TESIS - KI142502

PENGEMBANGAN ALGORITMA PENCARIAN NON INTERAKTIF UNTUK PENYELESAIAN PERMASALAHAN PENCARIAN ULAM DENGAN KEBOHONGAN JAMAK

RISYANGGI AZMI FAIZIN
NRP 5116201052

Dosen Pembimbing
Dr. Ir. R. V. Hari Ginardi, M.Sc
NIP: 196505181992031003

Program Magister
Rumpun Mata Kuliah Algoritma dan Pemrograman
DEPARTEMEN INFORMATIKA
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember
Surabaya, 2018

[*Halaman ini sengaja dikosongkan*]

Tesis disusun untuk memenuhi salah satu syarat memperoleh gelar
Magister Komputer (M.Kom.)
di
Institut Teknologi Sepuluh Nopember Surabaya

oleh:
Risyanggi Azmi Faizin
NRP. 05111650010052

Dengan judul :
Pengembangan Algoritma Pencarian Non Interaktif untuk Penyelesaian Permasalahan Pencarian Ulam dengan Kebohongan Jamak

Tanggal Ujian : 25 Juli 2018
Periode Wisuda : September 2018

Disetujui oleh:

Dr. Ir. Raden Venantius Hari Ginardi, M.Sc.
NIP. 196505181992031003 ..... (Pembimbing 1)

Royyana Muslim I, S.Kom., M.Kom., Ph.D.
NIP. 197708242006041001 ..... (Penguji 1)

Dr.Eng. Darlis Herumurti, S.Kom., M.Kom.
NIP. 197712172003121001 ..... (Penguji 2)

Dr.Eng. Radityo Anggoro, S.Kom., M.Sc.
NIP. 198410162008121002 ..... (Penguji 3)

Dekan Fakultas Teknologi Informasi dan Komunikasi,

Dr. Agus Zainal Arifin, S.Kom., M.Kom.
NIP. 197208091995121001

[*Halaman ini sengaja dikosongkan*]

ABSTRAK

PENGEMBANGAN ALGORITMA PENCARIAN NON INTERAKTIF UNTUK PENYELESAIAN PERMASALAHAN PENCARIAN ULAM DENGAN KEBOHONGAN JAMAK

Nama	:	Risyanggi Azmi Faizin
NRP	:	5116201052
Pembimbing	:	Dr. Ir. R. V. Hari Ginardi, M.Sc

Abstrak

Pada permasalahan permainan klasik pencarian Ulam dan Rényi, penanya harus mengajukan beberapa pertanyaan iya dan tidak untuk mencari sebuah nilai dalam range pencarian yang sudah disepakati, namun penjawab diperbolehkan berbohong. Sudah ada solusi dari beberapa variasi pada permasalahan pencarian Ulam dan Rényi, yaitu pada jenis query antara rentang atau subset dan jumlah maksimal bohong antara satu, dua, tiga, dan seterusnya. Namun belum ada solusi yang sempurna untuk query yang non interaktif yaitu penjawab hanya boleh menjawab query penanya setelah penanya selesai menanyakan semua querynya. Belum ada penelitian yang menyelesaikan permasalahan ini. Pada paper ini akan dijelaskan solusi sempurna untuk permainan Ulam dan Rényi non interaktif dengan maksimal kebohongan jamak menggunakan kode biner dengan jarak Hamming. Hasil algoritma pada paper ini menunjukkan jumlah query yang jauh lebih sedikit dari algoritma umum repetisi biner dan hasil terbaik pada pengujian online ternama.

Kata Kunci: permainan ulam; bohong; jarak hamming; kode biner; query;

[*Halaman ini sengaja dikosongkan*]

ABSTRACT

DEVELOPMENT OF NON-INTERACTIVE SEARCHING ALGORITHM FOR SOLVING ULAM'S SEARCHING PROBLEM WITH MANY LIES

Name : Risyanggi Azmi Faizin

Student ID : 5116201052

Supervisor : Dr. Ir. R. V. Hari Ginardi, M.Sc

Abstract

On the classic Ulam and Rényi searching problem, the questioner has to ask some yes-no questions to find an unknown value within the agreed search space, but the answerer is allowed to lie. There are already solutions of some variations in the Ulam and Rényi searching problem, i.e. on the type of query between range or subset and the maximum number of lies between one, two, three, and so on. But there is no perfect solution for non-interactive queries which the answerer can only answer the questioner's query after the questioner has finished querying all the queries. No research has resolved this problem yet. In this paper we will describe the perfect solution for non-interactive Ulam and Rényi searching problem with many lies using binary code with Hamming distance. The algorithm results in this paper shows a much smaller number of queries than the common binary repetition algorithm and the best results on a reputable online judge.

Keywords: ulam game; lies; hamming distances; binary codes; query;

[*Halaman ini sengaja dikosongkan*]

KATA PENGANTAR

Bismillahirrohmaanirrohiim. Puji syukur penulis panjatkan kepada Tuhan Yang Maha Esa. Atas rahmat dan kasih sayangNya, penulis dapat menyelesaikan tesis dalam bentuk buku ini yang berjudul **Pengembangan Algoritma Pencarian Non Interaktif untuk Penyelesaian Permasalahan Pencarian Ulam dengan Kebohongan Jamak**.

Pengerjaan buku 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 sekaligus pengerjaan buku ini dapat menjadi batu loncatan 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 tesis maupun masa studi.

1. Allah SWT yang selalu memberi kebahagiaan dan makna pada hidup.
2. Ibu dan Ayah yang selalu mendukung dalam segala-gala-gala hal. Terbaik.
3. Bapak Rully Soelaiman S.Kom.,M.Kom., selaku pembimbing bayangan penulis. Ucapan terima kasih juga penulis sampaikan atas segala perhatian, didikan, pengajaran, dan nasihat yang telah diberikan oleh beliau selama masa studi penulis.
4. Bapak Dr. Ir. R. V. Hari Ginardi, M.Sc, selaku pembimbing penulis yang telah memberikan arahan semasa pengerjaan tesis.
5. Rekan-rekan satu angkatan 2016 mahasiswa magister Teknik Informatika yang tidak lelah membantu penulis semasa masa studi, juga karena kesabaran mereka yang luar biasa dalam menghadapi kelakuan penulis.
6. Rekan-rekan satu lab DTK yang senantiasa memberi kebahagiaan nyata meskipun hanya sementara.

Penulis menyadari bahwa buku ini jauh dari kata sempurna. Maka dari itu, penulis

memohon maaf apabila terdapat salah kata maupun makna pada buku ini. Akhir kata, penulis mempersembahkan buku ini sebagai wujud nyata kontribusi penulis dalam ilmu pengetahuan.

Surabaya, Juni 2018

Risyanggi Azmi Faizin

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

[*Halaman ini sengaja dikosongkan*]

DAFTAR KODE SUMBER

[*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 tesis.

1.1. Latar Belakang

Dalam perkembangan dunia teknologi informasi selama beberapa dekade terakhir, teknologi informasi seringkali dijadikan solusi bagi permasalahan-permasalahan yang pernah ada, yang sebelumnya diselesaikan secara manual oleh manusia. Contoh permasalahan yang pernah ada adalah salah satu permasalahan klasik pencarian Ulam dan Rényi. Permasalahan ini dapat diilustrasikan dengan adanya dua pemain yang disebut penanya dan penjawab. Diberikan range pertanyaan $S_M = 0, \ldots, M-1$, penjawab menentukan sebuah bilangan $x \in S_M$. Penanya harus menemukan nilai x dengan memberikan beberapa query berupa pertanyaan iya dan tidak, apakah "$x \in Q$?", di mana Q adalah subset dari S_M, lalu penjawab menjawab "ya" atau "tidak". Permasalahan utama adalah penjawab dapat berbohong sampai e kali. Masalah dari permainan ini adalah mencari jumlah query minimal untuk dapat menentukan nilai x.

Pada permasalahan pencarian Ulam, penanya dan penjawab harus menyepakati beberapa peraturan sebelum bermain. Peraturan tersebut meliputi batasan ruang pencarian, batasan bagaimana penjawab diperbolehkan berbohong, format pertanyaan, dan bagaimana interaksi antara penjawab dan penanya [1]. Pertama penanya dan penjawab harus menyepakati batas ruang pencarian S_M, yaitu M angka, penjawab hanya boleh menentukan angka x diantara dalam set $\{0, \ldots, M-1\}$.

Aturan bagaimana penjawab diperbolehkan berbohong adalah aturan yang fundamental dalam permainan pencarian Ulam dan Rényi. Aturan probabilitas berbohong dicetuskan oleh Rényi dan aturan jumlah bohong dicetuskan oleh Ulam [2]. Pada kebohongan probabilitas yang diikiat secara global, probabilitas penjawab melakukan kebohongan di-

tentukan oleh r, sehingga maksimal penjawab melakukan kebohongan adalah rn di mana n adalah jumlah pertanyaan dan $r < 1$ [3]. Pada aturan jumlah bohong, variasi beragam antara maksimal penjawab dapat berbohong hanya satu [4] [5], dua [6], tiga [7], dan lebih dari tiga [8] [9].

Pada aturan format pertanyaan, terdapat beberapa variasi. Yang pertama adalah pertanyaan komparasi, bentuk pertanyaannya adalah "Apakah $x < a$?" di mana $a \in S_M$ [10] [11]. Lalu ada pertanyaan interval dan bi-interval, bentuk pertanyaannya adalah "Apakah x ada dalam interval $[a, b]$?" [12] dan "Apakah x ada dalam interval $[a, b] \cup [c, d]$?" [13]. Lalu format pertanyaan subset, bentuk pertanyaannya adalah "Apakah x ada dalam A di mana $A \subseteq S_M$" [14] [15].

Pada aturan interaksi antara penjawab dan penanya, terdapat tiga macam variasi yaitu interaktif, batch, dan non interaktif. Aturan yang paling umum digunakan adalah interaktif, yaitu penjawab harus menjawab setiap pertanyaan yang diajukan penanya sebelum penanya menanyakan pertanyaan selanjutnya. Pada aturan batch, penanya dan penjawab menyepakati berapa jumlah batch. Lalu pada setiap batch, penanya memberikan beberapa pertanyaan, lalu penjawab memberikan jawaban sejumlah pertanyaan yang diberikan oleh penanya [6]. Aturan yang terakhir adalah non interaktif, yaitu penjawab harus menjawab semua pertanyaan penanya sekaligus [15].

Salah satu variasi permasalahan Ulam dan Rényi yang diangkat dalam penelitian ini adalah pencarian Ulam dengan n query subset $q_1, q_2, \ldots, q_n | q_i \in S_M$, maksimal bohong adalah e, dan penjawab hanya boleh menjawab query penanya setelah penanya selesai menanyakan semua query. Permasalahan ini diangkat oleh Michał Miodek pada kategori *challenge* di *online judge* ternama yaitu SPOJ dengan identifikasi soal GUESSN5 [16]. Kategori *challenge* pada SPOJ adalah solusi yang diajukan oleh user akan dinilai berdasarkan skor untuk ditentukan rankingnya, sehingga ada solusi lebih bagus daripada solusi yang lain.

Pengujian dan evaluasi algoritma yang dirumuskan pada tesis ini dilakukan dengan pengujian dataset lokal dan pengujian dataset online menggunakan SPOJ. Pengujian data-

set lokal dilakukan untuk membandingkan jumlah query, waktu eksekusi, dan penggunaan memory dari semua algoritma dari semua kasus uji dataset. Pengujian pada SPOJ dilakukan untuk membandingkan algoritma solusi pada tesis ini dengan algoritma orang lain.

1.2. Rumusan Masalah

Permasalahan yang akan diselesaikan pada tesis ini adalah sebagai berikut:

1. Bagaimana merumuskan query untuk mencari bilangan diskrit pada interval yang diberikan pada permasalahan pencarian Ulam dengan pertanyaan seminimal mungkin?
2. Apakah algoritma kode biner dengan jarak Hamming dapat menyelesaikan permasalahan pencarian Ulam non interaktif lebih baik?
3. Bagaimana implementasi struktur data yang efisien dan optimal untuk menyelesaikan permasalahan pencarian Ulam?
4. Apakah solusi meggunakan algoritma kode biner dengan tabel pencarian dapat diterima pada pengujian online SPOJ?

1.3. Tujuan

Tujuan tesis ini adalah sebagai berikut:

1. Melakukan analisis dan mendesain algoritma dan struktur data untuk mencari bilangan dengan kebohongan dalam studi kasus permasalahan pencarian Ulam non interaktif dengan kebohongan.
2. Mengevaluasi hasil dan kinerja algoritma dan struktur data yang telah dirancang untuk permasalahan pencarian Ulam non interaktif dengan kebohongan.

1.4. Manfaat penelitian

Permasalahan Ulam non-interaktif dapat menjadi penting dalam beberapa aplikasi praktis. Salah satunya dapat digunakan untuk membuat enkode untuk mengirimkan data

pada jarak yang sangat jauh seperti pada luar angkasa, yaitu pada saat M macam data yang dikirimkan, terdapat kemungkinan ada e bit yang error. Dengan solusi yang dihasilkan pada permasalahan Ulam non-interaktif, data yang error dapat dikembalikan ke data semula dengan asumsi error yang terjadi tidak melebihi batas maksimal error yang sebelumnya sudah ditentukan. Demikian pula pada tes medis yang mungkin harus dilakukan secara paralel tanpa menunggu hasilnya untuk menentukan tes selanjutnya seperti pada rangkaian tes seri, bertujuan untuk mempercepat proses diagnostik [1].

1.5. Kontribusi penelitian

Belum ada pembahasan tentang permasalahan Ulam dengan query non-interaktif dan jumlah bohong lebih dari satu, dimana penanya harus menyiapkan seluruh query-nya sebelum penjawab menjawab seluruh query penanya sekaligus.

1.6. Batasan Masalah

Masalah yang akan diselesaikan memiliki batasan-batasan berikut:

1. Implementasi algoritma menggunakan bahasa pemrograman C++.
2. Batas maksimum kasus uji adalah 2^7.
3. Pada setiap permainan, penjawab dan penanya menyepakati jumlah M dan e.
4. Interval ruang pencarian adalah $[1, M]$, dengan M maksimum 2^{12}.
5. Jumlah maksimal penjawab dapat berbohong adalah $2 \leq e \leq 16$.
6. Dataset yang digunakan adalah dataset pada permasalahan SPOJ GUESSN5.

BAB II

KAJIAN PUSTAKA DAN DASAR TEORI

Bab ini berisi pembahasan tentang dasar teori dan kajian pustaka yang digunakan sebagai teori pendukung penelitian ini. Terlebih dahulu akan dijelaskan mengenai kasus pencarian Ulam. Kemudian dilanjutkan dengan beberapa solusi pencarian Ulam yang sudah ada maupun teori pendukung solusi pencarian Ulam non interaktif yaitu teori pengkodean dan kode biner.

2.1. Kasus-kasus pencarian Ulam di online judge

Pada online judge SPOJ, ada lima variasi soal yang diajukan seperti pada Tabel 2.1. Jenis soal classical adalah jenis soal yang jawaban yang diajukan hanya dapat bernilai benar atau salah, sedangkan jenis soal challenge membolehkan jawaban yang diajukan adalah yang paling buruk ataupun yang paling baik. Pada soal challenge, terdapat nilai skor dari jawaban yang diajukan.

Soal GUESSN1 dan GUESSN3 adalah pencarian Ulam dengan jumlah maksimal kebohongan satu dan tipe query interaktif. Soal jenis ini dapat diselesaikan dengan menentukan query dengan pembobotan Berlekamp $w_j(a,b) = a(j+1)+b$ di mana j adalah jumlah query tersisa, a dan b berturut adalah *truth*-set dan *lie*-set [17].

Tabel 2.1: Daftar kasus pencarian Ulam di SPOJ

Kode	Jumlah bohong	Jenis query	Tipe query	Jenis soal	Kesulitan
GUESSN1	1	Subset	Interaktif	challenge	Mudah
GUESSN2	2-16	Subset	Interaktif	challenge	Susah
GUESSN3	1	Range	Interaktif	classical	Menengah
GUESSN4	1	Subset	Non-interaktif	challenge	Menengah
GUESSN5	2-16	Subset	Non-interaktif	challenge	Sangat susah

2.2. Formulasi permasalahan

Bentuk permasalahan Ulam yang dibahas pada paper ini diangkat dari *online judge* SPOJ oleh Michał Miodek [16]. Penjawab menentukan sebuah bilangan x pada rentang $S_M = \{0, \ldots, M-1\}$. Anda sebagai penanya harus mencari nilai x dengan memberikan maksimal n query khusus apakah "$x \in Q$?", lalu penjawab menjawab 'ya' atau 'tidak' pada setiap query yang ditanyakan. Permasalahan utama adalah penjawab dapat berbohong sampai e kali. Selain itu, penjawab hanya boleh menjawab query penanya setelah penanya selesai menanyakan semua query-nya. Tujuan dari Ulam adalah mencari jumlah query minimal untuk dapat menentukan nilai x.

Bentuk dari query adalah string $s_1 s_2 s_3 \ldots s_n$ di mana s_i bernilai 0 atau 1. Jawaban dari penjawab adalah "Ya" jika $s_x = 1$ atau "Tidak" jika $s_x = 0$ dengan asumsi penjawab menjawab jujur.

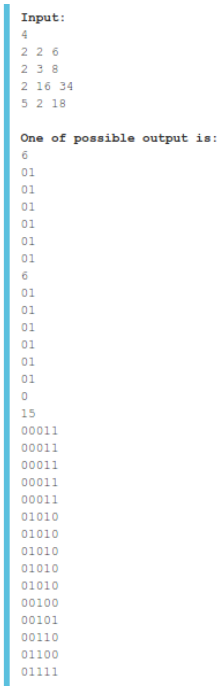
Tugas sesungguhnya dari permasalahan ini adalah bukan untuk mencari nilai x, tapi hanya menyiapkan query yang dapat memeungkinkan untuk mendapatkan nilai x dari semua kemungkinan jawaban dari penjawab. Penjawab tidak akan menjawab query yang diberikan penanya. Jika penjawab menemukan ada suatu set jawaban yang menyebabkan ada lebih dari satu kemungkinan nilai x, maka pengujian dianggap gagal.

Gambar 2.1 adalah contoh empat uji kasus dari permasalahan SPOJ GUESSN5. Pada uji kasus yang pertama hanya terdapat dua angka. Jika penjawab tidak berbohong, penjawab dapat menjawab pilihan

```
111111   000000.
```

Jika penjawab berbohong satu kali, penjawab dapat menjawab

```
111110   111101   111011
110111   101111   011111
000001   000010   000100
001000   010000   100000.
```



Gambar 2.1: Contoh kasus uji pada GUESSN5

Jika penjawab berbohong dua kali, penjawab dapat menjawab

```
000011   000101   000110
001001   001010   001100
010001   010010   010100
011000   100001   100010
100100   101000   110000
001111   010111   011011
011101   011110   100111
101011   101101   101110
110011   110101   110110
111001   111010   111100.
```

Penanya memenangkan permainan karena kemungkinan jawaban selain tersebut di atas membuat penjawab akan berbohong tiga kali.

Pada uji kasus yang kedua penjawab mencoba memberikan solusi namun jawaban-

nya salah. Penanya dapat menjawab `111000` yaitu jawaban yang valid karena jumlah bohong tiga kali untuk kedua kemungkinan angka. Pada kasus ini penanya kalah karena penanya membutuhkan query tambahan.

Pada uji kasus yang ketiga penanya tidak memberikan solusi.

Pada uji kasus yang keempat penanya memberikan query yang lebih sedikit dari jumlah query maksimal yang diperbolehkan. Dari semua kemungkinan jawaban penjawab, pasti hanya ada satu jawaban nilai x, jadi penanya memenangkan permainan.

2.3. Solusi pada permasalahan Ulam secara umum

Pada permasalahan Ulam, penjawab menentukan sebuah angka x di mana $x \in S_n$, lalu penanya akan menanyakan query untuk membantu mencari nilai x. Pada kenyataannya, penjawab tidak benar-benar memilih sebuah angka x, namun mempersiapkan n angka dengan state keboongan dari setiap angka [17]. State kebohongan setiap angka dapat digambarkan dengan bipartite graph kanal untuk status kebohongan dari setiap bilangan S_n [18]. Setiap kanal $C_i \subseteq S_n$ pada $C = \{C_0, C_1, \ldots, C_e\}$ adalah kanal berbentuk vektor yang berisi elemen dari S_n dengan status jumlah kebohongan i [19].

Setiap query $Q = \{q_1, q_2, \ldots\} \mid q_i \in S_n$, dengan jawaban penjawab adalah ”ya” maka set angka yang dianggap benar pada query tersebut adalah $Q_t = \{t_1, t_2, \ldots\} \mid t_i = q_i$ sedangkan jika jawaban penjawab adalah ”tidak” maka set angka yang dianggap benar pada query tersebut adalah $Q_t = S_n - Q$. Semua angka yang dianggap salah yaitu pada $S_n - Q_t = \{t_1, t_2, \ldots\}$ akan dipindahkan ke state kanal setelahnya. Jika t_i berasal dari C_e, maka t_i akan dikeluarkan dari kanal, sehingga nilai x pasti bukan t_i karena melampaui jumlah bohong maksimal.

Kanal pada setiap state query dan jawaban query memiliki bobot, yaitu berapa kesempatan setiap nilai angka pada setiap kanal untuk berbohong, sampai akhirnya tereliminasi karena melampaui jumlah bohong maksimal. Setiap state dapat dinotasikan pada Persamaan 2.1, di mana wb adalah fungsi Berlekamp, C adalah vektor kanal state, c_i adalah jumlah elemen pada C_i, k adalah jumlah query tersisa, dan e adalah jumlah bohong

maksimal. Notasi kombinatorik pada Persamaan 2.2 yaitu q kombinasi i.

$$wb(C,k) = \sum_{i=0}^{e} c_i \binom{k}{\leq e-i} \tag{2.1}$$

$$\binom{k}{\leq e} = \sum_{i=0}^{e} \binom{q}{i} \tag{2.2}$$

Karakter dari state C yang dapat dinotasikan sebagai $ch(C)$ yang ditunjukkan pada Persamaan 2.3 adalah jumlah pertanyaan minimal untuk dapat memenangkan permainan pada state C [9]. Oleh karena itu penanya memiliki kemungkinan untuk memenangkan permainan jika $wb(C,k) \geq 2^k$ [19].

$$ch(C) = \min\{k \mid wb(C,k) \leq 2^k\} \tag{2.3}$$

Berdasarkan teori konservasi Berlekamp [9] $wb(C,k) = wb_y(C,k) + wb_n(C,k)$ yaitu bobot pada state k sama dengan jumlah bobot pada state $k-1$ dengan jawaban penjawab ”ya” dan ”tidak”. Oleh karena itu yang harus dilakukan adalah mempersiapkan query agar selisih bobot jawaban ”ya” dan ”tidak” sekecil mungkin untuk meminimalkan bobot.

Untuk mengetahui sebuah query memiliki selisih bobot jawaban ”ya” dan ”tidak” sekecil mungkin maka dibuatlah sebuah rumus seperti pada Persamaan 2.4 di mana j adalah state sisa pertanyaan, Y dan N adalah hasil C ketika jawaban ”ya” dan ”tidak”. Jika nilai $\Delta_j(C,Q) = 0$ maka dapat dikatakan bahwa query a adalah sempurna pada state j. Namun karena penanya tidak dapat selalu membuat pertanyaan yang sempurna, maka akan dicari pertanyaan yang dapat menghasilkan $\Delta_j(C,Q)$ mendekati 0.

$$\Delta_j(C,Q) = wb(Y(C,Q),j) - wb(N(C,Q),j) \tag{2.4}$$

2.4. Repetisi kode biner

Ide pertama yang mungkin untuk menyelesaikan permasalahan ini adalah dengan mempersiapkan pencarian biner [20]. Query awal pencarian biner berjumlah $q_b = \log_2(M)$, agar setiap kemungkinan jawaban dari penjawab dapat mewakili semua kemungkinan nilai x. Lalu setiap query akan diulang sebanyak $q_e = 2e + 1$ kali agar penjawab pasti menjawab dengan jujur, karena e query untuk jawaban bohong, ditambah dengan e query untuk mengeliminasi jawaban bohong, ditambah dengan satu query jawaban pasti jujur karena kesempatan penjawab untuk berbohong sudah habis. Total jumlah query $q = q_b \times q_e$.

Tabel 2.2: Contoh pencarian biner pada $M = 8$

x	1	2	3	4	5	6	7	8
Q_1	0	0	0	0	1	1	1	1
Q_2	0	0	1	1	0	0	1	1
Q_3	0	1	0	1	0	1	0	1
Jawaban	NNN	NNY	NYN	NYY	YNN	YNY	YYN	YYY

Misalnya jika $M = 8, e = 2$, maka jumlah q_b untuk pencarian biner adalah `00001111`, `00110011` dan `01010101` seperti pada Tabel 2.2. Dari tiga query, tersebut, semua jawaban penjawab mulai dari ”NNN” sampai ”YYY” dapat mewakili semua nilai x dalam $S_M = 1, 2, ..., 8$ sehingga nilai $q_b = 3$. Lalu masing-masing query diulang sebanyak $q_e = 2e + 1 = 5$ kali. Maka total dari $q = q_b \times q_e = 9$.

2.5. Teori pengkodean

Tujuan utama dari teori pengkodean (*coding theory*) adalah bagaimana mengirimkan pesan pada kanal yang mengandung derau (*noisy channel*) [20]. Misal jika ada delapan macam kata pesan yang akan dikirim, maka kita merepresentasikan pesan tersebut menjadi bitstring biner dengan panjang 3. Namun jika pesan tersebut dikirm langsung melewati kanal yang mengandung derau, bisa jadi misalkan ada 1 bit akan tertukar, misal `001` menjadi `011`. Jika terjadi seperti itu, maka sebuah kata dapat tertukar menjadi kata yang lain.

Kita tahu bahwa jika kode biner sepanjang n digunakan untuk membuat 2^n bitstring

$$
\begin{array}{ll}
0+0=0 & 0\cdot 0=0 \\
0+1=1 & 0\cdot 1=0 \\
1+0=1 & 1\cdot 0=0 \\
1+1=0 & 1\cdot 1=1
\end{array}
$$

Gambar 2.2: Operasi linear pada $\mathbb{F}_2$

tidak dapat mendeteksi error. Ide yang paling mungkin adalah pengirim dan penerima menyetujui sebuah metode enkripsi bitstring menjadi bitstring yang lebih panjang dan dapat mendeteksi maksimal sebanyak e error menggunakan kode linear.

Kode linear adalah kode yang paling banyak dipelajari karena struktur aljabar yang mudah dipelajari dibandingkan kode non-linear [21]. Bidang kode linear dapat dinotasikan sebagai $\mathbb{F}_q^n$, yaitu kode memiliki q jenis elemen dan memiliki panjang n. Bentuk kode biner adalah $\mathbb{F}_2^n$, memiliki struktur aljabar penjumlahan dan perkalian pada Gambar 2.2.

$$d_H(\vec{x},\vec{y}) = |\{i \in 1,\ldots,n \mid x_i \neq y_i\}| \tag{2.5}$$

Jarak Hamming dari bitstring $\vec{x}$ dan $\vec{y}$ dengan panjang n didefinisikan dengan Persamaan 2.5 [6]. Sebagai contohnya $d_H(\texttt{0000},\texttt{1111}) = 4$ dan $d_H(\texttt{00110},\texttt{00101}) = 2$. $d_H(\vec{x},\vec{y})$ juga dapat dikatakan jumlah minimal untuk mentransformasi dari $\vec{x}$ ke $\vec{y}$. Contoh $\vec{x} = \texttt{00110}$ dan $\vec{y} = \texttt{00101}$ memiliki perbedaan pada 2 bit terakhir dengan jarak Hamming 2, dapat dikatakan $\vec{x} + \texttt{00011} = \vec{y}$.

$$wt(\vec{x}) = |\{i \in 1,\ldots,n \mid x_i \neq 0\}| \tag{2.6}$$

Bobot dari bitstring $\vec{x}$ didefinisikan dengan $wt(\vec{x})$, yaitu jumlah digit pada $\vec{x}$ yang bukan $\texttt{0}$ seperti pada Persamaan 2.6. Sebagai contohnya, $wt(\texttt{00101}) = 2$ dan $wt(\texttt{11111}) = 5$. Jika dihubungkan dengan jarak Hamming, jika $\vec{x}+\vec{e} = \vec{y}$ maka $d_H(\vec{x},\vec{y}) = wt(\vec{x}+\vec{y})$.

Terdapat sebuah sifat pada jarak Hamming yang bernama segitiga pertidaksamaan

```
000000  100110
001011  101101
010101  110011
011110  111000
```
Gambar 2.3: Kode biner $(6, 8, 3)_2$

(*triangle inequality*) [20].

$$d_H(\vec{x}, \vec{y}) \leq d_H(\vec{x}, \vec{z}) + d_H(\vec{y}, \vec{z}) \tag{2.7}$$

Dari Persamaan 2.7, misalkan $\vec{z}$ adalah $\vec{0}$ yaitu string biner yang berisi semua `0`, maka didapatkan Persamaan 2.8.

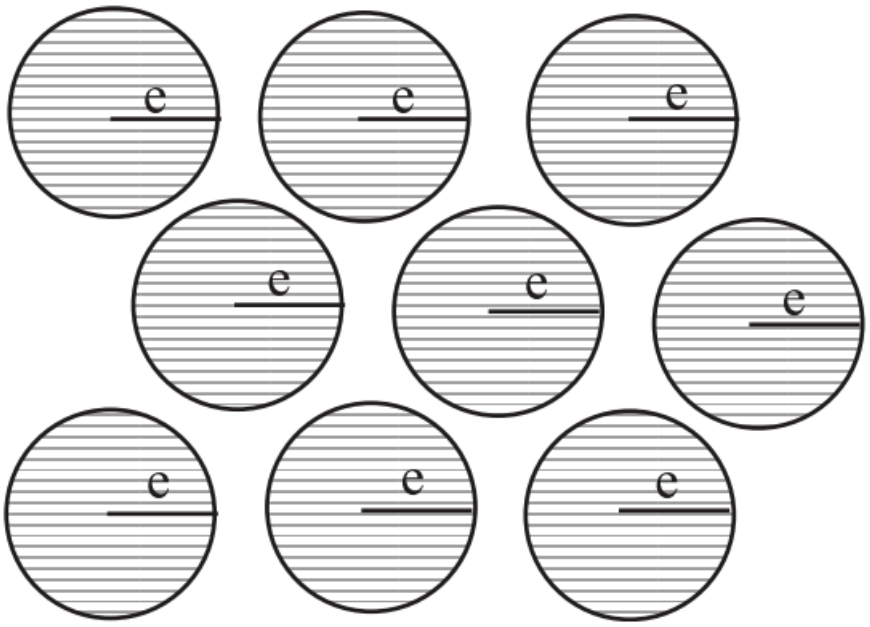
$$wt(\vec{x} + \vec{y}) \leq wt(\vec{x}) + wt(\vec{y}) \tag{2.8}$$

2.6. Kode biner

Kode biner (*binary code*) adalah sejumlah M bitstring biner dengan panjang masing-masing bitstring adalah n dan jarak Hamming pada masing masing bitstring adalah d. Mari kita ambil contoh $M = 8$, $n = 6$, dan $d = 3$ pada Gambar 2.3. Parameter pada kode ini adalah $(6, 8, 3)_2$, yaitu kode biner yang ditunjukkan pada angka 2, panjang bitstring 6, berisi 8 bitstring, dengan jarak Hamming minimal 3. Bitstring pada kode biner selanjutnya disebut kata kode (*codeword*).

Dengan kode biner $(6, 8, 3)_2$ pada Gambar 2.3, pengirim dan penerima menyepakati hanya kata kode yang akan dikirim dan diterima. Dengan asumsi hanya ada satu bit yang dapat error, pesan error tetap dapat dikembalikan ke bentuk semula. Misal `111100` menjadi `111000`, `000011` menjadi `001011`, dan seterusnya. Jarak Hamming antara setiap dua kata kode yang berbeda adalah 3, berarti dari setiap kata kode, terdapat sejumlah bitstring selain kata kode berjarak 1.

Notasi umum kode biner adalah $(n, M, d)_2$, dikatakan valid jika jarak setiap dua codeword yang berbeda adalah tidak kurang dari d. Kita bisa asumsikan ada M bola yang



Gambar 2.4: Bola codeword yang tidak saling overlap

```
100110
010101
001011
```

Gambar 2.5: Generator matrix $[6, 3, 3]_2$

tidak saling bersinggungan atau berpotongan, dengan radius bola e seperti pada Gambar 2.4. Jarak antara satu bola dengan bola yang lain adalah minimal d, sehingga didapatkan Persamaan 2.9.

$$d = 2e + 1 \tag{2.9}$$

Sebuah kode biner $(n, M, d)_2$ dapat dibangkitkan dari kombinasi linear dari setiap baris pada generator matrix $[n, m, d]_2$ di mana $2^m = M$. Generator matriks G adalah matriks berukuran $n \times m$, dapat juga disebut basis dari sebuah kode biner [20]. Contoh kode biner $(6, 8, 3)_2$ pada Gambar 2.3 dapat dibangkitkan dari generator matriks $[6, 3, 3]_2$ pada Gambar 2.5. Asumsikan bahwa setiap baris dari $[6, 3, 3]_2$ adalah g_1, g_2, and g_3, lalu kode biner $(6, 8, 3)_2$ adalah semua kombinasi linear penjumlahan dari semua vektor dalam bentuk $\lambda_1 g_1 + \lambda_2 g_2 + \lambda_3 g_3$ di mana $\lambda_i \in \mathbb{F}_2^6$.

Dari sebuah kode biner, kita dapat membuat kode biner yang baru dengan memodifikasi kode biner yang sudah ada [21]. Modifikasi yang dapat dilakukan adalah menambah dan mengurangi kode. Dari sebuah kode biner $(n, M, d)_2$, dapat dilakukan pemotongan pada kolom tertentu sehingga kode biner menjadi $(n-1, M, d')_2$ di mana $d' = d$ atau $d' = d - 1$. Begitu pula dengan penambahan kode, dari sebuah kode biner $(n, M, d)_2$, dapat dilakukan penambahan kode sehingga kode biner menjadi $(n+1, M, d')_2$ di mana $d' = d$ atau $d' = d + 1$.

2.7. Kolerasi kode biner dengan permasalahan Ulam non interaktif

Tujuan dari permasalahan ini adalah untuk menghasilkan n query. Diberikan sebuah matriks L berukuran $n \times M$ berisi n query. Kumpulan query ini dinotasikan dengan $L = \{\vec{q_1}, \vec{q_2}, \ldots, \vec{q_n}\}$ di mana $\vec{q_i} = \{s_1, s_2, \ldots, s_M\}$. Himpunan nilai s_j yang mungkin adalah $s_j \in \{0, 1\}$.

Diberikan sebuah vektor $\vec{z} \in \{z_1, z_2, \ldots, z_n\}$ di mana $z_i \in \{0, 1\}$ berisi jawaban dari seluruh query secara berurutan, z_i adalah jawaban dari $\vec{q_i}$. Jika $z_i = 0$ berarti jawaban query q_i adalah 'tidak' sehingg seluruh s_j pada $\vec{q_i}$ harus ditambah dengan 1 dan jika $z_i = 1$ berarti jawaban dari query $\vec{q_i}$ adalah 'ya' sehingga seluruh s_j pada $\vec{q_i}$ ditambah dengan 0 (diabaikan).

Matriks $L^\top$ berukuran $M \times n$ adalah hasil transpose dari matriks L. Asumsikan $L^\top$ juga dapat disebut sebagai kode biner $(n, M, d)_2$ di mana n adalah jumlah query, M adalah panjang S_M, dan d adalah $2e + 1$ di mana e adalah jumlah maksimal bohong. Tambahkan seluruh baris $\vec{c_j}$ pada $L^\top$ dengan $\vec{z}$. Maka jawaban dari permainan Ulam non interaktif adalah $x = j$ di mana j adalah index dari baris c pada $L^\top$ yang memiliki bobot $wt(\vec{c_j}) > n - e$.

Lemma 2.7.1. *Diketahui integer n, M, dan d. Jika $L^\top$ adalah kode biner $(n, M, d)_2$ yang valid, maka jika setiap codeword $\vec{c}$ pada $L^\top$ ditambah dengan $\vec{z} \mid \vec{z} \in \mathbb{F}_2^n$ maka hasilnya akan tetap menjadi kode biner $(n, M, d)_2$ yang valid.*

Bukti. Jika $d_H(\vec{v_i}, \vec{w_j}) \geq d \mid \vec{v_i}, \vec{w_j} \in L^\top$ maka

$$\begin{aligned} d_H(\vec{v_i} + \vec{z}, \vec{w_j} + \vec{z}) &\geq d \\ wt(\vec{v_i} + \vec{w_j} + \vec{z} + \vec{z}) &\geq d \\ wt(\vec{v_i} + \vec{w_j}) &\geq d \\ d_H(\vec{v_i}, \vec{w_j}) &\geq d \end{aligned}$$

Jadi jika $d_H(\vec{v_i}, \vec{w_j}) \geq d$ maka $d_H(\vec{v_i} + \vec{z}, \vec{w_j} + \vec{z}) \geq d$. □

Penanya memenangkan permainan jika $L^\top$ memiliki paling banyak satu baris dengan $wt(\vec{c}) \geq n - e$. Jika hanya ada satu row, maka row tersebut adalah jawaban permainan. Jika tidak ada satu row pun yang memenuhi, penanya tetap memenangkan permainan karena penjawab melakukan kecurangan, melakukan bohong untuk semua angka lebih dari batas yang ditetapkan.

Untuk meyakinkan bahwa setelah seluruh jawaban $\vec{z}$ diberikan dan diaplikasikan ke matrix L dan tidak pasti hanya ada 1 baris yang memiliki nilai 1 antara $n - e \leq wt(\vec{c_j}) \leq n$, adalah dengan memastikan bahwa jarak Hamming setiap row yang berbeda pada $L^\top$ adalah minimal d.

Lemma 2.7.2. *Diketahui integer n, M, dan d. Jika $L^\top$ adalah kode biner $(n, M, d)_2$ yang valid, maka pasti hanya ada paling banyak satu codeword $\vec{c}$ yang memiliki $0 \leq wt(\vec{c}) \leq e$.*

Bukti. Jika ada codeword c di mana $0 \leq wt(c) \leq e$ maka $wt(\vec{v}) > e$ di mana $\vec{v} \neq c$. Pembuktian dapat dibuktikan dengan dua kasus.

1) Jika $wt(\vec{c}) = 0$ maka

$$\begin{aligned} d_H(\vec{c}, \vec{v}) &\geq d \mid \vec{c} \neq \vec{v}, \vec{v} \in L^{\top} \\ wt(\vec{c} + \vec{v}) &\geq d \\ wt(\vec{v}) &\geq d \end{aligned} \tag{2.10}$$

Sebelumnya telah disebutkan hubungan d dan e pada Persamaan 2.9. Persamaan tersebut dapat diturunkan menjadi

$$d > e \tag{2.11}$$

Dengan memasukkan Persamaan 2.11 ke Persamaan 2.10, maka didapatkan

$$wt(\vec{v}) > e$$

2) Jika $wt(\vec{c}) = e$ maka

$$\begin{aligned} d_H(\vec{c}, \vec{v}) &\geq d \mid \vec{c} \neq \vec{v}, \vec{v} \in L^{\top} \\ wt(\vec{c} + \vec{v}) &\geq d \\ wt(\vec{c}) + wt(\vec{v}) \geq wt(\vec{c} + \vec{v}) &\geq d \\ wt(\vec{c}) + wt(\vec{v}) &\geq d \\ e + wt(\vec{v}) &\geq d \end{aligned}$$

Masukkan Persamaan 2.9 untuk mensubstitusi d

$$\begin{aligned} wt(\vec{v}) &\geq 2e + 1 - e \\ &\geq e + 1 \\ wt(\vec{v}) &> e \end{aligned}$$

Jadi jika ada codeword c di mana $0 \leq wt(c) \leq e$ maka $wt(\vec{v}) > e$ di mana $\vec{v} \neq c$. $\square$

Dari pembuktian diatas, dapat disimpulkan bahwa untuk menyelesaikan permainan pencarian Ulam non interaktif dengan batas pencarian M dan maksimal kebohongan e, transpose dari n query yang dibuat harus membentuk kode biner $(n, M, d)_2$.

[*Halaman ini sengaja dikosongkan*]

BAB III

METODE PENELITIAN

Bab ini memaparkan tentang metodologi penelitian yang digunakan pada penelitian ini yaitu tahap analisis penyelesaian masalah dan implementasi.

3.1. Desain dan analisis penyelesaian masalah

Pada permasalahan pencarian Ulam non interaktif, penjawab tidak diperbolehkan menjawab sebelum penanya selesai menanyakan seluruh query. Ada dua pendekatan berbeda yang dapat dilakukan untuk menyelesaikan masalah tersebut, yaitu menggunakan repetisi pencarian biner dan menggunakan kode biner. Perlu diketahui bahwa kedua solusi tersebut tidak saling berhubungan.

3.1.1. Solusi menggunakan kode biner

Kita tahu bahwa permasalahan Ulam dengan ruang pencarian M dan maksimal bohong e dapat deselesaikan dengan membuat q query yang jika ditranspose akan membentuk kode biner $(n, M, d)_2$ dimana $d = 2 * e + 1$. Oleh karena itu tahap selanjutnya adalah jika diketahui M dan d, bagaimana membuat seminimal mungkin n agar kode biner $(n, M, d)_2$ valid. Langkah utama untuk membuat kode biner tersebut dengan M dan e yang diketahui adalah dengan menambahkan kode secara iteratif agar minimal jarak Hamming kode biner mencapai e.

Gambar 3.1 menunjukkan ilustrasi jika $M = 8$, $n = 3$, dan $d = 1$. Tabel pada gambar menunjukkan jarak Hamming antar codeword x dan y, sehingga dapat disimpulkan jarak Hamming minimal adalah 1. Gambar 3.2 adalah penambahan dua kode dari Gambar 3.1 sehingga $n = 5$ dan $d = 2$.

Diagram alir algoritma untuk menyelesaikan permasalahan Ulam non interaktif ada pada Gambar 3.3. Setelah dilakukan input ruang pencarian M dan maksimal bohong e, simpan M untuk digunakan pada saat output query terakhir. Ubah M menjadi perpang-

Distance matrix

X	1	2	3	4	5	6	7	8
1	X	1	1	2	1	2	2	3
2		X	2	1	2	1	3	2
3			X	1	2	3	1	2
4				X	3	2	2	1
5					X	1	1	2
6						X	2	1
7							X	1
8								X

Codewords

1: 000
2: 001
3: 010
4: 011
5: 100
6: 101
7: 110
8: 111

Gambar 3.1: Hasil tabel jarak Hamming pada $(3, 8, 1)_2$

Distance matrix

X	1	2	3	4	5	6	7	8
1	X	3	2	3	2	3	4	3
2		X	3	2	3	2	3	4
3			X	3	4	3	2	3
4				X	3	4	3	2
5					X	3	2	3
6						X	3	2
7							X	3
8								X

Codewords

1: 00000
2: 00111
3: 01010
4: 01101
5: 10001
6: 10110
7: 11011
8: 11100

Gambar 3.2: Hasil tabel jarak Hamming pada $(5, 8, 2)_2$

katan dua terdekat, karena algoritma selanjutnya hanya dapat berjalan jika M adalah perpangkatan dua. Pada subbab selanjutnya akan dijelaskan algoritma untuk menggunakan query generator dan juga algoritma utama untuk menyelesaikan permasalahan Ulam non interaktif.

Proses pencarian mendalam (*exhaustive search*) menggunakan pendekatan *greedy* akan dilakukan sebagai pra proses dan hasil pra proses tersebut akan langsung dimasukkan ke dalam kode sumber solusi, karena proses pencarian mendalam memiliki kompleksitas $O(M^3)$ sedangkan jika dilakukan sebagai pra proses dan dimasukkan langsung ke dalam

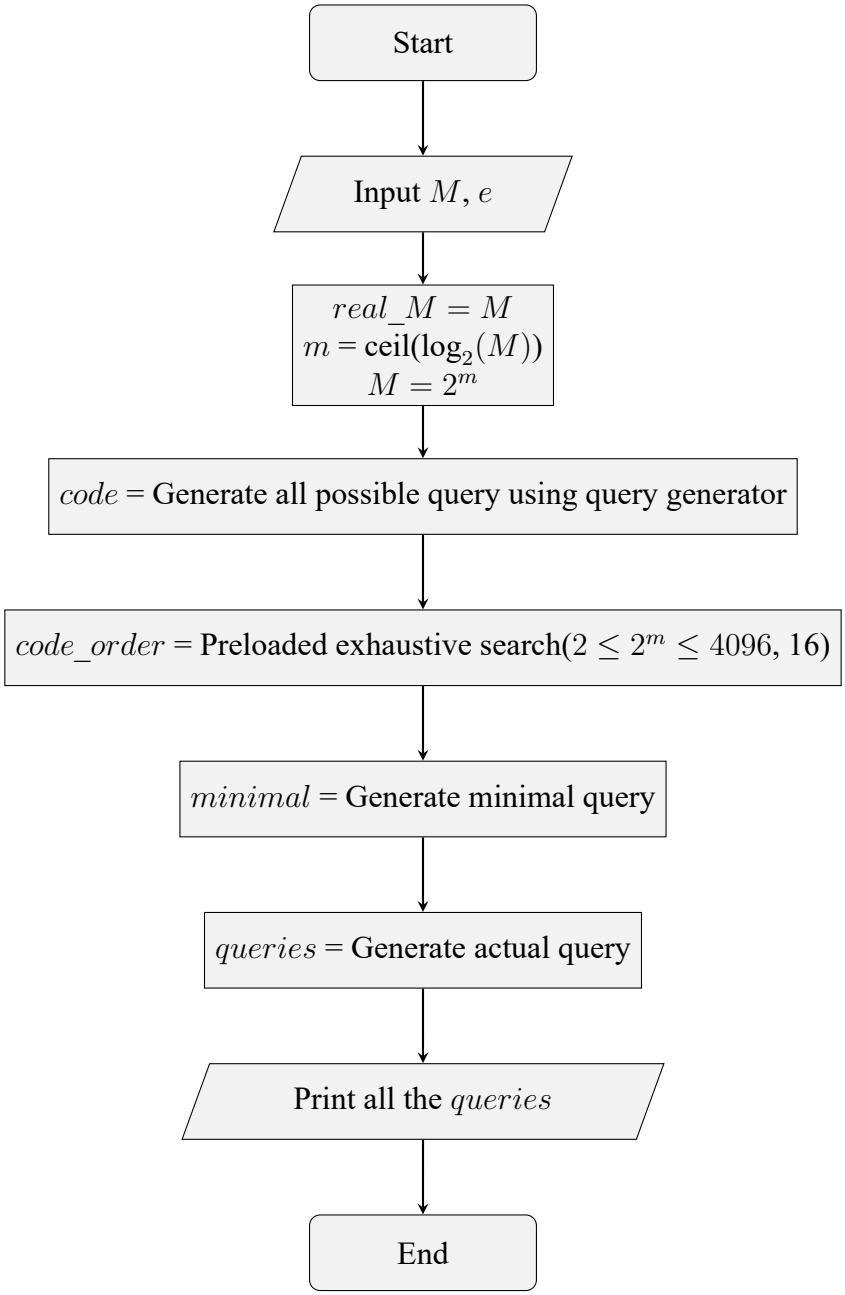
kode sumber solusi sebagai array akan mengurangi kompleksitas menjadi $O(1)$. Namun untuk keperluan pengujian, algoritma kode biner tanpa tabel pencarian (algoritma kode biner yang melakukan proses pencarian mendalam) akan dibandingkan dengan algoritma kode biner dengan tabel pencarian (proses pencarian mendalam dilakukan sebagai pra proses).

3.1.1.1. Algoritma pembangkitan query

Untuk membuat kode biner $(n, M, d)_2$, maka diperlukan algoritma untuk membangkitkan kode. Karena setiap satu query adalah hasil transpose dari sebuah kolom pada kode biner, setiap kolom pada kode biner dapat dibuat dari bitstring pembangkit $s = \mathbb{F}_2^m$ yang jika diubah menjadi integer akan memiliki rentang $0 \leq s < M$. Oleh karena itu ada $M-1$ macam s karena $s = 0$ akan menghasilkan query $\vec{0}$ yang tidak akan menghasilkan jarak Hamming pada kode biner.

Algoritma untuk membuat sebuah query dengan code generator ditunjukkan pada Kode Sumber 1. Input dari fungsi ini adalah ruang pencarian M dan bilangan bulat s yang akan diubah menjadi sebuah query. Pertama-tama siapkan bilangan bulat $0 \leq i < M$ untuk perulangan kombinasi linier pada s seperti yang ditunjukkan pada baris 3. Lalu kalikan semua bit dari s dan i seperti yang ditunjukkan pada baris 4. Lalu jumlahkan semua bit hasil perkalian $s \cdot i$ seperti yang ditunjukkan pada baris 6. Lalu hasil semua kombinasi linear akan ditampung pada string $query$ seperti yang ditunjukkan pada baris 10. Total query akan memiliki panjang M.

Setelah melakukan percobaan menggunakan seluruh $M-1$ query untuk M ruang pencarian, didapatkan seluruh jarak hamming setiap codeword adalah $M/2$. Gambar 3.4 adalah contoh pada $M = 8$ dan $n = 7$. Lalu Gambar 3.5 adalah contoh pada $M = 16$ dan $n = 15$. Dari hasil tersebut dapat disimpulkan bahwa dengan melakukan seluruh $M-1$ macam query dengan generator untuk ruang pencarian M, akan didapatkan kode biner dengan $d = M/2$. Kode biner pada Persamaan 3.1 disebut sempurna karena seluruh jarak Hamming pada setiap codeword yang berbeda bernilai $d = M/2$.



Gambar 3.3: Diagram alir solusi menggunakan pembobotan Berlekamp

Distance matrix

X	1	2	3	4	5	6	7	8
1	X	4	4	4	4	4	4	4
2		X	4	4	4	4	4	4
3			X	4	4	4	4	4
4				X	4	4	4	4
5					X	4	4	4
6						X	4	4
7							X	4
8								X

Codewords

1: 0000000
2: 0011101
3: 0101011
4: 0110110
5: 1000111
6: 1011010
7: 1101100
8: 1110001

Gambar 3.4: Hasil tabel jarak Hamming pada $M = 8$ dan $n = 7$

Distance matrix

X	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	X	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8
2		X	8	8	8	8	8	8	8	8	8	8	8	8	8	8
3			X	8	8	8	8	8	8	8	8	8	8	8	8	8
4				X	8	8	8	8	8	8	8	8	8	8	8	8
5					X	8	8	8	8	8	8	8	8	8	8	8
6						X	8	8	8	8	8	8	8	8	8	8
7							X	8	8	8	8	8	8	8	8	8
8								X	8	8	8	8	8	8	8	8
9									X	8	8	8	8	8	8	8
10										X	8	8	8	8	8	8
11											X	8	8	8	8	8
12												X	8	8	8	8
13													X	8	8	8
14														X	8	8
15															X	8
16																X

Gambar 3.5: Hasil tabel jarak Hamming pada $M = 16$ dan $n = 15$

```
1  Function generate_query(M, s)
       Data: M search space, s initial generator number
       Result: query
2      query = ""
3      for i = 0 to M do
4          bitstring = i & s
5          binary = 0
6          while bitstring do
7              binary ∧= bitstring & 1
8              bitstring >>= 1
9          end
10         query += binary
11     end
12     return query
13 end
```

Kode sumber 1: Algoritma membuat sebuah query dengan code generator

$$\text{Terdapat kode biner sempurna } (M-1, M, M/2)_2 \tag{3.1}$$

$$q_{rep} = d \setminus (M/2) \tag{3.2}$$

$$d_{mod} = d \bmod (M/2) \tag{3.3}$$

Oleh karena itu jika kita membutuhkan query yang menghasilkan $d = M/2$, maka gunakan seluruh $M-1$ query dengan generator. Sedangkan jika kita membutuhkan query yang menghasilkan $d < M/2$, pilih sesedikit mungkin n query agar menghasilkan d yang dibutuhkan. Dari kebutuhan tersebut, dibuatlah rumus q_{rep} dan d_{mod}. Nilai q_{rep} yang didapat dari Persamaan 3.2 adalah jumlah perulangan yang dilakukan terhadap $M-1$ query untuk mencapai solusi. Nilai d_{mod} yang didapat dari Persamaan 3.3 adalah sisa jarak Hamming d yang dibutuhkan untuk mencapai solusi. Cara untuk mencari jumlah query sisa untuk menghasilkan jarak Hamming distance d_{mod} ada pada subbab selanjutnya.

3.1.1.2. Pencarian mendalam dengan greedy

Algoritma solusi menggunakan generator matrix harus membuat urutan s tertentu sedemikian hingga seminimal mungkin jumlah query n yang dihasilkan dari $\vec{s} = \{s_1, \ldots, s_n\}$ akan menghasilkan semaksimal mungkin jarak Hamming d. Untuk itu dibuatlah algoritma pencarian mendalam *exhaustive search* dengan pendekatan *greedy* untuk mendapatkan urutan $\vec{s}$ yang paling optimal seperti yang ditunjukkan pada Kode Sumber 2.

Pertama-tama program membuat seluruh $M-1$ kemungkinan query menggunakan algoritma pembangkit query sehingga terbentuk $M-1$ query seperti yang ditunjukkan pada baris 6. Lalu pada baris 9, program melakukan perulangan pencarian mendalam untuk membuat query sampai batas jarak Hamming d yang diinginkan menggunakan. Dilakukan pendekatan *greedy* karena *global optimum* yang dicari adalah sebesar mungkin nilai jarak minimal dengan query sesedikit mungkin. *Global optimum* tersebut dapat didapat dengan mencari *local optimum* di setiap iterasi.

Pada algoritma *greedy*, fungsi seleksi untuk memilih kandidat query terbaik pada setiap iterasi ada empat macam:

1. Jarak minimum terbesar;
2. Jarak minimum terbesar lalu jarak maximum terkecil;
3. Pengurangan jarak maximum dengan jarak minimum terkecil;
4. Jarak minimum terbesar lalu jumlah angka yang memiliki nilai minimum terkecil.

Keempat fungsi seleksi tersebut ditentukan karena *global optimum* yang dicari adalah jarak minimum terbesar, selisih nilai jarak maximum dan jarak minimum harus sekecil mungkin, dan query yang dihasilkan dapat menghabiskan sebanyak mungkin angka dengan jarak minimum. Keempat fungsi seleksi akan diuji untuk dipilih fungsi seleksi terbaik dan digunakan pada program utama.

Pada baris 12, program melakukan perulangan untuk mencoba semua macam query yang belum digunakan untuk mencari yang terbaik, dan mengabaikan query yang sudah digunakan pada baris 13. Untuk mencari query yang terbaik pada state `order_pointer`,

parameter yang digunakan adalah sesuai dengan empat fungsi seleksi yang ditunjukkan pada baris 16. Setelah setiap satu pencarian best query pada satu state, program akan memperbarui jarak Hamming seperti yang ditunjukkan pada baris 23, lalu masukkan kandidat query terbaik ke `code_order` seperti yang ditunkukkan pada baris 29.

Keluaran dari algoritma ini dengan inputan $2 \leq 2^m \leq 4096$ dan $e = 16$ serta dengan fungsi seleksi terpilih akan menghasilkan sejumlah query beserta urutan seednya. Hasil query tersebut akan dimasukkan langsung ke dalam code pada solusi optimal pencarian Ulam non interaktif yang akan dijelaskan pada subbab selanjutnya.

3.1.1.3. Algoritma mencari query minimal

Dari keluaran algoritma mencari urutan kode biner, akan dibuat algoritma mencari query minimal untuk menghasilkan array $minimal$ berisi pasangan *key-value*, minimal jarak Hamming d sebagai *key* dan n jumlah query yang dibutuhkan untuk membuat codeword dengan jarak minimal jarak Hamming d. Pembuatan array $minimal$ pada baris 8. Array $minimal$ akan digunakan untuk mendapatkan nilai d_{mod} yang sebelumnya dijelaskan pada Persamaan 3.3. Dari algoritma ini, kita bisa mendapatkan q_{mod} dari d_{mod} dengan memasukkan d_{mod} ke index pada array $minimal$ seperti pada Persamaan 3.4.

$$q_{mod} = minimal[d_{mod}] \tag{3.4}$$

3.1.1.4. Menyatukan query repetisi dan query mod

Setelah mendapatkan array $minimal$, tahap terakhir adalah menyatukan query. Total query yang dihasilkan ada pada Persamaan 3.5. Query tersebut terdiri dari $q_{rep} \times M/2$ seperti ditunjukkan pada baris 4 dan q_{mod} seperti ditunjukkan pada baris 9.

$$q = q_{rep} \times (M - 1) + q_{mod} \tag{3.5}$$

3.1.2. Solusi menggunakan repetisi pencarian biner

Algoritma pencarian biner ada pada Kode Sumber 5. Baris 6 menunjukkan perulangan untuk setiap jenis query q_b. Nilai q_b adalah hasil $\log_2$ dari M, lalu dibulatkan ke atas karena M harus merupakan perpangkatan dari 2 sehingga nilai $q_b = \text{ceil}(\log_2(M))$. Baris 8 menunjukkan perulangan untuk membuat setiap satu jenis query pencarian biner. Baris 9 menunjukkan proses untuk mencari bit pada posisi tertentu pada sebuah integer [22]. Setiap query akan diulang sebanyak qe seperti yang ditunjukkan pada baris 13.

3.2. Desain pengujian keabsahan algoritma

Untuk menguji kebenaran query akan dibuat program yang dapat menguji query yang dihasilkan oleh algoritma solusi, agar tidak ada satu set pun yang menyebabkan ada lebih dari satu kemungkinan nilai x. Isi dari program pengujian kebenaran algoritma adalah mengecek apakah minimal jarak Hamming setiap query yang berbeda pada setiap kasus uji bernilai minimal $2e + 1$ seperti yang ditunjukkan pada Kode Sumber 6. Variabel $dist$ pada baris 2 digunakan untuk menyimpan jarak antara bit ke-0 dengan seluruh bit yang lain. Baris 4 menunjukkan bahwa jarak akan bertambah setiap ada perbedaan nilai bit.

3.3. Implementasi algoritma

Implementasi merupakan tahap untuk membangun algoritma yang akan digunakan. Pada tahap ini dilakukan implementasi dari rancangan struktur data yang akan dimodelkan sesuai dengan permasalahan. Implementasi dilakukan dengan menggunakan bahasa pemrograman C++ agar dapat disubmit ke SPOJ. Selain itu akan dilakukan implementasi program dalam bahasa C++ yang dapat mengecek kebenaran query dengan menghitung apakah jarak Hamming dari query sudah melampaui $2e + 1$.

```
1  Function exhaustive_search(M, e)
       Data: M search space, e max lies allowed
       Result: code_order
2      code[M] = [ ]
3      code_distance[M] = [ ]
4      code_min = 0
5      code_order[M] = [ ]
6      for i = 1 to M do
7          code[i] = generate_query(M, i)
8      end
9      while code_min < d do
10         most_min = 0
11         least_min_count = ∞
12         for i = 1 to M do
13             if code[i] already used then continue
14             min = minimal hamming distance if using code[i]
15             min_count = how many node which have hamming distance is min
16             if meet the greedy selection function then
17                 best = i
18                 most_min = min
19                 least_min_count = min_count
20             end
21         end
22         code_min = ∞
23         for j = 1 to M do
24             code_distance[j] += (code[best][0]! = code[best][j])
25             if code_distance[j] < code_min then
26                 code_min = code_distance[j]
27             end
28         end
29         code_order[order_pointer++] = best
30     end
31     return code_order
32 end
```

Kode sumber 2: Algoritma mencari urutan bibit generator terbaik

```
   Result: minimal array
 1 minimal = [ ]
 2 distances[M] = [ ]
 3 for i = 0 to M − 1 do
 4 |   min = ∞
 5 |   query = generate_query(M, code_order[m][i])
 6 |   update the distances after query
 7 |   min = min(distances)
 8 |   minimal[min] = i
 9 |   if min ≥ d then break
10 end
11 return minimal
```

Kode sumber 3: Algoritma mencari query minimal

```
   Result: solutions
 1 qrep = d/(M/2)
 2 dmod = d mod (M/2)
 3 solution = [ ]
 4 for i = 0 to qrep do
 5 |   for j = 0 to M − 1 do
 6 |   |   solution.push(code[j].substr(real_M))
 7 |   end
 8 end
 9 for i = 0 to minimal[dmod] do
10 |   solution.push(code[code_order[i]].substr(real_M))
11 end
12 return solution
```

Kode sumber 4: Algoritma membuat query yang sebenarnya

```
1  Function repetitive_binary_search(M, e)
       Data: M search space, e max lies allowed
       Result: queries
2      queries = [ ]
3      qb = ceil(log_2(M))
4      qe = 2 * e + 1
5      twopower = 1
6      for i = 0 to qb do
7          string = ""
8          for j = 0 to M do
9              if j & twopower then string += "1"
10             else  string += "0"
11
12         end
13         for j = 0 to qe do
14             queries.push(string)
15         end
16         twopower *= 2
17     end
18     return queries
19 end
```

Kode sumber 5: Algoritma repetisi pencarian biner

```
1  Function query_checker_search(M, e, queries)
       Data: M search space, e max lies allowed, queries to be checked
       Result: is_win
2      dist[M] = [0, . . . , 0]
3      for i = 1 to queries.size do
4          dist[j] += (queries[i][0] != queries[i][j])
5      end
6      for i = 1 to queries.size do
7          if dist[i] < 2 * e + 1 then
8              return false
9          end
10     end
11     return true
12 end
```

Kode sumber 6: Algoritma pengujian kebenaran query

BAB IV

HASIL DAN PEMBAHASAN

Bab ini menjelaskan uji coba yang dilakukan pada sistem yang telah dibangun beserta analisis dari uji coba yang telah dilakukan. Pembahasan pengujian meliputi lingkungan uji coba, skenario uji coba serta analisis setiap pengujian.

4.1. Skenario uji coba

Subbab ini akan menjelaskan hasil pengujian program penyelesaian permasalahan. Pengujian dilakukan untuk tiga metode: metode repetisi pencarian biner, kode biner, dan kode biner dengan tabel pencarian. Ada dua metode pengujian yang digunakan:

1. Pengujian lokal. Pengujian ini menggunakan mesin yang digunakan dalam pemgembang untuk mengukur kebenaran dan informasi hasil query, waktu eksekusi, dan penggunaan memori. Selain itu akan dilakukan pengujian untuk menentukan fungsi seleksi pada algoritma pencarian mendalam dengan pendekatan *greedy*
2. Pengujian luar. Pengujian ini menggunakan Online Judge untuk mengukur kebenaran dan informasi lain mengenai program.

4.2. Lingkungan Uji Coba

Pengujian dibagi menjadi dua jenis, yaitu dengan dataset buatan dan dengan dataset online. Uji coba dilakukan pada perangkat dengan spesifikasi berikut.

1. Perangkat Keras
   (a) Processor Intel® Core™ i7-7400 CPU @ 3.00GHz (4 CPUs), 3.0GHz
   (b) Random Access Memory 3837MB
2. Perangkat Lunak
   (a) Sistem Operasi Linux Ubuntu 16.04

(b) Bahasa Pemrograman C++

(c) gcc 5.4.0

3. Pengujian online SPOJ

(a) Time limit: 20s

(b) Source limit: 50000B

(c) Memory limit: 1536MB

(d) Cluster: Cube (Intel G860)

(e) Languages: C++ 4.3.2

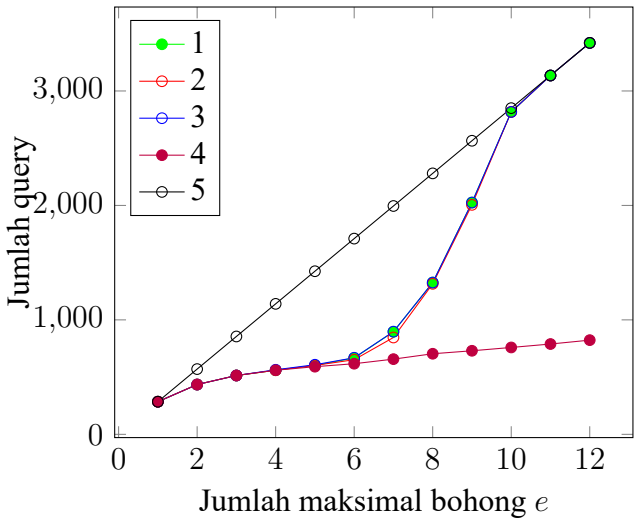
4.3. Uji coba menggunakan dataset lokal

Tahap pengujian adalah melakukan uji coba menggunakan dataset sesuai pada batasan permasalahan untuk mengetahui hasil dan performa dari algoritma dan struktur data yang dibangun. Pengujian juga akan digunakan untuk membandingkan jumlah query, waktu eksekusi, dan penggunaan memory dari algoritma dari semua kasus uji dataset. Dataset akan divariasikan sesuai dengan jumlah ruang pencarian M dan jumlah maksimal bohong e.

4.3.1. Hasil pengujian fungsi seleksi greedy

Pada pengujian empat fungsi seleksi *gredy*, dataset yang digunakan adalah ruang pencarian $M = 2^m \mid 1 \leq m \leq 12$ dan jumlah maksimal bohong $2 \leq e \leq 16$. Hasil yang akan dilihat adalah jumlah query dari setiap fungsi seleksi, semakin sedikit query pada semua kasus uji maka akan semakin bagus. Algoritma *greedy* dengan empat variasi fungsi seleksi akan dibandingkan dengan algoritma repetisi biner *ground truth*. Hasil algoritma repetisi biner juga akan menjadi batas maksimal query, jika jumlah query hasil algoritma *greedy* melebihi jumlah query hasil repetisi biner, maka jumlah query repetisi biner yang akan dipilih.

Pada Gambar 4.1, legenda nomor satu sampai lima secara berurutan adalah jarak minimum terbesar, jarak minimum terbesar lalu jarak maximum terkecil, pengurangan ja-



Gambar 4.1: Grafik pengujian fungsi seleksi greedy

rak maximum dengan jarak minimum terkecil, jarak minimum terbesar lalu jumlah angka memiliki nilai minimum terkecil, dan terakhir algoritma repetisi biner. Dari gambar tersebut dapat disimpulkan bahwa fungsi seleksi yang paling optimal adalah minimum terbesar diikuti dengan jumlah angka memiliki nilai minimum terkecil.

Tabel 4.1: Jumlah query keluaran dari Algoritma *exhaustive search*

M	jumlah query	M	jumlah query
2	1	128	72
4	3	256	76
8	7	512	79
16	15	1024	81
32	31	2048	84
64	63	4096	86

Setelah kandidat fungsi seleksi terpilih, keluaran dari algoritma pencarian mendalam dengan inputan $2 \leq 2^m \leq 4096$ dan $e = 16$ akan menghasilkan jumlah query seperti pada Tabel 4.1. Setiap sejumlah query pada setiap M ini akan menghasilkan $d = 33$ pada $128 \leq M \leq 4096$ atau telah menggunakan semua seed yang berjumlah $M - 1$ pada $2 \leq M \leq 64$. Hasil query tersebut akan dimasukkan langsung ke dalam code pada solusi optimal pencarian Ulam non interaktif, pada program utama seperti pada Gambar 4.2.

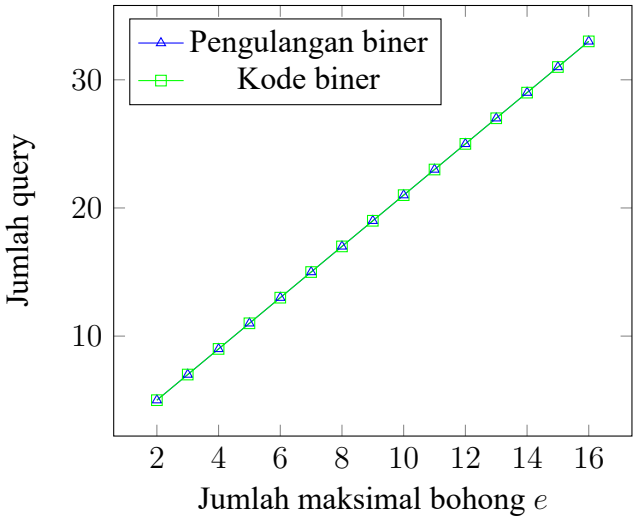
```
short minimal[MAX_M / 2 + 1]; // param: d needed; return: query

const int code_order[][MAX_QUERY] = {
    /* 0  */ {},
    /* 1  */ {1},
    /* 2  */ {1,2,3},
    /* 3  */ {1,2,4,7,3,5,6},
    /* 4  */ {1,2,4,8,15,3,5,9,14,6,10,13,7,11,12},
    /* 5  */ {1,2,4,8,16,31,7,9,19,29,10,22,12,15,17,18,20,21,24,27,30,3,
    /* 6  */ {1,2,4,8,16,32,63,7,25,42,52,13,22,35,56,14,19,37,11,48,28,5
    /* 7  */ {1,2,4,8,16,32,64,127,15,51,85,105,27,45,71,113,58,86,29,99,
    /* 8  */ {1,2,4,8,16,32,64,128,255,15,51,85,150,232,45,78,139,116,178
    /* 9  */ {1,2,4,8,16,32,64,128,256,511,31,99,165,302,470,203,344,146,
    /* 10 */ {1,2,4,8,16,32,64,128,256,512,1023,31,227,293,622,950,184,33
    /* 11 */ {1,2,4,8,16,32,64,128,256,512,1024,2047,63,455,585,1242,1899
    /* 12 */ {1,2,4,8,16,32,64,128,256,512,1024,2048,4095,63,455,1609,270
};

cin >> t;
while (t--) {
    cin >> M >> e >> max_query_allowed;

    the_real_M = M;
    M = pow(2, ceil(log2(M))); // M is the closest power(2)
```

Gambar 4.2: Potongan kode sumber tabel pencarian

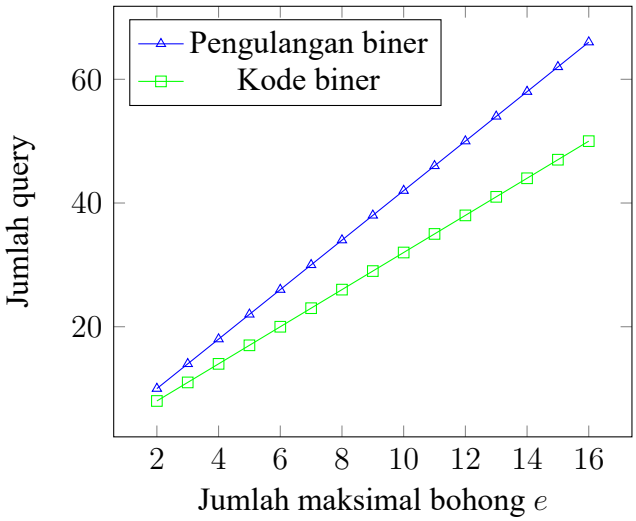
4.3.2. Hasil pengujian jumlah query

Pada pengujian jumlah query, pada setiap ruang pencarian $M = 2^m \mid 1 \leq m \leq 12$, pada setiap algoritma, diujikan 15 kasus uji sesuai dengan jumlah maksimal bohong $2 \leq e \leq 16$. Algoritma yang diujikan pada pengujian ini hanya ada dua yaitu algoritma kode biner dan algoritma repetisi biner karena algoritma kode biner dengan tabel pencarian dan algoritma kode biner tanpa tabel pencarian menghasilkan jumlah query yang sama.

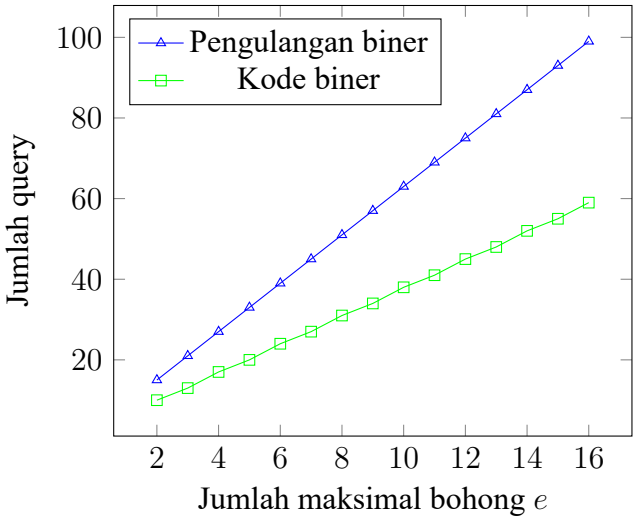
Pada Gambar 4.3 dapat dilihat bahwa jumlah query yang dihasilkan oleh algoritma kode biner dan repetisi biner sama. Namun pada kasus uji selanjutnya yaitu pada $4 \leq M \leq 4096$, algoritma yang diusulkan secara konsisten menunjukkan hasil yang lebih bagus karena menghasilkan jumlah query yang lebih sedikit, seperti yang ditunjukkan pada Gambar 4.4, Gambar 4.5, Gambar 4.6, Gambar 4.7, Gambar 4.8, Gambar 4.9, Gambar 4.10, Gambar 4.11, Gambar 4.12, dan Gambar 4.13. Dari hasil tersebut dapat disimpulkan bahwa algoritma kode biner menghasilkan jumlah query yang lebih sedikit untuk semua $M > 2$ dan untuk semua e.



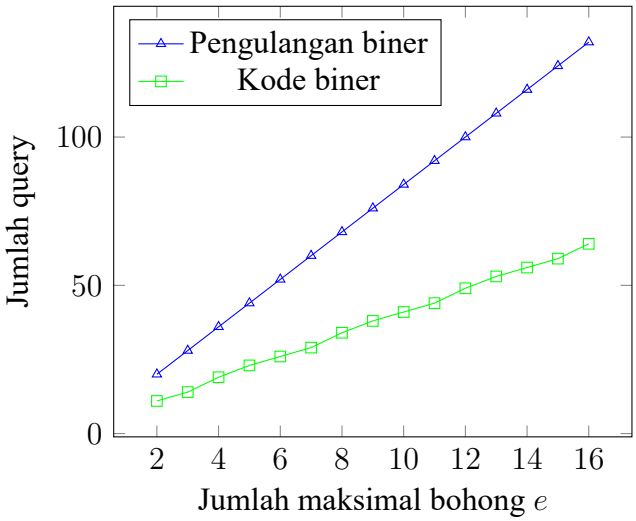
Gambar 4.3: Grafik perbandingan jumlah query pada $M = 2$



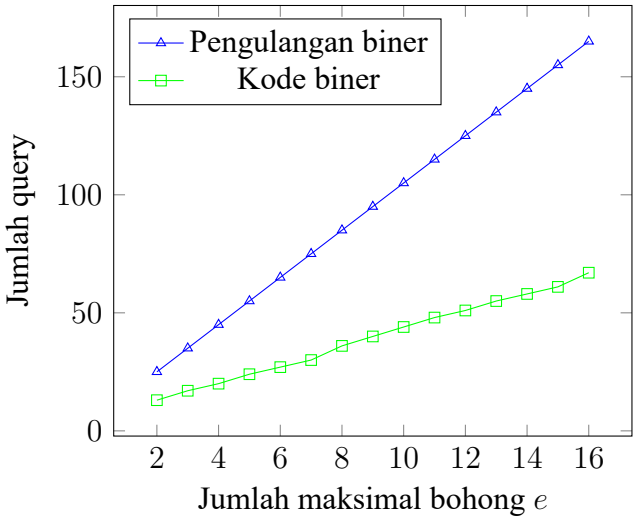
Gambar 4.4: Grafik perbandingan jumlah query pada $M = 4$



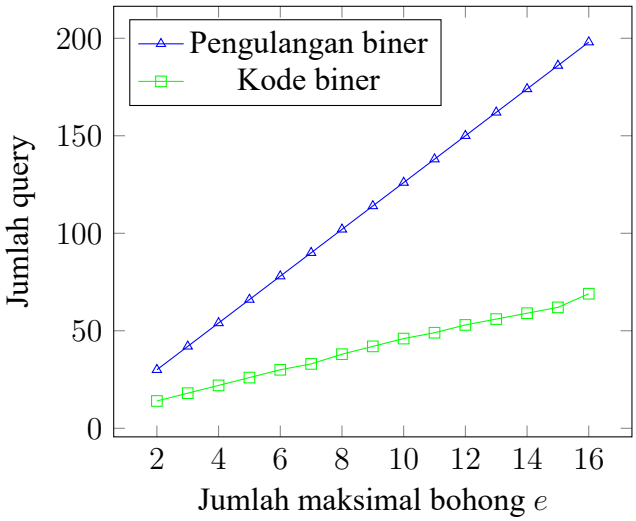
Gambar 4.5: Grafik perbandingan jumlah query pada $M = 8$



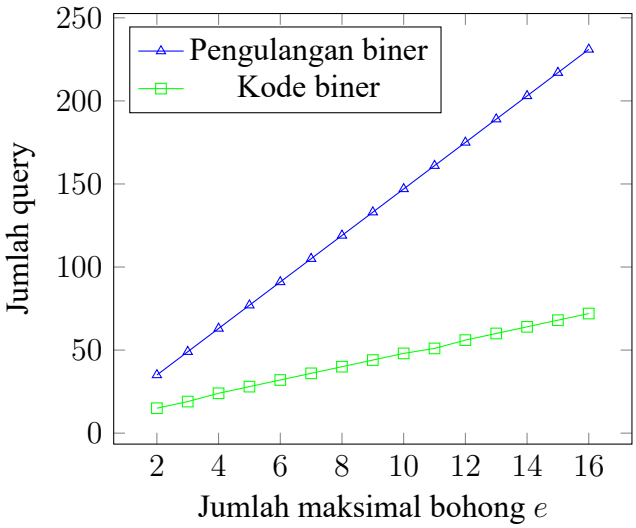
Gambar 4.6: Grafik perbandingan jumlah query pada $M = 16$



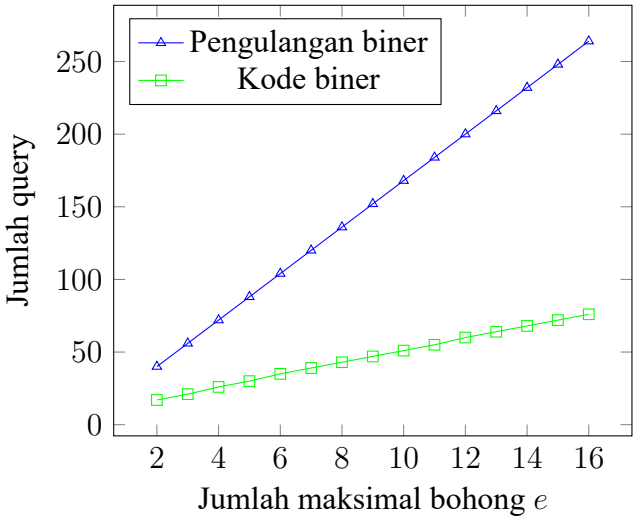
Gambar 4.7: Grafik perbandingan jumlah query pada $M = 32$



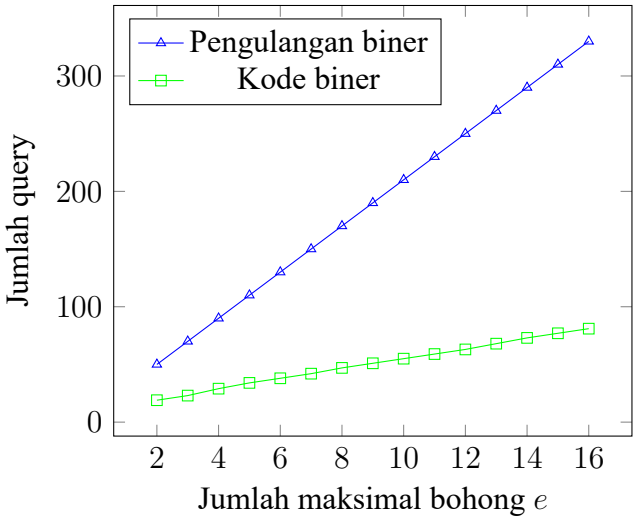
Gambar 4.8: Grafik perbandingan jumlah query pada $M = 64$



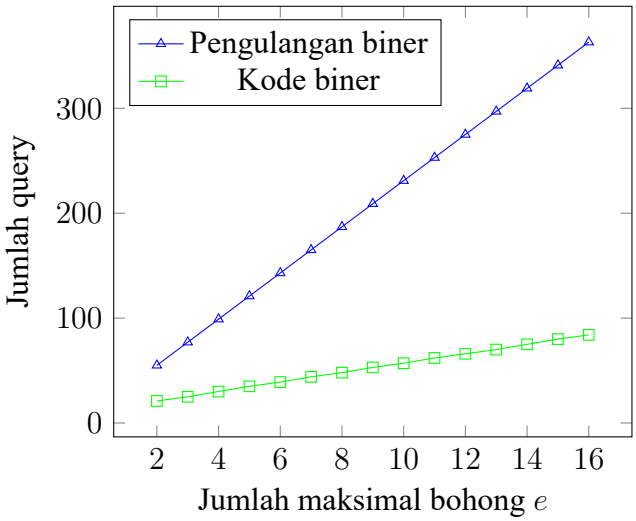
Gambar 4.9: Grafik perbandingan jumlah query pada $M = 128$



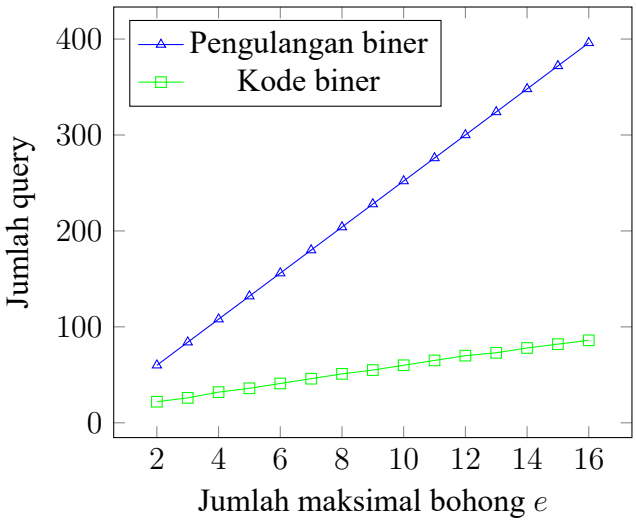
Gambar 4.10: Grafik perbandingan jumlah query pada $M = 256$



Gambar 4.11: Grafik perbandingan jumlah query pada $M = 1024$



Gambar 4.12: Grafik perbandingan jumlah query pada $M = 2048$



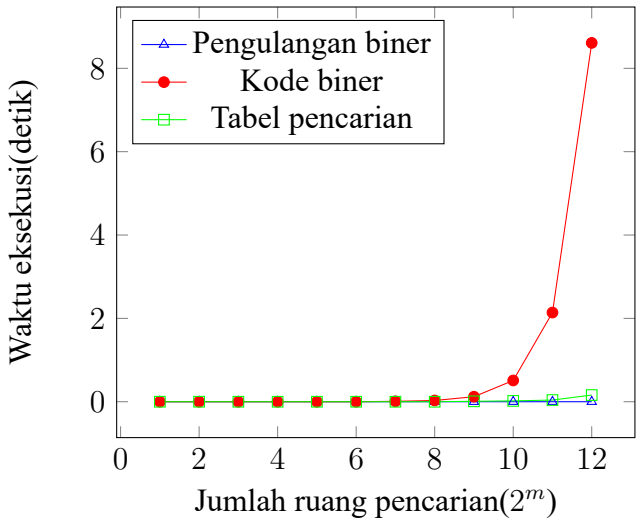
Gambar 4.13: Grafik perbandingan jumlah query pada $M = 4096$

4.3.3. Hasil pengujian waktu eksekusi

Pada pengujian waktu eksekusi, pada setiap algoritma, diujikan 12 kasus uji sesuai dengan jumlah variasi ruang pencarian $M = 2^m \mid 1 \leq m \leq 12$. Pada setiap pengujian, terdapat 15 kasus uji sesuai dengan variasi jumlah maksimal bohong $2 \leq e \leq 16$. Hasil pengujian ditunjukkan pada Gambar 4.14, dapat disimpulkan bahwa algoritma kode biner tanpa tabel pencarian menghasilkan laju peningkatan waktu yang tinggi, sedangkan untuk repetisi biner dan kode biner dengan tabel pencarian menghasilkan waktu yang stabil. Hal ini dikarenakan algoritma kode biner tanpa tabel pencarian memiliki kompleksitas $O(M^3)$ pada proses pencarian mendalam, sedangkan pada algoritma kode biner dengan tabel pencarian memiliki kompleksitas $O(1)$ karena proses pencarian mendalam dilakukan sebagai pra proses.

4.3.4. Hasil pengujian penggunaan memori

Pada pengujian penggunaan memori, pada setiap algoritma, diujikan 12 kasus uji sesuai dengan jumlah variasi ruang pencarian $1 \leq m \leq 12$. Pada setiap pengujian, terdapat 15 kasus uji sesuai dengan variasi jumlah maksimal bohong $2 \leq e \leq 16$. Hasil pengujian ditunjukkan pada Gambar 4.14, terlihat bahwa algoritma kode biner tanpa ta-



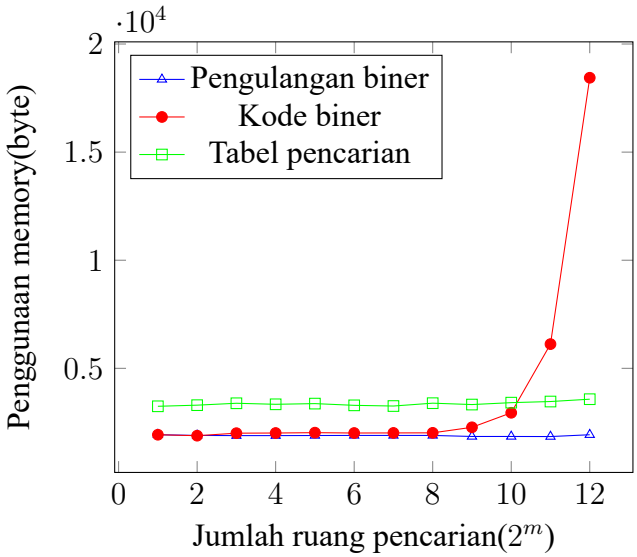
Gambar 4.14: Grafik perbandingan waktu eksekusi

bel pencarian menghasilkan laju peningkatan penggunaan memori yang tinggi, sedangkan untuk repetisi biner dan kode biner dengan tabel pencarian menghasilkan penggunaan memori yang stabil, namun algoritma repetisi biner menunjukkan penggunaan memori yang lebih sedikit.

4.4. Uji coba menggunakan dataset online SPOJ

Selain dataset buatan, program juga akan diuji dengan dataset pada Online Judge SPOJ GUESSN5. Hal yang dinilai pada pengujian adalah banyaknya query yang dibuat, waktu yang dibutuhkan, dan penggunaan memory. Pada pengujian online SPOJ terdapat 30 dataset, yaitu 15 dataset dengan 30 kasus uji dan 15 dataset dengan 99 kasus uji dengan persebaran ruang pencarian $2 \leq M \leq 4096$ dan maksimal bohong $2 \leq e \leq 16$.

Pembobotan skor adalah jika penjawab menemukan ada suatu set jawaban yang menyebabkan lebih dari satu kemungkinan nilai x, maka pengujian dianggap gagal. Jika berhasil, maka nilai skor bertambah q^2. Jika gagal, maka nilai skor bertambah $4m^2$. Total skor adalah jumlah semua skor dari setiap kasus uji. Semakin kecil skor maka semakin baik algoritma yang digunakan. Skor akan dibandingkan dengan pengajuan peserta lain pada SPOJ GUESSN5. Subbab ini akan menjelaskan hasil pengujian program penyelesa-



Gambar 4.15: Grafik perbandingan penggunaan memory

ian permasalahan menggunakan algoritma repetisi pencarian biner dan kode biner.

4.4.1. Uji coba algoritma repetisi pencarian biner pada SPOJ

Algoritma repetisi pencarian biner disubmit ke SPOJ dalam bahasa C, menghasilkan penilaian yang ditunjukkan pada Tabel 4.2. Karena ini adalah algoritma yang pasti benar dengan cara termudah, maka dapat diasumsikan bahwa skor yang didapat dari algoritma ini adalah skor minimal yang dapat menjadi tolok ukur keberhasilan algoritma yang lain.

Tabel 4.2: Hasil algoritma repetisi pencarian biner pada SPOJ

ID	20152331
Tanggal	2017-09-14 06:08:19
Skor	42,787,090
Waktu	0.00
Memori	2.7M

4.4.2. Uji coba algoritma kode biner pada SPOJ

Algoritma repetisi pencarian biner disubmit ke SPOJ dalam bahasa C, menghasilkan penilaian yang ditunjukkan pada Tabel 4.3. Algoritma ini menghasilkan error SIGSEGV

karena penggunaan memori melewati batas yang disediakan oleh SPOJ.

Tabel 4.3: Hasil algoritma kode biner pada SPOJ

ID	21726709
Tanggal	2018-05-26 01:10:34
Skor	runtime error (SIGSEGV)
Waktu	0.00
Memori	3.1M

4.4.3. Uji coba algoritma kode biner dengan tabel pencarian pada SPOJ

Algoritma repetisi pencarian biner dengan tabel pencarian disubmit ke SPOJ dalam bahasa C, menghasilkan penilaian yang ditunjukkan pada Tabel 4.4. Skor ini menjadi skor tertinggi pada SPOJ GUESSN5 seperti yang ditunjukkan pada Gambar 4.16.

Tabel 4.4: Hasil algoritma kode biner pada SPOJ

ID	21732463
Tanggal	2018-05-27 00:45:01
Skor	4,225,555
Waktu	0.44
Memori	2.8M

Guess The Number With Lies v5 statistics & best solutions

Users accepted	Submissions	Accepted	Wrong Answer	Compile Error	Runtime Error	Time Limit Exceeded
5	21	16	1	0	4	0

All ADA95 ASM32 ASM32-GCC MAWK GAWK BASH BC BF C-CLANG C CSHARP CPP C++ 4.3.2 CPP14-CLANG CPP14 C99 CLPS CLOJURE COBOL COFFEE LISP sbcl LISP clisp D-DMD D-CLANG D DART ELIXIR ERL FSHARP FANTOM FORTH FORTRAN GO GOSU GRV HASK ICON ICK JAR JAVA JS-MONKEY JS-RHINO KTLN LUA NEM NICE NIM NODEJS OBJC OBJC-CLANG OCAML OCT PAS-FPC PAS-GPC PERL PERL6 PHP PICO PIKE PRLG-swi PROLOG PYTHON PYPY PYTHON3 PY_NBC R RACKET RUBY RUST SCALA SCM guile SCM qobi CHICKEN SED ST SQLITE SWIFT TCL UNLAMBDA VB.NET WHITESPACE

RANK	DATE	USER	RESULT	TIME	MEM	LANG
1	2018-05-27 00:45:01	risyanggi	**4225555**	0.44	2.8M	C++ 4.3.2
2	2018-05-27 02:34:05	Rully Soelaiman	**4225555**	0.25	16M	CPP
3	2014-01-05 22:11:52	Mitch Schwartz	**5775304**	11.51	12M	PYTHON3
4	2017-01-30 04:43:49	Stilwell	**37537878**	0.03	2.8M	C++ 4.3.2
5	2015-09-03 21:41:25	efekcik	**186469552**	0.00	2.1M	C

Gambar 4.16: Urutan ranking solusi permasalahan GUESSN5 pada SPOJ

BAB V

KESIMPULAN DAN SARAN

Bab ini membahas mengenai kesimpulan dan saran yang dapat diambil dari hasil uji coba yang telah dilakukan sebagai jawaban dari rumusan masalah yang dikemukakan.

5.1. Kesimpulan

Berdasarkan penjabaran di bab-bab sebelumnya, dapat disimpulkan beberapa poin terkait penyelesaian permasalahan Ulam non interaktif.

1. Permasalahan Ulam non interaktif dapat diselesaikan dengan algoritma repetisi biner maupun kode biner dengan jarak Hamming.
2. Algoritma kode biner dengan jarak Hamming memiliki hasil yang lebih baik dibandingkan algoritma repetisi biner dilihat dari sisi jumlah query.
3. Pada algoritma kode biner terdapat proses pencarian mendalam yang memiliki kompleksitas $O(M^3)$, namun dapat direduksi menjadi $O(1)$ dengan melakukan proses pencarian mendalam sebagai pra proses dan menghasilkan waktu dan penggunaan memori yang sama bagusnya dengan algoritma repetisi biner.
4. Solusi meggunakan algoritma kode biner dengan tabel pencarian dapat diterima pada pengujian online SPOJ dan menghasilkan peringkat paling tinggi dibandingkan solusi yang sudah diselesaikan peserta lain.

5.2. Saran

Algoritma solusi menggunakan kode biner dengan pra proses pencarian mendalam terkendala dengan pra proses yang harus dilakukan dengan batasan jumlah bohong dan ruang pencarian yang harus ditentukan sejak awal. Saran untuk penelitian terkait adalah bagaimana solusi dapat dilakukan secara dinamis tanpa harus menambahkan kode yang didapat dari pra proses.

[*Halaman ini sengaja dikosongkan*]

DAFTAR PUSTAKA

[1] A. Pelc, “Searching games with errors - Fifty years of coping with liars”, *Theoretical Computer Science*, vol. 270, no. 1-2, pp. 71–109, 2002, ISSN: 03043975. DOI: 10.1016/S0304-3975(01)00303-6.

[2] S. M. Ulam, *Adventures of a mathematician*. Scribner, 1976.

[3] A. Dhagat, P. Gacs, and P. Winkler, “On Playing ”Twenty Questions” with a Liar”, *Proceedings of the third annual ACM-SIAM symposium on Discrete algorithms*, 1999.

[4] R. B. Ellis, V. Ponomarenko, and C. H. Yan, “How to play the one-lie Renyi–Ulam game”, vol. 308, pp. 5805–5808, 2008. DOI: 10.1016/j.disc.2007.09.052.

[5] A. Pelc, “Prefix Search with a Lie”, *International Journal of Mathematics and Mathematical Sciences*, vol. 48, pp. 165–173, 1988.

[6] F. Cicalese and D. Mundici, “Perfect two-fault tolerant search with minimum adaptiveness”, *Advances in Applied Mathematics*, vol. 25, no. 1, pp. 65–101, 2000, ISSN: 01968858. DOI: 10.1006/aama.2000.0688.

[7] A. Negro and M. Sereno, “Ulam’s Searching Game with Three Lies”, *Advances in Applied Mathematics*, vol. 13, pp. 404–428, 1992.

[8] E. Berlekamp, R. Hill, and J. Karim, “The solution of a problem of Ulam on searching with lies”, *International Symposium on Information Theory*, p. 244, 1998.

[9] C. Deppe, “Strategies for the Renyi–Ulam Game with fixed number of lies”, *Theoretical Computer Science*, vol. 314, pp. 45–55, 2004. DOI: 10.1016/j.tcs.2003.10.036.

[10] D. Innes, “Searching with a Lie Using Only Comparison Questions”, 1992.

[11] V. Auletta, A. Negro, and G. Parlati, “Some results on searching with lies”, *Proc. 4th Italian Conf. on Theoretical Computer Science, L’Aquila, Italy*, no. May, pp. 24–37, 1992. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.57.5209.

[12] J. S. Peter, “Penyelesaian permasalahan Ulam pada permasalahan SPOJ klasik 17320 Guess The Number With Lies v3”, PhD thesis, 2017, p. 117, ISBN: 1970021319940.

[13] D. Mundici and A. Trombetta, “Optimal comparison strategies in Ulam’s searching game with two errors”, *Theoretical Computer Science*, vol. 182, pp. 217–232, 1997.

[14] G. O. H. Katona, “Search with small sets in presence of a liar”, *Journal of Statistical Planning and Inference*, vol. 100, pp. 319–336, 2002.

[15] A. J. Macula, “A nonadaptive version of Ulam’s problem with one lie”, *Journal of Statistical Planning and Inference*, vol. 61, no. 1, pp. 175–180, 1997, ISSN: 0378-3758. DOI: 10.1016/S0378-3758(96)00145-0.

[16] M. Miodek, *GUESSN5 - Guess The Number With Lies v5*, http://www.spoj.com/problems/GUESSN5/, 2013.

[17] A. Pelc, “Solution of Ulam’s Problem on Searching with a Lie”, *Journal of Combinatorial Theory*, vol. 44, pp. 129–140, 1987.

[18] R. Ahlswede, F. Cicalese, and C. Deppe, “Searching with lies under error cost constraints”, *Discrete Applied Mathematics*, vol. 156, pp. 1444–1460, 2008. DOI: 10.1016/j.dam.2007.04.033.

[19] R. B. Ellis, V. Ponomarenko, and C. H. Yan, “The Rényi–Ulam pathological liar game with a fixed number of lies”, *Journal of Combinatorial Theory*, vol. 112, pp. 328–336, 2005.

[20] J. H. van Lint, *Introduction to coding theory*. 2016, vol. 201, pp. vii+136, ISBN: 9781482299809.

[21] W. C. Huffman and V. Pless, *Fundamentals of Error-Correcting Codes*. Cambridge University Press, 2003, ISBN: 9780521782807.

[22] S. E. Anderson, *Bit Twiddling Hacks*, http://www.graphics.stanford.edu/~seander/bithacks.html, 2005.

[*Halaman ini sengaja dikosongkan*]

BIODATA PENULIS

Penulis, Risyanggi Azmi Faizin, lahir di kota Surabaya pada tanggal 29 Juli 1993. Penulis dibesarkan di kota Sidoarjo, Jawa Timur.

Penulis menempuh pendidikan formal di SD Al-Hikmah Surabaya (1999-2005), SMP Al-Hikmah Surabaya (2005-2008), SMAN 15 Surabaya (2008-2011). Pada tahun 2012, penulis melanjutkan pendidikan S1 Departemen Informatika Fakultas Teknologi Informasi dan Komunikasi di Institut Teknologi Sepuluh Nopember, Surabaya, Jawa Timur.

Di Departemen Informatika, penulis mengambil bidang minat Algoritma dan Pemrograman atau biasa disingkat menjadi Alpro dan memiliki ketertarikan di bidang desain web, perancangan perangkat lunak, big data, dan Pemodelan 3D. Penulis aktif sebagai vokalis di Paduan Suara Mahasiswa ITS. Penulis dapat dihubungi melalui alamat surel (email) risyanggi@gmail.com.