ITS
Institut
Teknologi
Sepuluh Nopember

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

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 menyelesaikannya 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 $\mathcal{O}(2^{|S|} * MAX\_DIST)$*, 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

TO DO

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

Halaman ini sengaja dikosongkan

DAFTAR ISI

Halaman ini sengaja dikosongkan

DAFTAR TABEL

Halaman ini sengaja dikosongkan

DAFTAR GAMBAR

Halaman ini sengaja dikosongkan

DAFTAR KODE SUMBER

Halaman ini sengaja dikosongkan

BAB 1
PENDAHULUAN

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

1.1 Latar Belakang

Memasuki abad ke-20 perkembangan teknologi informasi semakin cepat. Tidak dapat dipungkiri bahwa hampir semua aspek kehidupan mulai terkomputerisasi. Tujuannya adalah konsistensi dan akurasi. Dengan desain dan implementasi yang tepat sebuah program komputer dapat membantu manusia untuk melakukan sebuah proses yang jika dikerjakan oleh manusia secara langsung akan membutuhkan waktu yang lama dan rentan terhadap error. Keunggulan inilah yang menyebabkan semakin banyak penggunaan teknologi informasi pada aspek-aspek kehidupan manusia.

Kesadaran akan keunggulan komputer dalam melakukan suatu pekerjaan menyebabkan semakin besar pula keinginan manusia untuk menyelesaikan masalah-masalah yang sebelumnya tidak dapat diselesaikan tanpa bantuan komputer. Masalah-masalah ini seringkali berkaitan dengan data yang memiliki ukuran sangat besar. Dari data tersebut tentu terdapat informasi-informasi tertentu yang ingin diambil. Dengan metode yang tepat informasi yang diperlukan dapat diambil dengan waktu yang relatif singkat.

Besarnya ukuran data tentu menjadi sebuah masalah tersendiri bagi para ilmuwan. Perkembangan perangkat keras yang ada belum dapat mengimbangi dengan kebutuhan komputasi yang semakin besar. Untuk inilah penyelesaian sebuah masalah membutuhkan pendekat-

an yang tepat, sehingga dapat ditemukan algoritma dan struktur data yang sesuai dengan permasalahan yang ada.

Topik 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 peforma 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. Algoritma yang digunakan diilhami oleh metode *meet in the middle* dan *dynamic programming*.
2. Implementasi algoritma menggunakan bahasa pemrograman C++.
3. *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. Melakukan analisis dan mendesain algoritma yang diilhami oleh metode *meet in the middle* dan *dynamic programming* untuk menyelesaikan permasalahan klasik SPOJ 9967 *Playing With Words*.
2. Melakukan implementasi algoritma dengan metode *meet in the middle* dan *dynamic programming* untuk menyelesaikan permasalahan klasik SPOJ 9967 *Playing With Words*.
3. Mengevaluasi algoritma dengan metode *meet in the middle* dan *dynamic programming* untuk menyelesaikan permasalahan klasik SPOJ 9967 *Playing With Words*.

1.5 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 dan struktur data 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 pengerjaan Tugas Akhir.

1.6 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 algortima 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.

Halaman ini sengaja dikosongkan

BAB 2
DASAR TEORI

Pada bab ini akan dijelaskan mengenai dasar teori yang menjadi dasar pengerjaan 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 $jarak(\text{"}ab\text{"}, \text{"}cd\text{"}) = |'a' -' c'| + |'b' -' d'| = 4$. Diberikan

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

Permasalahan ini adalah suatu permasalahan optimasi yang bertujuan untuk mencari banyak kemungkinan hasil yang ada. Teknik *meet in the middle* dapat diterapkan pada permasalahan ini karena permasalahan ini memiliki kriteria dapat dipecah menjadi beberapa submasalah yang dapat diselesaikan masing-masing tanpa bergantung dengan satu sama lain. Pendekatan *dynamic programming* dapat menyelesaikan masing-masing submasalah dari permasalahan ini karena submasalah yang dihasilkan memiliki kriteria submasalah optimal dan submasalah tumpang tindih.

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].

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) merajuk 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 menyelesaikannya 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 menyimpan 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].

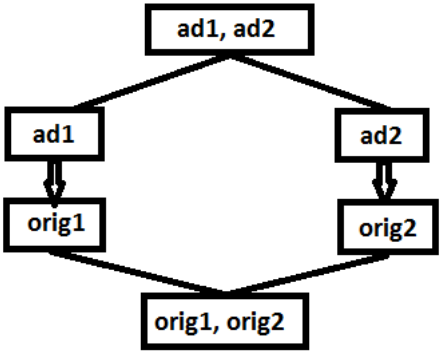
2.3 Analisa Submasalah Optimal

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 $dist(orig1, ad1) + dist(orig2, ad2) = X$. Sebagai contoh, dengan string $ad1 =$ "c" dan $ad2 =$ "n" dan $X = 1$, terdapat 4 kombinasi string $orig1$ dan $orig2$, yaitu $\{\{orig1 =$ "b", $orig2 =$ "n"$\}, \{orig1 =$ "d", $orig2 =$ "n"$\}, \{orig1 =$ "c", $orig2 =$ "m"$\}, \{orig1 =$ "c", $orig2 =$ "o"$\}\}$.

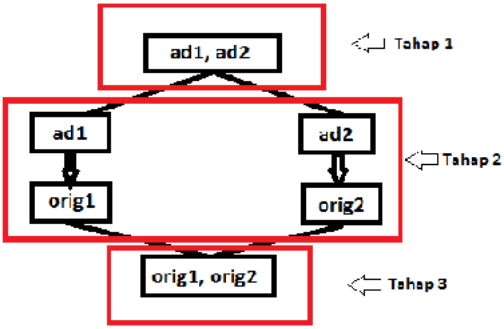
Teknik membagi permasalahan menjadi submasalah yang *independent* sering disebut dengan teknik *meet in the middle*. Teknik ini akan membagi permasalahan yang akan diberikan menjadi dua submasalah untuk masing-masing string yang diberikan. Sehingga string $ad1$ dan $ad2$ dapat diselesaikan masing-masing. Teknik ini tentu akan menghasilkan operasi yang lebih optimal dibandingkan dengan harus menghitung $ad1$ dan $ad2$ menjadi $orig1$ dan $orig2$ secara bersamaan.

Berdasarkan ilustrasi pada gambar 2.1, terlihat bahwa string $ad1$ dan $ad2$ bersifat *independent* karena memiliki karakteristik dapat diselesaikan tanpa saling bergantung satu sama lain.

Berdasarkan ilustrasi pada gambar 2.2, teknik *meet in the middle*akan membagi proses penyelesaian masalah menjadi tiga tahap, yaitu:



Gambar 2.1: Ilustrasi teknik *meet in the middle*.



Gambar 2.2: Ilustrasi tahapan teknik *meet in the middle*.

1. Tahap 1: Pembagian permasalahan menjadi submasalah yang *independent*.
2. Tahap 2: Penyelesaian submasalah.
3. Tahap 3: Penggabungan hasil penyelesaian submasalah.

2.3.1 Membagi Permasalah Menjadi Dua Submasalah yang *Independent*

Pada permasalahan klasik SPOJ 9967 *Playing With Words*, permasalahan akan dibagi menjadi dua submasalah yang *independent*. Permasalahan akan dibagi berdasarkan string yang diberikan. Dimana artinya permasalahan akan dibagi menjadi dua submasalah, yaitu submasalah untuk menghitung banyak kemungkinan string $orig1$ dari string $ad1$ dan submasalah untuk menghitung banyak kemungkinan string $orig2$ dari string $ad2$.

Tahap berikutnya yaitu menyelesaikan masing-masing submasalah. Karena submasalah yang kita pecah tidak saling bergantung satu sama lain, kita dapat menyelesaikannya secara terpisah. Pada permasalahan klasik SPOJ 9967 *Playing With Words*, kita dapat menghitung jumlah kemungkinan string $orig1$ dari string $ad1$ tanpa mempedulikan apa yang terjadi pada string $ad2$. Begitu pula sebaliknya, kita dapat menghitung jumlah kemungkinan string $orig2$ dari string $ad2$ tanpa mempedulikan apa yang terjadi pada string $ad1$.

Pada proses transformasi string $orig1$ dan $orig2$ menjadi string $ad1$ dan $ad2$, terdapat proses *replace* yaitu mengganti salah satu karakter pada string $orig1$ atau $orig2$. Sehingga untuk dapat menerapkan metode *meet in the middle* pada permasalahan ini, harus melakukan proses-proses perhitungan berikut:

1. Menghitung jumlah kemungkinan string $orig1$ dari string $ad1$ tanpa menggunakan operasi *replace*.
2. Menghitung jumlah kemungkinan string $orig1$ dari string $ad1$

dengan menggunakan operasi *replace*.
3. Menghitung jumlah kemungkinan string $orig2$ dari string $ad2$ tanpa menggunakan operasi *replace*.
4. Menghitung jumlah kemungkinan string $orig2$ dari string $ad2$ dengan menggunakan operasi *replace*.

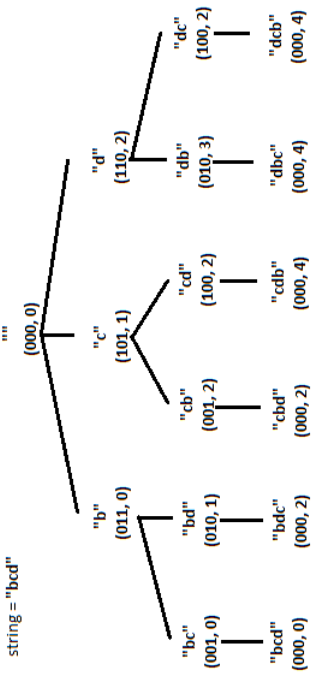
Jawaban akhir yang berupa jumlah kemungkinan string $orig1$ dan $orig2$ yang mungkin adalah hasil penjumlahan dari hasil perkalian proses 1 dengan proses 4 dengan hasil perkalian proses 2 dengan proses 3.

2.3.2 Submasalah Optimal untuk Menghitung Jumlah Kemungkinan String Awal Tanpa Operasi *Replace* dengan Jarak d

Submasalah jenis pertama pada permasalahan klasik SPOJ 9967 *Playing With Words* adalah menghitung jumlah kemungkinan string $orig$ dari string ad dengan jarak $dist(orig, ad) = d$ tanpa menggunakan operasi *replace*. Untuk dapat menyelesaikan permasalahan yang diberikan, diperlukan analisis submasalah-submasalah optimal. Pemecahan permasalahan menjadi submasalah-submasalah yang lebih kecil bertujuan untuk mempermudah penyelesaian masalah dengan menyelesaikan submasalah-submasalahnya terlebih dahulu.

Ilustrasi pada gambar 2.3 merupakan contoh penyelesaian submasalah tanpa operasi replace dengan string ad = "bcd". Berdasarkan ilustrasi pada gambar 2.3, didapatkan hasil dengan beberapa variasi jarak string $orig$ dan ad sebagai berikut:

1. Terdapat 1 solusi string $orig$ dengan jarak $d = 0$, yaitu string "bcd".
2. Terdapat 2 solusi string $orig$ dengan jarak $d = 2$, yaitu string "bdc" dan "cbd".
3. Terdapat 3 solusi string $orig$ dengan jarak $d = 4$, yaitu string



Gambar 2.3: Ilustrasi penyelesaian submasalah tanpa operasi replace.

"cdb", "dbc" dan "dcb".

Sehingga berdasarkan gambar 2.3 dapat kita simpulkan bahwa submasalah komputasi jumlah kemungkinan string awal tanpa operasi *replace* dapat dipecah menjadi submasalah yang lebih kecil untuk mempermudah menyelesaikan permasalahan utamanya.

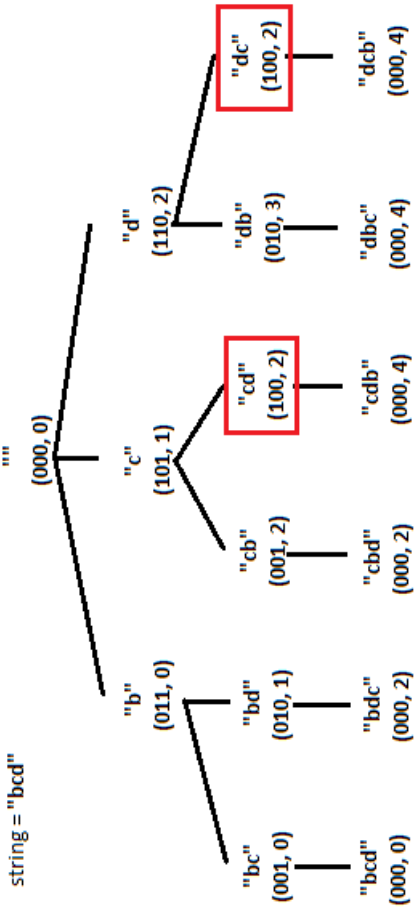
Berdasarkan gambar 2.4, dapat dilihat bahwa submasalah komputasi jumlah kemungkinan string awal tanpa operasi *replace* memiliki submasalah-submasalah optimal yang saling tumpang tindih. Sehingga, dapat digunakan teknik *dynamic programming* untuk mengoptimasi komputasi.

2.3.3 Submasalah Optimal untuk Menghitung Jumlah Kemungkinan String Awal dengan Operasi *Replace* dengan Jarak d

Submasalah jenis kedua pada permasalahan klasik SPOJ 9967 *Playing With Words* adalah menghitung jumlah kemungkinan string $orig$ dari string ad dengan jarak $dist(orig, ad) = d$ dengan menggunakan sekali operasi *replace*. Untuk dapat menyelesaikan permasalahan yang diberikan, diperlukan analisis submasalah-submasalah optimal. Pemecahan permasalahan menjadi submasalah-submasalah yang lebih kecil bertujuan untuk mempermudah penyelesaian masalah dengan menyelesaikan submasalah-submasalahnya terlebih dahulu.

Gambar 2.5 merupakan contoh dari ilustrasi penyelesaian submasalah menghitung jumlah string awal dengan sekali operasi *replace* jika diberikan string ad = "be". Berdasarkan ilustrasi pada gambar 2.5, didapatkan hasil dengan beberapa variasi jarak string $orig$ dan ad sebagai berikut:

1. Terdapat 4 solusi string $orig$ dengan jarak $d = 1$, yaitu string "ae", "bd", "bf" dan "ce".
2. Terdapat 1 solusi string $orig$ dengan jarak $d = 3$, yaitu string



Gambar 2.4: Contoh submasalah yang saling tumpang tindih

Gambar 2.5: Ilustrasi penyelesaian submasalah dengan sekali operasi replace.

"ec".
3. Terdapat 2 solusi string $orig$ dengan jarak $d = 5$, yaitu string "db" dan "ea".
4. Terdapat 1 solusi string $orig$ dengan jarak $d = 7$, yaitu string "fb".

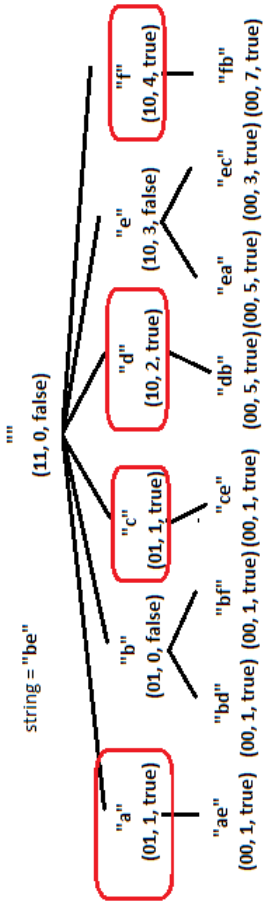
Berdasarkan gambar 2.6 dapat dilihat bahwa terdapat beberapa submasalah, yang ditandai dengan kotak berwarna merah, yang memiliki bentuk yang sama dengan submasalah tipe pertama, yaitu komputasi jumlah kemungkinan string awal tanpa operasi replace.

Submasalah komputasi kemungkinan string $orig$ dari string ad dengan jarak $dist(orig, ad) = d$ dengan menggunakan sekali operasi *replace* juga memiliki submasalah-submasalah yang saling tumpang tindih. Sehingga teknik *dynamic programming* dapat diterapkan untuk optimasi komputasi.

2.4 Pemodelan Relasi Rekurens

$$answer = \sum_{dist=0}^{dist=min_{(X,250)}} ((F_{(S_0,2^{|S_0|},bound-dist)} * G_{(S_1,2^{|S_1|},bound-X+dist)}) + (G_{(S_0,2^{|S_0|},bound-dist)} * F_{(S_1,2^{|S_1|},bound-X+dist)})) \quad (2.1)$$

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



Gambar 2.6: Komputasi submasalah tanpa operasi *replace* pada komputasi submasalah dengan operasi *replace*

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 2.1 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 $dist$

$$F_{(S,mask,dist)} = \begin{cases} 0, & \text{if } dist > bound, \\ & \text{or } (mask = 0 \text{ and} \\ & dist \neq bound) \\ 1, & \text{if } mask = 0 \text{ and}, \\ & dist = bound \\ \sum_{i=0}^{i=NSB_{(mask)}} & \\ F1_{(S,mask,set\_bit(mask)_i,dist)}, & \text{otherwise} \end{cases} \tag{2.2}$$

$$F1_{(S,mask,idx,dist)} = \begin{cases} F_{(S,mask-2^{idx},dist+ \\ |S_{idx}-S_{curIdx}|)}, & \text{idx = |S| - 1 or} \\ & duplicate\_rule1_{(S,mask,idx)} = \\ & True \\ 0, & \text{otherwise} \end{cases} \tag{2.3}$$

$$duplicate\_rule1_{(S,mask,idx)} = \begin{cases} True, & \text{if } idx < |S|-1 \text{ and} \\ & (S_{idx} \neq S_{idx+1} \\ & \text{or } ((S_{idx} = \\ & S_{idx+1}) \text{ and} \\ & (is\_off_{(mask,idx+1)})) \\ False, & \text{otherwise} \end{cases} \tag{2.4}$$

Pada persamaan 2.1 terdapat fungsi $F_{(S,mask,dist)}$ yang merupakan 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 $mask$. 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 2.3 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 2.2 adalah relasi rekurens dari submasalah perhitungan jumlah kemungkinan string $orig$ dari string S tanpa operasi *replace* dengan jarak $dist$.

2.4.2 Pemodelan Relasi Rekurens Submasalah Optimal untuk Menghitung Jumlah Kemungkinan String Awal dengan Sekali Operasi *Replace* dengan Jarak d

$$G_{(S,mask,dist)} = \begin{cases} 0, & dist > bound \text{ or } \\ & mask = bound \\ \sum_{i=0}^{i=NSB_{(mask)}} & \\ (G1_{(S,mask,} & \\ set\_bit_{(mask)i},dist) & \\ +G2_{(S,mask,} & \\ set\_bit_{(mask)i},dist) & \\ +G3_{(S,mask,} & \\ set\_bit_{(mask)i},dist)), & \text{otherwise} \end{cases} \tag{2.5}$$

Pada persamaan 2.1 terdapat fungsi $G_{(S,mask,d)}$ yang merupakan fungsi untuk menghitung jumlah kemungkinan string $orig$ dari string S dengan sekali operasi *replace* dengan jarak $dist$. Sama halnya dengan fungsi $F_{(S,mask,dist)}$, nilai dari fungsi $G_{(S,mask,d)}$ 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 keti-

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

1. Persamaan 2.6 adalah persamaan yang menentukan apakah *state* $G_{(S,mask-2^{idx},dist+|S_{idx}-S_{curIdx}|)}$ merupakan sebuah *state* yang valid dari *state* $G_{(S,mask,d)}$.
2. Persamaan 2.7 adalah persamaan yang menentukan apakah *state* $F_{(S,mask-2^{idx},dist+|S_{idx}+1-S_{curIdx}|)}$ merupakan sebuah *state* yang valid dari *state* $G_{(S,mask,d)}$ dengan $charFirstPos_{(S,C)}$ adalah posisi pertama karakter C pada string S. Apabila karakter C tidak ada pada string S makan akan bernilai -1.
3. Persamaan 2.8 adalah persamaan yang menentukan apakah *state* $F_{(S,mask-2^{idx},dist+|S_{idx}-1-S_{curIdx}|)}$ merupakan sebuah *state* yang valid dari *state* $G_{(S,mask,d)}$ dengan $charLastPos_{(S,C)}$ adalah posisi terakhir karakter C pada string S. Apabila karakter C tidak ada pada string S makan akan bernilai -1.

$$G1_{(S,mask,idx,dist)} = \begin{cases} G_{(S,mask} \\ -_{2^{idx},dist} \\ +|S_{idx} \\ -S_{curIdx}|), & idx = |S| - 1 \text{ or} \\ & duplicate\_rule1_{(S,mask,idx)} = \\ & True \\ 0, & \text{otherwise} \end{cases} \tag{2.6}$$

$$G2_{(S,mask,idx,dist)} = \begin{cases} F_{(S,mask} \\ -2^{idx},dist \\ +|S_{idx}+1 \\ -S_{curIdx}|), & idx = |S| - 1 \text{ or} \\ & (duplicate\_rule1_{(S,mask,idx)} = \\ & True \quad \text{and} \\ & duplicate\_rule2_{(S,mask,idx)} = \\ & True) \\ 0, & \text{otherwise} \end{cases} \tag{2.7}$$

$$G3_{(S,mask,idx,dist)} = \begin{cases} F_{(S,mask} \\ -2^{idx},dist \\ +|S_{idx}-1 \\ -S_{curIdx}|), & (idx = |S| - 1 \text{ or } duplicate\_rule1_{(S,mask,idx)} = True) \text{ and } (idx = 0 \text{ or } duplicate\_rule3_{(S,mask,idx)} = True) \\ 0, & \text{otherwise} \end{cases} \tag{2.8}$$

$$duplicate\_rule2_{(S,mask,idx)} = \begin{cases} True, & \text{if } idx < |S| - 1 \text{ and } (charFirstPos_{(S,S_{idx}+1)} = -1 \text{ or } (charFirstPos_{(S,S_{idx}+1)} \neq -1 \text{ and } is\_off_{(mask,} \\ & charFirstPos_{(S,S_{idx}+1)}) \\ False, & \text{otherwise} \end{cases} \tag{2.9}$$

$$duplicate\_rule3(S, mask, idx) = \begin{cases} True, & \text{if } idx > 0 - 1 \text{ and} \\ & (charLastPos(S, S_{idx} - \\ & 1) = -1 \text{ or} \\ & (charLastPos(S, S_{idx} - \\ & 1) \neq -1 \text{ and} \\ & is\_on_{(mask,} \\ & {}_{charFirstPos_{(S,S_{idx}+1)})} \\ False, & \text{otherwise} \end{cases} \tag{2.10}$$

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

BAB 3
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 terliat 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 mak-

```
Main()
1  preprocess()
2  TC = 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 isBitOn(num, bitPos)
4                 setBit_(powerNum).push(bitPos)
```

Gambar 3.2: Pseudocode Fungsi Preprocess

simal string $ad1$ dan $ad2$ yang mungkin. Gambar 3.2 adalah pseudocode untuk fungsi *preprocess*.

3.3 Desain Fungsi Init

Fungsi init merupakan fungsi yang bertujuan untuk melakukan inisialisasi 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 paradigma *dynamic programming* yang menggunakan teknik memoi-

sasi, maka algoritma yang dibangun harus melakukan inisialisasi nilai untuk setiap memo yang digunakan. Terdapat dua variabel memo yang digunakan pada algoritma yang dibangun, yaitu $memoF_{(idx,mask,dist)}$ untuk mencatat hasil perhitungan fungsi $F_{(S,mask,dist)}$ dan $memoG_{(idx,mask,dist)}$ 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 secara alfabetis ascending atau descending.

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. $maxMask_{(S)}$ yaitu nilai maksimal $mask$ untuk string S. Nilai maksimal $mask$ dari string S adalah $2^{|S|} - 1$.
2. $charFirstPos_{(S,C)}$ yaitu posisi pertama karakter C pada string S.
3. $charLastPos_{(S,C)}$ yaitu posisi terakhir karakter C pada string S.

Tentunya tidak semua karakter yang sesuai dengan batasan soal selalu muncul pada string S. Contohnya ketika S = "$contoh$" hanya karakter $\{c, h, n, o\}$ yang muncul. Sehingga fungsi $charFirstPos_{(S,C)}$ dan $charLastPos_{(S,C)}$ untuk setiap C, dimana $C \notin A$ dengan A adalah himpunan karakter yang muncul pada string S, tidak memiliki nilai atau bernilai $\varnothing$. Gambar 3.3 adalah pseudocode untuk fungsi init.

```
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,ad1_i) = i
 9       if charFirstPos_(ad1,ad1_i) = ∅
10            charFirstPos_(ad1,ad1_i) = i
11  maxMask_(ad2) = 2^|ad2| − 1
12  sort(ad2)
13  for i = 0 to |ad2| − 1
14       charLastPos_(ad2,ad2_i) = i
15       if charFirstPos_(ad2,ad2_i) = ∅
16            charFirstPos_(ad2,ad2_i) = i
```

Gambar 3.3: Pseudocode Fungsi Init

```
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,maxMask_ad1,bound−dist)
          +G(ad2,maxMask_ad2,bound−rem)
10       ret = ret + G(ad1,maxMask_ad1,bound−dist)
          +F(ad2,maxMask_ad2,bound−rem)
11  return ret
```

Gambar 3.4: Pseudocode Fungsi Solve

3.4 Desain Fungsi Solve

Fungsi solve adalah fungsi yang bertujuan untuk menyelesaikan permasalahan sesuai dengan deskripsi permasalah untuk setiap input yang diberikan. Setalah melalui proses *preprocessing*, membaca 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 2.1. 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 seperti 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 2.2. Gambar 3.5 adalah pseudocode dari fungsi $F_{(S,mask,dist)}$.

Karena tidak semua *state* yang terhubung dengan *state* $F_{(S,mask,dist)}$ valid, maka diperlukan sebuah fungsi $F1_{(S,mask,idx,dist)}$ 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 2.3. Gambar 3.6 adalah pseudocode dari fungsi $F1_{(S,mask,idx,dist)}$ dan gambar 3.7 adalah pseudocode dari fungsi $duplicate\_rule1(S, mask, idx)$.

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 $G_{(S,mask,d)}$ 2 yang merupakan fungsi untuk menghitung jumlah kemungkinan string $orig$ dari string S dengan sekali operasi *replace* dengan jarak $dist$. Nilai dari fungsi $G_{(S,mask,dist)}$ adalah hasil penjumlahan dari seluruh *state* yang berhubungan dengan dengan *state* $G_{(S,mask,dist)}$ yang valid. Gambar 3.8 adalah pseudocode dari fung-

```
F(S, mask, dist)
 1  if dist > bound ∨ (mask = 0 ∧ dist ≠ bound)
 2       return 0
 3  if mask = 0 ∧ dist = bound
 4       return 1
 5  if memoF_(S,mask,dist) ≠ ∅
 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 ≥ 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

```
F1(S, mask, idx, dist)
1  curIdx = length_(S) − size_(setBit_(mask))
2  if idx = length_(S) − 1 ∨ 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 < length_(S) − 1 ∧ (S_idx ≠idx+1 ∨
   (S_idx = S_idx+1 ∧ ¬isBitOn_(mask,idx+1)))
```

Gambar 3.7: Pseudocode Fungsi duplicate_rule1

si $G_{(S,mask,dist)}$. 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 $duplicate\_rule3(S, mask, idx)$.

```
G(S, mask, dist)
 1  if dist > bound ∨ mask = 0
 2      return 0
 3  if memoG_(S,mask,dist) ≠ ∅
 4      return memoG_(S,mask,dist)
 5  numberOfSetBit = size_(setBit_(mask))
 6  retVal = 0
 7  for i = 0 to numberOfSetBit − 1
 8      if setBit_(mask)i ≥ 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) − size_(setBit_(mask))
2  if idx = length_(S) − 1 ∨ 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 $curIdx = \text{length}_{(S)} - \text{size}_{(setBit_{(mask)})}$
2 **if** $idx = \text{length}_{(S)} - 1 \vee (\text{duplicateRule2}_{(S,mask,idx)} \wedge$
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 $curIdx = \text{length}_{(S)} - \text{size}_{(setBit_{(mask)})}$
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** $idx < length_{(S)} - 1 \wedge (charFirstPos_{(S,S_{idx}+1)}$
$= \varnothing \vee (charFirstPos_{(S,S_{idx}+1)} \neq \varnothing$
$\wedge \neg isBitOn_{(mask,charFirstPos_{(S,S_{idx}+1)})}))$

Gambar 3.12: Pseudocode Fungsi duplicate_rule2

duplicate_rule3$(S, mask, idx)$

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

Gambar 3.13: Pseudocode Fungsi duplicate_rule3

Halaman ini sengaja dikosongkan

BAB 4

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 Disjoint Subtrees 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 3.

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 cstdio, algorithm, vector, string dan cstring, seperti yang terlihat pada

kode sumber 4.1.

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

Kode Sumber 4.1: 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 menyala dari sebuah bilangan bulat. Header algorithm berisi modul yang memiliki fungsi - fungsi yang sangat berguna dalam membantu mengimplementasi algortima 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 4.2.

```
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 4.2: Variabel global

4.3.3 Implementasi Fungsi Main

Berdasarkan pada desain fungsi main yang telah dirancang pada gambar 3.1, awalnya program menerima masukan data berupa banyaknya kasus uji. Berikutnya, untuk setiap kasus uji, program akan membaca input data berupa string $ad1$, string $ad2$ dan sebuah bilangan bulat X. Berikutnya program akan memanggil fungsi init() yang berfungsi melakukan inisialisasi nilai pada data-data yang diperlukan untuk menyelesaikan permasalahan klasik SPOJ 9967 *Playing With Words*. Berikutnya program akan melakukan perhitungan penyelesaian permasalahan klasik SPOJ 9967 *Playing With Words* untuk kasus uji tersebut dengan cara memanggil fungsi solveProbelm. Berikutnya hasil dari perhitungan fungsi solveProbelm akan dicetak ke layar dengan cara memanggil fungsi writeOutput. Implementasi fungsi main dapat dilihat pada kode sumber 4.3.

```
 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: 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 4.4.

```
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))
                               ↪ set_bit[i].
                               ↪ push_back(j);
5                  }
6          }
7  }
```

Kode Sumber 4.4: Fungsi main

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. Fungsi yang digunakan untuk membaca masukan adalah fungsi scanf yang disediakan oleh header *cstdio*. Implementasi fungsi readInput dapat dilihat pada kode sumber 4.5.

```
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 4.5: Fungsi readInput

4.3.6 Implementasi Fungsi Init

Seperti yang telah dipaparkan pada subbab 3.3, Fungsi init merupakan fungsi yang bertujuan untuk melakukan inisialisasi nilai awal dan perhitungan data - data yang diperlukan untuk menyelesaikan permasalahan klasik SPOJ 9967 *Playing With Words* untuk setiap kasus uji.

Pada fungsi ini akan memberikan nilai awal pada variabel $memoF$ dan $memoG$ yang berfungsi untuk menyimpan hasil perhitungan solusi atau biasa disebut dengan memoisasi. Pada pseudocode 3.3, variabel $memoF$ dan $memoG$ diinisialisasi sebagai sebuah himpunan kosong. Namun, untuk mempermudah implementasi algoritma, nilai variabel $memoF$ dan $memoG$ akan diinisialilasi dengan array yang terdiri dari nilai -1 sebanyak ukuran array $memoF$ dan $memoG$. Selain inisialisasi nilai $memoF$ dan nilai $memoG$, pada fungsi init juga akan dilakukan perhitungan nilai $charFirstPos$ dan $charLastPos$ dimana $charFistPos$ adalah posisi kemunculan pertama sebuah karakter pada sebuah string dan $charLastPos$ adalah posisi kemunculan terakhir sebuah karakter pada sebuah string. Apabila sebuah karakter tidak muncul pada string tersebut, maka akan diberikan nilai *default* -1. Yang terakhir adalah inisialisasi nilai $bound$ yang bernilai nilai minimal antara nilai masukan X dengan konstanta 250. Implementasi dari fungsi init dapat dilihat pada kode sumber 4.6.

```
1  void init() {
2          memset(charLastPos, -1, sizeof
               ↪ charLastPos);
3          memset(charFirstPos, -1, sizeof
               ↪ charFirstPos);
4          memset(memoF, -1, sizeof memoF);
5          memset(memoG, -1, sizeof memoG);
6          for (int idx = 0; idx < 2; idx++) {
7                  sort(S[idx].begin(), S[idx].end()
                       ↪ );
8                  maxMask[idx] = (1 << S[idx].
```

```
                    ↪ length()) - 1;
 9                  for (int i = 0; i < S[idx].length
                        ↪ (); i++) {
10                      charLastPos[idx][S[idx][i
                            ↪ ] - 'a'] = i;
11                      if (charFirstPos[idx][S[
                            ↪ idx][i] - 'a'] ==
                            ↪ -1) {
12                          charFirstPos[idx
                                ↪ ][S[idx][i]
                                ↪  - 'a'] = i
                                ↪ ;
13                      }
14                  }
15          }
16          bound = min(X, 250);
17  }
```

Kode Sumber 4.6: 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 2.1. Implementasi dari fungsi solve dapat dilihat pada kode sumber 4.7.

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

```
                    ↪ bound - rem);
 9          }
10          return ret;
11  }
```

Kode Sumber 4.7: Fungsi solve

4.3.8 Implementasi Fungsi F

Pada pseudocode fungsi solve pada gambar 3.4, terdapat fungsi F yang telah dirancang algoritmanya pada subbab 3.4.1. Implementasi dari fungsi F dapat dilihat pada kode sumber 4.8. Algoritma pada fungsi F menggunakan pendekatan *dynamic programming*. Baris 2 dan 3 pada kode sumber adalah implementasi *base case* pada algoritma. Baris 4 adalah pengecekan memo sehingga algoritma tidak melakukan perhitungan yang sudah pernah dikerjakan sebelumnya.

```
 1  long long F(int idx, int mask, int dist) {
 2          if (dist > bound || (mask == 0 && dist !=
                ↪  bound)) return 0;
 3          if (mask == 0 && dist == bound) return 1;
 4          if (memoF[idx][mask][dist] != -1) return
                ↪ memoF[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].
                        ↪ length()) break;
 9                  ret += F1(idx, mask, set_bit[mask
                        ↪ ][i], dist);
10          }
11          return memoF[idx][mask][dist] = ret;
12  }
```

Kode Sumber 4.8: Fungsi F

4.3.9 Implementasi Fungsi F1

Fungsi F1 adalah implementasi dari perancangan pada pseudocode 3.6 yang dirancang berdasarkan persamaan 2.3. Implementasi dari fungsi F1 dapat dilihat pada kode sumber 4.9.

```
1  long long F1(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)
               ↪ ) {
4                  return F(idx, mask - (1 <<
                       ↪ charIdx), dist + abs(S[idx
                       ↪ ][charIdx] - S[idx][curIdx
                       ↪ ]));
5          }
6          return 0;
7  }
```

Kode Sumber 4.9: Fungsi F1

4.3.10 Implementasi Fungsi G

Pada pseudocode fungsi solve pada gambar 3.4, terdapat fungsi G yang telah dirancang algoritmanya pada subbab 3.4.2. Implementasi dari fungsi G dapat dilihat pada kode sumber 4.10. Algoritma pada fungsi F menggunakan pendekatan *dynamic programming*. Baris 2 pada kode sumber adalah implementasi *base case* pada algoritma. Baris 3 adalah pengecekan memo sehingga algoritma tidak melakukan perhitungan yang sudah pernah dikerjakan sebelumnya.

```
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
               ↪ memoG[idx][mask][dist];
4          int NSB = set_bit[mask].size();
```

```
5           long long ret = 0;
6           for (int i=0; i<NSB; i++) {
7                   if (set_bit[mask][i] >= S[idx].
                        ↪ length()) break;
8                   ret += G1(idx, mask, set_bit[mask
                        ↪ ][i], dist);
9                   ret += G2(idx, mask, set_bit[mask
                        ↪ ][i], dist);
10                  ret += G3(idx, mask, set_bit[mask
                        ↪ ][i], dist);
11          }
12          return memoG[idx][mask][dist] = ret;
13  }
```

Kode Sumber 4.10: Fungsi G

4.3.11 Implementasi Fungsi G1

Fungsi G1 adalah implementasi dari perancangan pada pseudocode 3.9 yang dirancang berdasarkan persamaan **??**. Implementasi dari fungsi G1 dapat dilihat pada kode sumber 4.11.

```
1   long long G1(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)
                ↪ ) {
4                   return G(idx, mask - (1 <<
                        ↪ charIdx), dist + abs(S[idx
                        ↪ ][charIdx] - S[idx][curIdx
                        ↪ ]));
5           }
6           return 0;
7   }
```

Kode Sumber 4.11: Fungsi G1

4.3.12 Implementasi Fungsi G2

Fungsi G2 adalah implementasi dari perancangan pada pseudocode 3.10 yang dirancang berdasarkan persamaan **??**. Implementasi dari fungsi G2 dapat dilihat pada kode sumber 4.12.

```
1  long long G2(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_rule2(idx, mask, charIdx)
               ↪  && duplicate_rule1(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 4.12: Fungsi G2

4.3.13 Implementasi Fungsi G3

Fungsi G3 adalah implementasi dari perancangan pada pseudocode 3.11 yang dirancang berdasarkan persamaan **??**. Implementasi dari fungsi G3 dapat dilihat pada kode sumber 4.13.

```
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 4.13: 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. Implementasi dari fungsi duplicate_rule1 dapat dilihat pada kode sumber 4.14.

```
1   bool duplicate_rule1(int idx, int mask, int
        ↪ charIdx) {
2           return (charIdx < S[idx].length() - 1
3                   && (S[idx][charIdx] != S[idx][
                        ↪ charIdx + 1]
4                   || (S[idx][charIdx] == S[idx][
                        ↪ charIdx + 1]
5                   && !isBitOn(mask, charIdx + 1))))
                        ↪ ;
6   }
```

Kode Sumber 4.14: Fungsi duplicate_rule1

4.3.15 Implementasi Fungsi Duplicate Rule 2

Fungsi duplicate_rule2 adalah implementasi dari perancangan pada pseudocode 3.12 yang dirancang berdasarkan persamaan 2.9. Implementasi dari fungsi duplicate_rule2 dapat dilihat pada kode sumber 4.15.

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

Kode Sumber 4.15: 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 2.10. Implementasi dari fungsi duplicate_rule1 dapat dilihat pada kode sumber 4.16.

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

Kode Sumber 4.16: Fungsi duplicate_rule3

Halaman ini sengaja dikosongkan

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.

[1]

Halaman ini sengaja dikosongkan