

TUGAS AKHIR - KI141502

OPTIMASI KASISKI EXAMINATION PADA STUDI KASUS SPOJ THE BYTELANDIAN CRYPTOGRAPHER (ACT IV)

FREDDY HERMAWAN YUWONO
NRP 5113100040

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

Dosen Pembimbing II
Wijayanti Nurul Khotimah, S.Kom., M.Sc

Departemen INFORMATIKA
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember
Surabaya, 2018

(Halaman ini sengaja dikosongkan)

TUGAS AKHIR - KI141502

**OPTIMASI KASISKI EXAMINATION PADA STUDI KASUS
SPOJ THE BYTELANDIAN CRYPTOGRAPHER (ACT IV)**

FREDDY HERMAWAN YUWONO
NRP 5113100040

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

Dosen Pembimbing II
Wijayanti Nurul Khotimah, S.Kom., M.Sc

Departemen INFORMATIKA
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember
Surabaya, 2018

(Halaman ini sengaja dikosongkan)

UNDERGRADUATE THESIS - KI141502

**OPTIMIZATION KASISKI EXAMINATION ON STUDY CASE
SPOJ THE BYTELANDIAN CRYPTOGRAPHER (ACT IV)**

FREDDY HERMAWAN YUWONO
NRP 5113100040

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

Supervisor II
Wijayanti Nurul Khotimah, S.Kom., M.Sc

Department of INFORMATICS
Faculty of Information Technology
Institut Teknologi Sepuluh Nopember
Surabaya, 2018

(Halaman ini sengaja dikosongkan)

LEMBAR PENGESAHAN
OPTIMASI KASISKI EXAMINATION PADA STUDI
KASUS SPOJ THE BYTELANDIAN CRYPTOGRAPHER
(ACT IV)

TUGAS AKHIR

Diajukan Guna Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Bidang Studi Algoritma Pemrograman
Program Studi S1 Departement Informatika
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember

Oleh:

FREDDY HERMAWAN YUWONO
NRP: 5113100040

Disetujui oleh Dosen Pembimbing Tugas Akhir :

| | |
|-------------------------------|----------------|
| Rully Soelaiman, S.Kom, M.Kom | |
| NIP: 197002131994021001 | (Pembimbing 1) |

| | |
|--|----------------|
| Wijayanti Nurul Khotimah, S.Kom., M.Sc | |
| NIP: 198603122012122004 | (Pembimbing 2) |

SURABAYA
Januari 2018

(Halaman ini sengaja dikosongkan)

OPTIMASI KASISKI EXAMINATION PADA STUDI KASUS SPOJ THE BYTELANDIAN CRYPTOGRAPHER (ACT IV)

Nama : FREDDY HERMAWAN YUWONO
NRP : 5113100040
Departemen : Informatika FTIK
Pembimbing I : Rully Soelaiman, S.Kom, M.Kom
Pembimbing II : Wijayanti Nurul Khotimah, S.Kom., M.Sc

Abstrak

Pada Era Digitalisasi ini, tingkat kebutuhan masyarakat akan informasi semakin meningkat. Hal ini menyebabkan pertukaran informasi menjadi sangat mudah. Hal ini membuat informasi yang bersifat sensitif dapat terjadi kebocoran informasi kepada pihak - pihak yang tidak berkepentingan. Kebocoran informasi terbagi menjadi 2 apabila dilihat dari keutuhan informasi yang didapat, yaitu sebagian dan seutuhnya. Kebocoran informasi yang bersifat sebagian, membuat pihak-pihak yang tidak berkepentingan tetapi yang meminginkan informasi tersebut, berusaha untuk mendapatkan informasi yang utuh dari potongan-potongan informasi yang telah didapatkan.

Permasalahan dalam buku tugas akhir ini adalah permasalahan untuk mendapatkan plaintext sebanyak-banyaknya dari ciphertext dan batas atas panjang kunci pada metode enkripsi yang menggunakan teknik Vigenere Cipher. Dalam permasalahan ini diberikan plaintext dan ciphertext, akan tetapi terdapat informasi yang hilang pada keduanya. Diberikan batas atas panjang kunci, dimana batas atas ini belum tentu panjang kunci yang sesungguhnya. Untuk dapat merekonstruksi plaintext dari kepingan informasi yang

didapatkan diperlukan untuk mencari panjang kunci yang di dapatkan dengan cara memodifikasi Kasiski Examination dan Intersection. Beberapa hal yang perlu diperhatikan seperti mempercepat dari Kasiski Examination dan juga Intersection terhadap hasil yang diperoleh dari pencarian panjang kunci.

Hasil dari tugas akhir ini telah berhasil untuk menyelesaikan permasalahan yang telah diangkat dengan benar. Waktu yang diperlukan untuk dapat menyelesaikan masukan sebesar 2MB dalam 4,42 detik dengan alokasi memori sebesar 26,5MB.

Kata-Kunci: *Plaintext, Ciphertext, Kasiski Examination, Optimisasi.*

OPTIMIZATION KASISKI EXAMINATION ON STUDY CASE SPOJ THE BYTELANDIAN CRYPTOGRAPHER (ACT IV)

Name : FREDDY HERMAWAN YUWONO
NRP : 5113100040
Major : Informatics FTIK
Supervisor I : Rully Soelaiman, S.Kom, M.Kom
**Supervisor II : Wijayanti Nurul Khotimah, S.Kom.,
M.Sc**

Abstract

In this Digitalization era, the level of community needs for information is increasing. This make exchanging the information very easy. This makes the sensitive information leakage to the unauthorized party. The leakage of information divided into 2 when viewed from the integrity of information they get. First they get all information or second they only get a partial of information. Partial information leakage, make unauthorized party to reconstruct the information they get from all piece information they have already obtain.

The problem in this undergraduate thesis book is the problem to get plaintext as much as possible from the ciphertext and upper bound of key length. The encryption method is using the Vigenere Cipher technique. Given the plaintext and ciphertext, but there is missing information in both of them. Given the upper bound of key length, that is not real key length. To reconstruct plaintext from piece information, must find the length of key from modify Kasiski Examination and Intersection. Some things to note that using only modify Kasiski Examination and Intersection can not solving the problem. Required to optimize both of them to get the answer and right time.

The results show that this problem is successfully solved. Averaging time of 4.42 second and averaging memory use about 26.5MB to solve 2MB data.

Keywords: *Plaintext, Ciphertext, Kasiski Examination, Optimization.*

KATA PENGANTAR

Puji Syukur kepada Tuhan yang Maha Esa, atas berkatNya penulis dapat menyelesaikan buku berjudul **Optimasi Kasiski Examination pada Studi Kasus SPOJ The Bytelandian Cryptographer (Act IV)**.

Selain itu, pada kesempatan ini penulis menghaturkan terima kasih sebesar-besarnya kepada pihak-pihak yang tanpa mereka, penulis tidak akan dapat menyelesaikan buku ini:

1. **Tuhan Yesus Kristus** atas segala berkat, limpahan karunia, kesempatan dan rancangan jalanNya-lah penulis masih diberi nafas kehidupan, waktu, tenaga dan pikiran untuk menyelesaikan buku ini.
2. **Alm. Papa** yang selalu menguatkan, menasehati, dan luar biasa sabar dalam mengingatkan penulis agar tidak lupa menjaga kesehatan dan selalu bersyukur selama masa studi.
3. **Mama dan saudara** yang selalu memberikan saran, dukungan, doa dan tidak lupa untuk selalu bersyukur selama masa studi.
4. **Yth Bapak Rully Soelaiman** sebagai dosen pembimbing I yang telah banyak memberikan ilmu, bimbingan, nasihat, motivasi, serta waktu diskusi sehingga penulis dapat menyelesaikan tugas akhir ini; dan
Yth Ibu Wijayanti Nurul Khotimah sebagai dosen pembimbing II yang memberi bimbingan, saran teknis dan administratif, diskusi dan pemecahan masalah dalam pembuatan dan penulisan buku tugas akhir.
5. **Teman-teman Sarjana Komedi** yang telah mengingatkan, memberikan semangat dan inspirasi untuk terus melanjutkan tugas akhir di saat penulis kehilangan semangat.
6. **Teman-teman S1 Teknik Informatika 2013** yang membantu, menyemangati dan bertukar pikiran dengan penulis selama pengerjaan tugas akhir ini.
7. **Teman-teman S1 Teknik Informatika bukan 2013**, yang

telah banyak membantu, menyemangati dan bertukar pikiran dengan penulis selama pengerjaan tugas akhir ini, terutama pada Steven, Theo, Daniel, dan Glenn.

8. Serta semua pihak yang tidak tertulis, baik yang membantu dalam proses pengujian, membantu memikirkan saat ada masalah, dan lainnya yang telah turut membantu penulis dalam menyelesaikan Tugas Akhir ini.

Penulis menyadari bahwa Tugas Akhir ini masih memiliki banyak kekurangan. Oleh karena itu, penulis berharap kritik dan saran dari pembaca sekalian untuk memperbaiki buku ini ke depannya. Semoga tugas akhir ini dapat memberikan manfaat yang sebaik-baiknya.

Surabaya, Januari 2018

Freddy Hermawan Yuwono

DAFTAR ISI

| | |
|---|-------------|
| ABSTRAK | iii |
| ABSTRACT | v |
| KATA PENGANTAR | vii |
| DAFTAR ISI | ix |
| DAFTAR TABEL | xi |
| DAFTAR GAMBAR | xiii |
| DAFTAR KODE SUMBER | xv |
| BAB I PENDAHULUAN | 1 |
| 1.1 Latar Belakang | 1 |
| 1.2 Rumusan Masalah | 2 |
| 1.3 Batasan Masalah..... | 2 |
| 1.4 Tujuan | 3 |
| 1.5 Metodologi..... | 3 |
| 1.6 Sistematika Penulisan | 4 |
| BAB II LANDASAN TEORI | 7 |
| 2.1 Definisi Umum..... | 7 |
| 2.1.1 Polyalphabetic Cipher | 7 |
| 2.1.2 Ciphertext..... | 8 |
| 2.1.3 Plaintext | 8 |
| 2.1.4 Secret Key | 8 |
| 2.1.5 Kasiski Examination | 9 |
| 2.1.6 Intersection | 10 |
| 2.2 Deskripsi Permasalahan | 10 |
| 2.3 Contoh Kasus Permasalahan | 15 |
| 2.4 Penyelesaian Masalah The Bytelandian Cryptographer (Act IV)..... | 23 |
| BAB III DESAIN | 27 |
| 3.1 Desain Umum Sistem | 27 |
| 3.2 Desain Algoritma | 27 |
| 3.2.1 Desain fungsi SOLVE | 28 |
| 3.2.2 Desain Fungsi VALIDITY | 30 |

| | |
|--|-----------|
| BAB IV IMPLEMENTASI | 33 |
| 4.1 Lingkungan Implementasi | 33 |
| 4.2 Rancangan Data | 33 |
| 4.2.1 Data Masukan | 33 |
| 4.2.2 Data Keluaran | 34 |
| 4.3 Implementasi Algoritma | 34 |
| 4.3.1 <i>Header</i> yang Diperlukan..... | 34 |
| 4.3.2 <i>Preprocessor Directives</i> | 35 |
| 4.3.3 Variabel Global | 36 |
| 4.3.4 Implementasi Fungsi Main..... | 36 |
| 4.3.5 Implementasi Fungsi SOLVE..... | 37 |
| 4.3.6 Implementasi Fungsi VALIDITY..... | 39 |
| BAB V UJI COBA DAN EVALUASI | 41 |
| 5.1 Lingkungan Uji Coba | 41 |
| 5.2 Uji Coba Kebenaran | 41 |
| 5.3 Analisa Kompleksitas Waktu | 43 |
| BAB VI PENUTUP | 47 |
| 6.1 Kesimpulan..... | 47 |
| 6.2 Saran..... | 47 |
| DAFTAR PUSTAKA..... | 49 |
| BAB A Lampiran | 51 |
| BAB B Hasil Percobaan dengan menggunakan Algoritma Naive dan Kasiski Examination..... | 57 |
| BIODATA PENULIS..... | 67 |

DAFTAR TABEL

| | |
|--|--------|
| Tabel 2.1 Contoh <i>Kasiski Examintaion</i> | 9 |
| Tabel 2.2 Contoh 1..... | 16 |
| Tabel 2.3 Langkah 1 Contoh 1..... | 16 |
| Tabel 2.4 Contoh 2..... | 17 |
| Tabel 2.5 Panjang Kunci 1 Contoh 2..... | 17 |
| Tabel 2.6 Panjang Kunci 2 Contoh 2..... | 18 |
| Tabel 2.7 Panjang Kunci 3 Contoh 2..... | 18 |
| Tabel 2.8 Panjang Kunci 4 Contoh 2..... | 19 |
| Tabel 2.9 Hasil Contoh 2 | 20 |
| Tabel 2.10Contoh 3..... | 20 |
| Tabel 2.11Panjang Kunci 1 Contoh 3..... | 20 |
| Tabel 2.12Panjang Kunci 2 Contoh 3..... | 21 |
| Tabel 2.13Panjang Kunci 3 Contoh 3..... | 22 |
| Tabel 2.14Panjang Kunci 4 Contoh 3..... | 22 |
| Tabel 2.15Hasil dari Contoh 3..... | 23 |
| Tabel 5.1 Kecepatan Maksimal, Minimal, dan Rata-Rata dari Hasil Uji Coba Pengumpulan 30 Kali pada Situs Pengujian Daring Spoj | 42 |
| Tabel 2.1 Hasil Percobaan Penyelesaian Studi Kasus SPOJ The Bytelandian Cryptographer(Act IV) dengan menggunakan algoritma <i>Optimasi Kasiski Examination</i> dan <i>Intersection</i> (1) | 57 |
| Tabel 2.2 Hasil Percobaan Penyelesaian Studi Kasus SPOJ The Bytelandian Cryptographer(Act IV) dengan menggunakan algoritma <i>Optimasi Kasiski Examination</i> dan <i>Intersection</i> (2) | 58 |

| | |
|--|----|
| Tabel 2.3 Hasil Percobaan Penyelesaian Studi Kasus SPOJ The Bytelandian Cryptographer(Act IV) dengan menggunakan algoritma <i>Optimasi Kasiski Examination</i> dan <i>Intersection</i> (3) | 59 |
| Tabel 2.4 Hasil Percobaan Penyelesaian Studi Kasus SPOJ The Bytelandian Cryptographer(Act IV) dengan menggunakan algoritma <i>Optimasi Kasiski Examination</i> dan <i>Intersection</i> (4) | 60 |
| Tabel 2.5 Hasil Percobaan Penyelesaian Studi Kasus SPOJ The Bytelandian Cryptographer(Act IV) dengan menggunakan algoritma <i>Optimasi Kasiski Examination</i> dan <i>Intersection</i> (5) | 61 |
| Tabel 2.6 Hasil Percobaan Penyelesaian Studi Kasus SPOJ The Bytelandian Cryptographer(Act IV) dengan menggunakan algoritma <i>Naive</i> (1) . | 62 |
| Tabel 2.7 Hasil Percobaan Penyelesaian Studi Kasus SPOJ The Bytelandian Cryptographer(Act IV) dengan menggunakan algoritma <i>Naive</i> (2) . | 63 |
| Tabel 2.8 Hasil Percobaan Penyelesaian Studi Kasus SPOJ The Bytelandian Cryptographer(Act IV) dengan menggunakan algoritma <i>Naive</i> (3) . | 64 |
| Tabel 2.9 Hasil Percobaan Penyelesaian Studi Kasus SPOJ The Bytelandian Cryptographer(Act IV) dengan menggunakan algoritma <i>Naive</i> (4) . | 65 |
| Tabel 2.10 Hasil Percobaan Penyelesaian Studi Kasus SPOJ The Bytelandian Cryptographer(Act IV) dengan menggunakan algoritma <i>Naive</i> (5) . | 66 |

DAFTAR GAMBAR

| | |
|--|----|
| Gambar 2.1 Aturan <i>Polyalphabetical Cipher</i> | 8 |
| Gambar 2.2 Deskripsi Permasalahan dalam Bahasa Inggris pada SPOJ <i>The Bytelandian Cryptographer (Act IV)</i> | 12 |
| Gambar 2.3 Deskripsi Format Masukan dan Keluaran pada SPOJ <i>The Bytelandian Cryptographer (Act IV)</i> | 15 |
| Gambar 3.1 Gamba Fungsi Main | 27 |
| Gambar 3.2 Gambar Fungsi SOLVE (1) | 29 |
| Gambar 3.3 Gambar Fungsi SOLVE (2) | 30 |
| Gambar 3.4 Gambar Fungsi VALIDITY..... | 31 |
| Gambar 5.1 Perbandingan Kinerja Algoritma Optimasi <i>Kasiski Examination</i> dan <i>Naive</i> | 45 |
| Gambar A.1 Hasil Uji Coba pada Situs Penilaian SPOJ.. | 51 |
| Gambar A.2 Grafik Hasil Uji Coba pada Situs SPOJ Sebanyak 30 Kali | 51 |
| Gambar A.3 Hasil Pengujian Sebanyak 30 Kali pada Situs Penilaian Daring SPOJ (1)..... | 52 |
| Gambar A.4 Hasil Pengujian Sebanyak 30 Kali pada Situs Penilaian Daring SPOJ (2)..... | 53 |
| Gambar A.5 Daftar Peringkat Berdasarkan Kecepatan yang Diperoleh dari Dari SPOJ(1) | 54 |
| Gambar A.6 Daftar Hasil Peringkat Berdasarkan Kecepatan yang Diperoleh dari Dari SPOJ(2) | 55 |

(Halaman ini sengaja dikosongkan)

DAFTAR KODE SUMBER

| | | |
|------|---|----|
| IV.1 | <i>Header</i> yang diperlukan | 34 |
| IV.2 | Preprocessor Directives | 35 |
| IV.3 | Variabel Global | 36 |
| IV.4 | Fungsi main | 36 |
| IV.5 | Fungsi SOLVE | 38 |
| IV.6 | Fungsi VALIDITY | 39 |

(Halaman ini sengaja dikosongkan)

BAB I

PENDAHULUAN

Pada bab ini akan dipaparkan mengenai garis besar Tugas Akhir yang meliputi latar belakang, tujuan, rumusan dan batasan permasalahan, metodologi pembuatan Tugas Akhir, dan sistematika penulisan.

1.1 Latar Belakang

Ketergantungan seseorang terhadap informasi tidak terlepas dari kebutuhan manusia akan informasi yang berada disekitarnya. Informasi yang diterima seseorang pada masa sekarang dapat melalui media fisik dan media digital. Media fisik seperti koran dan majalah, sedangkan media digital seperti facebook dan twitter. Media-media tersebut sanggup untuk menyebarkan informasi sangat cepat, sehingga orang-orang dengan cepat mengetahui informasi yang berada disekitarnya.

Pada zaman modern ini suatu informasi, terutama yang bersifat rahasia menjadi semakin rentan akan penyalahgunaan informasi tersebut. Oleh Karena itu, informasi ini disimpan akan disimpan pada tempat-tempat yang aman dan penulisan dari informasi ini pada umumnya menggunakan sandi yang hanya dimengerti oleh orang-orang yang berkepentingan terhadap informasi tersebut.

Informasi digital yang beredar di dunia maya pun tidak lepas dari penyalahgunaan informasi. Dibutuhkan suatu teknik penyandian terhadap data yang dimiliki agar data yang bersifat rahasia itu tidak diketahui dengan orang – orang yang tidak berkepentingan. Teknik penyandian terhadap data digital dapat dibagi menjadi 2 jika melihat dari teknik penyandiannya yaitu *symmetric cipher* dan *asymmetric cipher*. Teknik *symmetric cipher* dapat dibagi menjadi menjadi 4 bagian jika dilihat dari penyubtitusiannya yaitu *Caesar cipher*, *monoalphabetic cipher*, *polyalphabetic cipher*, *one time pad*. Pada dasarnya

pendeskripsian dari data yang terenkripsi dengan penyediaan *symmetric cipher* dengan cara mengetahui kuncinya dan tipe dari penyubstitusiannya.

Dalam Tugas Akhir ini penulis akan mencoba mendeskripsikan informasi terbut dengan menggunakan metode *symmetric cipher* dan teknik substitusinya menggunakan *polyalphabetic cipher*. Salah satunya dengan menggunakan modifikasi *Kasiski Examination*, akan tetapi dalam permasalahan ini apabila hanya menggunakan *Kasiski Examination* waktu yang dibutuhkan sangatlah besar, oleh karena itu penulis mengoptimasi metode yang telah ada.

1.2 Rumusan Masalah

Rumusan masalah yang diangkat dalam tugas akhir ini adalah sebagai berikut:

1. Bagaimana penerapan Optimasi *Kasiski Examination* untuk menyelesaikan studi kasus SPOJ *The Bytelandian Cryptographer(Act IV)*
2. Bagaimana hasil dari kinerja Optimasi *Kasiski Examination* yang digunakan untuk menyelesaikan studi kasus SPOJ *The Bytelandian Cryptographer(Act IV)*

1.3 Batasan Masalah

Dari permasalahan yang telah diuraikan di atas, terdapat beberapa batasan masalah pada tugas akhir ini, yaitu:

1. Bahasa pemrograman yang akan digunakan adalah bahasa pemrograman C/C++.
2. Batasan maksimum panjang dari *input file* sebesar 2 MB.
3. Batasan maksimum panjang dari batas atas *key* sebesar 100,000 karakter.

4. *Dataset* yang digunakan adalah *dataset* pada problem SPOJ *The Bytelandian Cryptographer (Act IV)*.

1.4 Tujuan

Tujuan dari pengerjaan Tugas Akhir ini adalah:

1. Menerapkan Optimasi *Kasiski Examination* untuk menyelesaikan studi kasus SPOJ *The Bytelandian Cryptographer (Act IV)*.
2. Mengevaluasi hasil dan kinerja Optimasi *Kasiski Examination* dalam menyelesaikan studi kasus SPOJ *The Bytelandian Cryptographer(Act IV)*

1.5 Metodologi

Langkah-langkah yang ditempuh dalam pengerjaan Tugas Akhir ini yaitu:

1. Penyusunan proposal Tugas Akhir

Pada tahap ini dilakukan penyusunan proposal Tugas Akhir yang berisi permasalahan dan gagasan solusi yang akan diteliti pada SPOJ *The Bytelandian Cryptographer (Act IV)*.

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 SPOJ *The*

Bytelandian Cryptographer (Act IV)

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. Pengujian kebenaran dilakukan pada sistem penilaian daring SPOJ sesuai dengan masalah yang dikerjakan untuk diuji apakah luaran dari program telah sesuai.

6. Penyusunan buku Tugas Akhir Pada tahap ini dilakukan penyusunan buku Tugas Akhir yang berisi dokumentasi hasil pengerjaan Tugas Akhir.

1.6 Sistematika Penulisan

Buku Tugas Akhir ini bertujuan untuk mendapatkan gambaran dari pengerjaan Tugas Akhir ini. Secara garis besar, buku Tugas Akhir terdiri atas beberapa bagian seperti berikut ini:

Bab I Pendahuluan

Bab ini berisi latar belakang masalah, tujuan dan manfaat pembuatan Tugas Akhir, permasalahan, batasan masalah, metodologi yang digunakan, dan sistematika penyusunan Tugas Akhir.

Bab II Dasar Teori

Bab ini berisi dasar teori mengenai permasalahan dan garis besar penyelesaian yang digunakan dalam Tugas Akhir dan deskripsi permasalahan yang digunakan dalam Tugas Akhir.

Bab III Desain

Bab ini berisi desain algoritma yang digunakan dalam penyelesaian permasalahan.

Bab IV Implementasi

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

Bab V Pengujian dan Evaluasi

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

Bab VI Kesimpulan dan Saran

Bab ini berisi kesimpulan dari hasil pengujian yang dilakukan, dan membahas saran untuk pengembangan algoritma lebih lanjut.

Daftar Pustaka

Merupakan daftar referensi yang digunakan untuk mengembangkan Tugas Akhir.

Lampiran

Merupakan bab tambahan yang berisi hal-hal terkait yang penting dalam aplikasi ini.

(Halaman ini sengaja dikosongkan)

BAB II

LANDASAN TEORI

Bab ini akan membahas mengenai dasar teori dan literatur yang menjadi dasar pengerjaan tugas akhir ini. Pada subbab 2.1 membahas mengenai definisi umum yang digunakan dalam memecahkan permasalahan ini. Pada subbab 2.2 membahas mengenai deskripsi permasalahan. Pada subbab 2.3 membahas mengenai contoh permasalahan. Pada subbab 2.4 membahas mengenai penyelesaian masalah secara lengkap.

2.1 Definisi Umum

Pada subbab ini membahas definisi-definisi yang digunakan sebagai dasar untuk memahami permasalahan ini dan pemecahannya.

2.1.1 Polyalphabetic Cipher

Polyalphabetic Cipher merupakan salah satu teknik untuk menenkripsi dengan menggunakan substitusi huruf untuk menyubstitusikannya. Secara garis besar yang dimaksud dengan *polyalphabetic cipher* memiliki 2 aturan dasar yang harus dipenuhi yaitu :

1. Memiliki satu set aturan substitusi *monoalphabetic cipher* yang digunakan.
2. Sebuah kunci mengatur suatu aturan tertentu yang dipilih untuk mengatur transformasi yang dilakukan.

Untuk memperjelas aturan diatas, dapat dilihat pada gambar 2.1.

$$\begin{aligned}
C &= c_0, c_1, c_2, \dots, c_{n-1} \\
E(K, P) &= E[(k_0, k_1, \dots, k_{m-1})(p_0, p_1, \dots, p_{n-1})] \\
&= (p_0 + k_0) \bmod 26, (p_1 + k_1) \bmod 26, \dots, (p_{m-1} + k_{m-1}) \bmod 26, \\
&\quad (p_m + k_0) \bmod 26, (p_{m+1} + k_1) \bmod 26, \dots, (p_{2m-1} + k_{m-1}) \bmod 26, \dots
\end{aligned}$$

Gambar 2.1 Aturan *Polyalphabetical Cipher*

Salah satu turunan dari *polyalphabetic cipher* adalah teknik *Vigenere Cipher* yang menjadi dasar permasalahan yang diangkat dalam tugas akhir ini[1].

2.1.2 Ciphertext

Ciphertext adalah suatu pesan / teks acak yang dihasilkan dari suatu algoritma kriptografi. Contoh dari *Ciphertext* dalam kasus *polyalphabetical cipher* adalah "RTPPRKGFI" yang merupakan hasil enkripsi dari "PLAINTEXT" dan menggunakan kunci "CIPHER"[2].

2.1.3 Plaintext

Plaintext adalah data original sebagai inputan dari suatu metode enkripsi yang akan dilakukan[2]. Biasanya merupakan suatu rangkaian kata yang masih dapat dipahami artinya atau hasil keluaran dari suatu algoritma kriptografi yang akan dienkripsi lagi.

2.1.4 Secret Key

Secret Key atau yang lebih dikenal dengan *key* adalah suatu inputan dari algoritma enkripsi yang akan menentukan suatu transformasi dan substitusi yang akan dilakukan oleh algoritma

enskripsi[2]. Dalam kasus *polyalphabetical cipher* pada permasalahan yang diangkat dalam tugas akhir ini, panjang kunci yang digunakan setidaknya 1.

2.1.5 Kasiski Examination

Kasiski Examination merupakan suatu teknik yang digunakan untuk mendeskripsikan secara paksa suatu *ciphertext* yang menggunakan teknik substitusi, baik itu *polyalphabetical cipher* maupun *monoalphabetical cipher*. Teknik menggunakan kelemahan yang ditimbulkan oleh teknik substitusi itu sendiri, yaitu apabila suatu *subtring* dari *plaintext* dan *subtring* dari suatu set kunci yang berulang terdapat yang berulang, maka dapat dipastikan untuk menebak panjang huruf / karakter kunci yang digunakan, tanpa diketahui *plaintext* dan kuncinya. Sebagai contoh dapat dilihat pada table 2.1.

| | | | | | | | | | | | | | | | |
|-------------------|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| <i>ciphertext</i> | C | S | A | S | X | S | J | O | S | P | C | S | A | S | X |
|-------------------|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

Tabel 2.1 Contoh *Kasiski Examintaion*

Dari table 2.1 yang ada dapat disimpulkan bahwa kata "CSASX" berulang. Sehingga setidaknya dapat disimpulkan bahwa panjang kuncinya mungkin 5 karakter atau faktor dari 10[8]. Cara pencarian panjangnya:

1. Mencari semua subtring yang berulang. Seperti tabel 2.1.
 2. Mencari semua faktor kapan subtring itu berulang lagi sebagai contoh table 2.1 itu selisih antara sub kalimat "CSASX" adalah 10. Maka, faktor dari 10 adalah 10,5,2,1.
 3. Faktor yang sering berulang biasanya adalah jawabannya.
- Hal ini yang menjadi dasar pengerjaan permasalahan yang diangkat dalam tugas akhir ini.

2.1.6 Intersection

Intersection adalah himpunan A dan Himpunan B dimana ada bagian dari A juga merupakan bagian dari B. Sehingga dapat ditulis dengan persamaan II.1.

$$A \cap B = \{x : x \in A \text{ dan } x \in B\} \quad (\text{II.1})$$

Sebagai contoh *intersection* antara $\{1, 2, 3\}$ dan $\{1, 4, 5\}$ adalah $\{1\}$ [5].

2.2 Deskripsi Permasalahan

Permasalahan yang diangkat dalam tugas akhir ini diangkat dari suatu permasalahan yang terdapat pada suatu situs penilaian daring atau *online judge* SPOJ yaitu *The Bytelandian Cryptographer (Act IV)* dengan nomer soal 20 dengan kode soal CRYPTO4. Deskripsi soal yang asli menggunakan bahasa Inggris dapat dilihat pada 2.2[3].

Permasalahan pada *The Bytelandian Cryptographer (Act IV)* diberikana pesan dengan panjang N huruf, huruf yang digunakan adalah huruf kapital latin dari A sampai dengan Z, yang dapat ditafsirkan menjadi bilangan bulat dari 0 sampai dengan 25. Diberikan kunci untuk mentransmisikan pesan yang diketahui oleh kedua belah pihak yang terdiri dari M bilangan bulat. Dengan menggunakan kunci yang ada bahwa pada index ke i dari pesan pada index x_i akan di enkripsikan ke dalam bentuk index ke i dari pesan hasil enkripsi y , yang mengikuti aturan

$$y_i = x_i + k_{1+(i-1) \bmod M} \bmod 26 \quad (\text{II.2})$$

Diketahui *plain text* dan *ciphertext* yang diberikan hanya berupa potongan-potongan dari kedua pesan tersebut. Dicari bagaimana menkonstruksi ulang pesan yang telah didapat sehingga bisa

membentuk *plain text* yang asli dari pesan yang telah didapatkan sebanyak-banyaknya.

CRYPTO4 - The Bytelandian Cryptographer (Act IV)

no tags

The Bytelandian Cryptographer has been requested by the BBFO to put forward an encryption scheme which would allow the BBFO to communicate with its foreign associates. After some intensive studies, he has decided upon the Vigenère cipher. Messages written using 26 upper case characters of the Latin alphabet: A, B, ..., Z which are interpreted as integers 0, 1, ..., 25 respectively. The secret cypher for transmitting a message is known to both sides and consists of n integers k_1, k_2, \dots, k_n . Using this cypher, the i -th number x_i of the input message x is encrypted to the form of the i -th number of the output message y , as follows:

$$y_i = (x_i + k_i + ((i-1) \bmod n)) \bmod 26.$$

You are trying to find out the content of a message transmitted by the BBFO. By a lucky stroke of fortune, your spies managed to intercept the message in both its plaintext and encrypted form (x and y respectively). Unfortunately, during their dramatic escape the files they were carrying were pierced by bullets and fragments of messages x and y were inadvertently lost. Or were they? It is your task to reconstruct as much of message x as you possibly can.

Gambar 2.2 Deskripsi Permasalahan dalam Bahasa Inggris pada
SPOJ *The Bytelandian Cryptographer (Act IV)*

Format masukan pada baris pertama diberikan T ujicoba kasus. Pada baris selanjutnya diberikan M batas atas panjang kunci. Pada baris selanjutnya diberikan *plain text*. Pada baris selanjutnya di berikan *ciphertext*, *plain text* dan *ciphertext* menggunakan karakter A sampai dengan Z yang dapat ditafsirkan kedalam bilangan bulat 0 sampai dengan 25 dan '*' dapat ditafsirkan sebagai karakter yang hilang.

Format keluaran yang dihasilkan adalah 1 baris yang mengandung *plain text* dan '*' apabila nilai dari karakter tersebut tidak dapat ditentukan. Deskripsi mengenai Format masukan dan keluaran beserta dengan contohnya dalam bahasa Inggris dapat lihat pada gambar 2.3

Input

The first line of input contains a single integer $t \leq 200$ denoting the number of test cases. t test case descriptions follow.

For each test case, the first line contains one integer m which is some upper bound on the length of the cypher ($1 \leq n \leq m \leq 100000$). The second line of input contains the original message x , while the third line contains the encrypted message y . The messages are expressed using characters 'A'-'Z' (interpreted as integers 0-25) and '*' (denoting a single character illegible due to damage). The total length of the input file is not more than 2MB.

Output

For each test case output a single line containing the original message x , with asterisks '*' in place of only those characters whose value cannot be determined.

Example

```

Input:
4
1
A*X*C
**CM*
4
*B***A
AAAAAA
6
*B***A
AAAAAA
4
*AA*****
AAAAAAAAAA

Output:
A*XHC
*BA*BA
*B***A
*AA**A****

```

Gambar 2.3 Deskripsi Format Masukan dan Keluaran pada SPOJ *The Bytelandian Cryptographer (Act IV)*

Batasan permasalahan *The Bytelandian Cryptographer (Act IV)* adalah sebagai berikut:

1. $T \leq 200$
2. $1 \leq M \leq 100,000$
3. Panjang *input file* tidak melebihi dari 2MB.
4. Lingkungan penilaian Intel Pentium G860 3GHz.
5. Batas Waktu: ≤ 17 detik
6. Batas Sumber Code : 50000B
7. Batas Memory : 1536 MB.

2.3 Contoh Kasus Permasalahan

Dalam Permasalahan yang diangkat ini huruf A sampai dengan Z ditafsirkan sebagai 0 sampai dengan 25. Contoh 1. Diketahui M bernilai 1 yang menunjukkan batas atas dari

panjang kunci. Diketahui *plaintext* adalah A*X*C dan *ciphertext* adalah **CM*.

| | | | | | |
|------------------------|---|---|---|---|---|
| <i>index</i> | 0 | 1 | 2 | 3 | 4 |
| <i>plain text</i> | A | * | X | * | C |
| <i>ciphertext</i> | * | * | C | M | * |
| Selisih yang diketahui | | | 5 | | |
| Hasil | A | * | X | H | C |

Tabel 2.2 Contoh 1

Dari table 2.2 dapat diketahui bagaimana hasil yang akan dicapai. Cara mencapai hasil yang diinginkan adalah sebagai berikut. Pertama-tama pencarian panjang kunci. Pada pencarian panjang kunci harus mengetahui dahulu batas atas dari panjang kunci tersebut. Pada contoh kasus ini M bernilai 1, maka kemungkinan jawabannya hanya ada 1, yaitu M itu sendiri, karena suatu panjang kunci yang digunakan dalam enkripsi tidak mungkin 0. Diasumsikan bahwa 1 warna mewakili satu blok kunci yang akan digunakan. Maka akan membentuk seperti pada tabel 2.3.

| | | | | | |
|------------------------|---|---|---|---|---|
| <i>index</i> | 0 | 1 | 2 | 3 | 4 |
| <i>plain text</i> | A | * | X | * | C |
| <i>ciphertext</i> | * | * | C | M | * |
| Selisih yang diketahui | | | 5 | | |

Tabel 2.3 Langkah 1 Contoh 1

Karena selisih yang di ketahui hanya ada 1, maka itulah yang menjadi jawabannya. Selanjutnya mencari yang *plaintext* yang bernilai "*" dan *ciphertext* tidak kosong, seperti indeks ketiga. Maka indeks ketiga jawabannya adalah "H". Karena "M"—5 adalah "H".

Contoh 2. Diketahui bahwa M bernilai 4 yang menunjukkan batas atas dari panjang kunci. Diketahui *plaintext* adalah *B***A dan *ciphertext* adalah AAAAAA. Dalam Contoh ini panjang kuncinya bisa dari 1 sampai dengan 4.

| | | | | | | |
|------------------------|---|----|---|---|---|---|
| <i>index</i> | 0 | 1 | 2 | 3 | 4 | 5 |
| <i>plain text</i> | * | B | * | * | * | A |
| <i>ciphertext</i> | A | A | A | A | A | A |
| Selisih yang diketahui | | 25 | | | | 0 |
| Panjang Kunci 1 | tidak bisa karena ada yang <i>collision</i> | | | | | |
| Panjang Kunci 2 | tidak bisa karena ada yang <i>collision</i> | | | | | |
| Panjang Kunci 3 | * | B | A | * | B | A |
| Panjang Kunci 4 | tidak bisa karena ada yang <i>collision</i> | | | | | |

Tabel 2.4 Contoh 2

Cara yang harus dilalui adalah sebagai berikut. Pertama mencari panjang kunci yang bernilai diantara 1 - 4. Pada table 2.5 akan diperlihatkan proses pencarian ketika panjang kunci adalah 1.

| | | | | | | |
|------------------------|---|----|---|---|---|---|
| <i>index</i> | 0 | 1 | 2 | 3 | 4 | 5 |
| <i>plain text</i> | * | B | * | * | * | A |
| <i>ciphertext</i> | A | A | A | A | A | A |
| Selisih yang diketahui | | 25 | | | | 0 |
| Panjang Kunci 1 | tidak bisa karena ada yang <i>collision</i> | | | | | |

Tabel 2.5 Panjang Kunci 1 Contoh 2

Apabila 1 warna direpresentasikan menjadi 1 blok kunci, maka terdapat 5 blok kunci yang terbentuk. Pada bagian tabel 2.5 bahwa hanya ada 2 indeks dimana *plaintext* dan *ciphertext* yang diketahui. Kedua-duanya terletak pada blok kunci yang berbeda. Akan tetapi karena panjang bloknya 1 maka tidak mungkin

panjang kunci 1 dilakukan karena kedua-duanya saling bertabrakan dan membawa nilai yang berbeda. Yang menjadi permasalahan pada panjang kunci 1 adalah ketika blok yang diketahui bertabrakan dan masing masing membawa nilai yang berbeda juga. Sehingga dapat disimpulkan bahwa panjang kunci 1 pada contoh kasus 2 tidak mungkin terjadi.

Proses pencarian ketika panjang kunci adalah 2.

| <i>index</i> | 0 | 1 | 2 | 3 | 4 | 5 |
|------------------------|---|----|---|---|---|---|
| <i>plain text</i> | * | B | * | * | * | A |
| <i>ciphertext</i> | A | A | A | A | A | A |
| Selisih yang diketahui | | 25 | | | | 0 |
| Panjang Kunci 2 | tidak bisa karena ada yang <i>collision</i> | | | | | |

Tabel 2.6 Panjang Kunci 2 Contoh 2

Apabila 1 warna direpresentasikan menjadi 1 blok kunci, maka terdapat 3 blok kunci yang terbentuk. Pada tabel 2.6 terlihat bahwa terdapat 2 indeks dimana *plaintext* dan *ciphertext* yang diketahui. Indeks yang pertama terletak pada blok pertama bagian akhir, begitu juga indeks yang kedua terletak pada blok 3 bagian akhir. Karena kedua-duanya terletak pada bagian akhir dari masing-masing blok maka dapat disimpulkan panjang kunci 2 juga tidak dapat digunakan.

Proses Pencarian ketika panjang kunci adalah 3.

| <i>index</i> | 0 | 1 | 2 | 3 | 4 | 5 |
|------------------------|---|----|---|---|---|---|
| <i>plain text</i> | * | B | * | * | * | A |
| <i>ciphertext</i> | A | A | A | A | A | A |
| Selisih yang diketahui | | 25 | | | | 0 |
| Panjang Kunci 3 | * | B | A | * | B | A |

Tabel 2.7 Panjang Kunci 3 Contoh 2

Apabila 1 warna direpresentasikan menjadi 1 blok kunci, maka terdapat 2 blok kunci yang terbentuk. Pada tabel 2.7 terlihat bahwa terdapat 2 indeks dimana *plaintext* dan *ciphertext* yang diketahui. Indeks yang pertama terletak dibagian tengah pada blok 1 dan indeks yang kedua terletak pada bagian akhir pada blok 2. Maka, *plaintext* yang terbentuk adalah pada bagian akhir dari blok kunci 1 dan bagian tengah dari blok kunci 2. Karena kedua-duanya saling melengkapi antara satu bagian dengan bagian lainnya. Sehingga hasil yang terbentuk pada bagian akhir dari blok kunci 1 adalah "A"—0 adalah "A" dan bagian tengah dari blok kunci 2 adalah "A"—25 adalah "B". Proses pencarian ketika panjang kunci adalah 4

| <i>index</i> | 0 | 1 | 2 | 3 | 4 | 5 |
|------------------------|---|----|---|---|---|---|
| <i>plain text</i> | * | B | * | * | * | A |
| <i>ciphertext</i> | A | A | A | A | A | A |
| Selisih yang diketahui | | 25 | | | | 0 |
| Panjang Kunci 4 | tidak bisa karena ada yang <i>collision</i> | | | | | |

Tabel 2.8 Panjang Kunci 4 Contoh 2

Apabila 1 warna direpresentasikan menjadi 1 blok kunci, maka terdapat 2 blok kunci yang terbentuk. Pada tabel 2.8 terlihat bahwa terdapat 2 indeks dimana *plaintext* dan *ciphertext* yang diketahui. Indeks yang pertama terletak dibagian kedua dari depan pada blok 1 dan indeks yang kedua terletak pada bagian kedua dari depan pada blok 2. Maka dalam ini tidak bisa digunakan panjang kunci 4, karena terdapat bagian dari kedua blok pada panjang kunci 4 yang bertabrakan dan membawa nilai yang berbeda. Sehingga hasil yang terbentuk dapat dilihat pada tabel 2.9

| | | | | | | |
|------------------------|---|----|---|---|---|---|
| <i>index</i> | 0 | 1 | 2 | 3 | 4 | 5 |
| <i>plain text</i> | * | B | * | * | * | A |
| <i>ciphertext</i> | A | A | A | A | A | A |
| Selisih yang diketahui | | 25 | | | | 0 |
| Hasil | * | B | A | * | B | A |

Tabel 2.9 Hasil Contoh 2

Contoh 3. Diketahui bahwa M bernilai 4 yang menunjukkan batas atas dari panjang kunci. Diketahui *plaintext* adalah *AA***** dan *ciphertext* adalah AAAAAAAAAA. Indeks akan dihitung mulai dari 0.

| | | | | | | | | | | |
|------------------------|---|---|---|---|---|---|---|---|---|---|
| <i>index</i> | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| <i>plain text</i> | * | A | A | * | * | * | * | * | * | * |
| <i>ciphertext</i> | A | A | A | A | A | A | A | A | A | A |
| Selisih yang diketahui | | 0 | 0 | | | | | | | |

Tabel 2.10 Contoh 3

Cara yang harus dilalui adalah sebagai berikut. Pertama mencari panjang kunci yang bernilai diantara 1 - 4. Pada table 2.11 akan diperlihatkan proses pencarian ketika panjang kunci adalah 1.

| | | | | | | | | | | |
|------------------------|---|---|---|---|---|---|---|---|---|---|
| <i>index</i> | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| <i>plain text</i> | * | A | A | * | * | * | * | * | * | * |
| <i>ciphertext</i> | A | A | A | A | A | A | A | A | A | A |
| Selisih yang diketahui | | 0 | 0 | | | | | | | |
| Panjang kunci 1 | A | A | A | A | A | A | A | A | A | A |

Tabel 2.11 Panjang Kunci 1 Contoh 3

Apabila 1 warna merepresentasikan sebagai 1 blok kunci

maka, dapat terbentuk sebanyak 10 blok kunci. Dapat dilihat pada table 2.11. Indek yang diketahui baik *plaintext* dan *ciphertext* adalah indek 1 dan indek 2, yang mana keduanya terletak pada blok yang berbeda. Akan tetapi, karena mereka membawa 1 nilai yang sama maka semua plaintext yang kosong pasti bernilai "A".

Pada table 2.12 akan diperlihatkan proses pencarian ketika panjang kunci adalah 2.

| <i>index</i> | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|------------------------|---|---|---|---|---|---|---|---|---|---|
| <i>plain text</i> | * | A | A | * | * | * | * | * | * | * |
| <i>ciphertext</i> | A | A | A | A | A | A | A | A | A | A |
| Selisih yang diketahui | | 0 | 0 | | | | | | | |
| Panjang kunci 2 | A | A | A | A | A | A | A | A | A | A |

Tabel 2.12 Panjang Kunci 2 Contoh 3

Apabila 1 warna merepresentasikan sebagai 1 blok kunci maka, dapat terbentuk sebanyak 5 blok kunci. Dapat dilihat pada tabel 2.12. Indek yang diketahui baik *plaintext* dan *ciphertext* adalah indek 1 dan indek 2, yang mana keduanya terletak pada blok yang berbeda dan posisi yang berbeda pula. Hal ini dapat dilihat pada tabel 2.12. Indek yang diketahui pertama terletak pada bagian akhir dari blok 1 dan indek yang diketahui kedua terletak pada bagian awal dari blok 2. Maka dapat disimpulkan bahwa seluruh plaintext yang nilai "*" dan ciphertext yang nilainya tidak "*", nilai dari plaintext yang terbentuk seluruhnya "A", seperti yang terlihat pada tabel 2.12.

Pada table 2.13 akan diperlihatkan proses pencarian ketika panjang kunci adalah 3.

| <i>index</i> | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|------------------------|---|---|---|---|---|---|---|---|---|---|
| <i>plain text</i> | * | A | A | * | * | * | * | * | * | * |
| <i>ciphertext</i> | A | A | A | A | A | A | A | A | A | A |
| Selisih yang diketahui | | 0 | 0 | | | | | | | |
| Panjang kunci 3 | * | A | A | * | A | A | * | A | A | * |

Tabel 2.13 Panjang Kunci 3 Contoh 3

Apabila 1 warna merepresentasikan sebagai 1 blok kunci maka, dapat terbentuk sebanyak 4 blok kunci. Dapat dilihat pada tabel 2.13. Indek yang diketahui baik *plaintext* dan *ciphertext* adalah indek 1 dan indek 2, dimana smuanya terletak pada satu blok yang sama, yaitu blok kunci 1 . Jadi, sisanya menngikuti perulangan dari blok kunci 1.

Pada table 2.14 akan diperlihatkan proses pencarian ketika panjang kunci adalah 4.

| <i>index</i> | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|------------------------|---|---|---|---|---|---|---|---|---|---|
| <i>plain text</i> | * | A | A | * | * | * | * | * | * | * |
| <i>ciphertext</i> | A | A | A | A | A | A | A | A | A | A |
| Selisih yang diketahui | | 0 | 0 | | | | | | | |
| Panjang kunci 4 | * | A | A | * | * | A | A | * | * | A |

Tabel 2.14 Panjang Kunci 4 Contoh 3

Apabila 1 warna merepresentasikan sebagai 1 blok kunci maka, dapat terbentuk sebanyak 4 blok kunci. Dapat dilihat pada tabel 2.14. Indek yang diketahui baik *plaintext* dan *ciphertext* adalah indek 1 dan indek 2, dimana smuanya terletak pada satu blok yang sama, yaitu blok kunci 1 . Jadi, sisanya menngikuti perulangan dari blok kunci 1.

| <i>index</i> | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|------------------------|---|---|---|---|---|---|---|---|---|---|
| <i>plain text</i> awal | * | A | A | * | * | * | * | * | * | * |
| Panjang kunci 1 | A | A | A | A | A | A | A | A | A | A |
| Panjang kunci 2 | A | A | A | A | A | A | A | A | A | A |
| Panjang kunci 3 | * | A | A | * | A | A | * | A | A | * |
| Panjang kunci 4 | * | A | A | * | * | A | A | * | * | A |
| Hasil Akhir | * | A | A | * | * | A | * | * | * | * |

Tabel 2.15 Hasil dari Contoh 3

Pada Contoh ini kalau dilihat pada indeks selisih yang telah diketahui, terjadi bahwa panjang kunci 1 sampai dengan 4 dapat tercipta sedangkan pada contoh sebelumnya tidak bisa. hal ini juga di pengaruhi oleh karena isinya itu sama. Walaupun terjadi *collision* pada indeks yang selisih yang diketahui, tetapi kalau hasilnya sama maka hal itu jawaban untuk setiap panjang kunci bisa terbentuk. Keempat panjang kunci tersebut benar terhadap pesan tersebut. Hasil keluaran yang diinginkan hanya 1 saja tetapi keempat-empatnya benar, oleh karena itu perlu dilakukan *intersection* atau perpotongan dari himpunan keempat kunci tersebut. Perpotongan itu menghasilkan ,A,A,,A,,,,,. Terdapat bagian bagian yang kosong yang dapat diisi dengan *. Sehingga hasil akhirnya dapat dilihat pada tabel 2.15 pada bagian hasil akhir.

2.4 Penyelesaian Masalah The Bytelandian Cryptographer (Act IV)

Permasalahan *The Bytelandian Cryptographer (Act IV)* dapat diselesaikan dengan menggunakan *Kasiski Examination* dan *Intersection*. Untuk menyelesaikan masalah ini perlu ditafsirkan bahwa karakter A sampai dengan Z menjadi 0 sampai dengan 25, karena untuk memudahkan perhitungan mencari selisih dan merekonstruksi *plaintext* dari *ciphertext* dan karakter kunci.

Berikut ini tahapan-tahapan untuk menyelesaikan masalah ini:

1. Menyimpan posisi indeks karakter, dimana pada indeks tersebut baik *ciphertext* maupun *plaintext* tidak bernilai '*', beserta menyimpan hasil perhitungan selisih antara *ciphertext* dan *plaintext* [4].
2. Menyimpan posisi indeks, apabila *ciphertext* diketahui dan *plaintext* tidak diketahui. Untuk mengurangi *running time* dari program[4].
3. Pada Tahapan ini adalah modifikasi dari *Kasiski Examination*, apabila menggunakan *Kasiski Examination* pada umumnya yang hanya mencari posisi berulang dari suatu sub kalimat dalam suatu kalimat tidak bisa digunakan untuk menyelesaikan permasalahan ini. Oleh karena itu dirubah menjadi mengiterasi M dari 1 sampai dengan M , yang akan digunakan untuk membagi selisih yang diketahui antara *plaintext* dan *ciphertext* sebesar posisi iterasi yang telah berjalan. Melihat apakah dalam blok-blok yang telah terbentuk ini terdapat *collision* dan indeks yang saling bertabrakan memiliki value yang sama atau tidak, apabila sama maka tidak terjadi tabrakan sebalik jika terjadi tabrakan maka harus mencari panjang kunci yang baru. Mengiterasi panjang kunci dari $\frac{M}{2} + 1 \leq N \leq M$, alasannya dimulai dari $\frac{M}{2} + 1$ tidak dari 1 karena apabila suatu panjang kunci bernilai benar maka kelipatan dari panjang kunci itu pun juga pasti benar dan untuk mempersingkat waktu *running time* yang seharusnya terjadi. Yang mendasari ini adalah dari tabel 2.4 pada bagian 2.2. Pada bagian ini dilakukan untuk mencari panjang kunci yang benar dengan cara mengiterasi hasil yang diperoleh pada tahap 1 dimodulo dengan posisi iterasi yang dilakukan, apabila tidak terjadi konflik maka panjang kunci tersebut benar jika sebaliknya yang terjadi maka panjang kunci tersebut tidak salah. Contohnya dapat

dilihat pada contoh 2 pada bagian 2.2. Pada bagian ini memungkinkan bahwa bisa jadi lebih dari 1 panjang kunci yang bernilai benar. Contohnya seperti yang terjadi pada table 2.10 pada bagian 2.2.

4. Melakukan *intersection* terhadap himpunan dari kunci yang telah di hasilkan[4]. *intersection* yang dilakukan berada didalam perulangan panjang kunci pada waktu generate setiap karakter yang terdapat dalam penyimpanan tahap 2 pada panjang kunci tersebut dengan ketentuan:
 - (a) Panjang kunci harus benar
 - (b) Apabila terdapat indeks yang tidak dapat dipastikan isinya maka posisinya harus di buang dari penyimpanan dan hasil yang diperoleh pasti '*'.
 - (c) Apabila *plaintext* pada indeks tersebut nilainya masih '*' dan pada penyimpanan masih menyimpan indek tersebut maka, hasilnya adalah *ciphertext* pada indeks tersebut dikurangi dengan hasil yang diperoleh dari tahap 1 bagian perhitungan selisih pada indeks yang sama kemudian dimodulo 26.

Sehingga untuk setiap *textcase* kompleksitasnya

$$\mathcal{O}(T * \frac{M}{2} * (N + S))$$

Dimana $\frac{M}{2}$ adalah batas atas kunci dibagi dengan 2, N adalah jumlah posisi karakter yang terdapat pada tahap 2, dan S adalah jumlah posisi karakter yang terdapat pada tahap 1. Pada kondisi *worst case* $T * (N + S) = 1.000.000$, sedangkan $\frac{M}{2}$ adalah 50.000. Hasilnya 50 miliar perulangan, dengan asumsi 1 detik adalah 1 miliar perulangan maka waktu yang dibutuhkan adalah 50 detik. Oleh karena itu hal ini tidak mungkin bisa dilakukan begitu saja, diperlukan pruning pada sumber kode yang ada untuk memangkas waktu eksekusi program.

(Halaman ini sengaja dikosongkan)

BAB III

DESAIN

Pada bab ini akan dijelaskan mengenai desain algoritma yang digunakan untuk menyelesaikan permasalahan pada Tugas Akhir ini.

3.1 Desain Umum Sistem

Sistem pertama kali akan menjalankan fungsi MAIN terlebih dahulu. Desain dari fungsi main sendiri dapat dilihat pada gambar 3.1. Didalam fungsi main akan di panggil fungsi SOLVE yang digunakan untuk menyelesaikan permasalahan yang diangkat pada tugas akhir ini dan didalam fungsi SOLVE akan terdapat fungsi VALIDITY yang digunakan untuk mengecek kebenaran suatu panjang kunci yang sedang diproses. Secara garis besar fungsi Main.

Algorithm 1 Gambar Fungsi Main

```
1: function MAIN
2:    $T \leftarrow INPUT$ 
3:   while  $T \neq 0$  do
4:      $T = T - 1$ 
5:      $m \leftarrow input$  ▷ masukkan batas atas dari kunci
6:      $message[] \leftarrow input$  ▷ masukkan plaintext
7:      $cipher[] \leftarrow input$  ▷ masukkan ciphertext
8:     SOLVE( $message, cipher, m$ )
9:   end while
10: end function
```

Gambar 3.1 Gamba Fungsi Main

3.2 Desain Algoritma

Pada bagian ini akan dibahas secara rinci mengenai fungsi-fungsi yang digunakan dalam sistem.

3.2.1 Desain fungsi SOLVE

Fungsi ini digunakan untuk menyelesaikan permasalahan yang diangkat pada tugas akhir ini yang didalamnya terdapat tahapan yang telah disebutkan di subbab 2.2 dan subbab 2.4, kecuali untuk mengecek kebenaran dari suatu panjang kunci. Gambar mengenai fungsi SOLVE dapat dilihat pada gambar 3.3. Mengenai modulo 26 yang terdapat pada gambar digunakan untuk memastikan bahwa sesilih dari *plaintext* dan *ciphertext* adalah 0 sampai dengan 25, dan ditambah 26 pada gambar dimaksudkan agar silisih antara *plaintext* dan *ciphertext* selalu bernilai positif.

Algorithm 2 Gambar Fungsi SOLVE

```

1: function SOLVE(message, cipher, m)
2:   counter_diketahui  $\leftarrow$  0
3:   counter_yang_ingin_diketahui  $\leftarrow$  0
4:   diketahui[]
5:   Selisih_diketahui[]
6:   ingin_diketahui[]
7:   Kegj[]
8:   for i = 0 to message[i]  $\neq$  0 ; i + = 1 do
9:     if message[i]  $\neq$  '*' dan cipher[i]  $\neq$  '*' then
10:      diketahui[counter_diketahui] = i
11:      Selisih_diketahui[i] = (message[i] - cipher[i] + 26) % 26
12:      counter_diketahui = counter_diketahui + 1
13:    else if message[i] = '*' dan cipher[i]  $\neq$  '*' then
14:      ingin_diketahui[counter_yang_ingin_diketahui] = i
15:      counter_yang_ingin_diketahui + 1
16:    end if
17:  end for

```

Gambar 3.2 Gambar Fungsi SOLVE (1)

```

18:   $m = \min(m, \text{panjang message})$ 
19:  for  $n = \frac{m}{25} + 1$  to  $n \leq m; n++ = 1$  do
20:    if VALIDITY ( $K_{eg}$ , counter_diketahui, diketahui, Selisih_diketahui,  $n$ ) = True
    then
21:      counter  $\leftarrow 0$ 
22:      while counter  $\neq \text{sizeof}(\text{ingin\_diketahui})$  do
23:        if  $K_{eg}[\text{ingin\_diketahui}[\text{counter}]\%n] = \text{null}$  then
24:          message[ingin_diketahui[counter]] = '*'
25:          remove element index  $i$  in ingin_diketahui
26:        else if message[ingin_diketahui[counter]] = '*' then
27:          message[ingin_diketahui[counter]] = (ciphertext[ingin_diketahui[counter]] -
             $K_{eg}[\text{ingin\_diketahui}[\text{counter}] + 26] \% 26$ 
            counter = counter + 1
28:        else if message[ingin_diketahui[counter]]
29:          (ciphertext[ingin_diketahui[counter]] -  $K_{eg}[\text{ingin\_diketahui}[\text{counter}] + 26] \% 26$ ) then
30:            message[ingin_diketahui[counter]] = '*'
31:            remove element index  $i$  in ingin_diketahui
32:          else
33:            counter = counter + 1
34:          end if
35:        end while
36:      end if
37:    end for
38:  end function

```

Gambar 3.3 Gambar Fungsi SOLVE (2)

3.2.2 Desain Fungsi VALIDITY

Fungsi ini digunakan untuk memvalidasi suatu panjang kunci yang sekarang di cek kebenarannya. Gambar mengenai fungsi VALIDITY dapat dilihat pada gambar 3.4. Penjelasan mengenai

fungsi ini terdapat pada subbab 2.4

Algorithm 3 Gambar Fungsi VALIDITY

```

1: function VALIDITY(Key, counter_diketahui, diketahui, Selisih_diketahui, n)
2:   Initialize(Key, -1)
3:   for i = 0 to i < counter_diketahui; i += 1 do
4:     temp = diketahui[i]
5:     if Key[temp%n] = -1 then
6:       Key[temp%n] = Selisih_diketahui[temp]
7:     else if Key[temp%n] ≠ Selisih_diketahui[temp] then return False
8:     end if
9:   end for
10:  return True
11: end function

```

Gambar 3.4 Gambar Fungsi VALIDITY

(Halaman ini sengaja dikosongkan)

BAB IV

IMPLEMENTASI

Pada bab ini menjelaskan implementasi yang sesuai dengan desain algoritma yang telah ditentukan sebelumnya.

4.1 Lingkungan Implementasi

Lingkungan uji coba yang digunakan adalah sebagai berikut:

1. Perangkat Keras
 - *Processor* Intel(R) Core(TM)i7-5700 @ 2.7GHz.
 - Memori 8 GB
2. Perangkat Lunak
 - Sistem Operasi Windows 10 Home 64 bit
 - *Text editor* Bloodshed Dev-C++ 5.11.
 - *Compiler* g++ (TDM-GCC 4.9.2 32-bit).

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 *The Bytelandian Cryptographer (Act IV)*. 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, baris pertama berupa sebuah bilangan bulat yang merepresentasikan batas atas dari kunci.

baris kedua berupa *string* yang merepresentasikan *plaintext* dan baris ketiga berupa *string* yang merepresentasikan *ciphertext* .

4.2.2 Data Keluaran

Data keluaran yang dihasilkan oleh program hanya berupa satu kalimat yang berisikan *plaintext* yang bisa didapatkan dari *ciphertext* dan batas atas panjang kunci yang telah di berikan.

4.3 Implementasi Algoritma

Pada subbab ini akan dijelaskan tentang implementasi proses algoritma secara keseluruhan berdasarkan desain yang telah dijelaskan pada bab III. Pada bagian ini menggunakan kode untuk mengoptimasi kompiler yang bertujuan untuk mempersingkat waktu eksekusi, seperti "inline" dan "noexcept". Inline berguna untuk membuat baris kode dalam kompiler menjadi berurutan, karena bisa saja baris kode yang terjadi pada kompiler tidak berurutan. Noexcept adalah membuang *exception* apabila terjadinya *exception*.

4.3.1 Header yang Diperlukan

Implementasi algoritma dengan teknik *Kasiski Examination* untuk menyelesaikan *The Bytelandian Cryptographer (Act IV)* untuk membutuhkan 4 *header* yaitu *cstdio*, *cstring*, *algorithm*, dan *unordered_map*. Seperti yang terdapat pada kode sumber

```

1 #include <cstdio>
2 #include <cstring>
3 #include <unordered_map>
4 #include <algorithm>

```

Kode Sumber IV.1 *Header* yang diperlukan

Header cstdio berisi modul untuk menerima masukan dan memberikan keluaran. *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 cstring* berisi modul yang memiliki fungsi-fungsi untuk melakukan pemrosesan *string*. Contoh fungsi yang membantu mengimplementasikan algoritma yang dibangun adalah fungsi *memset*. *Header unordered_map* berisi modul-modul untuk membuat suatu tempat penyimpanan data yang dapat diisi, dihapus untuk setiap elemennya, tetapi hanya dapat menyimpan data dalam bentuk seperti array 1 dimensi, akan tetapi media penyimpanannya seperti memetakan suatu elemen himpunan kedalam elemen lainnya. Pengindeksan yang ada menggunakan *hashing function*.

4.3.2 Preprocessor Directives

Preprocessor directives digunakan untuk memudahkan dalam menyingkat kode-kode yang akan dibuat dan biasanya berupa fungsi ataupun suatu konstanta yang akan digunakan dalam proses perhitungan, yang nantinya akan diterjemahkan terlebih dahulu sebelum mengeksekusi kode. Kode Sumber implementasi konstanta variabel dapat dilihat pada Kode Sumber IV.2.

```

1 #define mp make_pair
2 #define ins insert
3 #define MAX (int)(1e6)+1
4 #define MAXK (int)(1e5)+1

```

Kode Sumber IV.2 Preprocessor Directives

4.3.3 Variabel Global

Variabel global digunakan untuk memudahkan dalam mengakses data yang digunakan lintas fungsi. Kode sumber implementasi variabel global dapat dilihat pada Kode Sumber IV.3.

```

1 char plaintext[MAX], ciphertext[MAX];
2 int key[MAXK], both[MAX], known[MAX], knownall;
3 unordered_map<int, int> tf;

```

Kode Sumber IV.3 Variabel Global

4.3.4 Implementasi Fungsi Main

Fungsi Main adalah implementasi algoritma yang dirancang pada Gambar 3.1. Implementasi fungsi Main dapat dilihat pada Kode Sumber IV.4.

```

1 int main()noexcept{
2     int tc;
3     scanf("%d", &tc);
4     while(tc--){
5         int m;
6         scanf("%d", &m);
7         scanf("%s %s", plaintext, ciphertext);
8         tf.clear();
9         knownall=0;
10        SOLVE(m);
11    }

```

Kode Sumber IV.4 Fungsi main

4.3.5 Implementasi Fungsi SOLVE

Fungsi SOLVE adalah implementasi algoritma yang dirancang pada Gambar 3.3. Implementasi fungsi SOLVE dapat dilihat pada Kode Sumber IV.5.

```

1  inline void SOLVE(int m) noexcept
2  {
3      int ntofind = 0;
4      for(int i=0; plaintext[i]!=0; i++)
5          if(plaintext[i]!='*' && ciphertext[i]!='*'){
6              known[knownall++]=i;
7              both[i]=((ciphertext[i]-plaintext[i]
8                  +26)%26);
9          }
10         else if(plaintext[i]=='*' && ciphertext[i]!='*'){
11             tf.ins(mp(ntofind,i));
12             ntofind++;
13         }
14         unordered_map<int,int>::iterator it;
15         m = min(m, (int)strlen(plaintext));
16         for(int n=m/2+1;n<=m;n++)
17             {
18                 if(VALIDITY(n))
19                     {
20                         it=tf.begin();
21                         while(it!=tf.end())
22                             if(key[(it->second)%n]==-1){
23                                 plaintext[it->second]='*';
24                                 it=tf.erase(it);
25                             }
26                             else if(plaintext[it->second]=='*'){
27                                 plaintext[it->second]=
28                                     (ciphertext[it->second]- 'A'
29                                     -key[(it->second)%n]+26)%26 + 'A';
30                                 it++;
31                             }
32                             else if(plaintext[it->second] !=
33                                 (ciphertext[it->second]- 'A'
34                                 -key[(it->second)%n]+26)%26 + 'A'){
35                                 plaintext[it->second]='*';
36                                 it=tf.erase(it);
37                             }
38                             else it++;
39                     }
40             }
41         printf("%s\n", plaintext);
42     }

```

4.3.6 Implementasi Fungsi VALIDITY

Fungsi VALIDITY adalah implementasi algoritma yang dirancang pada Gambar 3.4. Implementasi fungsi VALIDITY dapat dilihat pada Kode Sumber IV.6.

```

1 inline bool VALIDITY(int n) noexcept
2 {
3     memset(key, -1, sizeof(int)*n);
4     for(int x=0; x<knownall; x++){
5         int temp=known[x];
6         if(key[temp%n]==-1) {
7             key[temp%n]=both[temp];
8         }
9         else if(key[temp%n]!=both[temp])
10             return false;
11     }
12     return true;
13 }

```

Kode Sumber IV.6 Fungsi VALIDITY

(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

Lingkungan 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* CPP14.

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 *The Bytelandian Cryptographer (Act IV)*. Hasil uji coba dengan waktu terbaik pada situs SPOJ ditunjukkan pada Gambar A.1.

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 A.3, A.4 dan A.2.

Dari hasil uji coba pada Gambar A.3, A.4 dan A.2 dapat ditarik beberapa informasi seperti yang tertera pada Tabel 5.1.

Berdasarkan Tabel 5.1, dari percobaan yang dilakukan, didapatkan waktu eksekusi rata-rata 4.418 detik dan waktu maksimal 4,47 detik. Waktu eksekusi tersebut 3,8 kali lebih

| | |
|------------------|-------------|
| Waktu Maksimal | 4, 49 detik |
| Waktu Minimal | 4, 38 detik |
| Waktu Rata-Rata | 4.418 detik |
| Memori Maksimal | 27 MB |
| Memori Minimal | 26 MB |
| Memori Rata-Rata | 26.5 MB |

Tabel 5.1 Kecepatan Maksimal, Minimal, dan Rata-Rata dari Hasil Uji Coba Pengumpulan 30 Kali pada Situs Pengujian Daring Spoj

cepat dari batas waktu eksekusi yang tertera pada deskripsi permasalahan, yaitu 17 detik.

Uji Coba dengan menggunakan contoh kasus ujicoba yang tersedia didalam SPOJ *The Bytelandian Cryptographer (Act IV)*. Sebagai contoh akan digunakan kasus ujicoba yang menggunakan baik mencari panjang kunci maupun *intersection* yang terjadi dalam permasalahan ini.

Sesuai dengan algoritma yang telah dirancang pada pseudocode yang terdapat pada gambar 3.3 maupun pada gambar 3.4. Algoritma ini akan melakukan iterasi yang terdapat pada *plaintext* dan *ciphertext* yang diperoleh dari inputan dan akan menyimpan posisi karakter dengan ketentuan apabila pada indeks tersebut diketahui baik *plaintext* dan *ciphertext* beserta dengan selisih antara *ciphertext* dan *plaintext* . Selanjutnya juga menyimpan posisi karakter yang diperoleh dari *ciphertext* dengan ketentuan apabila *ciphertext* pada indeks tersebut diketahui karakternya dan *plaintext* diindeks tersebut tidak diketahui karakternya. Misalnya diambilkan contoh dari tabel 2.10. Maka yang disimpan untuk bagian yang diketahui keduanya adalah indeks ke 1 dengan selisih 0 dan indeks ke 2 dengan selisih 0, sedangkan untuk yang disimpan pada diketahui *ciphertext* saja maka jawabannya semua indeks kecuali indeks ke 1 dan 2. Selanjutnya, akan membandingkan antara panjang

plaintext atau *ciphertext* dengan m untuk dicari yang lebih kecil yang mana. Selanjutnya, mulai mengiterasi panjang kunci yang akan muncul dari $\frac{m}{2} + 1$ sampai dengan m . Didalam iterasi tersebut akan dilakukan pengecekan apakah nilai m dengan fungsi $\text{VALIDITY}(m)$. Jika gagal program akan melanjutkan untuk mencari panjang kunci selanjutnya, sebaliknya jika hasil dari fungsi tersebut benar maka akan melanjutkan proses mengenerate hasil yang telah diperoleh dari panjang kunci secara satu persatu dan dibandingkan dengan hasil yang sudah ada sebelumnya. Perbandingan tersebut akan mengikuti aturan apabila suatu indeks ternyata ada yang konflik (baik yang nilainya berubah ataupun tidak memiliki suatu aturan kunci dari panjang kunci yang saat itu tersedia) (maka hasil dari indeks tersebut adalah '*' dan menghapus element dari tempat penyimpanan yang menampung indeks *plaintext* yang '*' dan *ciphertext* yang tidak '*', apabila tidak ada konflik maka *plaintext* pada indeks tersebut tidak menjadi '*'. Seperti contohnya terdapat pada table 2.4 dan tabel 2.10

Sehingga hasil keluaran yang diperoleh dari algoritma ini adalah seluruh *plaintext* yang dapat dibentuk.

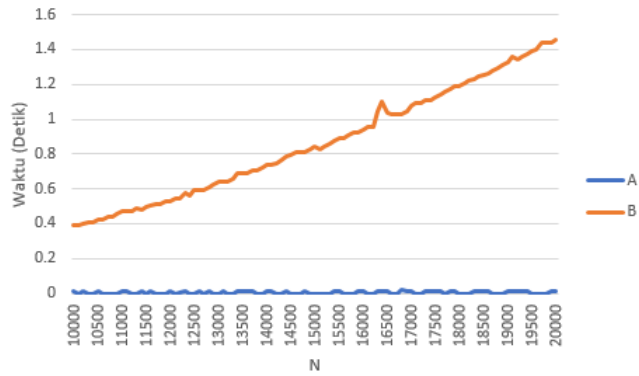
5.3 Analisa Kompleksitas Waktu

Pada *pseudocode* yang terdapat pada Gambar 3.1. Untuk setiap kasus ujicoba terdapat 2 fungsi utama. Dengan menggunakan *Kasiski Examination* fungsi SOLVE dan VALIDITY memiliki kompleksitas $\mathcal{O}((n + (\frac{M}{2} * (N + S))))$. n adalah panjang karakter *plaintext* atau *ciphertext*, $M/2$ adalah batas atas kunci dibagi dengan 2, N adalah jumlah posisi karakter yang terdapat bisa jadi memiliki suatu nilai yang bisa didapat dari *ciphertext*, dan S adalah jumlah posisi karakter yang diketahui. Untuk kompleksitas fungsi VALIDITY $\mathcal{O}(S)$. Sehingga kompleksitas dapat disederhanakan menjadi

$\mathcal{O}(T * \frac{M}{2} * (N + S))$. Sehingga secara keseluruhan kompleksitas dari algoritma yang dirancang pada tugas akhir ini adalah $\mathcal{O}(T * \frac{M}{2} * (N + S))$.

Pada umumnya, eksekusi program pada situs penilaian daring SPOJ adalah 1 detik untuk setiap 1.000.000.000 proses. Eksekusi program dengan kompleksitas $\mathcal{O}(T * \frac{M}{2} * (N + S))$. Pada *worst case* $T * (N + S) = 1.000.000$, sedangkan $\frac{M}{2}$ adalah 50,000. Hasilnya melebihi dari 50 miliar perulangan, maka waktu yang dibutuhkan adalah 50 detik. Jika hal ini terjadi maka waktu eksekusi akan berjalan dengan sangat lama. Estimasi waktu kurang lebih 100 detik, tetapi kenyataan yang terjadi tidak demikian, alasannya karena jumlah S bisa berkurang seringin dengan perulangan yang ada. Karena dari pernyataan soal bahwa file masukkan tidak akan lebih besar daripada 2 MB[3].

Sebagai perbandingan ujicoba kinerja antara menggunakan algoritma *Naive* dan algoritma optimasi *Kasiski Examination* ditambah dengan *Intersection*, dapat dilihat pada gambar 5.1. Dimana A adalah performa algoritma *Naive* dan B adalah performa algoritma optimasi *Kasiski Examination* dengan *Intersection*. N pada gambar 5.1 adalah jumlah banyak *plaintext* atau *ciphertext* sebagai masukan program.



Gambar 5.1 Perbandingan Kinerja Algoritma Optimasi *Kasiski Examination* dengan *Intersection* dan *Naive*

(Halaman ini sengaja dikosongkan)

BAB VI

PENUTUP

Bab ini membahas kesimpulan yang dapat diambil dari tujuan pembuatan sistem dan hubungannya dengan hasil uji coba yang telah dilakukan. Selain itu, terdapat beberapa saran yang bisa dijadikan acuan untuk melakukan pengembangan dan penelitian lebih lanjut.

6.1 Kesimpulan

Dari hasil uji coba yang telah dilakukan terhadap perancangan dan implementasi algoritma untuk menyelesaikan SPOJ *The Bytelandian Cryptographer (Act IV)* dapat diambil kesimpulan sebagai berikut:

1. Implementasi algoritma dengan menggunakan teknik *Kasiski Examination* dengan adanya optimasi dapat menyelesaikan permasalahan SPOJ *The Bytelandian Cryptographer (Act IV)* dengan benar.
2. Kompleksitas waktu $\mathcal{O}(T * \frac{M}{2} * (N + S))$ masih dapat menyelesaikan permasalahan SPOJ *The Bytelandian Cryptographer (Act IV)*.
3. Waktu yang dibutuhkan oleh program untuk menyelesaikan SPOJ *The Bytelandian Cryptographer (Act IV)* minimum 4,38 detik, maksimum 4,49 detik dan rata-rata 4.418 detik. Memori yang dibutuhkan berkisar antara 26-27 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. Dengan teknik *Kasiski Examination* masih cenderung lambat, karena masih menggunakan teknik *brute force*

sehingga hasil yang diperoleh kurang optimal, perlu adanya optimisasi lanjutan yang dapat mencari suatu panjang kunci. Dapat lihat sebenarnya pada gambar A.5 dan A.6.

2. Perlu adanya Optimisasi dalam hal pencarian suatu indeks yang perlu dirubah atau tidak. Dengan teknik yang dipakai oleh penulis tidak dapat memenuhi ekspektasi jika berharap dengan hasil yang sangat cepat.

DAFTAR PUSTAKA

- [1] W. Stallings and L. Brown, *Computer Security Principles and Practice*, 3rd ed. Pearson, 2015.
- [2] S. William, *Cryptography and Network Security*, 5th ed. Pearson, 2011.
- [3] K. Piwakowski, "CRYPTO4 - The Bytelandian Cryptographer (Act IV)," 2004. [Online]. Available: <http://www.spoj.com/problems/CRYPTO4/>
- [4] john_jones, "SPOJ Discussion Board," 2009. [Online]. Available: <http://discuss.spoj.com/t/problemset-3/242/13>
- [5] K. Devlin, *The Joy of Sets: Fundamentals of Contemporary Set Theory*, 2nd ed. New York: Springer, 1993.
- [6] Center for History and New Media, "Zotero Quick Start Guide." [Online]. Available: http://zotero.org/support/quick_start_guide
- [7] C. Henk, *Encyclopedia of Cryptography and Security*. Springer, 2005.
- [8] "Kasiski Method." [Online]. Available: <http://pages.mtu.edu/~shene/NSF-4/Tutorial/VIG/Vig-Kasiski.html>

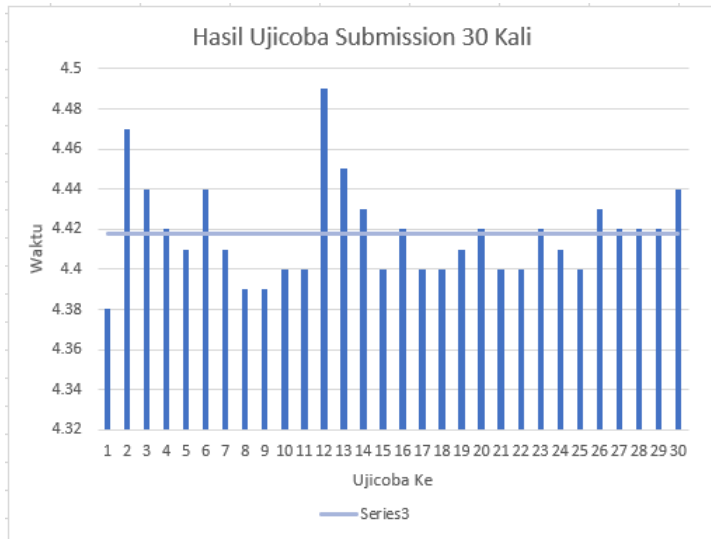
(Halaman ini sengaja dikosongkan)

BAB A

LAMPIRAN

| | | | | | | | |
|----------|---|------------------------|--|-----------------------------------|------|-----|-------|
| 20609690 |  | 2017-11-16 06:24:28 | The Bytelandian Cryptographer (Act IV) | accepted edit ideone.it | 4.38 | 26M | CPP14 |
|----------|---|------------------------|--|-----------------------------------|------|-----|-------|











Gambar A.1 Hasil Uji Coba pada Situs Penilaian SPOJ



Gambar A.2 Grafik Hasil Uji Coba pada Situs SPOJ Sebanyak 30 Kali

| | | | | | | | |
|----------|---|------------------------|--|----------------------------------|------|-----|-------|
| 20609690 | ■ | 2017-11-09 06:24:26 | The Bytelandian Cryptographer (Act IV) | accepted edit delete 0 | 4,38 | 26M | CPP14 |
| 20609689 | ■ | 2017-11-09 06:24:20 | The Bytelandian Cryptographer (Act IV) | accepted edit delete 0 | 4,47 | 27M | CPP14 |
| 20609688 | ■ | 2017-11-09 06:24:10 | The Bytelandian Cryptographer (Act IV) | accepted edit delete 0 | 4,44 | 26M | CPP14 |
| 20609686 | ■ | 2017-11-09 06:24:02 | The Bytelandian Cryptographer (Act IV) | accepted edit delete 0 | 4,42 | 26M | CPP14 |
| 20609684 | ■ | 2017-11-09 06:23:52 | The Bytelandian Cryptographer (Act IV) | accepted edit delete 0 | 4,41 | 26M | CPP14 |
| 20609682 | ■ | 2017-11-09 06:23:43 | The Bytelandian Cryptographer (Act IV) | accepted edit delete 0 | 4,44 | 26M | CPP14 |
| 20609681 | ■ | 2017-11-09 06:23:36 | The Bytelandian Cryptographer (Act IV) | accepted edit delete 0 | 4,41 | 26M | CPP14 |
| 20609680 | ■ | 2017-11-09 06:23:20 | The Bytelandian Cryptographer (Act IV) | accepted edit delete 0 | 4,39 | 26M | CPP14 |
| 20609678 | ■ | 2017-11-09 06:23:20 | The Bytelandian Cryptographer (Act IV) | accepted edit delete 0 | 4,39 | 26M | CPP14 |
| 20609675 | ■ | 2017-11-09 06:23:10 | The Bytelandian Cryptographer (Act IV) | accepted edit delete 0 | 4,40 | 27M | CPP14 |
| 20609673 | ■ | 2017-11-09 06:23:05 | The Bytelandian Cryptographer (Act IV) | accepted edit delete 0 | 4,40 | 26M | CPP14 |
| 20609672 | ■ | 2017-11-09 06:23:52 | The Bytelandian Cryptographer (Act IV) | accepted edit delete 0 | 4,49 | 27M | CPP14 |
| 20609671 | ■ | 2017-11-09 06:23:43 | The Bytelandian Cryptographer (Act IV) | accepted edit delete 0 | 4,43 | 27M | CPP14 |
| 20609669 | ■ | 2017-11-09 06:23:26 | The Bytelandian Cryptographer (Act IV) | accepted edit delete 0 | 4,43 | 27M | CPP14 |
| 20609667 | ■ | 2017-11-09 06:23:16 | The Bytelandian Cryptographer (Act IV) | accepted edit delete 0 | 4,40 | 27M | CPP14 |
| 20609664 | ■ | 2017-11-09 06:23:07 | The Bytelandian Cryptographer (Act IV) | accepted edit delete 0 | 4,42 | 27M | CPP14 |
| 20609663 | ■ | 2017-11-09 06:23:56 | The Bytelandian Cryptographer (Act IV) | accepted edit delete 0 | 4,40 | 26M | CPP14 |
| 20609663 | ■ | 2017-11-09 06:23:44 | The Bytelandian Cryptographer (Act IV) | accepted edit delete 0 | 4,40 | 26M | CPP14 |
| 20609662 | ■ | 2017-11-09 06:23:36 | The Bytelandian Cryptographer (Act IV) | accepted edit delete 0 | 4,41 | 26M | CPP14 |
| 20609661 | ■ | 2017-11-09 06:23:20 | The Bytelandian Cryptographer (Act IV) | accepted edit delete 0 | 4,42 | 26M | CPP14 |

Gambar A.3 Hasil Pengujian Sebanyak 30 Kali pada Situs Penilaian Daring SPOJ (1)

| | | | | | | | |
|----------|---|------------------------|--|-----------------------------------|------|-----|-------|
| 20609659 |  | 2017-11-15 06:21:21 | The Bytelandian Cryptographer (Act IV) | accepted edit ideone it | 4.40 | 26M | CPP14 |
| 20609657 |  | 2017-11-16 06:21:12 | The Bytelandian Cryptographer (Act IV) | accepted edit ideone it | 4.40 | 27M | CPP14 |
| 20609656 |  | 2017-11-16 06:21:02 | The Bytelandian Cryptographer (Act IV) | accepted edit ideone it | 4.42 | 26M | CPP14 |
| 20609655 |  | 2017-11-16 06:20:50 | The Bytelandian Cryptographer (Act IV) | accepted edit ideone it | 4.41 | 26M | CPP14 |
| 20609654 |  | 2017-11-16 06:20:32 | The Bytelandian Cryptographer (Act IV) | accepted edit ideone it | 4.40 | 27M | CPP14 |
| 20609652 |  | 2017-11-16 06:20:11 | The Bytelandian Cryptographer (Act IV) | accepted edit ideone it | 4.43 | 26M | CPP14 |
| 20609651 |  | 2017-11-16 06:20:00 | The Bytelandian Cryptographer (Act IV) | accepted edit ideone it | 4.42 | 27M | CPP14 |
| 20609650 |  | 2017-11-16 06:19:35 | The Bytelandian Cryptographer (Act IV) | accepted edit ideone it | 4.42 | 26M | CPP14 |
| 20603932 |  | 2017-11-15 12:19:55 | The Bytelandian Cryptographer (Act IV) | accepted edit ideone it | 4.42 | 27M | CPP14 |
| 20603856 |  | 2017-11-15 12:06:01 | The Bytelandian Cryptographer (Act IV) | accepted edit ideone it | 4.44 | 27M | CPP14 |

Gambar A.4 Hasil Pengujian Sebanyak 30 Kali pada Situs Penilaian Daring SPOJ (2)

| RANK | DATE | USER | RESULT | TIME | MEM | LANG |
|------|------------------------|---------------------------------------|----------|------|------|--------------|
| 1 | 2014-07-09 20:26:13 | sidharth jain | accepted | 0.21 | 18M | C++ 4.3.2 |
| 2 | 2011-02-04 10:11:14 | Josef K. | accepted | 0.32 | 9.0M | CPP |
| 3 | 2010-02-25 22:16:52 | Carlos Eduardo Rodrigues Alves [USJT] | accepted | 0.33 | 31M | CPP |
| 4 | 2011-06-04 01:25:06 | NiHaobin | accepted | 0.34 | 151M | CPP |
| 5 | 2009-12-29 16:48:15 | [Rampage] Blue.Mary | accepted | 0.39 | 28M | C++ 4.3.2 |
| 6 | 2010-05-03 01:07:25 | Carlos Eduardo Rodrigues Alves [USJT] | accepted | 0.39 | 30M | C++ 4.3.2 |
| 7 | 2011-02-04 10:12:30 | Josef K. | accepted | 0.39 | 8.4M | C++ 4.3.2 |
| 8 | 2004-12-08 18:13:03 | Jakub Łopuszański | accepted | 0.42 | 14M | CPP |
| 9 | 2009-03-07 01:11:57 | Robert Gerbicz | accepted | 0.42 | 33M | CPP |
| 10 | 2010-04-23 11:23:55 | Oleg | accepted | 0.44 | 7.5M | C++ 4.3.2 |
| 11 | 2004-11-27 07:22:27 | Tomek Czajka | accepted | 0.53 | 4.3M | CPP |
| 12 | 2007-05-31 03:54:03 | Huacheng Yu | accepted | 0.60 | 17M | C |
| 13 | 2010-07-02 12:47:04 | 刘启鹏 | accepted | 0.61 | 17M | C |
| 14 | 2010-12-18 14:59:02 | sevenkplus | accepted | 0.70 | 21M | C++ 4.3.2 |
| 15 | 2010-12-18 14:55:16 | sevenkplus | accepted | 0.77 | 21M | CPP |
| 16 | 2011-03-05 23:42:24 | Alexander Pivovarov | accepted | 0.77 | 118M | CPP |
| 17 | 2013-08-02 09:56:37 | Tomasz Stanislawski | accepted | 0.77 | 14M | C99 |
| 18 | 2011-06-06 13:20:16 | blashyrkh | accepted | 0.80 | 23M | C |
| 19 | 2007-05-31 01:58:27 | FG | accepted | 0.95 | 2.7M | PAS- FPC |
| 20 | 2008-03-19 03:57:13 | zhengxi | accepted | 0.98 | 52M | CPP |

Gambar A.5 Daftar Peringkat Berdasarkan Kecepatan yang Diperoleh dari Dari SPOJ(1)

| RANK | DATE | USER | RESULT | TIME | MEM | LANG |
|------|------------------------|-----------------|----------|------|------|--------------|
| 21 | 2009-06-22 16:57:33 | QIZIChao | accepted | 0.98 | 13M | CPP |
| 22 | 2007-09-08 22:37:16 | Sandeep Kumar | accepted | 1.24 | 17M | CPP |
| 23 | 2006-08-31 22:54:06 | Tijs van Bakel | accepted | 1.32 | 59M | CPP |
| 24 | 2007-02-04 18:33:29 | James Cook | accepted | 1.52 | 10M | C |
| 25 | 2010-09-14 13:48:55 | Laxminarayana | accepted | 1.57 | 12M | C++ 4.3.2 |
| 26 | 2013-02-05 03:27:04 | roginn | accepted | 1.60 | 3.9M | C++ 4.3.2 |
| 27 | 2009-11-01 21:00:00 | anonymous | accepted | 1.64 | 3.1M | C++ 4.3.2 |
| 28 | 2011-08-06 21:38:41 | Peutri | accepted | 1.91 | 2.8M | C++ 4.3.2 |
| 29 | 2004-11-18 17:26:14 | Pascal Zimmer | accepted | 1.94 | 7.2M | C |
| 30 | 2005-02-03 10:06:50 | Tim Green @ Ark | accepted | 1.94 | 13M | CPP |
| 31 | 2015-09-03 20:00:03 | vijaygbvv | accepted | 2.36 | 5.6M | C++ 4.3.2 |
| 32 | 2011-08-23 16:06:32 | Darren Izzard | accepted | 2.48 | 3.1M | C++ 4.3.2 |
| 33 | 2017-11-15 11:46:27 | freddy | accepted | 4.38 | 27M | CPP14 |

Gambar A.6 Daftar Hasil Peringkat Berdasarkan Kecepatan yang Diperoleh dari Dari SPOJ(2)

(Halaman ini sengaja dikosongkan)

BAB B

HASIL PERCOBAAN DENGAN MENGGUNAKAN ALGORITMA NAIVE DAN KASISKI EXAMINATION

| N | Waktu yang diperlukan dalam detik |
|-------|-----------------------------------|
| 10000 | 0.0156259 |
| 10100 | 0 |
| 10200 | 0.0156244 |
| 10300 | 0 |
| 10400 | 0 |
| 10500 | 0.0156255 |
| 10600 | 0 |
| 10700 | 0 |
| 10800 | 0 |
| 10900 | 0 |
| 11000 | 0.0156248 |
| 11100 | 0.0156351 |
| 11200 | 0 |
| 11300 | 0 |
| 11400 | 0.0156251 |
| 11500 | 0 |
| 11600 | 0.0156259 |
| 11700 | 0 |
| 11800 | 0 |
| 11900 | 0 |

Tabel 2.1 Hasil Percobaan Penyelesaian Studi Kasus SPOJ The Bytelandian Cryptographer(Act IV) dengan menggunakan algoritma *Optimasi Kasiski Examination* dan *Intersection* (1)

| N | Waktu yang diperlukan dalam detik |
|-------|-----------------------------------|
| 12000 | 0.0156252 |
| 12100 | 0 |
| 12200 | 0.001048 |
| 12300 | 0.0156267 |
| 12400 | 0 |
| 12500 | 0 |
| 12600 | 0.0156237 |
| 12700 | 0 |
| 12800 | 0.0156244 |
| 12900 | 0 |
| 13000 | 0 |
| 13100 | 0.0156244 |
| 13200 | 0 |
| 13300 | 0 |
| 13400 | 0.0156229 |
| 13500 | 0.0156251 |
| 13600 | 0.0156441 |
| 13700 | 0.0156351 |
| 13800 | 0 |
| 13900 | 0 |

Tabel 2.2 Hasil Percobaan Penyelesaian Studi Kasus SPOJ The Bytelandian Cryptographer(Act IV) dengan menggunakan algoritma *Optimasi Kasiski Examination* dan *Intersection* (2)

| N | Waktu yang diperlukan dalam detik |
|-------|-----------------------------------|
| 14000 | 0.0156149 |
| 14100 | 0.0156252 |
| 14200 | 0 |
| 14300 | 0 |
| 14400 | 0.0156343 |
| 14500 | 0 |
| 14600 | 0 |
| 14700 | 0 |
| 14800 | 0.0156229 |
| 14900 | 0 |
| 15000 | 0 |
| 15100 | 0 |
| 15200 | 0 |
| 15300 | 0 |
| 15400 | 0.0156354 |
| 15500 | 0.0156263 |
| 15600 | 0 |
| 15700 | 0 |
| 15800 | 0 |
| 15900 | 0.0156278 |

Tabel 2.3 Hasil Percobaan Penyelesaian Studi Kasus SPOJ The Bytelandian Cryptographer(Act IV) dengan menggunakan algoritma *Optimasi Kasiski Examination* dan *Intersection* (3)

| N | Waktu yang diperlukan dalam detik |
|-------|-----------------------------------|
| 16000 | 0.0156256 |
| 16100 | 0 |
| 16200 | 0 |
| 16300 | 0.0156351 |
| 16400 | 0.0156244 |
| 16500 | 0.0156278 |
| 16600 | 0 |
| 16700 | 0 |
| 16800 | 0.0227098 |
| 16900 | 0.0156351 |
| 17000 | 0.0156153 |
| 17100 | 0 |
| 17200 | 0 |
| 17300 | 0.015635 |
| 17400 | 0.0156259 |
| 17500 | 0.0156347 |
| 17600 | 0.0156153 |
| 17700 | 0 |
| 17800 | 0.0156191 |
| 17900 | 0.0156247 |

Tabel 2.4 Hasil Percobaan Penyelesaian Studi Kasus SPOJ The Bytelandian Cryptographer(Act IV) dengan menggunakan algoritma *Optimasi Kasiski Examination* dan *Intersection* (4)

| N | Waktu yang diperlukan dalam detik |
|-------|-----------------------------------|
| 18000 | 0 |
| 18100 | 0 |
| 18200 | 0 |
| 18300 | 0.0156229 |
| 18400 | 0.0156255 |
| 18500 | 0.0156247 |
| 18600 | 0.0156225 |
| 18700 | 0 |
| 18800 | 0 |
| 18900 | 0 |
| 19000 | 0.0156248 |
| 19100 | 0.0156275 |
| 19200 | 0.0156252 |
| 19300 | 0.0156206 |
| 19400 | 0.015626 |
| 19500 | 0 |
| 19600 | 0 |
| 19700 | 0 |
| 19800 | 0 |
| 19900 | 0.0156248 |

Tabel 2.5 Hasil Percobaan Penyelesaian Studi Kasus SPOJ The Bytelandian Cryptographer(Act IV) dengan menggunakan algoritma *Optimasi Kasiski Examination* dan *Intersection* (5)

| N | Waktu yang diperlukan dalam detik |
|-------|-----------------------------------|
| 10000 | 0.390619 |
| 10100 | 0.394834 |
| 10200 | 0.400778 |
| 10300 | 0.406246 |
| 10400 | 0.406737 |
| 10500 | 0.42187 |
| 10600 | 0.421361 |
| 10700 | 0.437496 |
| 10800 | 0.437661 |
| 10900 | 0.45313 |
| 11000 | 0.468254 |
| 11100 | 0.468746 |
| 11200 | 0.469241 |
| 11300 | 0.48438 |
| 11400 | 0.48388 |
| 11500 | 0.498918 |
| 11600 | 0.501698 |
| 11700 | 0.515621 |
| 11800 | 0.516122 |
| 11900 | 0.531256 |

Tabel 2.6 Hasil Percobaan Penyelesaian Studi Kasus SPOJ The Bytelandian Cryptographer(Act IV) dengan menggunakan algoritma *Naive* (1)

| N | Waktu yang diperlukan dalam detik |
|-------|-----------------------------------|
| 12000 | 0.530714 |
| 12100 | 0.546407 |
| 12200 | 0.54688 |
| 12300 | 0.578888 |
| 12400 | 0.562516 |
| 12500 | 0.593242 |
| 12600 | 0.594233 |
| 12700 | 0.593746 |
| 12800 | 0.608855 |
| 12900 | 0.621452 |
| 13000 | 0.640633 |
| 13100 | 0.640096 |
| 13200 | 0.641099 |
| 13300 | 0.656247 |
| 13400 | 0.687057 |
| 13500 | 0.687981 |
| 13600 | 0.687508 |
| 13700 | 0.702599 |
| 13800 | 0.703614 |
| 13900 | 0.718225 |

Tabel 2.7 Hasil Percobaan Penyelesaian Studi Kasus SPOJ The Bytelandian Cryptographer(Act IV) dengan menggunakan algoritma *Naive* (2)

| N | Waktu yang diperlukan dalam detik |
|-------|-----------------------------------|
| 14000 | 0.734383 |
| 14100 | 0.734832 |
| 14200 | 0.749211 |
| 14300 | 0.766334 |
| 14400 | 0.789337 |
| 14500 | 0.796879 |
| 14600 | 0.81308 |
| 14700 | 0.812038 |
| 14800 | 0.813018 |
| 14900 | 0.827625 |
| 15000 | 0.844241 |
| 15100 | 0.827581 |
| 15200 | 0.843761 |
| 15300 | 0.859857 |
| 15400 | 0.874509 |
| 15500 | 0.891106 |
| 15600 | 0.890121 |
| 15700 | 0.906285 |
| 15800 | 0.921589 |
| 15900 | 0.922729 |

Tabel 2.8 Hasil Percobaan Penyelesaian Studi Kasus SPOJ The Bytelandian Cryptographer(Act IV) dengan menggunakan algoritma *Naive* (3)

| N | Waktu yang diperlukan dalam detik |
|-------|-----------------------------------|
| 16000 | 0.937003 |
| 16100 | 0.952834 |
| 16200 | 0.953624 |
| 16300 | 1.04645 |
| 16400 | 1.10416 |
| 16500 | 1.0375 |
| 16600 | 1.03005 |
| 16700 | 1.03179 |
| 16800 | 1.03075 |
| 16900 | 1.04765 |
| 17000 | 1.07823 |
| 17100 | 1.09324 |
| 17200 | 1.09423 |
| 17300 | 1.10882 |
| 17400 | 1.10902 |
| 17500 | 1.12558 |
| 17600 | 1.14016 |
| 17700 | 1.15626 |
| 17800 | 1.17237 |
| 17900 | 1.18701 |

Tabel 2.9 Hasil Percobaan Penyelesaian Studi Kasus SPOJ The Bytelandian Cryptographer(Act IV) dengan menggunakan algoritma *Naive* (4)

| N | Waktu yang diperlukan dalam detik |
|-------|-----------------------------------|
| 18000 | 1.18803 |
| 18100 | 1.20266 |
| 18200 | 1.21944 |
| 18300 | 1.23395 |
| 18400 | 1.24748 |
| 18500 | 1.25243 |
| 18600 | 1.26614 |
| 18700 | 1.28063 |
| 18800 | 1.2969 |
| 18900 | 1.31404 |
| 19000 | 1.32662 |
| 19100 | 1.35917 |
| 19200 | 1.34388 |
| 19300 | 1.36163 |
| 19400 | 1.37299 |
| 19500 | 1.39379 |
| 19600 | 1.40303 |
| 19700 | 1.43746 |
| 19800 | 1.43739 |
| 19900 | 1.43803 |

Tabel 2.10 Hasil Percobaan Penyelesaian Studi Kasus SPOJ The Bytelandian Cryptographer(Act IV) dengan menggunakan algoritma *Naive* (5)

BIODATA PENULIS



Freddy Hermawan Yuwono, kelahiran & besar di Bondowoso-Jawa Timur, sangat suka membaca. Penulis menempuh jenjang pendidikan S1 Teknik Informatika ITS dari tahun 2013 sampai dengan dibuatnya buku ini.

Motto penulis yaitu "Segala sesuatu pasti akan terjadi dan pasti akan dilewati", membawa penulis mencoba belajar yang baru topik tugas akhir ini, dimana penulis dapat menerapkan sesuatu yang belum pernah penulis untuk melaluinya. Algoritma, optimasi dan pelajaran yang penulis petik dari yang pernah dilakukan oleh penulis sebelumnya, dengan bimbingan dosen-dosen pembimbing. Dalam pendalaman topik tugas akhir ini juga, penulis banyak belajar dan menjadi sangat tertarik mendalami *cryptography*, dan *data scientist*.

Dengan segala kerendahan hati, ilmu penulis masihlah setitik dibandingkan susu sebelanga. Penulis sangat mengharapkan diskusi, ajaran dan bantuan dalam memperbaiki diri. Apabila pembaca berkenan, penulis dapat dihubungi melalui *email* ke freddy.yuwono@gmail.com.