(behkn,31,1)}$	$F_{(behkn,30,1)} + F_{(behkn,29,4)} + F_{(behkn,27,7)} + F_{(behkn,23,10)} + F_{(behkn,15,13)}$	0

Tabel 5.52 Simulasi perhitungan jumlah kombinasi string *orig1* tanpa operasi *replace* dengan *dist* = 4 pada kasus string *ad1* = *kbenh*, string *ad2* = *kbenh* dan *X* = 5

Fungsi	Perhitungan Nilai	Nilai
$G_{(behkn,0,4)}$	<i>base case</i>	0
$F_{(behkn,0,5)}$	<i>base case</i>	1
$F_{(behkn,0,5)}$	<i>base case</i>	1
$G_{(behkn,16,4)}$	$G_{(behkn,0,4)} + F_{(behkn,0,5)} + F_{(behkn,0,5)}$	2
$F_{(behkn,16,5)}$	$memoF_{(behkn,16,5)}$	1
$F_{(behkn,16,5)}$	$memoF_{(behkn,16,5)}$	1
$G_{(behkn,8,7)}$	<i>base case</i>	0
$F_{(behkn,8,8)}$	<i>base case</i>	0
$F_{(behkn,8,6)}$	<i>base case</i>	0
$G_{(behkn,24,4)}$	$G_{(behkn,16,4)} + F_{(behkn,16,5)} + F_{(behkn,16,5)} + G_{(behkn,8,7)} + F_{(behkn,8,8)} + F_{(behkn,8,6)}$	4
$F_{(behkn,24,5)}$	$memoF_{(behkn,24,5)}$	1
$F_{(behkn,24,5)}$	$memoF_{(behkn,24,5)}$	1
$G_{(behkn,20,7)}$	<i>base case</i>	0
$F_{(behkn,20,8)}$	<i>base case</i>	0
$F_{(behkn,20,6)}$	<i>base case</i>	0
$G_{(behkn,12,10)}$	<i>base case</i>	0
$F_{(behkn,12,11)}$	<i>base case</i>	0
$F_{(behkn,12,9)}$	<i>base case</i>	0
$G_{(behkn,28,4)}$	$G_{(behkn,24,4)} + F_{(behkn,24,5)} + F_{(behkn,24,5)} + G_{(behkn,20,7)} + F_{(behkn,20,8)} + F_{(behkn,20,6)} + G_{(behkn,12,10)} + F_{(behkn,12,11)} + F_{(behkn,12,9)}$	6
$F_{(behkn,28,5)}$	$memoF_{(behkn,28,5)}$	1
$F_{(behkn,28,5)}$	$memoF_{(behkn,28,5)}$	1

Tabel 5.53 Simulasi perhitungan jumlah kombinasi string *orig2* dengan operasi *replace* dengan *dist* = 4 pada kasus string *ad1* = *kbenh*, string *ad2* = *kbenh* dan *X* = 5 (1)

Fungsi	Perhitungan Nilai	Nilai
$G_{(behkn,26,7)}$	<i>base case</i>	0
$F_{(behkn,26,8)}$	<i>base case</i>	0
$F_{(behkn,26,6)}$	<i>base case</i>	0
$G_{(behkn,22,10)}$	<i>base case</i>	0
$F_{(behkn,22,11)}$	<i>base case</i>	0
$F_{(behkn,22,9)}$	<i>base case</i>	0
$G_{(behkn,14,13)}$	<i>base case</i>	0
$F_{(behkn,14,14)}$	<i>base case</i>	0
$F_{(behkn,14,12)}$	<i>base case</i>	0
$G_{(behkn,30,4)}$	$G_{(behkn,28,4)} + F_{(behkn,28,5)} +$ $F_{(behkn,28,5)} + G_{(behkn,26,7)} +$ $F_{(behkn,26,8)} + F_{(behkn,26,6)} +$ $G_{(behkn,22,10)} + F_{(behkn,22,11)} +$ $F_{(behkn,22,9)} + G_{(behkn,14,13)} +$ $F_{(behkn,14,14)} + F_{(behkn,14,12)}$	8
$F_{(behkn,28,5)}$	<i>memoF<sub>(behkn,28,5)</sub></i>	1
$F_{(behkn,26,8)}$	<i>base case</i>	0
$F_{(behkn,22,11)}$	<i>base case</i>	0
$F_{(behkn,14,14)}$	<i>base case</i>	0
$F_{(behkn,30,5)}$	$F_{(behkn,28,5)} + F_{(behkn,26,8)} +$ $F_{(behkn,22,11)} + F_{(behkn,14,14)}$	1
$F_{(behkn,30,5)}$	<i>memoF<sub>(behkn,30,5)</sub></i>	1
$G_{(behkn,29,7)}$	<i>base case</i>	0
$F_{(behkn,29,8)}$	<i>base case</i>	0

Tabel 5.54 Simulasi perhitungan jumlah kombinasi string *orig2* dengan operasi *replace* dengan *dist* = 4 pada kasus string *ad1* = *kbenh*, string *ad2* = *kbenh* dan *X* = 5 (2)

Fungsi	Perhitungan Nilai	Nilai
$F_{(behkn,29,6)}$	<i>base case</i>	0
$G_{(behkn,27,10)}$	<i>base case</i>	0
$F_{(behkn,27,11)}$	<i>base case</i>	0
$F_{(behkn,27,9)}$	<i>base case</i>	0
$G_{(behkn,23,13)}$	<i>base case</i>	0
$F_{(behkn,23,14)}$	<i>base case</i>	0
$F_{(behkn,23,12)}$	<i>base case</i>	0
$G_{(behkn,15,16)}$	<i>base case</i>	0
$F_{(behkn,15,17)}$	<i>base case</i>	0
$F_{(behkn,15,15)}$	<i>base case</i>	0
$G_{(behkn,31,4)}$	$G_{(behkn,30,4)} + F_{(behkn,30,5)} +$ $F_{(behkn,30,5)} + G_{(behkn,29,7)} +$ $F_{(behkn,29,8)} + F_{(behkn,29,6)} +$ $G_{(behkn,27,10)} + F_{(behkn,27,11)} +$ $F_{(behkn,27,9)} + G_{(behkn,23,13)} +$ $F_{(behkn,23,14)} + F_{(behkn,23,12)} +$ $G_{(behkn,15,16)} + F_{(behkn,15,17)} +$ $F_{(behkn,15,15)}$	10

Tabel 5.55 Simulasi perhitungan jumlah kombinasi string *orig2* dengan operasi *replace* dengan *dist* = 4 pada kasus string *ad1* = *kbenh*, string *ad2* = *kbenh* dan *X* = 5 (3)

Fungsi	Perhitungan Nilai	Nilai
$G_{(behkn,0,1)}$	<i>base case</i>	0
$F_{(behkn,0,2)}$	<i>base case</i>	0
$F_{(behkn,0,2)}$	<i>base case</i>	0
$G_{(behkn,16,1)}$	$G_{(behkn,0,1)} + F_{(behkn,0,2)} + F_{(behkn,0,2)}$	0
$F_{(behkn,16,2)}$	$memoF_{(behkn,16,2)}$	0
$F_{(behkn,16,2)}$	$memoF_{(behkn,16,2)}$	0
$G_{(behkn,0,7)}$	<i>base case</i>	0
$F_{(behkn,0,6)}$	<i>base case</i>	0
$F_{(behkn,0,8)}$	<i>base case</i>	0
$G_{(behkn,8,4)}$	$G_{(behkn,0,7)} + F_{(behkn,0,6)} + F_{(behkn,0,8)}$	0
$F_{(behkn,8,5)}$	$memoF_{(behkn,8,5)}$	0
$F_{(behkn,0,6)}$	<i>base case</i>	0
$F_{(behkn,8,3)}$	$F_{(behkn,0,6)}$	0
$G_{(behkn,24,1)}$	$G_{(behkn,16,1)} + F_{(behkn,16,2)} + F_{(behkn,16,2)} + G_{(behkn,8,4)} + F_{(behkn,8,5)} + F_{(behkn,8,3)}$	0
$F_{(behkn,24,2)}$	$memoF_{(behkn,24,2)}$	0
$F_{(behkn,24,2)}$	$memoF_{(behkn,24,2)}$	0
$G_{(behkn,16,7)}$	<i>base case</i>	0
$F_{(behkn,16,6)}$	<i>base case</i>	0
$F_{(behkn,16,8)}$	<i>base case</i>	0
$G_{(behkn,4,7)}$	<i>base case</i>	0
$F_{(behkn,4,8)}$	<i>base case</i>	0
$F_{(behkn,4,6)}$	<i>base case</i>	0

Tabel 5.56 Simulasi perhitungan jumlah kombinasi string *orig1* dengan operasi *replace* dengan *dist* = 4 pada kasus string *ad1* = *kbenh*, string *ad2* = *kbenh* dan *X* = 5 (1)

Fungsi	Perhitungan Nilai	Nilai
$G_{(behkn,20,4)}$	$G_{(behkn,16,7)} + F_{(behkn,16,6)} + F_{(behkn,16,8)} + G_{(behkn,4,7)} + F_{(behkn,4,8)} + F_{(behkn,4,6)}$	0
$F_{(behkn,20,5)}$	$memoF_{(behkn,20,5)}$	0
$F_{(behkn,16,6)}$	<i>base case</i>	0
$F_{(behkn,4,6)}$	<i>base case</i>	0
$F_{(behkn,20,3)}$	$F_{(behkn,16,6)} + F_{(behkn,4,6)}$	0
$G_{(behkn,12,7)}$	<i>base case</i>	0
$F_{(behkn,12,8)}$	<i>base case</i>	0
$F_{(behkn,12,6)}$	<i>base case</i>	0
$G_{(behkn,28,1)}$	$G_{(behkn,24,1)} + F_{(behkn,24,2)} + F_{(behkn,24,2)} + G_{(behkn,20,4)} + F_{(behkn,20,5)} + F_{(behkn,20,3)} + G_{(behkn,12,7)} + F_{(behkn,12,8)} + F_{(behkn,12,6)}$	0
$F_{(behkn,28,2)}$	$memoF_{(behkn,28,2)}$	0
$F_{(behkn,28,2)}$	$memoF_{(behkn,28,2)}$	0
$G_{(behkn,24,7)}$	<i>base case</i>	0
$F_{(behkn,24,6)}$	<i>base case</i>	0
$F_{(behkn,24,8)}$	<i>base case</i>	0
$G_{(behkn,18,7)}$	<i>base case</i>	0
$F_{(behkn,18,8)}$	<i>base case</i>	0
$F_{(behkn,18,6)}$	<i>base case</i>	0
$G_{(behkn,10,10)}$	<i>base case</i>	0
$F_{(behkn,10,11)}$	<i>base case</i>	0
$F_{(behkn,10,9)}$	<i>base case</i>	0

Tabel 5.57 Simulasi perhitungan jumlah kombinasi string *orig1* dengan operasi *replace* dengan *dist* = 4 pada kasus string *ad1* = *kbenh*, string *ad2* = *kbenh* dan *X* = 5 (2)

Fungsi	Perhitungan Nilai	Nilai
$G_{(behkn,26,4)}$	$G_{(behkn,24,7)} + F_{(behkn,24,6)} + F_{(behkn,24,8)} + G_{(behkn,18,7)} + F_{(behkn,18,8)} + F_{(behkn,18,6)} + G_{(behkn,10,10)} + F_{(behkn,10,11)} + F_{(behkn,10,9)}$	0
$F_{(behkn,26,5)}$	$memoF_{(behkn,26,5)}$	0
$F_{(behkn,24,6)}$	<i>base case</i>	0
$F_{(behkn,18,6)}$	<i>base case</i>	0
$F_{(behkn,10,9)}$	<i>base case</i>	0
$F_{(behkn,26,3)}$	$F_{(behkn,24,6)} + F_{(behkn,18,6)} + F_{(behkn,10,9)}$	0
$G_{(behkn,22,7)}$	<i>base case</i>	0
$F_{(behkn,22,8)}$	<i>base case</i>	0
$F_{(behkn,22,6)}$	<i>base case</i>	0
$G_{(behkn,14,10)}$	<i>base case</i>	0
$F_{(behkn,14,11)}$	<i>base case</i>	0
$F_{(behkn,14,9)}$	<i>base case</i>	0
$G_{(behkn,30,1)}$	$G_{(behkn,28,1)} + F_{(behkn,28,2)} + F_{(behkn,28,2)} + G_{(behkn,26,4)} + F_{(behkn,26,5)} + F_{(behkn,26,3)} + G_{(behkn,22,7)} + F_{(behkn,22,8)} + F_{(behkn,22,6)} + G_{(behkn,14,10)} + F_{(behkn,14,11)} + F_{(behkn,14,9)}$	0
$F_{(behkn,30,2)}$	$memoF_{(behkn,30,2)}$	0
$F_{(behkn,30,2)}$	$memoF_{(behkn,30,2)}$	0
$G_{(behkn,28,7)}$	<i>base case</i>	0
$F_{(behkn,28,6)}$	<i>base case</i>	0
$F_{(behkn,28,8)}$	<i>base case</i>	0
$G_{(behkn,25,7)}$	<i>base case</i>	0

Tabel 5.58 Simulasi perhitungan jumlah kombinasi string *orig1* dengan operasi *replace* dengan *dist* = 4 pada kasus string *ad1* = *kbenh*, string *ad2* = *kbenh* dan *X* = 5 (3)

Fungsi	Perhitungan Nilai	Nilai
$F_{(behkn,25,8)}$	<i>base case</i>	0
$F_{(behkn,25,6)}$	<i>base case</i>	0
$G_{(behkn,21,10)}$	<i>base case</i>	0
$F_{(behkn,21,11)}$	<i>base case</i>	0
$F_{(behkn,21,9)}$	<i>base case</i>	0
$G_{(behkn,13,13)}$	<i>base case</i>	0
$F_{(behkn,13,14)}$	<i>base case</i>	0
$F_{(behkn,13,12)}$	<i>base case</i>	0
$G_{(behkn,29,4)}$	$G_{(behkn,28,7)} + F_{(behkn,28,6)} +$ $F_{(behkn,28,8)} + G_{(behkn,25,7)} +$ $F_{(behkn,25,8)} + F_{(behkn,25,6)} +$ $G_{(behkn,21,10)} + F_{(behkn,21,11)} +$ $F_{(behkn,21,9)} + G_{(behkn,13,13)} +$ $F_{(behkn,13,14)} + F_{(behkn,13,12)}$	0
$F_{(behkn,29,5)}$	<i>memo</i> $F_{(behkn,29,5)}$	0
$F_{(behkn,28,6)}$	<i>base case</i>	0
$F_{(behkn,25,6)}$	<i>base case</i>	0
$F_{(behkn,21,9)}$	<i>base case</i>	0
$F_{(behkn,13,12)}$	<i>base case</i>	0
$F_{(behkn,29,3)}$	$F_{(behkn,28,6)} + F_{(behkn,25,6)} +$ $F_{(behkn,21,9)} + F_{(behkn,13,12)}$	0
$G_{(behkn,27,7)}$	<i>base case</i>	0
$F_{(behkn,27,8)}$	<i>base case</i>	0
$F_{(behkn,27,6)}$	<i>base case</i>	0
$G_{(behkn,23,10)}$	<i>base case</i>	0
$F_{(behkn,23,11)}$	<i>base case</i>	0
$F_{(behkn,23,9)}$	<i>base case</i>	0

Tabel 5.59 Simulasi perhitungan jumlah kombinasi string *orig1* dengan operasi *replace* dengan *dist* = 4 pada kasus string *ad1* = *kbenh*, string *ad2* = *kbenh* dan *X* = 5 (4)

Fungsi	Perhitungan Nilai	Nilai
$G_{(behkn,15,13)}$	<i>base case</i>	0
$F_{(behkn,15,14)}$	<i>base case</i>	0
$F_{(behkn,15,12)}$	<i>base case</i>	0
$G_{(behkn,31,1)}$	$G_{(behkn,30,1)} + F_{(behkn,30,2)} +$ $F_{(behkn,30,2)} + G_{(behkn,29,4)} +$ $F_{(behkn,29,5)} + F_{(behkn,29,3)} +$ $G_{(behkn,27,7)} + F_{(behkn,27,8)} +$ $F_{(behkn,27,6)} + G_{(behkn,23,10)} +$ $F_{(behkn,23,11)} + F_{(behkn,23,9)} +$ $G_{(behkn,15,13)} + F_{(behkn,15,14)} +$ $F_{(behkn,15,12)}$	0

Tabel 5.60 Simulasi perhitungan jumlah kombinasi string  $orig1$  dengan operasi *replace* dengan  $dist = 4$  pada kasus string  $ad1 = kbenh$ , string  $ad2 = kbenh$  dan  $X = 5$  (5)

Fungsi	Perhitungan Nilai	Nilai
$F_{(behkn,30,4)}$	<i>memo</i> $F_{(behkn,30,4)}$	0
$F_{(behkn,29,7)}$	<i>base case</i>	0
$F_{(behkn,27,10)}$	<i>base case</i>	0
$F_{(behkn,23,13)}$	<i>base case</i>	0
$F_{(behkn,15,16)}$	<i>base case</i>	0
$F_{(behkn,31,4)}$	$F_{(behkn,30,4)} + F_{(behkn,29,7)} +$ $F_{(behkn,27,10)} + F_{(behkn,23,13)} +$ $F_{(behkn,15,16)}$	0

Tabel 5.61 Simulasi perhitungan jumlah kombinasi string  $orig2$  tanpa operasi *replace* dengan  $dist = 4$  pada kasus string  $ad1 = kbenh$ , string  $ad2 = kbenh$  dan  $X = 5$

Fungsi	Perhitungan Nilai	Nilai
$F_{(behkn,0,0)}$	<i>base case</i>	0
$F_{(behkn,16,0)}$	$F_{(behkn,0,0)}$	0
$F_{(behkn,8,3)}$	<i>memoF</i> $_{(behkn,8,3)}$	0
$F_{(behkn,24,0)}$	$F_{(behkn,16,0)} + F_{(behkn,8,3)}$	0
$F_{(behkn,20,3)}$	<i>memoF</i> $_{(behkn,20,3)}$	0
$F_{(behkn,12,6)}$	<i>base case</i>	0
$F_{(behkn,28,0)}$	$F_{(behkn,24,0)} + F_{(behkn,20,3)} + F_{(behkn,12,6)}$	0
$F_{(behkn,26,3)}$	<i>memoF</i> $_{(behkn,26,3)}$	0
$F_{(behkn,22,6)}$	<i>base case</i>	0
$F_{(behkn,14,9)}$	<i>base case</i>	0
$F_{(behkn,30,0)}$	$F_{(behkn,28,0)} + F_{(behkn,26,3)} + F_{(behkn,22,6)} + F_{(behkn,14,9)}$	0
$F_{(behkn,29,3)}$	<i>memoF</i> $_{(behkn,29,3)}$	0
$F_{(behkn,27,6)}$	<i>base case</i>	0
$F_{(behkn,23,9)}$	<i>base case</i>	0
$F_{(behkn,15,12)}$	<i>base case</i>	0
$F_{(behkn,31,0)}$	$F_{(behkn,30,0)} + F_{(behkn,29,3)} + F_{(behkn,27,6)} + F_{(behkn,23,9)} + F_{(behkn,15,12)}$	0

Tabel 5.62 Simulasi perhitungan jumlah kombinasi string *orig1* tanpa operasi *replace* dengan *dist* = 5 pada kasus string *ad1* = *kbenh*, string *ad2* = *kbenh* dan *X* = 5

Fungsi	Perhitungan Nilai	Nilai
$G_{(behkn,0,5)}$	<i>base case</i>	0
$F_{(behkn,0,6)}$	<i>base case</i>	0
$F_{(behkn,0,6)}$	<i>base case</i>	0
$G_{(behkn,16,5)}$	$G_{(behkn,0,5)} + F_{(behkn,0,6)} + F_{(behkn,0,6)}$	0
$F_{(behkn,16,6)}$	<i>base case</i>	0
$F_{(behkn,16,6)}$	<i>base case</i>	0
$G_{(behkn,8,8)}$	<i>base case</i>	0
$F_{(behkn,8,9)}$	<i>base case</i>	0
$F_{(behkn,8,7)}$	<i>base case</i>	0
$G_{(behkn,24,5)}$	$G_{(behkn,16,5)} + F_{(behkn,16,6)} + F_{(behkn,16,6)} + G_{(behkn,8,8)} + F_{(behkn,8,9)} + F_{(behkn,8,7)}$	0
$F_{(behkn,24,6)}$	<i>base case</i>	0
$F_{(behkn,24,6)}$	<i>base case</i>	0
$G_{(behkn,20,8)}$	<i>base case</i>	0
$F_{(behkn,20,9)}$	<i>base case</i>	0
$F_{(behkn,20,7)}$	<i>base case</i>	0
$G_{(behkn,12,11)}$	<i>base case</i>	0
$F_{(behkn,12,12)}$	<i>base case</i>	0
$F_{(behkn,12,10)}$	<i>base case</i>	0
$G_{(behkn,28,5)}$	$G_{(behkn,24,5)} + F_{(behkn,24,6)} + F_{(behkn,24,6)} + G_{(behkn,20,8)} + F_{(behkn,20,9)} + F_{(behkn,20,7)} + G_{(behkn,12,11)} + F_{(behkn,12,12)} + F_{(behkn,12,10)}$	0
$F_{(behkn,28,6)}$	<i>base case</i>	0
$F_{(behkn,28,6)}$	<i>base case</i>	0

Tabel 5.63 Simulasi perhitungan jumlah kombinasi string *orig2* dengan operasi *replace* dengan *dist* = 5 pada kasus string *ad1* = *kbenh*, string *ad2* = *kbenh* dan *X* = 5 (1)

Fungsi	Perhitungan Nilai	Nilai
$G_{(behkn,26,8)}$	<i>base case</i>	0
$F_{(behkn,26,9)}$	<i>base case</i>	0
$F_{(behkn,26,7)}$	<i>base case</i>	0
$G_{(behkn,22,11)}$	<i>base case</i>	0
$F_{(behkn,22,12)}$	<i>base case</i>	0
$F_{(behkn,22,10)}$	<i>base case</i>	0
$G_{(behkn,14,14)}$	<i>base case</i>	0
$F_{(behkn,14,15)}$	<i>base case</i>	0
$F_{(behkn,14,13)}$	<i>base case</i>	0
$G_{(behkn,30,5)}$	$G_{(behkn,28,5)} + F_{(behkn,28,6)} +$ $F_{(behkn,28,6)} + G_{(behkn,26,8)} +$ $F_{(behkn,26,9)} + F_{(behkn,26,7)} +$ $G_{(behkn,22,11)} + F_{(behkn,22,12)} +$ $F_{(behkn,22,10)} + G_{(behkn,14,14)} +$ $F_{(behkn,14,15)} + F_{(behkn,14,13)}$	0
$F_{(behkn,30,6)}$	<i>base case</i>	0
$F_{(behkn,30,6)}$	<i>base case</i>	0
$G_{(behkn,29,8)}$	<i>base case</i>	0
$F_{(behkn,29,9)}$	<i>base case</i>	0
$F_{(behkn,29,7)}$	<i>base case</i>	0
$G_{(behkn,27,11)}$	<i>base case</i>	0
$F_{(behkn,27,12)}$	<i>base case</i>	0
$F_{(behkn,27,10)}$	<i>base case</i>	0
$G_{(behkn,23,14)}$	<i>base case</i>	0
$F_{(behkn,23,15)}$	<i>base case</i>	0
$F_{(behkn,23,13)}$	<i>base case</i>	0
$G_{(behkn,15,17)}$	<i>base case</i>	0

Tabel 5.64 Simulasi perhitungan jumlah kombinasi string *orig2* dengan operasi *replace* dengan *dist* = 5 pada kasus string *ad1* = *kbenh*, string *ad2* = *kbenh* dan *X* = 5 (2)

Fungsi	Perhitungan Nilai	Nilai
$F_{(behkn,15,18)}$	<i>base case</i>	0
$F_{(behkn,15,16)}$	<i>base case</i>	0
$G_{(behkn,31,5)}$	$G_{(behkn,30,5)} + F_{(behkn,30,6)} +$ $F_{(behkn,30,6)} + G_{(behkn,29,8)} +$ $F_{(behkn,29,9)} + F_{(behkn,29,7)} +$ $G_{(behkn,27,11)} + F_{(behkn,27,12)} +$ $F_{(behkn,27,10)} + G_{(behkn,23,14)} +$ $F_{(behkn,23,15)} + F_{(behkn,23,13)} +$ $G_{(behkn,15,17)} + F_{(behkn,15,18)} +$ $F_{(behkn,15,16)}$	0

Tabel 5.65 Simulasi perhitungan jumlah kombinasi string  $orig2$  dengan operasi *replace* dengan  $dist = 5$  pada kasus string  $ad1 = kbenh$ , string  $ad2 = kbenh$  dan  $X = 5$  (3)

Terakhir adalah fungsi *writeOutput*. Kompleksitas waktu dari fungsi *writeOutput* adalah  $\mathcal{O}(1)$ . Sehingga secara keseluruhan, kompleksitas waktu dari algoritma yang telah dirancang pada Tugas Akhir ini adalah  $\mathcal{O}(2^{|S|} * MAX\_DIST^2 * T)$ .

Fungsi	Perhitungan Nilai	Nilai
$G_{(behkn,0,0)}$	<i>base case</i>	0
$F_{(behkn,0,1)}$	<i>base case</i>	0
$F_{(behkn,0,1)}$	<i>base case</i>	0
$G_{(behkn,16,0)}$	$G_{(behkn,0,0)} + F_{(behkn,0,1)} + F_{(behkn,0,1)}$	0
$F_{(behkn,16,1)}$	$memoF_{(behkn,16,1)}$	0
$F_{(behkn,16,1)}$	$memoF_{(behkn,16,1)}$	0
$G_{(behkn,0,6)}$	<i>base case</i>	0
$F_{(behkn,0,5)}$	<i>base case</i>	1
$F_{(behkn,0,7)}$	<i>base case</i>	0
$G_{(behkn,8,3)}$	$G_{(behkn,0,6)} + F_{(behkn,0,5)} + F_{(behkn,0,7)}$	1
$F_{(behkn,8,4)}$	$memoF_{(behkn,8,4)}$	0
$F_{(behkn,0,5)}$	<i>base case</i>	1
$F_{(behkn,8,2)}$	$F_{(behkn,0,5)}$	1
$G_{(behkn,24,0)}$	$G_{(behkn,16,0)} + F_{(behkn,16,1)} + F_{(behkn,16,1)} + G_{(behkn,8,3)} + F_{(behkn,8,4)} + F_{(behkn,8,2)}$	2
$F_{(behkn,24,1)}$	$memoF_{(behkn,24,1)}$	0
$F_{(behkn,24,1)}$	$memoF_{(behkn,24,1)}$	0
$G_{(behkn,16,6)}$	<i>base case</i>	0
$F_{(behkn,16,5)}$	$memoF_{(behkn,16,5)}$	1
$F_{(behkn,16,7)}$	<i>base case</i>	0
$G_{(behkn,4,6)}$	<i>base case</i>	0
$F_{(behkn,4,7)}$	<i>base case</i>	0
$F_{(behkn,0,11)}$	<i>base case</i>	0
$F_{(behkn,4,5)}$	$F_{(behkn,0,11)}$	0

Tabel 5.66 Simulasi perhitungan jumlah kombinasi string *orig1* dengan operasi *replace* dengan *dist* = 5 pada kasus string *ad1* = *kbenh*, string *ad2* = *kbenh* dan *X* = 5 (1)

Fungsi	Perhitungan Nilai	Nilai
$G_{(behkn,20,3)}$	$G_{(behkn,16,6)} + F_{(behkn,16,5)} + F_{(behkn,16,7)} + G_{(behkn,4,6)} + F_{(behkn,4,7)} + F_{(behkn,4,5)}$	1
$F_{(behkn,20,4)}$	$memoF_{(behkn,20,4)}$	0
$F_{(behkn,16,5)}$	$memoF_{(behkn,16,5)}$	1
$F_{(behkn,4,5)}$	$memoF_{(behkn,4,5)}$	0
$F_{(behkn,20,2)}$	$F_{(behkn,16,5)} + F_{(behkn,4,5)}$	1
$G_{(behkn,12,6)}$	<i>base case</i>	0
$F_{(behkn,12,7)}$	<i>base case</i>	0
$F_{(behkn,8,8)}$	<i>base case</i>	0
$F_{(behkn,4,5)}$	$memoF_{(behkn,4,5)}$	0
$F_{(behkn,12,5)}$	$F_{(behkn,8,8)} + F_{(behkn,4,5)}$	0
$G_{(behkn,28,0)}$	$G_{(behkn,24,0)} + F_{(behkn,24,1)} + F_{(behkn,24,1)} + G_{(behkn,20,3)} + F_{(behkn,20,4)} + F_{(behkn,20,2)} + G_{(behkn,12,6)} + F_{(behkn,12,7)} + F_{(behkn,12,5)}$	4
$F_{(behkn,28,1)}$	$memoF_{(behkn,28,1)}$	0
$F_{(behkn,28,1)}$	$memoF_{(behkn,28,1)}$	0
$G_{(behkn,24,6)}$	<i>base case</i>	0
$F_{(behkn,24,5)}$	$memoF_{(behkn,24,5)}$	1
$F_{(behkn,24,7)}$	<i>base case</i>	0
$G_{(behkn,18,6)}$	<i>base case</i>	0
$F_{(behkn,18,7)}$	<i>base case</i>	0
$F_{(behkn,16,11)}$	<i>base case</i>	0
$F_{(behkn,2,8)}$	<i>base case</i>	0
$F_{(behkn,18,5)}$	$F_{(behkn,16,11)} + F_{(behkn,2,8)}$	0
$G_{(behkn,10,9)}$	<i>base case</i>	0
$F_{(behkn,10,10)}$	<i>base case</i>	0

Tabel 5.67 Simulasi perhitungan jumlah kombinasi string *orig1* dengan operasi *replace* dengan *dist* = 5 pada kasus string *ad1* = *kbenh*, string *ad2* = *kbenh* dan *X* = 5 (2)

Fungsi	Perhitungan Nilai	Nilai
$F_{(behkn,10,8)}$	<i>base case</i>	0
$G_{(behkn,26,3)}$	$G_{(behkn,24,6)} + F_{(behkn,24,5)} +$ $F_{(behkn,24,7)} + G_{(behkn,18,6)} +$ $F_{(behkn,18,7)} + F_{(behkn,18,5)} +$ $G_{(behkn,10,9)} + F_{(behkn,10,10)} +$ $F_{(behkn,10,8)}$	1
$F_{(behkn,26,4)}$	<i>memoF<sub>(behkn,26,4)</sub></i>	0
$F_{(behkn,24,5)}$	<i>memoF<sub>(behkn,24,5)</sub></i>	1
$F_{(behkn,18,5)}$	<i>memoF<sub>(behkn,18,5)</sub></i>	0
$F_{(behkn,10,8)}$	<i>base case</i>	0
$F_{(behkn,26,2)}$	$F_{(behkn,24,5)} + F_{(behkn,18,5)} +$ $F_{(behkn,10,8)}$	1
$G_{(behkn,22,6)}$	<i>base case</i>	0
$F_{(behkn,22,7)}$	<i>base case</i>	0
$F_{(behkn,20,8)}$	<i>base case</i>	0
$F_{(behkn,18,5)}$	<i>memoF<sub>(behkn,18,5)</sub></i>	0
$F_{(behkn,6,11)}$	<i>base case</i>	0
$F_{(behkn,22,5)}$	$F_{(behkn,20,8)} + F_{(behkn,18,5)} +$ $F_{(behkn,6,11)}$	0
$G_{(behkn,14,9)}$	<i>base case</i>	0
$F_{(behkn,14,10)}$	<i>base case</i>	0
$F_{(behkn,14,8)}$	<i>base case</i>	0
$G_{(behkn,30,0)}$	$G_{(behkn,28,0)} + F_{(behkn,28,1)} +$ $F_{(behkn,28,1)} + G_{(behkn,26,3)} +$ $F_{(behkn,26,4)} + F_{(behkn,26,2)} +$ $G_{(behkn,22,6)} + F_{(behkn,22,7)} +$ $F_{(behkn,22,5)} + G_{(behkn,14,9)} +$ $F_{(behkn,14,10)} + F_{(behkn,14,8)}$	6
$F_{(behkn,30,1)}$	<i>memoF<sub>(behkn,30,1)</sub></i>	0

Tabel 5.68 Simulasi perhitungan jumlah kombinasi string *orig1* dengan operasi *replace* dengan *dist* = 5 pada kasus string *ad1* = *kbenh*, string *ad2* = *kbenh* dan *X* = 5 (3)

Fungsi	Perhitungan Nilai	Nilai
$F_{(behkn,30,1)}$	$memoF_{(behkn,30,1)}$	0
$G_{(behkn,28,6)}$	<i>base case</i>	0
$F_{(behkn,28,5)}$	$memoF_{(behkn,28,5)}$	1
$F_{(behkn,28,7)}$	<i>base case</i>	0
$G_{(behkn,25,6)}$	<i>base case</i>	0
$F_{(behkn,25,7)}$	<i>base case</i>	0
$F_{(behkn,24,11)}$	<i>base case</i>	0
$F_{(behkn,17,8)}$	<i>base case</i>	0
$F_{(behkn,9,11)}$	<i>base case</i>	0
$F_{(behkn,25,5)}$	$F_{(behkn,24,11)} + F_{(behkn,17,8)} + F_{(behkn,9,11)}$	0
$G_{(behkn,21,9)}$	<i>base case</i>	0
$F_{(behkn,21,10)}$	<i>base case</i>	0
$F_{(behkn,21,8)}$	<i>base case</i>	0
$G_{(behkn,13,12)}$	<i>base case</i>	0
$F_{(behkn,13,13)}$	<i>base case</i>	0
$F_{(behkn,13,11)}$	<i>base case</i>	0
$G_{(behkn,29,3)}$	$G_{(behkn,28,6)} + F_{(behkn,28,5)} + F_{(behkn,28,7)} + G_{(behkn,25,6)} + F_{(behkn,25,7)} + F_{(behkn,25,5)} + G_{(behkn,21,9)} + F_{(behkn,21,10)} + F_{(behkn,21,8)} + G_{(behkn,13,12)} + F_{(behkn,13,13)} + F_{(behkn,13,11)}$	1
$F_{(behkn,29,4)}$	$memoF_{(behkn,29,4)}$	0

Tabel 5.69 Simulasi perhitungan jumlah kombinasi string *orig1* dengan operasi *replace* dengan *dist* = 5 pada kasus string *ad1* = *kbenh*, string *ad2* = *kbenh* dan *X* = 5 (4)

Fungsi	Perhitungan Nilai	Nilai
$F_{(behkn,28,5)}$	$memoF_{(behkn,28,5)}$	1
$F_{(behkn,25,5)}$	$memoF_{(behkn,25,5)}$	0
$F_{(behkn,21,8)}$	<i>base case</i>	0
$F_{(behkn,13,11)}$	<i>base case</i>	0
$F_{(behkn,29,2)}$	$F_{(behkn,28,5)} + F_{(behkn,25,5)} + F_{(behkn,21,8)} + F_{(behkn,13,11)}$	1
$G_{(behkn,27,6)}$	<i>base case</i>	0
$F_{(behkn,27,7)}$	<i>base case</i>	0
$F_{(behkn,26,8)}$	<i>base case</i>	0
$F_{(behkn,25,5)}$	$memoF_{(behkn,25,5)}$	0
$F_{(behkn,19,11)}$	<i>base case</i>	0
$F_{(behkn,11,14)}$	<i>base case</i>	0
$F_{(behkn,27,5)}$	$F_{(behkn,26,8)} + F_{(behkn,25,5)} + F_{(behkn,19,11)} + F_{(behkn,11,14)}$	0
$G_{(behkn,23,9)}$	<i>base case</i>	0
$F_{(behkn,23,10)}$	<i>base case</i>	0
$F_{(behkn,23,8)}$	<i>base case</i>	0
$G_{(behkn,15,12)}$	<i>base case</i>	0
$F_{(behkn,15,13)}$	<i>base case</i>	0
$F_{(behkn,15,11)}$	<i>base case</i>	0
$G_{(behkn,31,0)}$	$G_{(behkn,30,0)} + F_{(behkn,30,1)} + F_{(behkn,30,1)} + G_{(behkn,29,3)} + F_{(behkn,29,4)} + F_{(behkn,29,2)} + G_{(behkn,27,6)} + F_{(behkn,27,7)} + F_{(behkn,27,5)} + G_{(behkn,23,9)} + F_{(behkn,23,10)} + F_{(behkn,23,8)} + G_{(behkn,15,12)} + F_{(behkn,15,13)} + F_{(behkn,15,11)}$	8

Tabel 5.70 Simulasi perhitungan jumlah kombinasi string *orig1* dengan operasi *replace* dengan *dist* = 5 pada kasus string *ad1* = *kbenh*, string *ad2* = *kbenh* dan *X* = 5 (5)

Fungsi	Perhitungan Nilai	Nilai
$F_{(behkn,30,5)}$	$memoF_{(behkn,30,5)}$	1
$F_{(behkn,29,8)}$	<i>base case</i>	0
$F_{(behkn,27,11)}$	<i>base case</i>	0
$F_{(behkn,23,14)}$	<i>base case</i>	0
$F_{(behkn,15,17)}$	<i>base case</i>	0
$F_{(behkn,31,5)}$	$F_{(behkn,30,5)} + F_{(behkn,29,8)} + F_{(behkn,27,11)} + F_{(behkn,23,14)} + F_{(behkn,15,17)}$	1

Tabel 5.71 Simulasi perhitungan jumlah kombinasi string *orig2* tanpa operasi *replace* dengan *dist* = 5 pada kasus string *ad1* = *kbenh*, string *ad2* = *kbenh* dan *X* = 5

## **BAB VI**

### **KESIMPULAN**

Pada bab ini dijelaskan mengenai kesimpulan dari hasil uji coba yang telah dilakukan.

#### **6.1 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 teknik *meet in the middle* dan pendekatan *dynamic programming* dapat menyelesaikan permasalahan permasalahan klasik SPOJ 9967 *Playing With Words* dengan benar.
2. Kompleksitas waktu sebesar  $\mathcal{O}(2^{|S|} * MAX\_DIST^2 * T)$  cukup untuk 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 1.08 detik dan rata-rata 1.01 detik. Memori yang dibutuhkan adalah sebesar 19 MB.

#### **6.2 Saran**

Pada Tugas Akhir kali ini tentunya terdapat kekurangan serta nilai - nilai yang dapat penulis ambil. Berikut adalah saran - saran yang dapat diambil melalui Tugas Akhir ini:

1. Teknik *meet in the middle* adalah teknik yang sesuai untuk

menyelesaikan permasalahan apabila permasalahan tersebut dapat dibagi menjadi dua atau lebih submasalah yang tidak memiliki ketergantungan satu sama lain.

2. Paradigma *dynamic programming* adalah pendekatan yang sesuai untuk menyelesaikan permasalahan yang memiliki submasalah yang bersifat tumpang tindih dengan submasalah lainnya.

## **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**, Second., 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. 523–557, Mar. 1970.

*Halaman ini sengaja dikosongkan*

## **BIODATA PENULIS**



**Dewangga Winasforcepta Winardi**, lahir di Surabaya tanggal 18 Mei 1995. Penulis merupakan anak kedua dari 4 bersaudara. Penulis telah menempuh pendidikan formal TK Aisyiyah Bustanul Athfal Denpasar, SD Negeri 1 Ubung (2001-2007), SMP Negeri 5 Denpasar (2007-2010) dan SMA Negeri 4 Denpasar (2010-2013). Penulis melanjutkan studi kuliah program sarjana di Jurusan Teknik Informatika ITS.

Selama kuliah di Teknik Informatika ITS, penulis mengambil bidang minat Algoritma Pemrograman (AP). Penulis pernah menjadi asisten dosen dan praktikum untuk mata kuliah Dasar Pemrograman (2015 dan 2016), Struktur data (2015 dan 2016). Selama menempuh perkuliahan penulis juga aktif mengikuti kompetisi pemrograman tingkat nasional dan menjadi Juara 2 kategori pemrograman pada lomba COMPFEST Universitas Indonesia 2014. Selain itu penulis juga aktif di kegiatan organisasi dan kepanitiaan diantaranya menjadi staff Departemen Riset dan Teknologi HMTC ITS, wakil ketua National Programming Contest Schematics 2014, ketua National Programming Contest 2014, panitia Pemusatan Latihan Nasional 2 TOKI 2014, 2015, 2016 dan 2017 di ITS dan technical comitee Olimpiade Sains Nasional 2015 di Jogjakarta. Penulis dapat dihubungi melalui surel di [dewangga.winardi@gmail.com](mailto:dewangga.winardi@gmail.com).