TUGAS AKHIR - IF184802

PERBANDINGAN KINERJA ALGORITMA BABY-STEP GIANT-STEP DAN POHLIG-HELLMAN SEBAGAI METODE PENYELESAIAN LOGARITMA DISKRIT DENGAN MODULUS BILANGAN PRIMA: STUDI KASUS URI ONLINE JUDGE 2711 UNLOCKING A CELL PHONE

TAUFIQ TIRTAJIWANGGA
NRP 05111640000016

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

Dosen Pembimbing 2
M. M. Irfan Subakti, S.Kom., M.Sc.Eng., M.Phil.

DEPARTEMEN TEKNIK INFORMATIKA
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember
Surabaya, 2020

[*Halaman ini sengaja dikosongkan*]

TUGAS AKHIR - IF184802

PERBANDINGAN KINERJA ALGORITMA BABY-STEP GIANT-STEP DAN POHLIG-HELLMAN SEBAGAI METODE PENYELESAIAN LOGARITMA DISKRIT DENGAN MODULUS BILANGAN PRIMA: STUDI KASUS URI ONLINE JUDGE 2711 UNLOCKING A CELL PHONE

TAUFIQ TIRTAJIWANGGA
NRP 05111640000016

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

Dosen Pembimbing 2
M. M. Irfan Subakti, S.Kom., M.Sc.Eng., M.Phil.

DEPARTEMEN TEKNIK INFORMATIKA
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember
Surabaya, 2020

[*Halaman ini sengaja dikosongkan*]

UNDERGRADUATE THESIS - IF184802

PERFORMANCE COMPARISON OF BABY-STEP GIANT-STEP AND POHLIG-HELLMAN ALGORITHMS AS DISCRETE LOGARITHM WITH MODULUS OF PRIME NUMBER RESOLUTION METHOD: CASE STUDY OF URI ONLINE JUDGE 2711 UNLOCKING A CELL PHONE

TAUFIQ TIRTAJIWANGGA
NRP 05111640000016

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

Supervisor 2
M. M. Irfan Subakti, S.Kom., M.Sc.Eng., M.Phil.

DEPARTMENT OF INFORMATICS ENGINEERING
Faculty of Intelligent Electrical and Informatics Technology
Institut Teknologi Sepuluh Nopember
Surabaya, 2020

[*Halaman ini sengaja dikosongkan*]

LEMBAR PENGESAHAN

PERBANDINGAN KINERJA ALGORITMA BABY-STEP GIANT-STEP DAN POHLIG-HELLMAN SEBAGAI METODE PENYELESAIAN LOGARITMA DISKRIT DENGAN MODULUS BILANGAN PRIMA: STUDI KASUS URI ONLINE JUDGE 2711 UNLOCKING A CELL PHONE

TUGAS AKHIR

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

Oleh:
Taufiq Tirtajiwangga
NRP. 05111640000016

Disetujui oleh Dosen Pembimbing Tugas Akhir:

Rully Soelaiman, S.Kom., M.Kom. .....
NIP. 19700213194021001 (Pembimbing 1)

M. M. Irfan Subakti, S.Kom., M.Sc.Eng., M.Phil. .....
NIP. 197402092002121001 (Pembimbing 2)

SURABAYA
Januari 2020

[*Halaman ini sengaja dikosongkan*]

ABSTRAK

PERBANDINGAN KINERJA ALGORITMA BABY-STEP GIANT-STEP DAN POHLIG-HELLMAN SEBAGAI METODE PENYELESAIAN LOGARITMA DISKRIT DENGAN MODULUS BILANGAN PRIMA: STUDI KASUS URI ONLINE JUDGE 2711 UNLOCKING A CELL PHONE

Nama	:	Taufiq Tirtajiwangga
NRP	:	05111640000016
Departemen	:	Departemen Teknik Informatika, Fakultas Teknologi Elektro dan Informatika Cerdas, ITS
Pembimbing I	:	Rully Soelaiman, S.Kom., M.Kom.
Pembimbing II	:	M. M. Irfan Subakti, S.Kom., M.Sc.Eng., M.Phil.

Abstrak

Keamanan adalah salah satu aspek terpenting dalam pertukaran data melalui jaringan internet. Salah satu contoh metode pengamanan data adalah dengan membuat asymmetric key yang terdiri dari public key untuk mengenkripsi data dan private key untuk mendekripsi data. Beberapa skema keamanan seperti ElGamal Encryption dan Digital Signature Algorithm menggunakan public key dan private key yang dibuat berdasarkan logaritma diskrit dengan memanfaatkan kemudahan melakukan dekripsi data menggunakan private key dan operasi pemangkatan modular. Namun demikian, tetaplah sulit untuk mengetahui private key bila yang diketahui hanyalah public key. Dalam Tugas Akhir ini dibahas mengenai kinerja penyelesaian permasalahan logaritma diskrit menggunakan algoritma Baby-Step Giant-Step dan Pohlig-Hellman untuk mendapatkan private key berdasarkan public key yang diketahui. Dari hasil uji

coba pada studi kasus yang digunakan, didapatkan hasil bahwa algoritma Pohlig-Hellman dengan rata-rata running time maksimal $3,195 \pm 0,720\ ms$ *memiliki kinerja yang lebih baik dibandingkan algoritma Baby-Step Giant-Step dengan rata-rata running time maksimal* $17,709 \pm 1,021\ ms$.

Kata Kunci: logaritma diskrit; Baby-Step Giant-Step; Pohlig-Hellman

ABSTRACT

PERFORMANCE COMPARISON OF BABY-STEP GIANT-STEP AND POHLIG-HELLMAN ALGORITHMS AS DISCRETE LOGARITHM WITH MODULUS OF PRIME NUMBER RESOLUTION METHOD: CASE STUDY OF URI ONLINE JUDGE 2711 UNLOCKING A CELL PHONE

Name	:	Taufiq Tirtajiwangga
Student ID	:	05111640000016
Department	:	Department of Informatics Engineering, Faculty of Intelligent Electrical and Informatics Technology, ITS
Supervisor I	:	Rully Soelaiman, S.Kom., M.Kom.
Supervisor II	:	M. M. Irfan Subakti, S.Kom., M.Sc.Eng., M.Phil.

Abstract

Security is one of most important aspect in data exchange through internet. One of data securing method is to create asymmetric key which consists of public key for data encryption and private key for data decryption. Security schemes like ElGamal Encryption and Digital Signature Algorithm have used public key and private key which have been built based on discrete logarithm that utilise the ease of data decryption by using private key and modular exponentiation operation. However, it is difficult to retrieve private key if only public key has been known. This thesis investigated the performance of Baby-Step Giant-Step and Pohlig-Hellman algorithm for solving the discrete logarithm problem. Based on the result of the given case study, it can be shown that the performance of Pohlig-Hellman algorithm with maximum average running time $3.195 \pm 0.720\ ms$ *is*

better than Baby-Step Giant-Step algorithm with maximum average running time $17.709 \pm 1.021\ ms.$

Keywords: discrete logarithm; Baby-Step Giant-Step; Pohlig-Hellman

KATA PENGANTAR

Puji syukur penulis ucapkan kepada Tuhan Yang Maha Esa. Atas rahmat dan kasih sayang-Nya, penulis dapat menyelesaikan Tugas Akhir ini.

Pengerjaan Tugas Akhir ini penulis tujukan untuk mengeksplorasi lebih mendalam topik-topik yang tidak diwadahi oleh kampus, namun banyak menarik perhatian penulis. Selain itu besar harapan penulis bahwa pengerjaan Tugas Akhir ini dapat menjadi sarana penulis dalam menimba ilmu yang bermanfaat.

Penulis ingin menyampaikan rasa terima kasih kepada banyak pihak yang telah membimbing, menemani, dan membantu penulis selama masa pengerjaan Tugas Akhir maupun dalam masa studi.

1. Bapak dan Ibu dari penulis yang telah memberikan dukungan kepada penulis untuk menyelesaikan pendidikan di Perguruan Tinggi.
2. Bapak Rully Soelaiman, S.Kom., M.Kom. dan Bapak M. M. Irfan Subakti, S.Kom., M.Sc.Eng., M.Phil. selaku pembimbing penulis yang telah memberikan perhatian, didikan, pengajaran, dan nasihat selama masa studi penulis.
3. Teman penulis Aguel Satria yang telah meminjamkan laptopnya untuk saya gunakan sejak semester 5 hingga saat ini.
4. Teman-teman saya yang telah menemani saya berdiskusi mengenai berbagai hal dan memberikan bantuan-bantuan lain.

Penulis menyadari buku Tugas Akhir jauh dari kata sempurna. Maka dari itu, penulis memohon maaf apabila terdapat salah kata maupun makna pada buku Tugas Akhir ini. Akhir kata, penulis mempersembahkan buku ini sebagai wujud nyata kontribusi

penulis dalam ilmu pengetahuan.

Surabaya, Januari 2020

Taufiq Tirtajiwangga

DAFTAR ISI

[*Halaman ini sengaja dikosongkan*]

DAFTAR GAMBAR

[*Halaman ini sengaja dikosongkan*]

DAFTAR TABEL

[*Halaman ini sengaja dikosongkan*]

DAFTAR PSEUDOCODE

[*Halaman ini sengaja dikosongkan*]

DAFTAR KODE SUMBER

DAFTAR NOTASI

$\mathbb{Z}$	Himpunan bilangan bulat.
$\mathbb{Z}_n$, $\mathbb{Z}/n\mathbb{Z}$	Himpunan bilangan bulat non-negatif terkecil modulo n .
$\mathbb{Z}_n^*$	Himpunan bilangan bulat $\mathbb{Z}_n$ dan koprima dengan n.
$\|\mathbb{Z}\|$	Banyaknya elemen dari himpunan $\mathbb{Z}$.

[*Halaman ini sengaja dikosongkan*]

BAB I
PENDAHULUAN

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

1.1 Latar Belakang

Keamanan adalah sebuah aspek penting pada proses pertukaran data. Hal ini dikarenakan terdapat berbagai faktor yang dapat mengakibatkan terjadinya pengubahan atau penyadapan data oleh pihak yang tidak memiliki hak terhadap data tersebut. Untuk itu dibuat berbagai usaha untuk melakukan penyandian data atau pemeriksaan terhadap keutuhan dan keaslian data.

Permasalahan logaritma diskrit digunakan dalam beberapa skema pengamanan data dikarenakan sulitnya untuk menentukan nilai dari logaritma diskrit dibandingkan dengan operasi inversnya, pemangkatan modular. Untuk mengilustrasikan pentingnya peran logaritma diskrit, digunakan contoh skema keamanan *ElGamal Encryption*. Pada *ElGamal*, dibuat *private key* x dan *public key* $P = (p, g, h)$ (dengan p adalah bilangan prima) yang memenuhi Persamaan 1.1 [1].

$$h = g^x \pmod{p} \tag{1.1}$$

Variabel x disebut sebagai logaritma diskrit dari Persamaan 1.1 [1]. Sebuah data yang dienkripsi menggunakan *public key* P hanya dapat didekripsi dengan mengetahui *private key* x. Hal ini mengimplikasikan bahwa dengan mengetahui *private key* x, seluruh data yang dienkripsi menggunakan *public key* P dapat didekripsi.

Pada Tugas Akhir ini akan dilakukan analisis perbandingan kinerja algoritma penyelesaian logaritma diskrit dengan modulus bilangan prima antara metode *Baby-Step Giant-Step* dan metode

Pohlig-Hellman menggunakan studi kasus *URI Online Judge* 2711 *Unlocking A Cell Phone* [2]. Dalam permasalahan, diketahui bilangan bulat positif B, N, dan bilangan prima M yang berturut-turut memenuhi Persamaan 1.2.

$$B^E = N \pmod{M} \tag{1.2}$$

Pada Persamaan 1.2, E adalah nilai logaritma diskrit dari B modulo M.

1.2 Rumusan Masalah

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

1. Bagaimana kinerja algoritma *Baby-Step Giant-Step* yang diimplementasikan untuk menyelesaikan permasalahan komputasi logaritma diskrit studi kasus *Unlocking A Cell Phone*?
2. Bagaimana kinerja algoritma Pohlig-Hellman yang diimplementasikan untuk menyelesaikan permasalahan komputasi logaritma diskrit pada studi kasus *Unlocking A Cell Phone*?
3. Bagaimana perbandingan kinerja penyelesaian permasalahan komputasi logaritma diskrit pada studi kasus *Unlocking A Cell Phone* antara metode *Baby-Step Giant-Step* dan metode Pohlig-Hellman?

1.3 Batasan Masalah

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

1. Perbandingan antara algoritma *Baby-Step Giant-Step* sebagai penyelesaian *URI Online Judge* 2711 *Unlocking A Cell Phone*.
2. Bilangan modulus yang digunakan adalah bilangan prima.

3. Algoritma *Baby-Step Giant-Step* yang dibahas terbatas pada analisis intuitif dan logis.
4. Algoritma Pohlig-Hellman yang dibahasa terbatas pada analisis intuitif dan logis.

Pada kasus *URI Online Judge* 2711 *Unlocking A Cell Phone*, diberikan batasan sebagai berikut.

1. Implementasi algoritma menggunakan bahasa pemrograman C++.
2. Batas nilai B dan N adalah bilangan bulat pada rentang 0 sampai 10^5 eksklusif.
3. Batas nilai M adalah bilangan bulat prima pada rentang 1 sampai 10^9 eksklusif.
4. Batas nilai E adalah bilangan bulat pada rentang -1 sampai $M - 1$ inklusif.
5. Batas waktu yang diberikan adalah 2 detik.

1.4 Tujuan

Tujuan dari pembuatan Tugas Akhir ini adalah untuk mengetahui algoritma dengan kinerja yang lebih baik antara algoritma *Baby-Step Giant-Step* dan Pohlig-Hellman dalam menyelesaikan permasalahan logaritma diskrit.

1.5 Metodologi

Metodologi pengerjaan yang digunakan pada Tugas Akhir ini memiliki beberapa tahapan. Tahapan-tahapan tersebut yaitu:

1. Penyusunan proposal
   Penjelasan mengenai apa yang penulis akan lakukan dan alasan Tugas Akhir ini dilakukan diberikan di tahapan ini. Penjelasan tersebut dituliskan dalam bentuk proposal Tugas Akhir.
2. Studi literatur

Referensi yang diperlukan guna mendukung pengerjaan Tugas Akhir dikumpulkan pada tahapan ini. Referensi yang digunakan dapat berupa hasil penelitian yang sudah pernah dilakukan, buku, artikel internet, atau sumber lain yang bisa dipertanggungjawabkan.

3. Implementasi algoritma
   Pada tahapan ini, algoritma yang digunakan mulai dikembangkan untuk menyelesaikan permasalahan logaritma diskrit.
4. Pengujian dan evaluasi
   Kinerja algoritma yang digunakan diuji pada tahapan ini. Hasil pengujian kemudian dievaluasi untuk kemudian dilakukan pengambilan kesimpulan.
5. Penyusunan buku
   Hasil pengerjaan Tugas Akhir disusun dalam bentuk buku laporan Tugas Akhir dengan mengikuti format penulisan buku Tugas Akhir.

1.6 Sistematika Penulisan

Sistematika laporan Tugas Akhir yang akan digunakan adalah sebagai berikut.

1. Bab I : Pendahuluan
   Bab ini berisi latar belakang, rumusan masalah, batasan masalah, tujuan, manfaat, 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 ini.
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 Analisis
   Bab ini berisi uji coba dan evaluasi dari hasil implementasi yang telah dilakukan pada tahap implementasi.
6. Bab VI : Kesimpulan dan Saran
   Bab ini berisi kesimpulan dan saran yang didapat dari hasil uji coba yang telah dilakukan.

[*Halaman ini sengaja dikosongkan*]

BAB II
DASAR TEORI

Pada bab ini, akan dijelaskan dasar teori yang digunakan sebagai landasaan pengerjaan Tugas Akhir.

2.1 Deskripsi Permasalahan

Permasalahan yang dibahas pada Tugas Akhir ini adalah perhitungan nilai E pada Persamaan 1.2. Pada subbab ini akan dibahas mengenai parameter masukan dan keluaran permasalahan yang digunakan dalam pengujian luar maupun pengujian lokal.

2.1.1 Parameter Masukan

Berdasarkan deskripsi soal dari *URI Online Judge*, permasalahan yang diberikan memiliki parameter masukan dengan batasan sebagai berikut.

1. B, bilangan bulat yang berperan sebagai basis pemangkatan.
$$0 < B < 10^5 \tag{2.1}$$
2. N, bilangan bulat hasil pemangkatan B terhadap E dalam $\mathbb{Z}_M$.
$$0 < N < 10^5 \tag{2.2}$$
3. M, bilangan bulat prima yang berperan sebagai modulus.
$$2 < M < 10^9 \tag{2.3}$$

Pada pengujian lokal akan digunakan batasan nilai B dan N yang lebih besar dari soal.

$$0 < B < 10^9 \tag{2.4}$$
$$0 < N < 10^9 \tag{2.5}$$
$$0 < B,\ N < M \tag{2.6}$$

2.1.2 Keluaran Permasalahan

Keluaran dari permasalahan ini adalah bilangan bulat E yang memenuhi Persamaan 1.2. Jika tidak ada nilai E yang memenuhi persamaan tersebut, keluaran dari permasalahan yang diberikan adalah -1.

$$E = \begin{cases} E, & 0 \leq E \leq M-1 \\ -1, & E \notin [0,\ M-1] \end{cases} \tag{2.7}$$

2.2 Deskripsi Umum

Subbab ini akan memaparkan definisi, deskripsi, dan landasan yang digunakan dalam penyelesaian permasalahan yang diberikan.

2.2.1 Group

Didefinisikan himpunan bilangan G disebut sebagai *group* dengan operasi perkalian (*multiplicative group*) jika memenuhi kondisi-kondisi berikut [1].

1. Asosiatif, untuk $g_i,\ g_j \in G$ berlaku $g_i \cdot g_j = g_j \cdot g_i$.
2. Memiliki elemen identitas $e \in G$ sedemikian sehingga untuk setiap $g \in G$ berlaku $g \cdot e = g$
3. Untuk setiap elemen $g \in G$ terdapat invers $g^{-1} \in G$ sedemikian sehingga $g \cdot g^{-1} = e$.

Dari definisi tersebut, untuk bilangan bulat positif n pada himpunan bilangan $\mathbb{Z}_n$ bukan *group* karena tidak semua elemen dari $\mathbb{Z}_n$ memiliki elemen invers. Namun himpunan bilangan $\mathbb{Z}_n^*$ adalah *group* karena seluruh elemennya memiliki invers. Eksistensi invers untuk setiap elemen dari *group* $\mathbb{Z}_n^*$ akan dibahas pada subbab 2.2.6.

Dapat diamati bahwa himpunan bilangan yang dibentuk dari pemangkatan B modulo M berdasarkan Persamaan 1.2 adalah *multiplicative group*. Hal ini mengimplikasikan bahwa teorema atau

sifat pada *multiplicative group* dapat diaplikasikan pada permasalahan logaritma diskrit.

2.2.2 Subgroup

Sebuah himpunan bilangan H didefinisikan sebagai *subgroup* dari *group* G jika memenuhi dua kondisi berikut [3].

1. H memiliki operasi elemen yang sama dengan G.
2. H adalah subset dari G.

Eksistensi *subgroup* ini menjadi dasar teorema Lagrange yang akan dibahas pada subbab 2.2.4.

2.2.3 Order

Order dari *group* G adalah banyaknya elemen $g \in G$ dan dinotasikan dengan $|G|$. *Group* $\mathbb{Z}_p^*$ dengan p bilangan prima memiliki elemen $h = \{1, 2, \cdots, p-1\}$. Sehingga *order* dari dari $\mathbb{Z}_p^*$ adalah $p-1$ [1]. *Group order* ini digunakan pada algoritma Pohlig-Hellman sebagai modulus dari logaritma diskrit yang dibahas pada subbab 2.5.

2.2.4 Teorema Lagrange

Misal terdapat *group* G. Maka untuk setiap *subgroup* H dari G berlaku Persamaan 2.8 [3].

$$|H| \mid |G| \tag{2.8}$$

Teorema ini dapat dimanfaatkan untuk mengurangi banyaknya iterasi algoritma *Baby-Step Giant-Step* yang diimplementasikan dalam algoritma Pohlig-Hellman berdasarkan Persamaan 2.48.

2.2.5 Teorema Bëzout

Misal bilangan $p,\ q \in \mathbb{Z}$ memiliki nilai $gcd(p,\ q) = k$. Maka terdapat bilangan $x,\ y \in \mathbb{Z}$ sedemikian sehingga berlaku Persama-

an 2.9. Persamaan ini dinamakan *Extended Euclidean Representation* [4].

$$px + qy = k \tag{2.9}$$

Teorema ini digunakan sebagai dasar penghitungan *modular multiplicative inverse* yang dibahas pada subbab 2.2.6.

2.2.6 Modular Multiplicative Inverse

Didefinisikan bilangan $p,\ q\ \in \mathbb{Z}$ dengan $gcd(p,\ q) = 1$. Bilangan $x\ \in \mathbb{Z}$ disebut sebagai *modular multiplicative inverse* modulo q jika memenuhi Persamaan 2.10 [1].

$$px \equiv 1 \pmod{q} \tag{2.10}$$

Persamaan 2.9 ekivalen dengan Persamaan 2.10 setelah dilakukan operasi modulo oleh bilangan q.

Untuk setiap elemen $s \in \mathbb{Z}_n^*$ berlaku $gcd(s,\ n) = 1$. Hal ini mengimplikasikan bahwa setiap elemen dalam $\mathbb{Z}_n^*$ memiliki invers. *Modular multiplicative inverse* dapat dihitung dengan menggunakan algoritma Euclid yang memiliki kompleksitas $\mathcal{O}(log(n))$. *Modular multiplicative inverse* ini digunakan untuk menentukan nilai *modular inverse* B^{-1} pada persamaan *giant-step* (ruas kanan pada Persamaan 2.35).

2.2.7 Fermat Little Theorem

Misalkan p adalah bilangan prima. Maka untuk sebarang bilangan bulat a berlaku Persamaan 2.11 [4].

$$a^p \equiv a \pmod{p} \tag{2.11}$$

Untuk $gcd(a,\ p) = 1$, Persamaan 2.11 ekivalen dengan Persamaan 2.12.

$$a^{p-1} \equiv 1 \pmod{p} \tag{2.12}$$

$$a^{p-1} \equiv a^0 \pmod{p} \tag{2.13}$$

Persamaan 2.13 mengimplikasikan bahwa nilai E dari Persamaan 1.2 memiliki rentang nilai $[0,\ p-1)$ dan ekivalen dengan Persamaan 2.14 dan 2.15.

$$B^E \equiv B^{E \pmod{M-1}} \pmod{M} \tag{2.14}$$

$$E \equiv E \pmod{M-1} \tag{2.15}$$

Persamaan 2.15 dan *Chinese Remainder Theorem* (yang dijelaskan pada subbab 2.2.10) digunakan sebagai dasar penghitungan logaritma diskrit $E_i \pmod{p_i^{r_i}}$ pada Persamaan 2.41.

2.2.8 Pemangkatan Modular Repeated-Squaring

Pemangkatan modular secara naif memiliki kompleksitas $\mathcal{O}(n)$. Proses ini dapat dipercepat dengan memanfaatkan observasi sebagai berikut [5].

Didefinisikan bilangan bulat $g,\ e,$ dan m berturut-turut adalah basis, eksponen, dan modulus. Bilangan bulat h memenuhi Persamaan 2.16.

$$h \equiv g^e \pmod{m} \tag{2.16}$$

Bilangan e dapat dituliskan sebagai Persamaan 2.17.

$$e = 2^0 e_0 + 2^1 e_1 + \cdots + 2^{k-1} e_{k-1};\ e_i \in \{0, 1\} \tag{2.17}$$

Berdasarkan Persamaan 2.16 dan 2.17 didapatkan Persamaan 2.18 - 2.20.

$$h \equiv g^{2^0 e_0 + 2^1 e_1 + \cdots + 2^{k-1} e_{k-1}} \pmod{m} \tag{2.18}$$

$$h \equiv (g^{2^0})^{e_0} (g^{2^1})^{e_1} \cdots (g^{2^{k-1}})^{e_{k-1}} \pmod{m} \tag{2.19}$$

$$h \equiv w_0^{e_0} w_1^{e_1} \cdots w_{k-1}^{e_{k-1}} \pmod{m} \tag{2.20}$$

Dari Persamaan 2.20, nilai dari w_i dapat dituliskan pada Persamaan 2.21.

$$w_i \equiv {w_{i-1}}^2 \pmod{m} \tag{2.21}$$

Menggunakan Persamaan 2.20 dan 2.21 pemangkatan modular bisa dilakukan dengan kompleksitas $\mathcal{O}(log_2(n))$.

2.2.9 Multiplicative Order

Didefinisikan bilangan $p,\ q\ \in \mathbb{Z}$. Bilangan bulat positif terkecil t disebut sebagai *multiplicative order* dari p modulo q jika memenuhi Persamaan 2.22 [4].

$$p^t \equiv 1 \pmod{q} \tag{2.22}$$

Bilangan p disebut sebagai *primitive element* jika *multiplicative order* dari p modulo q memiliki nilai yang sama dengan *order* dari *multiplicative group* $\mathbb{Z}_q^*$.

Misalkan *group* yang terbentuk dari pemangkatan p adalah H. Berdasarkan penjelasan pada subbab 2.2.2, H adalah *subgroup* dari $\mathbb{Z}_q^*$. Menggunakan teorema Lagrange, maka *order* dari H membagi *order* dari $\mathbb{Z}_q^*$. Informasi ini dapat digunakan untuk menghitung *multiplicative order* p modulo q dengan melakukan iterasi faktor-faktor dari q yang memenuhi Persamaan 2.22. Nilai dari *multiplicative order* ini digunakan sebagai modulus dari hasil penghitungan *Chinese Remainder Theorem* dalam algoritma Pohlig-Hellman.

Didefinisikan faktorisasi prima dari q adalah $\Pi_{i=1}^{k} b_i^{r_i}$. Desain efisien penghitungan *multiplicative order* ini dituliskan pada subbab 3.2.9 dengan kompleksitas $\mathcal{O}(\sum_{i=1}^{k}(log_2(b_i^{r_i}) + r_i))$.

2.2.10 Chinese Remainder Theorem

Misal $n_1,\ n_2,\ \cdots,\ n_k,\ a_1,\ a_2,\ \cdots,\ a_k,$ dan N adalah bilangan bulat positif yang memenuhi Persamaan 2.23, 2.24, dan 2.25.

$$gcd(n_i,\ n_j) = 1 \,\forall\, i,\ j\ =\ 1,\ 2,\ \cdots,\ k;\ i\ \neq\ j \tag{2.23}$$

$$0\ \leq\ a_i\ <\ n_i \tag{2.24}$$

$$N = n_1 n_2 \cdots n_k \tag{2.25}$$

Terdapat bilangan bulat x sedemikian sehingga memenuhi Persamaan 2.26 [3].

$$x \equiv a_i \pmod{n_i} \; \forall \; i = 1, \; 2, \; \cdots, \; k. \tag{2.26}$$

Nilai dari x dapat ditentukan dengan melakukan observasi berikut. Misal diketahui $N_i = \frac{N}{n_i}$. Berdasarkan penjelasan pada subbab 2.2.5, terdapat bilangan bulat N_i' dan n_i' sedemikian sehingga berlaku Persamaan 2.27.

$$N_i \cdot N_i' + n_i \cdot n_i' = 1 \tag{2.27}$$

Karena $gcd\,(N_i, n_j) > 1 \; \forall \; i \neq j$, nilai x dapat dituliskan menjadi Persamaan 2.28.

$$x \equiv \sum_{i=1}^{k} \left(a_i \cdot N_i \cdot N_i'\right) \pmod{N} \tag{2.28}$$

Metode *Chinese Remainder Theorem* ini digunakan untuk menghitung logaritma diskrit E menggunakan nilai dari $E_i \pmod{p_i^{r_i}}$ pada algoritma Pohlig-Hellman berdasarkan Persamaan 2.41 dengan kompleksitas $\mathcal{O}(k \cdot log(n))$.

2.2.11 Miller-Rabin Primality Test

Pengecekan keprimaan Miller-Rabin digunakan untuk mempercepat penentuan keprimaan suatu bilangan sebagai alternatif dari pengecekan prima dengan metode naif. Menggunakan bilangan bulat ganjil positif p, ditentukan bilangan bulat positif s dan r sedemikian sehingga $p - 1 = 2^s r$. Metode ini menyatakan bahwa untuk bilangan prima p dan suatu bilangan bulat positif a dengan $gcd(a, \; p) = 1$ setidaknya salah satu dari Persamaan 2.29 dan 2.30 bernilai benar [1].

$$a^r \equiv 1 \pmod{p} \tag{2.29}$$

$$a^{2^j r} \equiv -1 \pmod{p}, \; 0 \leq j < s \tag{2.30}$$

Metode ini merupakan metode probabilistik yang memiliki persentase galat $(\frac{1}{4})^t$ [5] dengan t adalah banyaknya nilai a berbeda yang digunakan dalam pengecekan.

Terdapat varian deterministik dari metode ini untuk nilai p lebih kecil dari batas tertentu. Untuk nilai $p < 3215031751$ hanya diperlukan pengecekan untuk nilai $a = \{2,\ 3,\ 5,\ 7\}$ [6]. Desain algoritma deterministik ini dituliskan pada subbab 3.2.6 dan memiliki kompleksitas $\mathcal{O}(k \cdot (log_2(n) + a))$ dengan $k = |a|$.

Algoritma ini digunakan dalam proses pemfaktoran menggunakan algoritma faktorisasi Pollard-Rho sebagai bagian dari strategi penyelesaian algoritma Pohlig-Hellman.

2.2.12 Faktorisasi Pollard-Rho

Algoritma ini digunakan untuk menemukan faktor *non-trivial* dari suatu bilangan komposit. Didefinisikan bilangan komposit n, bilangan bulat positif acak c (dengan $c \neq \{0,\ 2\}$ [1]), dan bilangan bulat positif $x_{i+1} = f(x_i) = x_i^2 + c$. Faktor *non-trivial* bisa didapatkan dengan melakukan perhitungan pada Persamaan 2.31 [7].

$$d = gcd(x_i - x_{i+1}, n) \tag{2.31}$$

Persamaan 2.31 ekivalen dengan Persamaan 2.32.

$$d \mid n \tag{2.32}$$

Persamaan 2.32 menunjukkan bahwa d adalah faktor dari n. Jika nilai d adalah faktor *trivial* ($d = 1$ atau $d = n$), lakukan perhitungan kembali dengan nilai c yang berbeda. Untuk nilai d memenuhi $1 < d < n$, maka d merupakan faktor *non-trivial* dari n. Kompleksitas dari algoritma ini adalah $\mathcal{O}(n^{\frac{1}{4}})$ [1]. Algoritma digunakan dalam proses pemfaktoran *group order* algoritma Pohlig-Hellman.

Algoritma akan berjalan akan berjalan secara *indefinite* jika n adalah bilangan prima. Untuk menghindari kejadian ini, digunakan pengecekan bilangan prima *Miller-Rabin* sebelum dilakukan pemfaktoran oleh algoritma Pollard-Rho.

2.3 Strategi Penyelesaian dengan Solusi Naif

Solusi paling sederhana untuk menyelesaikan permasalahan ini adalah dengan melakukan iterasi nilai E sedemikian sehingga Persamaan 1.2 terpenuhi. Menggunakan solusi naif, permasalahan

Pseudocode 2.1 Solusi Penyelesaian Logaritma Diskrit Naif

```
Input: B, N, M
Output: E
1: for i ← 0 to (M − 1) do
2:     if B^i (mod M) = N then
3:         return i
4: return (−1)
```

logaritma diskrit dapat diselesaikan dengan kompleksitas $\mathcal{O}(M)$. *Multiplicative Order* t dari B modulo M dapat dimanfaatkan untuk memperkecil daerah pencarian dari $[0,\ M-1]$ menjadi $[0,\ t]$. Namun karena nilai dari t berada pada $[0,\ M-1]$, untuk *worst-case scenario* akan memiliki kompleksitas yang sama dengan solusi naif.

2.4 Strategi Penyelesaian dengan Baby-Step Giant-Step

Berikut penjelasan dari penyelesaian logaritma diskrit menggunakan algoritma *Baby-Step Giant-Step*. Seluruh operasi pemangkatan modular dilakukan menggunakan metode sesuai dengan pembahasan pada subbab 2.2.8.

Berdasarkan Persamaan 1.2, nilai dari bilangan E dapat dituliskan menjadi Persamaan 2.33.

$$E = k \cdot q + r \tag{2.33}$$

Sehingga Persamaan 1.2 dapat dituliskan menjadi Persamaan 2.34.

$$B^{k \cdot q + r} \equiv N \pmod{M} \tag{2.34}$$

Untuk B dengan $gcd(B,\ M) = 1$, terdapat *modular multiplicative inverse* B^{-1} dari B modulo M. Persamaan 2.34 dapat dituliskan menjadi Persamaan 2.35

$$B^r \equiv (B^{-1})^{k \cdot q} N \pmod{M} \tag{2.35}$$

Menggunakan nilai $k = \lceil \sqrt{M} \rceil$ untuk meminimalkan iterasi pencarian r dan q pada Persamaan 2.35 [5], daerah pencarian nilai r dan q dapat dituliskan pada Persamaan 2.36.

$$0 \ \leq \ r, q \ \leq \ \lceil \sqrt{M} \rceil \tag{2.36}$$

Kompleksitas algoritma ini dapat dituliskan menjadi $\mathcal{O}(\sqrt{M})$.

2.5 Strategi Penyelesaian dengan Pohlig-Hellman

Berikut penjelasan dari penyelesaian logaritma diskrit menggunakan algoritma Pohlig-Hellman. Seluruh operasi pemangkatan modular dilakukan sesuai dengan pembahasan pada subbab 2.2.8.

Berdasarkan Persamaan 2.12, bilangan M dari Persamaan 1.2 memiliki *order* $M - 1$. Misalkan $p_1,\ p_2,\ \cdots,\ p_k$ adalah faktor prima dari $M - 1$ sedemikian sehingga berlaku Persamaan 2.37.

$$M - 1 = \prod_{i=1}^{k} p_i^{r_i} \tag{2.37}$$

Menggunakan penjelasan pada subbab 2.2.10 dan subbab 2.2.4, nilai dari E dapat dihitung dengan menemukan nilai E_i yang didapatkan dari *subgroup* dengan *order* $p_i^{r_i}$ dan memenuhi Persamaan 2.38-2.40.

$$B_i^{Ei} \equiv N_i \pmod{M} \tag{2.38}$$

$$B_i \equiv B^{(M-1)/p_i^{r_i}} \pmod{M} \tag{2.39}$$

$$N_i \equiv N^{(M-1)/p_i^{r_i}} \pmod{M} \tag{2.40}$$

Bilangan E dapat dituliskan dalam Persamaan 2.41.

$$E \equiv E_i \pmod{p_i^{r_i}} = \sum_{j=0}^{r_j-1} b_j \cdot p_i^j \tag{2.41}$$

Perhatikan bahwa berdasarkan penjelasan subbab 2.2.4, Persamaan 2.42 memiliki *order* p_i.

$$B' \equiv B^{(M-1)/p_i} \pmod{M} \tag{2.42}$$

Dengan melakukan pemangkatan Persamaan 1.2 dengan $(M-1)/p_i$ didapatkan Persamaan 2.43 - 2.45.

$$B^{E\cdot((M-1)/p_i)} \equiv N^{(M-1)/p_i} \pmod{M} \tag{2.43}$$

$$(B^{(M-1)/p_i})^E \equiv N^{(M-1)/p_i} \pmod{M} \tag{2.44}$$

$$(B')^E \equiv N^{(M-1)/p_i} \pmod{M} \tag{2.45}$$

Karena Persamaan 2.45 memiliki *order* p_i, Persamaan 2.45 ekivalen dengan Persamaan 2.46-2.48.

$$(B')^E \equiv N^{(M-1)/p_i} \pmod{M} \tag{2.46}$$

$$(B')^{E \pmod{p_i}} \equiv N^{(M-1)/p_i} \pmod{M} \tag{2.47}$$

$$(B')^{b_0} \equiv N^{(M-1)/p_i} \pmod{M} \tag{2.48}$$

Nilai dari b_0 bisa didapatkan menggunakan algoritma *Baby-Step Giant-Step*. Karena Persamaan 2.48 memiliki *order* p_i, maka daerah pencarian pada Persamaan 2.36 menjadi Persamaan 2.49.

$$0 \le r, q \le \lceil \sqrt{p_i} \rceil \tag{2.49}$$

Selanjutnya didefinisikan bilangan z yang memenuhi Persamaan 2.50.

$$z = N \cdot B^{-b_0} \equiv B^{E-b_0} \pmod{M} \tag{2.50}$$

Pemangkatan Persamaan 2.50 terhadap $(M-1)/p_i^2$ menghasilkan Persamaan 2.51 - 2.53.

$$z^{(M-1)/p_i^2} \equiv B^{(E-b_0)(M-1)/p_i^2} \pmod{M} \tag{2.51}$$

$$z^{(M-1)/p_i^2} \equiv (B^{(M-1)/p_i})^{(E-b_0)/p_i} \pmod{M} \tag{2.52}$$

$$z^{(M-1)/p_i^2} \equiv (B')^{(E-b_0)/p_i} \pmod{M} \tag{2.53}$$

Karena Persamaan 2.53 memiliki *order* p_i, Persamaan 2.53 ekivalen dengan Persamaan 2.54 dan dapat diselesaikan menggunakan cara yang sama dengan Persamaan 2.48.

$$z^{(M-1)/p_i^2} \equiv (B')^{b_1} \pmod{M} \tag{2.54}$$

Nilai dari E_i dapat dihitung sesuai dengan Persamaan 2.41 dan nilai dari E dapat dihitung dengan menggunakan nilai E_i berdasarkan penjelasan subbab 2.2.10 untuk setiap nilai $i = 1,\ 2,\ \cdots,\ k$. Kompleksitas algoritma ini dapat dituliskan menjadi $\mathcal{O}(\sum_{i=1}^{k} e_i \cdot (log_2 M + \sqrt{p_i}))$.

Namun diperlukan solusi alternatif untuk bilangan B yang memiliki *Multiplicative Order* modulo M sedemikian sehingga $B' \equiv 1 \pmod{M}$. Hal ini dapat diamati dari contoh kasus uji dengan $B = 229,\ N = 45,$ dan $M = 241 = 2^4 \cdot 3 \cdot 5 + 1$ saat menghitung b_1 untuk $p_i = 2$.

$$B' = 1,\ b_0 = 0 \tag{2.55}$$

$$z = 45 \cdot 229^{-1 \cdot 0} \equiv 45 \pmod{241} \tag{2.56}$$

$$1^{(E-0)/2} \equiv 45^{240/2^2} \pmod{241} \tag{2.57}$$

$$1^{(E-0)/2} \equiv 240 \pmod{241} \tag{2.58}$$

$$E_i = -1 \tag{2.59}$$

Dari Persamaan 2.38, terdapat solusi $E_i = 7$ yang kontradiktif dengan hasil pada Persamaan 2.59. Solusi alternatif yang digunakan

penulis untuk mengatasi masalah tersebut adalah dengan menggunakan algoritma *Baby-Step Giant-Step* untuk Persamaan 2.38 ketika nilai dari $B' = 1 \pmod{M}$. Algoritma ini tidak memerlukan implementasi solusi alternatif penulis ketika B adalah bilangan primitif (*multiplicative order* dari B modulo M adalah $M - 1$) [8].

[*Halaman ini sengaja dikosongkan*]

BAB III
DESAIN

Pada bab ini akan dijelaskan desain program yang akan digunakan untuk menyelesaikan permasalahan yang diberikan. Penjelasan dibagi menjadi tiga bagian, yaitu Desain Umum Sistem, Desain Program Utama, dan Desain Program Pembuat Kasus Uji.

3.1 Desain Umum Sistem

Sistem akan menerima masukan nilai B, N dan M yang memenuhi batasan pada subbab 2.1.1. Setelah menerima masukan, program akan melakukan penghitungan nilai E berdasarkan Persamaan 1.2 menggunakan algoritma *Baby-Step Giant-Step* dan Pohlig-Hellman yang telah dibahas dalam subbab 2.4 dan 2.5.

3.2 Desain Program Utama

Pada subbab ini akan dijelaskan desain fungsi dalam program yang digunakan untuk menyelesaikan permasalahan.

3.2.1 Desain Fungsi Main

Fungsi ini digunakan untuk menjalankan tiga perintah, yaitu membaca parameter masukan B, N, dan M, melakukan penghitungan logaritma diskrit E, dan menampilkan nilai E. Desain dari fungsi ini dituliskan pada *Pseudocode* 3.1.

Pseudocode 3.1 Fungsi MAIN

```
1: while not End of File do
2:     B, N, M ← INPUT()
3:     E ← SOLVE(B, N, M)
4:     PRINT(E)
```

3.2.2 Desain Fungsi Pemangkatan Modular

Pemangkatan modular pada penjelasan subbab 2.2.8 dituliskan dalam fungsi MODEXPONENTIATION pada *Pseudocode* 3.2. Fungsi ini menerima masukan bilangan bulat $base$ sebagai basis pemangkatan, bilangan bulat exp sebagai eksponen dari basis, dan bilangan bulat mod sebagai modulus pemangkatan modular. Keluaran dari fungsi ini adalah hasil pemangkatan modular yang memenuhi Persamaan 2.16.

Pseudocode 3.2 Fungsi MODEXPONENTATION

Input: $base$, exp, mod
Output: ret
1: $r \leftarrow 1$
2: $pow \leftarrow 0$
3: **while** $exp > 1$ **do**
4: **if** $exp \pmod 2 = 1$ **then**
5: $ret \leftarrow ret \cdot base \pmod{mod}$
6: $base \leftarrow base \cdot base \pmod{mod}$
7: $pow \leftarrow pow + 1$
8: $exp \leftarrow \lfloor \frac{exp}{2} \rfloor$
9: **return** ret

3.2.3 Desain Fungsi Penghitungan Eksponen

Penghitungan eksponen digunakan dalam proses faktorisasi prima dari bilangan $M - 1$ berdasarkan Persamaan 2.37. Fungsi ini menerima masukan bilangan bulat $target$ yang memiliki faktor d dan bilangan prima d yang membagi $target$. Keluaran dari fungsi ini adalah exp yang merupakan bilangan bulat terbesar sedemikian sehingga d^{exp} membagi $target$. res merupakan hasil pembagian $target$ oleh d^{exp}. Desain fungsi ini dituliskan dalam fungsi EXTRACTEXPONENT pada *Pseudocode* 3.3.

Pseudocode 3.3 Fungsi EXTRACTEXPONENT

Input: $target,\ d$
Output: $exp,\ res$
1: $res \leftarrow target$
2: $exp \leftarrow 0$
3: **while** $res \pmod{d} = 0$ **do**
4: $\quad exp \leftarrow exp + 1$
5: $\quad res \leftarrow \lfloor \frac{res}{d} \rfloor$
6: **return** $res,\ exp$

3.2.4 Desain Fungsi Extended GCD

Fungsi ini digunakan untuk menemukan faktor persekutuan terkecil dari dua parameter masukan p dan q. Keluaran dari fungsi ini adalah gcd yang merupakan faktor persekutuan terkecil p dan q. Bilangan x dan y adalah variabel penyelesaian persamaan Bëzout berdasarkan penjelasan pada subbab 2.2.5. Desain fungsi ini dituliskan dalam fungsi EXTENDEDGCD pada *Pseudocode* 3.4.

Pseudocode 3.4 Fungsi EXTENDEDGCD

Input: $p,\ q$
Output: $gcd,\ x,\ y$
1: $r,\ old_r \leftarrow q,\ p$
2: $s,\ old_s \leftarrow 0,\ 1$
3: $t,\ old_t \leftarrow 1,\ 0$
4: **while** $r > 0$ **do**
5: $\quad quotient \leftarrow \lfloor \frac{old_r}{r} \rfloor$
6: $\quad r,\ old_r \leftarrow (old_r - quotient \cdot r),\ r$
7: $\quad s,\ old_s \leftarrow (old_s - quotient \cdot s),\ s$
8: $\quad t,\ old_t \leftarrow (old_t - quotient \cdot t),\ t$
9: $gcd,\ x,\ y \leftarrow old_r,\ old_s,\ old_t$
10: **return** $gcd,\ x,\ y$

3.2.5 Desain Fungsi Faktorisasi Pollard-Rho

Fungsi ini digunakan untuk menemukan faktor *non-trivial* dari parameter masukan $target$ berdasarkan penjelasan pada subbab 2.2.12. Fungsi akan tetap berjalan selama faktor d yang ditemukan adalah faktor *trivial* dan akan digunakan bilangan c yang berbeda jika nilai dari d sama dengan $target$. Desain fungsi ini dituliskan pada *Pseudocode* 3.5.

Pseudocode 3.5 Fungsi POLLARDRHOFACTOR

Input: $target$
Output: d

1: $d \leftarrow 1$
2: **while** $d = 1$ **or** $d = target$ **do**
3: $c \leftarrow$ RANDOMINTEGER$(0,\ target - 1\ )$
4: $x_i \leftarrow$ RANDOMINTEGER$(0,\ target - 2\ ) + 2$
5: $x_m \leftarrow xi$
6: $x_i \leftarrow \left(x_i^2 + c\right) \pmod{target}$
7: $x_m \leftarrow \left((x_m^2 + c)^2 + c\right) \pmod{target}$
8: $d, , \leftarrow$ EXTENDEDGCD$(x_i - x_m,\ target)$
9: **return** d

3.2.6 Desain Fungsi Deterministic Miller-Rabin

Berdasarkan penjelasan pada subbab 2.2.11, fungsi ini digunakan untuk menentukan keprimaan suatu bilangan masukan $target$. Keluaran dari fungsi ini adalah **True** jika $target$ adalah bilangan prima dan **False** jika $target$ adalah bilangan komposit. Desain fungsi ini dituliskan dalam fungsi DETERMINISTICMILLERRABIN pada *Pseudocode* 3.6. Fungsi MIN pada *pseudocode* digunakan untuk menentukan nilai minimum dari parameter masukan.

Pseudocode 3.6 Fungsi DETERMINISTICMILLERRABIN

Input: $target$
Output: $isPrime$

1: **if** $target = 2$ **then**
2: $\quad isPrime \leftarrow$ **True**
3: **else if** $target \pmod 2 = 0$ **then**
4: $\quad isPrime \leftarrow$ **False**
5: **else**
6: $\quad arr \leftarrow \{2,\ 3,\ 5,\ 7\}$
7: $\quad d \leftarrow target - 1$
8: $\quad s \leftarrow 0$
9: $\quad$ **while** $d \pmod 2 = 0$ **do**
10: $\quad\quad d \leftarrow \lfloor \frac{d}{2} \rfloor$
11: $\quad\quad s \leftarrow s + 1$
12: $\quad$ **for** $i \leftarrow 0$ **to** $arr.size - 1$ **do**
13: $\quad\quad a \leftarrow$ MIN$(target - 2,\ arr\,[i])$
14: $\quad\quad now \leftarrow a^d \pmod{target}$
15: $\quad\quad$ **if** $now = 1$ **or** $now = target - 1$ **then**
16: $\quad\quad\quad$ continue
17: $\quad\quad$ **for** $j \leftarrow 0$ **to** $s - 1$ **do**
18: $\quad\quad\quad now \leftarrow now^2 \pmod{target}$
19: $\quad\quad\quad$ **if** $now = target - 1$ **then**
20: $\quad\quad\quad\quad$ break
21: $\quad\quad$ **if** j = s **then**
22: $\quad\quad\quad$ **return False**
23: $\quad$ **return True**

3.2.7 Desain Fungsi Faktorisasi Prima

Fungsi ini digunakan untuk melakukan faktorisasi prima dari parameter masukan $target$. Parameter masukan $primes$ digunakan untuk menyimpan faktorisasi prima yang telah diketahui. Fungsi POLLARDRHOFACTOR digunakan untuk menemukan faktor *non-*

trivial dari $target$ dan fungsi DETERMINISTICMILLERRABIN digunakan untuk menentukan keprimaan bilangan hasil dari pemfaktorkan POLLARDRHOFACTOR. Desain fungsi ini dituliskan pada fungsi FACTORIZE dalam *Pseudocode* 3.7.

Pseudocode 3.7 Fungsi FACTORIZE

Input: $target$, $primes$
Output: $primes$

```
1:  while target is not prime do
2:      f ← POLLARDRHOFACTOR(target)
3:      if f is not prime then
4:          primes ← FACTORIZE(f, primes)
5:          target ← ⌊target / f⌋
6:      else
7:          target, exp ← EXTRACTEXPONENT(target, f)
8:          primes[f] ← primes[f] + exp
9:      if target ≤ 1 then
10:         return primes
11: primes[f] ← primes[f] + 1
12: return primes
```

3.2.8 Desain Fungsi Chinese Remainder

Berdasarkan penjelasan pada subbab 2.2.10, fungsi ini digunakan untuk menentukan nilai E dengan mengetahui seluruh nilai E_i dan $p_i^{r_i}$. Parameter masukan fungsi ini adalah res yang berisi bilangan E_i, $mods$ yang berisi bilangan $p_i^{r_i}$, dan n yang merupakan hasil perkalian seluruh nilai $p_i^{r_i}$.

Fungsi SIZE digunakan untuk menghitung banyaknya elemen dalam res. Desain fungsi ini dituliskan dalam *Pseudocode* 3.8.

Pseudocode 3.8 Fungsi CHINESEREMAINDER

Input: res, $mods$, n
Output: E
1: $E \leftarrow 0$
2: **for** $i \leftarrow 0$ **to** SIZE(res) **do**
3: , inv, $\leftarrow$ EXTENDEDGCD($\lfloor \frac{n}{mods[i]} \rfloor$, $mods\,[i]$)
4: $E \leftarrow \left(E + res\,[i] \cdot inv \cdot \lfloor \frac{n}{mods[i]} \rfloor\right) \pmod{n}$
5: **return** E

3.2.9 Desain Fungsi Element Order

Fungsi ini digunakan untuk menentukan *order* elemen t dari $base$ modulo mod. Variabel $primes$ digunakan untuk melakukan iterasi faktor-faktor dari *group order* yang memiliki nilai $mod - 1$ berdasarkan penjelasan pada subbab 2.2.9. Proses ini dituliskan dalam fungsi ELEMENTORDER pada *Pseudocode* 3.9. p_b dan p_e dalam *pseudocode* berturut-turut adalah bilangan basis p_i dan eksponen r_i dari persamaan $p_i^{r_i}$.

Pseudocode 3.9 Fungsi ELEMENTORDER

Input: $base$, mod, $primes$
Output: ord
1: $ord \leftarrow mod - 1$
2: **for** p **in** $primes$ **do**
3: $pow \leftarrow p_b^{p_e} \pmod{mod}$
4: $ord \leftarrow \lfloor \frac{ord}{pow} \rfloor$
5: $a \leftarrow base^{ord} \pmod{mod}$
6: **while** $a \neq 1$ **do**
7: $a \leftarrow a^{p_b} \pmod{mod}$
8: $ord \leftarrow ord \cdot p_b$
9: **return** ord

3.2.10 Desain Fungsi Penyelesaian Baby-Step Giant-Step

Desain fungsi ini dibuat berdasarkan penjelasan pada subbab 2.4. Digunakan struktur data *hash map* pada variabel $baby$ untuk menyimpan nilai penghitungan *baby-step*. Penghitungan *giant-step* dilakukan jika nilai E tidak ditemukan pada penghitungan *baby-step*. Fungsi ini dituliskan pada *Pseudocode* 3.10.

Pseudocode 3.10 Solusi Logaritma Diskrit *Baby-Step Giant-Step*

Input: $B,\ N,\ M$
Output: E

1: $iter \leftarrow 1$
2: $limit \leftarrow \lceil \sqrt{M} \rceil$
3: $baby \leftarrow [\,]$
4: **for** $i \leftarrow 0$ to $limit$ **do**
5: $\quad baby\,[iter] \leftarrow i$
6: $\quad$ **if** $iter = N$ **then**
7: $\quad\quad$ **return** i
8: $\quad iter \leftarrow (iter * B) \pmod M$
9: $gcd,\ x, \leftarrow ExtendedGCD(B,\ M)$
10: $inv \leftarrow x^{limit} \pmod M$
11: **for** $i \leftarrow 0$ to $\lceil \frac{M}{limit} \rceil$ **do**
12: $\quad$ **if** $baby[\ N\ ] \neq$ **NULL then**
13: $\quad\quad$ **return** $(limit \cdot i + baby\,[N])$
14: $\quad N \leftarrow (N \cdot inv) \pmod M$
15: **return** -1

3.2.11 Desain Fungsi Penyelesaian Pohlig-Hellman

Desain fungsi ini dibuat berdasarkan penjelasan pada subbab 2.5. Digunakan struktur data *hash map* pada variabel $primes$ untuk menyimpan hasil pemfaktoran prima *group order* $M-1$ dengan indeks *hash map* adalah faktor prima dan nilai *hash map* adalah eks-

ponen dari bilangan prima indeks. Hasil penghitungan E dimodulokan dengan hasil dari fungsi $ElementOrder$ pada *Pseudocode* 3.9.

Pseudocode 3.11 Solusi Logaritma Diskrit Pohlig-Hellman (bagian 1)

Input: $B,\ N,\ M$
Output: E

$i \leftarrow 0,\ j \leftarrow 0$
$primes \leftarrow$ FACTORIZE($M-1,\ primes$)
$gcd,\ inv, \leftarrow$ EXTENDEDGCD($B,\ M$)
for f **in** $primes$ **do**
    $iter_{dislog} \leftarrow 0$
    $v \leftarrow f_b^{f_{exp}} \pmod M$
    $mods.pushback\,(v)$
    $iter_e \leftarrow \frac{M-1}{f_b}$
    $iter_b \leftarrow B^{iter_e} \pmod M$
    if $iter_b = 1$ **then**
        $iter_b \leftarrow B^{\frac{M-1}{mods[i]}} \pmod M$
        $iter_n \leftarrow N^{\frac{M-1}{mods[i]}} \pmod M$
        $iter_{dislog} \leftarrow$ BABYGIANTSTEP($iter_b,\ iter_n,\ M$)
        if $iter_{dislog} = -1$ **then**
            return -1
        $dislog.pushback\,(iter_{dislog})$
        $i \leftarrow i + 1$
        continue

Pseudocode 3.12 Solusi Logaritma Diskrit Pohlig-Hellman (bagian 2)

```
19:     for j ← 0 to (f_exp − 1) do
20:         iter_n ← (N · inv^{iter_dislog}) (mod M)
21:         iter_n ← iter_n^{iter_e} (mod M)
22:         temp ← BABYGIANTSTEP(iter_b, iter_n, M)
23:         if temp = −1 then
24:             return −1
25:         iter_dislog ← iter_dislog + f_b^j · temp
26:         iter_dislog ← iter_dislog (mod mods[i])
27:         iter_e ← iter_e / f_b
28:     dislog.pushback(iter_dislog)
29:     i ← i + 1
30: e_ord ← ELEMENTORDER(B, M, primes)
31: ret ← CHINESEREMAINDER(dislog, mods, M − 1)
32: return ret (mod e_ord)
```

3.3 Desain Program Pembuat Kasus Uji

Program pembuat kasus uji didesain untuk membuat kasus uji yang digunakan sebagai parameter masukan program utama pada pengujian lokal. Kasus uji yang dibuat terdiri dari bilangan B, N, dan M sesuai dengan penjelasan pada subbab 2.1.1.

Pseudocode 3.13 Desain Program Pembuat Kasus Uji

```
Input: −
1: B, N, M ← 0, 0, 0
2: while M is not prime do
3:     M ← RANDOMINTEGER(3, 10^9 − 1)
4: B ← RANDOMINTEGER(1, M − 1)
5: N ← RANDOMINTEGER(1, M − 1)
6: return B, N, M
```

BAB IV
IMPLEMENTASI

4.1 Lingkungan implementasi

Lingkungan implementasi dan pengembangan yang dilakukan adalah sebagai berikut.

1. Perangkat Keras
   (a) Prosesor Intel® Core™ i5-3317U CPU @ 1.70GHz (4 CPUs)
   (b) *Random Access Memory* 8092MB
2. Perangkat Lunak
   (a) Sistem Operasi Linux Mint 19.1 Tessa 64-bit
   (b) *Text Editor* Visual Studio Code
   (c) Bahasa Pemrograman C++
   (d) MinGW 64-bit

4.2 Implementasi Program Utama

Subbab ini menjelaskan implementasi program utama. Program ini digunakan untuk menyelesaikan permasalahan *URI Online Judge* 2711 *Unlocking A Cell Phone*. Hal yang perlu diperhatikan pada tahap implementasi ini adalah Bahasa Pemrograman C++ tidak mendukung *multiple return values*. Pada beberapa kode sumber akan digunakan variabel *pointer* atau *reference* untuk mengatasi permasalahan ini.

4.2.1 Penggunaan Library

Program ini menggunakan *library* seperti yang ditunjukkan pada Kode Sumber 4.1. Untuk mempercepat proses pembacaan masukan, digunakan fungsi GETNUM yang dituliskan pada Kode Sum-

ber 4.2.

```
#include<unordered_map>
#include<vector>
#include<cstring>
#include<cmath>
using namespace std;
```

Kode Sumber 4.1 Daftar *library*

```
template <typename T>
inline T getNum(){
  T res=0;char c;
  bool minus=false;
  while(1){
    c=getchar_unlocked();
    if(c==EOF) return -1;
    if(c==' '||c=='\n') continue;
    if(c=='-')minus=true;
    else break;
  }
  res=c-'0';
  while(1){
    c=getchar_unlocked();
    if(c==EOF) return -1;
    if(c>='0'&& c<='9') res=10*res+c-'0';
    else break;
  }
  return (minus)? -res : res;
}
```

Kode Sumber 4.2 Fungsi GETNUM

Fungsi yang diimplementasikan dari desain pada bab 3 dikelompokkan ke dalam empat *class*, yaitu ALGORITHMS, BABYGIANTSTEP, POHLIGHELLMAN, dan DISLOGSOLVER.

4.2.2 Class Algorithms

Class ini digunakan untuk menyimpan fungsi-fungsi pendukung untuk menyelesaikan permasalahan logaritma diskrit. Berikut

penjelasan dari masing-masing fungsi tersebut.

```
 1 class Algorithms{
 2 public:
 3   int64_t modExponentiation(int64_t b, int64_t e,
         int64_t mod);
 4   int64_t extractExponent(int64_t * target,
         int64_t divisor);
 5   bool deterministicMillerRabin(int64_t target);
 6   void factorize(int64_t target,
         unordered_map<int64_t, int64_t> &
         primefactorization);
 7   int64_t extendedGCD(int64_t a, int64_t b,
         int64_t * p, int64_t * q);
 8   int64_t chineseRemainder(vector<int64_t> &
         residues, vector<int64_t> & mods, int64_t
         modulo);
 9   int64_t getElementOrder(int64_t base, int64_t
         modulo, unordered_map<int64_t, int64_t> &
         primefactorization);
10 };
```

Kode Sumber 4.3 *Class* ALGORITHMS

4.2.2.1 Fungsi Pemangkatan Modular

Fungsi ini digunakan untuk menentukan hasil pemangkatan b terhadap eksponen e dan dimodulo mod berdasarkan penjelasan pada subbab 3.2.2. Implementasi fungsi ini dituliskan pada Kode Sumber 4.4.

```
 1 int64_t modExponentiation(int64_t b, int64_t e,
       int64_t mod){
 2   int64_t r = 1;
 3   while(e > 0){
 4     if((e & 1)){
 5       r = (r * b) % mod;
 6     }
 7     e >>= 1;
 8     b = (b * b) % mod;
 9   }
10   return r;
11 }
```

Kode Sumber 4.4 Fungsi MODEXPONENTIATION

4.2.2.2 Fungsi Penghitungan Eksponen

Fungsi ini digunakan untuk menentukan eksponen dari suatu bilangan pembagi *divisor* terhadap bilangan bulat *target* berdasarkan penjelasan pada subbab 3.2.3. Keluaran dari fungsi ini adalah eksponen dari *divisor* dan hasil pembagian *target* oleh *divisor*.

```
 1 int64_t extractExponent(int64_t * target, int64_t
       divisor){
 2   int64_t ret = 0;
 3   while( ( (*target) % divisor == 0 ) &&
         ((*target) /= divisor) )
 4   {
 5     ret++;
 6   }
 7   return ret;
 8 }
```

Kode Sumber 4.5 Fungsi EXTRACTEXPONENT

4.2.2.3 Fungsi Deterministic Miller-Rabin

Fungsi ini digunakan untuk menentukan keprimaan dari bilangan $target$ berdasarkan penjelasan pada subbab 3.2.6. Keluaran dari fungsi ini adalah **true** jika $target$ adalah bilangan prima dan **false** jika $target$ bukan bilangan prima.

```
 1 bool deterministicMillerRabin(int64_t target){
 2   if(target == 2)
 3     return true;
 4   else if((target & 1) == 0)
 5     return false;
 6   vector<int64_t> arr = { 2, 3, 5, 7 };
 7   int64_t d = target - 1;
 8   int64_t s = 0;
 9   int64_t a, now;
10   while((d & 1) == 0) {
11     d >>= 1, s++;
12   }
13   int i, j;
14   for(i = 0; i < arr.size(); i++){
15     a = min(target - 2, arr[i]);
16     now = this->modExponentiation(a, d, target);
17     if(now == 1 || now == target-1)
18       continue;
19     for(j = 0; j < s; j++){
20       now = (now * now) % target;
21       if(now == target-1)
22         break;
23     }
24     if(j == s)
25       return false;
26   }
27   return true;
28 }
```

Kode Sumber 4.6 Fungsi DETERMINISTICMILLERRABIN

4.2.2.4 Fungsi Faktorisasi Pollard-Rho

Fungsi ini digunakan untuk menemukan faktor *non-trivial* dari bilangan $target$ berdasarkan penjelasan pada subbab 3.2.5. Keluaran dari fungsi ini adalah bilangan s yang merupakan faktor *non-trivial* dari $target$.

```
 1 int64_t pollardRhoFactorization(int64_t target){
 2   if((target & (target - 1)) == 0)
 3     return 2;
 4   int64_t c, xi, xm;
 5   int64_t s = 1;
 6   while(s == 1 || s == target){
 7     c = rand()%(target-1);
 8     xi = rand()%(target-2)+2;
 9     xm = xi;
10     xi = ((xi * xi)%target + c)%target;
11     xm = ((xm * xm)%target + c)%target;
12     xm = ((xm * xm)%target + c)%target;
13     s = this->extendedGCD(xi-xm, target, NULL,
           NULL);
14   }
15   return s;
16 }
```

Kode Sumber 4.7 Fungsi POLLARDRHOFACTORIZATION

4.2.2.5 Fungsi Faktorisasi Prima

Fungsi ini digunakan untuk melakukan faktorisasi prima bilangan masukan $target$ dan disimpan dalam $primefactorization$. Untuk mempermudah pembuatan fungsi, parameter masukan $primefactorization$ juga digunakan sebagai variabel keluaran.

```
 1 void factorize(int64_t target,
       unordered_map<int64_t, int64_t> &
       primefactorization){
 2   int64_t ret;
 3   while(!this->deterministicMillerRabin(target)){
 4     ret = this->pollardRhoFactorization(target);
 5     if(!this->deterministicMillerRabin(ret)){
 6       this->factorize(ret, primefactorization);
 7       target /= ret;
 8     } else {
 9       primefactorization[ret] +=
             this->extractExponent(&target, ret);
10     }
11     if(target <= 1) return;
12   }
13   primefactorization[target]++;
14   return;
15 }
```

Kode Sumber 4.8 Fungsi FACTORIZE

4.2.2.6 Fungsi Extended GCD

Fungsi ini digunakan untuk mencari faktor persekutuan terbesar sekaligus mencari nilai dari konstanta pada identitas Bëzout atau *modular multiplicative inverse* berdasarkan penjelasan pada subbab 3.2.4. Keluaran dari fungsi ini adalah $oldr$ yang merupakan faktor persekutuan terbesar dari bilangan a dan b serta variabel *pointer* p dan q yang merupakan konstanta Bëzout dari a dan b.

```
 1 int64_t extendedGCD(int64_t a, int64_t b, int64_t
       * p, int64_t * q){
 2   int64_t tmp, quotient, s=0, olds=1, t=1,
         oldt=0, r=b, oldr=a;
 3   while(r > 0){
 4     quotient = oldr/r;
```

Kode Sumber 4.9a Fungsi EXTENDEDGCD (bagian 1)

```
 5      tmp = r;
 6      r = oldr - quotient * tmp;
 7      oldr = tmp;
 8
 9      tmp = s;
10      s = olds - quotient * tmp;
11      olds = tmp;
12
13      tmp = t;
14      t = oldt - quotient * tmp;
15      oldt = tmp;
16    }
17
18    if(p != NULL){
19      *p = olds;
20    }
21    if(q != NULL){
22      *q = oldt;
23    }
24
25    return oldr;
26 }
```

Kode Sumber 4.9b Fungsi EXTENDEDGCD (bagian 2)

4.2.2.7 Fungsi Chinese Remainder

Fungsi ini digunakan untuk mencari nilai logaritma diskrit E dari nilai E_i berdasarkan penjelasan pada subbab 3.2.8. Fungsi ini memerluan parameter nilai E_i yang disimpan pada variabel $residues$, modulo untuk masing-masing nilai E_i yang disimpan pada variabel $mods$, dan $modulo$ yang merupakan hasil perkalian nilai dalam variabel $mods$. Keluaran dari fungsi ini adalah bilangan bulat E.

```
1  int64_t chineseRemainder(vector<int64_t> &
       residues, vector<int64_t> & mods, int64_t
       modulo){
2    int64_t ret = 0;
3    int64_t tmp;
4    int64_t inv;
5    int64_t gcd;
6    for(int i = 0; i < residues.size(); i++){
7      tmp = modulo / mods[i];
8      gcd = this->extendedGCD(tmp, mods[i], &inv,
           NULL);
9      while(inv < 0) inv += (mods[i] / gcd);
10     ret = (ret + residues[i] * inv * tmp) %
           modulo;
11   }
12   return ret;
13 }
```

Kode Sumber 4.10 Fungsi CHINESEREMAINDER

4.2.2.8 Fungsi Element Order

Fungsi ini digunakan untuk menentukan *order* elemen dari $base$ modulo $modulo$ berdasarkan penjelasan pada subbab 3.2.9 dengan memanfaatkan hasil faktorisasi prima dari $modulo - 1$ yang telah dihitung pada saat pemanggilan fungsi FACTORIZE. Keluaran dari fungsi ini adalah bilangan bulat ret yang merupakan elemen order $base$ modulo $modulo$.

```
1  int64_t getElementOrder(int64_t base, int64_t
       modulo, unordered_map<int64_t, int64_t> &
       primefactorization){
2    int64_t ret = modulo - 1;
3    int64_t a;
```

Kode Sumber 4.11a Fungsi ELEMENTORDER (bagian 1)

```
4    for(unordered_map<int64_t, int64_t>::iterator i
         = primefactorization.begin(); i !=
         primefactorization.end(); i++){
5      ret /= this->modExponentiation(i->first,
           i->second, modulo);
6      a = this->modExponentiation(base, ret,
           modulo);
7      while(a != 1){
8        a = this->modExponentiation(a, i->first,
             modulo);
9        ret *= i->first;
10     }
11   }
12   return ret;
13 }
```

Kode Sumber 4.11b Fungsi ELEMENTORDER (bagian 2)

4.2.3 Class BabyGiantStep

Class ini digunakan untuk melakukan penyelesaian permasalahan logaritma diskrit menggunakan algoritma *Baby-Step Giant-Step*. Variabel $babystep$ dan alg berturut-turut digunakan untuk menyimpan nilai dari B^r berdasarkan penjelasan pada subbab 2.4 dan melakukan pemanggilan fungsi dari *class* ALGORITHMS seperti yang telah dituliskan pada subbab 4.2.2.

```
1 class BabyGiantStep{
2 private:
3   unordered_map<int64_t, int> babystep;
4   Algorithms alg;
5 public:
6   int64_t solve(int64_t base, int64_t res,
        int64_t mod, int64_t lim, int64_t pinv);
7 };
```

Kode Sumber 4.12 *Class* BABYGIANTSTEP

4.2.3.1 Fungsi Penyelesaian Baby-Step Giant-Step

Fungsi ini merupakan implementasi dari desain fungsi pada subbab 3.2.10. Parameter fungsi ini adalah bilangan bulat $base$ sebagai basis, $residue$ sebagai sisa dari operasi modulo dan $modulo$ sebagai modulus. Parameter $limit$ dan inv digunakan untuk menjalankan algoritma *Baby-Step Giant-Step* dengan *group order* sesuai masukan yang diterima. Keluaran dari fungsi ini adalah bilangan bulat exp sedemikian sehingga $base^{exp} \equiv residue \pmod{modulo}$.

```
 1 int64_t solve(int64_t base, int64_t res, int64_t
       mod, int64_t lim, int64_t  pinv){
 2   int64_t expinv;
 3   int64_t iter;
 4   int64_t gcd;
 5
 6   int64_t inv = pinv;
 7   int64_t ub1 = lim;
 8   iter = 1;
 9   this->babystep.clear();
10
11   for(int j = 0; j < ub1; j++){
12     this->babystep[iter] = j;
13     if (iter == res) return j;
14     iter = (iter * base) % mod;
15     if (iter == 1) break;
16   }
17   iter = res;
18   expinv = this->alg.modExponentiation(inv, ub1,
         mod);
```

Kode Sumber 4.13a Fungsi SOLVE (bagian 1)

```
19   for(int i = 0; i < ub1; i++){
20     if(this->babystep.find(iter) !=
           this->babystep.end()){
21       return (ub1 * i + this->babystep[iter]);
22     }
23     iter = (iter * expinv) % mod;
24   }
25   return -1;
26 }
```

Kode Sumber 4.13b Fungsi SOLVE (bagian 2)

4.2.4 Class PohligHellman

Class ini digunakan untuk melakukan penyelesaian permasalahan logaritma diskrit menggunakan algoritma Pohlig-Hellman berdasarkan penjelasan pada subbab 2.5. *Class* ini memiliki lima variabel *private*, yaitu variabel $residues$ yang digunakan untuk menyimpan sisa bilangan N_i, variabel $mods$ untuk menyimpan bilangan $p_i^{r_i}$, variabel $element\_dislog$ untuk menyimpan logaritma diskrit E_i, variabel $primefactorization$ untuk menyimpan faktorisasi prima $M - 1$, dan variabel alg yang digunakan untuk melakukan pemanggilan fungsi dari *class* ALGORITHMS seperti yang telah dituliskan pada subbab 4.2.2.

```
1 class PohligHellman{
2 private:
3   vector<int64_t> residues, mods, element_dislog;
4   unordered_map<int64_t, int64_t>
        primefactorization;
5   Algorithms alg;
```

Kode Sumber 4.14a *Class* POHLIGHELLMAN (bagian 1)

```
6 public:
7   void reset();
8   int64_t solve(int64_t base, int64_t residue,
        int64_t modulo, BabyGiantStep & bsgs);
9 };
```

Kode Sumber 4.14b *Class* POHLIGHELLMAN (bagian 2)

4.2.4.1 Fungsi Reset

Fungsi ini digunakan untuk mengembalikan variabel $residues$, $mods$, $element\_dislog$, dan $primefactorization$ ke keadaan semula agar dapat digunakan kembali untuk melakukan penghitungan logaritma diskrit kasus uji selanjutnya. Implementasi fungsi ini dituliskan pada Kode Sumber 4.15.

```
1 void reset(){
2   this->residues.clear();
3   this->mods.clear();
4   this->element_dislog.clear();
5   this->primefactorization.clear();
6 }
```

Kode Sumber 4.15 Fungsi RESET

4.2.4.2 Fungsi Penyelesaian Pohlig-Hellman

Fungsi ini merupakan implementasi dari desain fungsi pada subbab 3.2.11. Parameter fungsi ini adalah bilangan bulat $base$ sebagai basis, $residue$ sebagai sisa dari operasi modulo, $modulo$ sebagai modulus, dan variabel $bsgs$ untuk melakukan pemanggilan fungsi penyelesaian logaritma diskrit *Baby-Step Giant-Step*. Keluaran dari fungsi ini adalah bilangan bulat exp sedemikian sehingga $base^{exp} \equiv residue \pmod{modulo}$.

```
 1 int64_t solve(int64_t base, int64_t residue,
       int64_t modulo, BabyGiantStep & bsgs){
 2   int64_t order;
 3   int64_t inv;
 4   int64_t tmp;
 5   int64_t gcd;
 6   int64_t i, j;
 7   int64_t iter_e, iter_r, iter_b, iter_inv,
         primepow, x0, dislog, limit;
 8
 9   i = 0;
10   j = 0;
11   this->alg.factorize(modulo - 1,
         this->primefactorization);
12   gcd = this->alg.extendedGCD(base, modulo, &inv,
         NULL);
13   while(inv < 0){
14     inv += (modulo / gcd);
15   }
16
17   for(unordered_map<int64_t, int64_t>::iterator
         iter = this->primefactorization.begin();
         iter != this->primefactorization.end();
         iter++, i++){
18     this->mods.push_back(
           this->alg.modExponentiation(iter->first,
           iter->second, modulo));
19     dislog = 0;
20     iter_e = (modulo - 1) / iter->first;
21     x0 = 0;
22     primepow = 1;
23     iter_b = this->alg.modExponentiation(base,
           iter_e, modulo);
```

Kode Sumber 4.16a Fungsi SOLVE (bagian 1)

```
25      // case iter_b not primitive element
26      if(iter_b == 1){
27        iter_b = this->alg.modExponentiation(base,
              (modulo - 1) / this->mods[i], modulo);
28        iter_r =
              this->alg.modExponentiation(residue,
              (modulo - 1) / this->mods[i], modulo);
29
30        gcd = this->alg.extendedGCD(iter_b, modulo,
              &iter_inv, NULL);
31        while(iter_inv < 0){
32          iter_inv += (modulo /gcd);
33        }
34
35        x0 = bsgs.solve(iter_b, iter_r, modulo,
              (int64_t)(ceil(sqrt(this->mods[i]))),
              iter_inv);
36        if(x0 == -1) return -1;
37        this->element_dislog.push_back(x0);
38        continue;
39      }
40
41      gcd = this->alg.extendedGCD(iter_b, modulo,
            &iter_inv, NULL);
42      while(iter_inv < 0){
43        iter_inv += (modulo / gcd);
44      }
45      limit = (int64_t)(ceil(sqrt(iter->first)));
46      for(j=0; j < iter->second; j++){
47        iter_r = (residue *
              this->alg.modExponentiation(inv,
              dislog, modulo)) % (modulo);
48        iter_r =
              this->alg.modExponentiation(iter_r,
              iter_e, modulo);
49
50        x0 = bsgs.solve(iter_b, iter_r, modulo,
              limit, iter_inv);
```

Kode Sumber 4.16b Fungsi SOLVE (bagian 1)

```
51        if(x0 == -1){
52          return -1;
53        }
54
55        dislog = (dislog + primepow * x0) %
              this->mods[i];
56        primepow *= iter->first;
57        iter_e /= iter->first;
58      }
59
60      element_dislog.push_back(dislog);
61
62    }
63    order = this->alg.getElementOrder(base, modulo,
          this->primefactorization);
64    return this->alg.chineseRemainder(
          this->element_dislog, this->mods, modulo -
          1) % order;
65  }
```

Kode Sumber 4.16c Fungsi SOLVE (bagian 3)

4.2.5 Class DislogSolver

Class ini memiliki dua kegunaan utama, yaitu menyimpan parameter masukan B, N, dan M serta melakukan pemanggilan fungsi penyelesaian logaritma diskrit. Proses penyimpanan parameter masukan dilakukan menggunakan fungsi SETPARAMETER dan proses penghitungan logaritma diskrit dilakukan menggunakan fungsi SOLVE.

```
 1 class DislogSolver
 2 {
 3 private:
 4   int64_t base, residue, modulo;
 5   BabyGiantStep bsgs;
 6   PohligHellman ph;
 7 public:
 8   void setParameter(int64_t base, int64_t
         residue, int64_t modulo);
 9   int64_t solve(string method);
10 };
```

Kode Sumber 4.17 *Class* DISLOGSOLVER

4.2.5.1 Fungsi Penyimpan Masukan Kasus Uji

Fungsi ini digunakan untuk menyimpan nilai B, N, dan M dari masukan kasus uji. Proses modulo pada parameter $base$ dan $residue$ dilakukan untuk penyederhanaan nilai dan tidak mempengaruhi kebenaran nilai logaritma diskrit E.

```
1 void setParameter(int64_t base, int64_t residue,
      int64_t modulo){
2   this->base = base % modulo;
3   this->residue = residue % modulo;
4   this->modulo = modulo;
5 }
```

Kode Sumber 4.18 Fungsi SETPARAMETER

4.2.5.2 Fungsi Penyelesaian Logaritma Diskrit

Fungsi ini digunakan sebagai antarmuka penyelesaian permasalahan logaritma diskrit menggunakan algoritma yang ada, dalam hal ini algoritma *Baby-Step Giant-Step* dan Pohlig-Hellman. Parameter masukan fungsi ini adalah string $method$ yang digunakan

untuk menentukan algoritma yang akan digunakan, ”bsgs” untuk algoritma *Baby-Step Giant-Step* dan ”ph” untuk algoritma Pohlig-Hellman.

```
 1 int64_t solve(string method){
 2   int64_t ret;
 3   this->base = this->base % this->modulo;
 4   this->residue = this->residue % this->modulo;
 5   // trivia case
 6   if(this->base == 0){
 7     if(this->residue != 0) return -1;
 8     else return 1;
 9   }
10   else if(this->residue == 1){ return 0; }
11   else if(this->base == 1){ return -1; }
12   else{
13     if(method == "bsgs"){
14       int64_t limit =
             (int64_t)(ceil(sqrt(this->modulo - 1)));
15       int64_t inv;
16       int64_t gcd = this->alg.extendedGCD(
             this->base, this->modulo, &inv, NULL);
17       while(inv < 0) {
18         inv += (this->modulo / gcd);
19       }
20       ret = this->bsgs.solve(this->base,
             this->residue, this->modulo, limit,
             inv);
21       return ret;
22     }
23     else if(method == "ph"){
24       ret = this->ph.solve(this->base,
             this->residue, this->modulo,
             this->bsgs);
25       this->ph.reset();
26       return ret;
27     }
28   }
29 }
```

Kode Sumber 4.19 Fungsi SOLVE

4.3 Implementasi Program Pembuat Kasus Uji

Subbab ini menjelaskan implementasi program pembuat kasus uji. Program ini digunakan untuk membuat kasus uji yang dapat digunakan sebagai parameter masukan program utama pada subbab 4.2.

Fungsi yang diimplementasikan pada program ini dikelompokkan menjadi 2 *class*, yaitu *class* Algorithms yang telah dibahas pada subbab 4.2.2 dan *class* TestDataGenerator.

4.3.1 Class TestDataGenerator

Class ini digunakan untuk membuat kasus uji sesuai dengan spesifikasi nilai B, N, dan M pada subbab 2.1.1. Variabel $testcase$ digunakan untuk menyimpan nilai dari B, N, dan M. Berikut penjelasan dari masing-masing fungsi di *class* ini.

```
 1 class TestdataGenerator
 2 {
 3 public:
 4     void generate(int t, int limit);
 5     void print_testcase(int idx, int limit);
 6 private:
 7     Algorithms alg;
 8     map<int64_t, set<string>> testcase;
 9     int64_t str_to_int64(string value);
10     string create_set_key(int64_t base, int64_t
           res, int64_t modulo);
11     void get_parameter(string set_key, int64_t *
           base, int64_t * res, int64_t * modulo);
12     int64_t interval_random(int64_t lb, int64_t
           ub);
13     void create_parameter(int64_t lb, int64_t ub,
           int64_t * base, int64_t * res, int64_t *
           modulo);
14     string generate_testcase(int64_t lb, int64_t
           ub);
15 };
```

Kode Sumber 4.20 *Class* TESTDATAGENERATOR

4.3.1.1 Fungsi Generator

Fungsi ini digunakan sebagai fungsi utama untuk membuat kasus uji dan menyimpannya di variabel *testcase* serta menampilkannya. Implementasi fungsi ini dituliskan pada Kode Sumber 4.21.

```
 1 void generate(int t, int limit){
 2     int maxnum = t;
 3     int64_t i;
 4     string key;
 5     for(int64_t i = 10; i < limit; i *= 10){
 6         while(testcase[i].size() < maxnum){
 7             key = this->generate_testcase(i,
                   i*10);
 8             this->testcase[i].insert(key);
 9         }
10     }
11     this->print_testcase(t, limit);
12     return;
13 }
```

Kode Sumber 4.21 Fungsi Generator

4.3.1.2 Fungsi untuk Menampilkan Kasus Uji

Fungsi ini digunakan untuk menampilkan seluruh data dalam variabel *testcase*. Implementasi fungsi ini dituliskan pada Kode Sumber 4.22.

```
 1 void print_testcase(int idx, int limit){
 2     for(int64_t i = 10; i < limit; i *= 10){
 3         for(auto iter =
               this->testcase[i].begin(); iter !=
               this->testcase[i].end(); iter++){
 4             cout << *iter << endl;
 5         }
 6     }
 7 }
```

Kode Sumber 4.22 Fungsi untuk Menampilkan Kasus Uji

4.3.1.3 Fungsi Konversi String menjadi Integer

Fungsi ini digunakan untuk melakukan konversi data dari tipe data *string* menjadi *integer*. Implementasi fungsi ini dituliskan pada Kode Sumber 4.23.

```
 1 int64_t str_to_int64(string value){
 2     int64_t ret = 0;
 3     for(int i = 1; i < value.length(); i++){
 4         if(value[0] == '-'){
 5             continue;
 6         }
 7         ret = ret * 10 + ((int64_t) value[i] -
               (int64_t) '0');
 8     }
 9     if(value[0] == '-'){
10         ret *= (-1);
11     }
12     return ret;
13 }
```

Kode Sumber 4.23 Fungsi Konversi String menjadi Integer

4.3.1.4 Fungsi Set Key

Fungsi ini digunakan untuk melakukan konversi nilai variabel B, N, dan M menjadi string dengan format ”B N M” guna menghindari pembuatan kasus uji yang sama. Implementasi fungsi ini dituliskan pada Kode Sumber 4.24.

```
 1 string create_set_key(int64_t base, int64_t res,
       int64_t modulo){
 2     string ret = "";
 3     ret += to_string(base); ret += " ";
 4     ret += to_string(res); ret += " ";
 5     ret += to_string(modulo);
 6     return ret;
 7 }
```

Kode Sumber 4.24 Fungsi Set Key

4.3.1.5 Fungsi Get Parameter

Fungsi ini digunakan untuk mendapatkan nilai variabel B, N, dan M dari string dengan format "B N M". Implementasi fungsi ini dituliskan pada Kode Sumber 4.25.

```
1  void get_parameter(string set_key, int64_t *
       base, int64_t * res, int64_t * modulo){
2      string buf = "";
3      int counter_param = 0;
4      for(int i = 0; i < set_key.length(); i++){
5          if (set_key[i] != ' ') {
6              buf += set_key[i];
7          }
8          else if (set_key[i] == ' ' || (i ==
               set_key.length() - 1) ) {
9              if (counter_param == 0) {
10                 *(base) = this->str_to_int64(buf);
11             }
12             else if (counter_param == 1) {
13                 *(res) = this->str_to_int64(buf);
14             }
15             else if (counter_param == 2) {
16                 *(modulo) =
                       this->str_to_int64(buf);
17             }
18             buf = "";
19         }
20     }
21 }
```

Kode Sumber 4.25 Fungsi Get Parameter

4.3.1.6 Fungsi Interval Random

Fungsi ini digunakan untuk mendapatkan bilangan acak pada rentang nilai tertentu. Implementasi fungsi ini dituliskan pada Kode Sumber 4.26.

```
1 int64_t interval_random(int64_t lb, int64_t ub){
2     return ( lb + rand()%(ub - lb) );
3 }
```

Kode Sumber 4.26 Fungsi Set Key

4.3.1.7 Fungsi Create Parameter

Fungsi ini digunakan untuk mendapatkan nilai acak B, N dan bilangan prima acak M. Implementasi fungsi ini dituliskan pada Kode Sumber 4.27.

```
1 void create_parameter(int64_t lb, int64_t ub,
      int64_t * base, int64_t * res, int64_t *
      modulo){
2     *(modulo) = 4;
3     while(!this->alg.deterministicMillerRabin(
          *modulo)) {
4         *(modulo) = this->interval_random(lb, ub);
5     }
6     *(base) = 1 + this->interval_random(lb, ub-1)
          % (*modulo);
7     *(res) = 1 + this->interval_random(lb, ub-1)
          % (*modulo);
8     return;
9 }
```

Kode Sumber 4.27 Fungsi Create Parameter

4.3.1.8 Fungsi Pembuat Kasus Uji

Fungsi ini digunakan untuk membuat kasus uji dengan parameter variabel B, N, dan M yang berbeda dari data dalam variabel $testcase$. Implementasi fungsi ini dituliskan pada Kode Sumber 4.28.

```
1 string generate_testcase(int64_t lb, int64_t ub){
2     int64_t b = 0, n = 0, m = 0;
3     while(this->testcase[lb].find(
          this->create_set_key(b, n, m)) !=
          this->testcase[lb].end() || m == 0){
4         this->create_parameter(lb, ub, &b, &n,
              &m);
5     }
6     return this->create_set_key(b, n, m);
7 }
```

Kode Sumber 4.28 Fungsi Pembuat Kasus Uji

BAB V
UJI COBA DAN ANALISIS

5.1 Lingkungan Uji Coba

Uji coba dilakukan pada perangkat dengan spesifikasi sebagai berikut.

1. Perangkat Keras
   (a) Prosesor Intel® Core™ i3-3240 CPU @ 3.40GHz (4 CPUs)
   (b) *Random Access Memory* 4096MB
2. Perangkat Lunak
   (a) Sistem Operasi Linux Mint 19.1 Tessa 64-bit

5.2 Skenario Uji Coba

Subbab ini akan menjelaskan hasil pengujian program penyelesaian permasalahan. Pengujian dilakukan untuk dua algoritma, *Baby-Step Giant-Step* dan Pohlig-Hellman. Metode pengujian yang akan dilakukan adalah sebagai berikut.

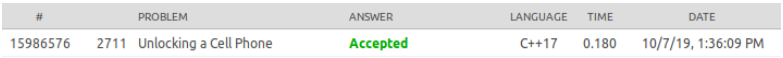
1. Pengujian luar. Pengujian ini menggunakan *URI Online Judge* untuk mengukur kebenaran dan informasi lain mengenai program.
2. Pengujian lokal. Pengujian ini menggunakan mesin yang digunakan dalam pengembangan untuk mengukur kebenaran dan informasi lain mengenai program.

Sampel yang digunakan sebagai kasus uji dalam pengujian lokal dibuat menggunakan program pembuat kasus uji dengan mengambil 10 nilai M acak dari setiap rentang nilai $10^i < M < 10^{i+1} \; \forall \; i \in \{1,\ 2,\ \cdots,\ 8\}$. Bilangan acak $B,\ N,$ dan M memenuhi batasan pada subbab 2.1.1.

5.3 Uji Coba Kebenaran

Kebenaran setiap metode diujikan dengan mengirim kode sumber terkait ke *URI Online Judge*. Berikut bukti hasil pengujian.

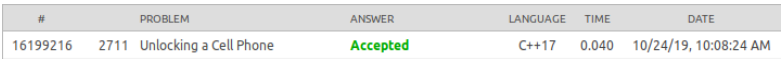
1. Kode sumber dengan metode *Baby-Step Giant-Step*.



#		PROBLEM	ANSWER	LANGUAGE	TIME	DATE
15986576	2711	Unlocking a Cell Phone	Accepted	C++17	0.180	10/7/19, 1:36:09 PM

Gambar 5.1 Umpan Balik Online Judge Metode *Baby-Step Giant-Step*

2. Kode sumber dengan metode Pohlig-Hellman.



#		PROBLEM	ANSWER	LANGUAGE	TIME	DATE
16199216	2711	Unlocking a Cell Phone	Accepted	C++17	0.040	10/24/19, 10:08:24 AM

Gambar 5.2 Umpan Balik Online Judge Metode Pohlig-Hellman

5.4 Uji Coba Kinerja

Pada subbab ini akan ditampilkan hasil uji coba kinerja dari algoritma *Baby-Step Giant-Step* dan Pohlig-Hellman untuk menyelesaikan permasalahan logaritma diskrit dengan modulus bilangan prima. Pengujian dilakukan terhadap kelompok masukan yang telah dijelaskan pada subbab 5.2. Detail masukan dan hasil uji coba kinerja dapat dilihat pada Lampiran A.

Langkah pengujian kinerja dilakukan dengan langkah sebagai berikut.

1. Rekam waktu tepat sebelum komputasi penyelesaian masalah dilakukan.
2. Melakukan komputasi penyelesaian masalah untuk masukan kasus uji yang sama sebanyak 10.000 kali secara berturut-turut.

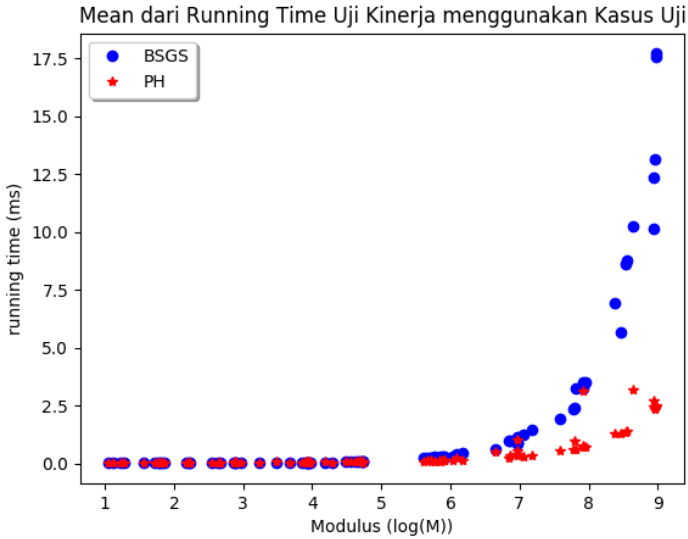
RANKING	SUBMISSION	USER	LANGUAGE	RUNTIME
1	15446691	Taufiq Tirtajiwangga	C++17	0.020
2	15406417	Rachmadi Rizki	C++	0.024
3	09453205	Muhammad Ghazian	C++17	0.188
4	09424691	Pedro Papa Paniago	C++	0.224
5	10581108	Erick Leonardo de Sousa Monteiro	C++	0.236
6	13733036	João Sobral	C++	0.304
7	13746691	Ciafrino	C++17	0.396
8	15455215	[GAPA] William Azevedo da Silva	C++	0.568
9	08972102	Diego Rangel Piranga Costa	C++	0.872
10	09163199	Luis Fernando Veronese Trivelatto	C++	0.940
11	09036689	Felipe Mota	C++17	0.956
12	08870821	Marcello Marques	C++	1.720
13	08844720	Gabriel Duarte	C++	1.748
14	08951248	Maycon Alves	C++	1.760
15	08884849	Gustavo Policarpo	C++	1.832
16	09051766	Luis Felipe Braga Gebrim Silva	C++	1.980

Gambar 5.3 Peringkat Solusi dengan Metode Pohlig-Hellman

3. Rekam waktu tepat setelah komputasi penyelesaian masalah dilakukan.
4. Menghitung durasi waktu komputasi dengan mengurangi waktu selesai komputasi dengan waktu sebelum komputasi.
5. Ulangi untuk seluruh kasus uji.

Karena memungkinkan terjadi fluktuasi *running time* dari solusi yang digunakan, proses penyelesaian kasus uji dilakukan 10.000 kali untuk mendapatkan rata-rata durasi waktu untuk setiap kasus uji.

Grafik 5.4 menampilkan rata-rata kinerja masing-masing metode. Berdasarkan hasil uji coba kinerja lokal didapatkan rata-rata *running time* dari algoritma *Baby-Step Giant-Step* dan Pohlig-Hellman berturut-turut adalah $(0,012 - 17,709)$ ms dan $(0,021 -$



Gambar 5.4 Rata-rata *Running Time* untuk tiap Nilai Modulus M Kasus Uji

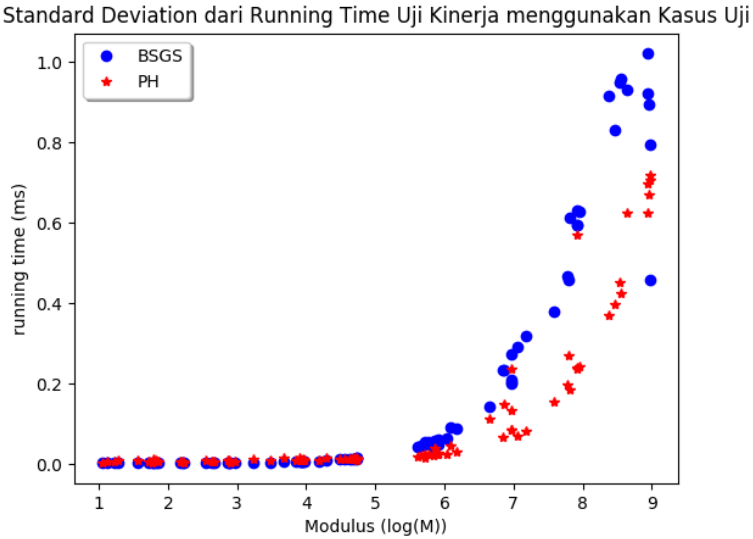
$3,195)$ ms.

Grafik 5.5 menunjukkan nilai standar deviasi dari kedua algoritma. Algoritma *Baby-Step Giant-Step* memiliki nilai standar deviasi $(0,002 - 1,021)$ ms. Sedangkan algoritma Pohlig-Hellman memiliki nilai standar deviasi $(0,004 - 0,720)$ ms.

5.5 Analisis dan Kesimpulan Umum dari Uji Kinerja

Dari pengujian kinerja didapatkan informasi mengenai kinerja algoritma *Baby-Step Giant-Step* dan Pohlig-Hellman.

1. Algoritma *Baby-Step Giant-Step* dapat menyelesaikan permasalahan logaritma diskrit dengan *running time* $(0.012 - 17.709) \pm (0.002 - 1.021)$ ms.



Gambar 5.5 Standar Deviasi *Running Time* untuk tiap Nilai Modulus M Kasus Uji

2. Algoritma Pohlig-Hellman dapat menyelesaikan permasalahan logaritma diskrit dengan *running time* $(0.021 - 3.195) \pm (0.004 - 0.720)$ ms.
3. Rata-rata *running time* algoritma *Baby-Step Giant-Step* dan Pohlig-Hellman relatif sama pada rentang nilai $M < 10^7$.
4. Rata-rata *running time* algoritma Pohlig-Hellman relatif lebih rendah daripada algoritma *Baby-Step Giant-Step* pada rentang nilai $M > 10^7$.

Berdasarkan penjelasan di atas, dapat ditarik beberapa kesimpulan terkait waktu yang dibutuhkan kedua algoritma untuk menyelesaikan permasalahan logaritma diskrit menggunakan kasus uji.

1. Algoritma Pohlig-Hellman dengan rata-rata *running time* mi-

nimum $0,021 \pm 0,004\ ms$ dan maksimum $3,195 \pm 0,720\ ms$ dapat menyelesaikan permasalahan logaritma diskrit lebih baik dari algoritma Pohlig-Hellman yang memiliki rata-rata *running time* minimum $0,012 \pm 0,002\ ms$ dan maksimum $17,709 \pm 1,021\ ms$.

2. Algoritma Pohlig-Hellman yang memiliki kompleksitas $\mathcal{O}(\sum_{i=1}^{k} e_i \cdot (log_2 M + \sqrt{p_i}))$ dapat menyelesaikan permasalahan logaritma diskrit lebih baik dari algoritma Pohlig-Hellman yang memiliki kompleksitas $\mathcal{O}(\sqrt{M})$.
3. Besar nilai modulus M tidak memiliki pengaruh yang signifikan terhadap selisih *running time* kedua algoritma untuk nilai $M < 10^7$.

BAB VI
KESIMPULAN DAN SARAN

6.1 Kesimpulan

Berdasarkan penjabaran di bab-bab sebelumnya, dapat dibuat beberapa kesimpulan terkait penyelesaian permasalahan logaritma diskrit studi kasus *URI Online Judge* 2711 *Unlocking A Cell Phone*.

1. Algoritma Pohlig-Hellman yang memiliki kompleksitas $\mathcal{O}(\sum_{i=1}^{k} e_i \cdot (log_2 M + \sqrt{p_i}))$ dapat menyelesaikan permasalahan logaritma diskrit lebih baik dari algoritma *Baby-Step Giant-Step* yang memiliki kompleksitas $\mathcal{O}(\sqrt{M})$. Pernyataan ini dibuktikan dengan hasil uji kinerja algoritma Pohlig-Hellman yang memiliki rata-rata *running time* minimum $0,021 \pm 0,004\ ms$ dan maksimum $3,195 \pm 0,720\ ms$ serta algoritma *Baby-Step Giant-Step* yang memiliki rata-rata *running time* minimum $0,012 \pm 0,002\ ms$ dan maksimum $17,709 \pm 1,021\ ms$.
2. Besar nilai modulus M tidak memiliki pengaruh yang signifikan terhadap selisih rata-rata *running time* kedua algoritma untuk nilai $M < 10^7$.

6.2 Saran

Hal yang perlu diperhatikan pada penggunaan algoritma Pohlig-Hellman adalah adanya dependensi terhadap algoritma pemfaktoran prima. Efisiensi dari algoritma pemfaktoran ini dan besarnya nilai modulus M dapat dijadikan sebagai bahan penelitian lebih lanjut untuk mengetahui kinerja dari algoritma Pohlig-Hellman.

[*Halaman ini sengaja dikosongkan*]

DAFTAR PUSTAKA

[1] A. Menezes, P. van Oorschot, and S. Vanstone, *Handbook of Applied Cryptography*, 5th ed. CRC Press, 1996. [Online] Available: http://cacr.uwaterloo.ca/hac/, [Accessed November 2, 2018].

[2] URI Online Judge. (2017). *Unlocking A Cell Phone*, [Online]. Available: https://www.urionlinejudge.com.br/judge/en/problems/view/2711.

[3] F. M. Goodman, *Algebra Abstract and Concrete*, 2nd ed. SemiSimple Press, 2015, [Online] Available: http://homepage.divms.uiowa.edu/~goodman/algebrabook.dir/book.2.6.pdf. [Accessed February 15, 2019].

[4] W. Stein, *Elementary Number Theory: Primes, Congruences, and Secrets*. Springer, 2017, [Online] Available: https://wstein.org/ent/ent.pdf. [Accessed September 7, 2018].

[5] M. Ghazian, *Perbandingan Kinerja Metode Baby-step Giant-step dan Pollard Rho dengan Brent Cycle Detection sebagai Metode Penyelesaian Permasalahan Logaritma Diskret: Studi Kasus Persoalan "DSA ATTACK" pada Timus Online Judge*. Institut Teknologi Sepuluh Nopember, 2018.

[6] C. Pomerance, J. L. Selfridge, and S. S. Wagstaff, "The pseudoprimes to $25 \cdot 10^9$", *Mathematics of Computation*, vol. 35, no. 151, pp. 1003–1026, 1980. [Online] Available: https://math.dartmouth.edu/~carlp/PDF/paper25.pdf. [Accessed January 30, 2019].

[7] C. Barnes, *Integer Factorization Algorithms*. 2004. [Online] Available: http://www.connellybarnes.com/documents/factoring.pdf, [Accessed January 25, 2019].

[8] S. Pohlig and M. Hellman, “An Improved Algorithm for Computing Logarithms over GF(p) and its Cryptographic Significance”, *IEEE Transactions on Information Theory*, vol. 24, no. 1, pp. 106–110, 1978. [Online] Available: https://ee.stanford.edu/ hellman/publications/28.pdf. [Accessed Desember 11, 2018].

LAMPIRAN A: Hasil Uji Kinerja Algoritma Baby-Step Giant-Step dan Pohlig-Hellman

Tabel A.1 Tabel hasil uji kinerja untuk nilai $10^1 < M < 10^2$

No	B	N	M	BSGS		PH	
				Mean	Std	Mean	Std
1	7	8	11	0.012	0.002	0.024	0.005
2	11	7	13	0.012	0.003	0.033	0.007
3	4	13	19	0.012	0.003	0.022	0.011
4	5	10	17	0.012	0.003	0.04	0.008
5	22	13	37	0.012	0.003	0.044	0.009
6	15	11	53	0.013	0.003	0.022	0.006
7	47	27	59	0.012	0.002	0.024	0.005
8	40	58	61	0.013	0.003	0.023	0.012
9	43	46	67	0.014	0.003	0.03	0.009
10	49	48	73	0.014	0.003	0.021	0.007

Tabel A.2 Tabel hasil uji kinerja untuk nilai $10^2 < M < 10^3$

No	B	N	M	BSGS		PH	
				Mean	Std	Mean	Std
1	77	91	149	0.014	0.003	0.036	0.007
2	23	2	163	0.016	0.003	0.024	0.007
3	131	69	167	0.015	0.002	0.027	0.004
4	11	253	349	0.017	0.002	0.036	0.011
5	427	411	433	0.016	0.003	0.046	0.007
6	133	240	461	0.018	0.003	0.047	0.007
7	314	297	733	0.018	0.004	0.049	0.009
8	493	608	757	0.019	0.003	0.067	0.01
9	235	381	797	0.02	0.003	0.029	0.004
10	534	4	947	0.021	0.003	0.041	0.006

Tabel A.3 Tabel hasil uji kinerja untuk nilai $10^3 < M < 10^4$

No	B	N	M	BSGS		PH	
				Mean	Std	Mean	Std
1	389	1670	1693	0.024	0.003	0.038	0.011
2	540	2293	3079	0.029	0.004	0.071	0.01
3	4025	4603	4729	0.034	0.006	0.05	0.016
4	4783	6451	7207	0.039	0.007	0.047	0.009
5	2751	7826	8161	0.036	0.006	0.092	0.016
6	6728	3465	8263	0.04	0.007	0.027	0.009
7	1643	8460	8521	0.042	0.007	0.04	0.01
8	5119	6649	8753	0.039	0.006	0.067	0.011
9	6931	8610	9137	0.037	0.006	0.068	0.011
10	3561	2805	9479	0.039	0.007	0.053	0.009

Tabel A.4 Tabel hasil uji kinerja untuk nilai $10^4 < M < 10^5$

No	B	N	M	BSGS		PH	
				Mean	Std	Mean	Std
1	7180	14130	15451	0.046	0.006	0.069	0.009
2	4577	1265	19739	0.058	0.01	0.059	0.015
3	13880	3201	30427	0.07	0.012	0.061	0.013
4	7555	25375	35279	0.075	0.013	0.061	0.013
5	27726	19727	41879	0.079	0.012	0.084	0.013
6	19608	22931	46237	0.083	0.013	0.081	0.014
7	11792	1656	48871	0.077	0.013	0.086	0.015
8	39088	35982	54679	0.075	0.012	0.074	0.012
9	40599	18771	54869	0.091	0.015	0.064	0.016
10	2714	43775	55009	0.091	0.015	0.04	0.011

Tabel A.5 Tabel hasil uji kinerja untuk nilai $10^5 < M < 10^6$

No	B	N	M	BSGS		PH	
				Mean	Std	Mean	Std
1	113655	370706	402593	0.23	0.043	0.081	0.019
2	401495	382627	463763	0.238	0.046	0.131	0.023
3	299113	246011	514001	0.26	0.054	0.136	0.026
4	526107	104591	526943	0.258	0.049	0.088	0.017
5	471293	291906	602317	0.278	0.054	0.108	0.025
6	137147	263668	693757	0.236	0.043	0.129	0.023
7	62975	151170	716161	0.302	0.059	0.091	0.039
8	423768	545102	769151	0.281	0.055	0.146	0.025
9	76374	590214	815459	0.321	0.062	0.12	0.025
10	80643	357272	829709	0.268	0.05	0.134	0.027

Tabel A.6 Tabel hasil uji kinerja untuk nilai $10^6 < M < 10^7$

No	B	N	M	BSGS		PH	
				Mean	Std	Mean	Std
1	373240	769768	1067611	0.32	0.065	0.139	0.026
2	948656	1112376	1244591	0.401	0.092	0.234	0.047
3	674573	515215	1513957	0.436	0.088	0.14	0.031
4	668119	2579193	4475707	0.636	0.143	0.497	0.111
5	4872433	750272	6949777	0.964	0.234	0.238	0.067
6	639071	1984213	7157587	0.977	0.235	0.347	0.148
7	8567851	6964216	9184171	0.873	0.203	0.423	0.086
8	6793756	7756842	9259837	0.862	0.2	0.561	0.135
9	180564	5884811	9283403	0.884	0.211	0.326	0.084
10	5635329	6687702	9290543	1.121	0.274	1.025	0.238

Tabel A.7 Tabel hasil uji kinerja untuk nilai $10^7 < M < 10^8$

No	B	N	M	BSGS		PH	
				Mean	Std	Mean	Std
1	2877833	1790484	11236237	1.238	0.292	0.286	0.07
2	417791	13894749	15121391	1.436	0.318	0.359	0.082
3	20811865	20576834	38184701	1.939	0.379	0.54	0.154
4	52761637	15179489	60011947	2.358	0.468	0.621	0.199
5	30995139	16862721	63555389	2.378	0.459	0.981	0.269
6	39704667	62124048	65312591	3.249	0.612	0.602	0.187
7	60138769	58584585	82249019	3.353	0.595	3.118	0.571
8	18484611	40331901	82448263	3.237	0.595	0.751	0.239
9	25951093	42303469	84916891	3.533	0.632	0.74	0.236
10	25115013	9554424	89440331	3.497	0.628	0.728	0.243

Tabel A.8 Tabel hasil uji kinerja untuk nilai $10^8 < M < 10^9$

No	B	N	M	BSGS		PH	
				Mean	Std	Mean	Std
1	96522901	136429769	239469869	6.917	0.915	1.272	0.371
2	29262050	173837537	289439881	5.681	0.831	1.292	0.398
3	189684200	301604600	340934389	8.595	0.95	1.35	0.453
4	173305408	159911576	358091501	8.75	0.958	1.385	0.425
5	85956726	329844158	446710541	10.266	0.933	3.195	0.624
6	290657162	835799488	862083709	10.13	1.021	2.468	0.626
7	241821926	232758591	885598787	12.383	0.923	2.693	0.697
8	467556180	161880633	906547193	13.136	0.896	2.371	0.67
9	178059513	245127752	949904467	17.551	0.794	2.492	0.707
10	202168195	1138376	967871923	17.709	0.46	2.481	0.72

BIODATA PENULIS

Penulis bernama Taufiq Tirtajiwangga, putra ketiga dari 3 bersaudara yang lahir pada tanggal 1 November 1997 di Tulungagung. Penulis melaksanakan pendidikan dasar di Sekolah Dasar Negeri 1 Kauman pada tahun 2004 hingga 2010, Sekolah Menengah Pertama Negeri 1 Kauman pada tahun 2010 hingga 2013, dan Sekolah Menengah Atas Negeri 1 Boyolangu pada tahun 2013 hingga 2016. Pada masa penulisan Tugas Akhir ini, penulis sedang menempuh studi S1 di Institut Teknologi Sepuluh Nopember, Surabaya di Departemen Teknik Informatika.

Selama masa studi, penulis memiliki ketertarikan dalam bidang matematika diskrit, rancang bangun aplikasi web, rancang bangun aplikasi sistem informasi, interaksi manusia dan komputer, kompresi data, dan riset operasi. Keinginan penulis dalam mengajar juga mendorong penulis menjadi asisten dosen pada mata kuliah Dasar Pemrograman, Struktur Data, Jaringan Komputer, dan Sistem Operasi.

Selain kesibukan akademik, penulis juga berperan aktif dalam berbagai kegiatan dan kepanitiaan. Beberapa diantaranya adalah staf soal pada kegiatan *National Programming Contest* Schematics tahun 2017 dan 2018, serta staf untuk bidang perlombaan *Capture the Flag* Gemastik tahun 2018.