TUGAS AKHIR – KI141502

PENYELESAIAN PERMASALAHAN ULAM PADA PERMASALAHAN SPOJ KLASIK 17320 GUESS THE NUMBER WITH LIES V3

JOHN STEPHANUS PETER
5113100106

Dosen Pembimbing
Rully Soelaiman, S.Kom., M.Kom.
Fajar Baskoro, S.Kom., M.T.

DEPARTEMEN TEKNIK INFORMATIKA
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember
Surabaya 2017

TUGAS AKHIR – KI141502

PENYELESAIAN PERMASALAHAN ULAM PADA PERMASALAHAN SPOJ KLASIK 17320 GUESS THE NUMBER WITH LIES V3

JOHN STEPHANUS PETER
5113100106

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

Dosen Pembimbing II
Fajar Baskoro, S.Kom., M.T.

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

[*Halaman ini sengaja dikosongkan*]

FINAL PROJECT - KI141502

SOLUTION OF ULAM'S PROBLEM ON CLASSICAL SPOJ PROBLEM 17320 GUESS THE NUMBER WITH LIES V3

JOHN STEPHANUS PETER
5113100106

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

Supervisor II
Fajar Baskoro, S.Kom., M.T.

DEPARTMENT OF INFORMATICS
FACULTY OF INFORMATION TECHNOLOGY
Sepuluh Nopember Institute of Technology
Surabaya, 2017

[*Halaman ini sengaja dikosongkan*]

LEMBAR PENGESAHAN

PENYELESAIAN PERMASALAHAN ULAM PADA PERMASALAHAN SPOJ KLASIK 17320 GUESS THE NUMBER WITH LIES V3

TUGAS AKHIR

Diajukan Untuk Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Rumpun Mata Kuliah Algoritma Pemrograman
Program Studi S-1 Departemen Teknik Informatika
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember

Oleh:
John Stephanus Peter
NRP: 5113 100 106

Disetujui oleh Dosen Pembimbing Tugas Akhir:

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

Fajar Baskoro, S.Kom., M.T.
NIP. 197404031999031002 ..... (Pembimbing 2)

SURABAYA
JULI 2017

[*Halaman ini sengaja dikosongkan*]

PENYELESAIAN PERMASALAHAN ULAM PADA PERMASALAHAN SPOJ KLASIK 17320 GUESS THE NUMBER WITH LIES V3

Nama Mahasiswa : John Stephanus Peter
NRP : 5113 100 106
Departemen : Teknik Informatika FTIf - ITS
Dosen Pembimbing 1 : Rully Soelaiman, S.Kom., M.Kom.
Dosen Pembimbing 2 : Fajar Baskoro, S.Kom, M.T.

Abstrak

Permasalahan Ulam adalah permasalahan klasik yang dimulai pada tahun 1976. Permasalahan asli dari Permasalahan Ulam adalah permasalahan mencari banyak pertanyaan minimum untuk mencari sebuah bilangan dari 1 hingga 1.000.000, jika penjawab boleh berbohong paling banyak satu kali selama proses pencarian.

Pada Tugas Akhir ini, akan dirancang penyelesaian permasalahan Ulam dalam pencarian bilangan pada rentang yang diberikan, dengan batasan maksimal rentang adalah 10^{18}. Permasalahan ini dapat diselesaikan dengan menggunakan beberapa teori dan lemma, dan membuat 2 buah set, yaitu truth-set dan lie-set. Beberapa hal yang harus diperhatikan adalah mengenai cara mengatasi overflow dan mengatur elemen-elemen pada truth-set dan lie-set. Tipe data yang banyak digunakan pada permasalahan ini adalah long double dan long long, sedangkan struktur data yang digunakan untuk mengatur elemen pada truth-set maupun lie-set adalah struktur data set dengan tipe pair.

Hasil dari tugas akhir ini telah berhasil menyelesaikan permasalahan di atas dengan cukup efisien, dengan rata-rata waktu penyelesaian 1.000 permainan dalam 0,17 detik dengan penggunaan memori 2,8 MB.

Kata kunci: Permasalahan Ulam, Set, Pair, Lemma, Bohong

[*Halaman ini sengaja dikosongkan*]

SOLUTION OF ULAM'S PROBLEM ON CLASSICAL SPOJ PROBLEM 17320 GUESS THE NUMBER WITH LIES V3

Student Name	: John Stephanus Peter
Registration Number	: 5113 100 106
Department	: Informatics Department Faculty of IT – ITS
First Supervisor	: Rully Soelaiman, S.Kom., M.Kom.
Second Supervisor	: Fajar Baskoro, S.Kom, M.T.

Abstract

Ulam's problem is a classic mathematical problem stated on 1976. The original Ulam's Problem was about to find the minimal number of yes-no questions needed to find an integer between one and one million, if one lie is allowed among the answers.

This undergraduate thesis will design the problem solving of modified Ulam's problem on searching a number with a given range, with the maximum constraint of the range is 10^{18}. This problem is solved within the help of 2 sets, truth-set and lie-set, and using some lemmas. Some conditions which should be carefully considered during the implementation including how to handle overflow and how to manage the elements of the truth-set and lie-set. Long double and long long data type is frequently used in this undergraduate thesis, with the combination with set and pair data structure.

The result shows that Ulam's problem is successfully solved efficiently with average time of 0.17 seconds and memory use of 2.8 MB to solve 1,000 games.

Keywords: Ulam's problem, Set, Pair, Lemma, Lies.

[*Halaman ini sengaja dikosongkan*]

KATA PENGANTAR

Puji syukur penulis panjatkan kepada Tuhan Yang Maha Esa atas pimpinan, penyertaan, dan karunia-Nya sehingga penulis dapat menyelesaikan Tugas Akhir yang berjudul :

PENYELESAIAN PERMASALAHAN ULAM PADA PERMASALAHAN SPOJ KLASIK 17320 *GUESS THE NUMBER WITH LIES V3*

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

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

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

1. Tuhan Yesus Kristus atas segala hikmat, kesehatan, dan penyertaan selama ini.
2. Ayah, ibu, dan keluarga penulis yang selalu memberikan dukungan, perhatian, dan kasih sayang bagi penulis yang menjadi semangat selama perkuliahan maupun pengerjaan Tugas Akhir ini.
3. Bapak Rully Soelaiman, S.Kom., M.Kom. selaku dosen pembimbing yang telah banyak meluangkan waktu untuk memberikan ilmu, nasihat, motivasi, bimbingan dan didikan kepada penulis dengan sabar selama perkuliahan maupun pengerjaan Tugas Akhir ini.
4. Bapak Fajar Baskoro, S.Kom., M.T. selaku dosen pembimbing yang telah memberikan masukan dan bimbingan kepada penulis selama pengerjaan Tugas Akhir ini.

5. Seluruh tenaga pengajar dan karyawan Departemen Teknik Informatika ITS yang telah memberikan tenaga dan waktunya demi kelancaran proses belajar mengajar penulis selama perkuliahan.
6. Cynthia Dewi T. yang telah membantu penulis dalam membuat ilustrasi dalam penyusunan Tugas Akhir ini.
7. Teman-teman administrator Laboratorium Pemrograman 2 yang telah menerima penulis dan menjadi keluarga penulis selama di kampus.
8. Teman-teman dari rumpun mata kuliah dan laboratorium Algoritma dan Pemrograman yang memberikan dukungan dan semangat kepada penulis selama pengerjaan Tugas Akhir ini.
9. Teman-teman angkatan 2013 jurusan Teknik Informatika ITS yang telah menemani perjuangan penulis selama 4 tahun masa perkuliahan.
10. Serta pihak-pihak lain yang tidak dapat disebutkan disini yang telah banyak membantu penulis dalam penyusunan Tugas Akhir ini.

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

Surabaya, Juli 2017

John Stephanus Peter

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

[*Halaman ini sengaja dikosongkan*]

DAFTAR KODE SUMBER

[*Halaman ini sengaja dikosongkan*]

BAB I
PENDAHULUAN

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

1.1 Latar Belakang

Perkembangan dunia teknologi informasi selama beberapa dekade terakhir sangatlah pesat. Dalam perkembangannya, 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 Ulam.

Pada tahun 1976, S. M. Ulam mengangkat sebuah permasalahan, yaitu berapakah pertanyaan ya-tidak minimal yang diperlukan untuk menemukan sebuah bilangan diantara satu hingga satu juta, jika diperbolehkan terdapat satu jawaban bohong diantara semua respon[1]. Permasalahan klasik ini selanjutnya diselesaikan secara matematis oleh Andrzej Pelc[2]. Dalam Tugas Akhir ini, permasalahan klasik Ulam diatas akan dikembangkan. Sehingga, rentang bilangan yang diperbolehkan dalam permainan adalah 1 hingga 10^{18}. Selain itu, solusi dari permasalahan Ulam akan diimplementasikan dalam bentuk kode.

Hasil dari Tugas Akhir ini diharapkan dapat mengimplementasikan solusi dari pengembangan permasalahan Ulam dengan pertanyaan seminimal mungkin sehingga diharapkan dapat memberikan kontribusi pada pengembangan ilmu pengetahuan dan teknologi informasi.

1.2 Rumusan Permasalahan

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

1. Bagaimana mencari sebuah bilangan diskrit pada interval yang diberikan pada permasalahan SPOJ Klasik 17320 *Guess The Number With Lies v3* dengan pertanyaan seminimal mungkin?
2. Bagaimana implementasi algoritma dan struktur data yang efisien dan optimal untuk menyelesaikan permasalahan SPOJ Klasik 17320 *Guess The Number With Lies v3*?

1.3 Batasan Permasalahan

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

1. Implementasi algoritma menggunakan bahasa pemrograman C++.
2. Batas maksimum kasus uji adalah 2^{10}.
3. Interval bilangan yang yang dicari berada pada [1,n], dengan n maksimum 10^{18}.
4. *Dataset* yang digunakan adalah *dataset* pada problem SPOJ 17320 *Guess The Number With Lies v3* (GUESSN3).

1.4 Tujuan Pembuatan Tugas Akhir

Tujuan dari pembuatan Tugas Akhir ini adalah sebagai berikut:

1. Melakukan analisis dan mendesain algoritma dan struktur data untuk mencari bilangan dengan kebohongan dalam studi kasus permasalahan SPOJ Klasik 17320 Guess The Number With Lies v3.
2. Mengevaluasi hasil dan kinerja algoritma dan struktur data yang telah dirancang untuk permasalahan SPOJ Klasik 17320 Guess The Number With Lies v3.

1.5 Manfaat Tugas Akhir

Tugas Akhir ini diharapkan dapat mengimplementasikan solusi dari permasalahan Ulam sehingga dapat memberikan

kontribusi pada pengembangan ilmu pengetahuan dan teknologi informasi.

1.6 Metodologi

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

1.6.1 Penyusunan Proposal Tugas Akhir

Tahap awal untuk memulai pengerjaan Tugas Akhir adalah penyusunan proposal Tugas Akhir. Pada proposal ini, penulis mengajukan gagasan untuk menyelesaikan permasalahan deteksi kesalahan pada studi kasus SPOJ klasik 17320 Guess The Number With Lies v3.

1.6.2 Studi Literatur

Tahap kedua adalah mencari informasi dan studi literatur yang relevan untuk dijadikan referensi dalam melakukan pengerjaan Tugas Akhir. Informasi dan studi literatur ini didapatkan dari buku, *internet*, dan materi-materi kuliah yang berhubungan dengan metode yang akan digunakan.

1.6.3 Implementasi Perangkat Lunak

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 ini dikakukan dengan menggunakan bahasa pemrograman C++.

1.6.4 Pengujian dan Evaluasi

Tahap berikutnya adalah melakukan uji coba menggunakan *dataset* pada *Online Judge* SPOJ Klasik 17320 Guess The Number With Lies v3 untuk mengetahui hasil dan performa dari metode dan struktur data yang dibangun. Evaluasi dan perbaikan juga akan dilakukan pada *Online Judge* hingga perangkat lunak yang diuji mengeluarkan hasil dan performa yang sesuai dengan data uji pada *Online Judge* SPOJ.

1.6.5 Penyusunan Buku Tugas Akhir

Pada tahap ini dilakukan penyusunan laporan yang menjelaskan dasar teori dan metode yang digunakan dalam tugas akhir ini serta hasil dari implementasi aplikasi perangkat lunak yang telah dibuat.

1.7 Sistematika Penulisan

Buku Tugas Akhir ini merupakan laporan secara lengkap mengenai Tugas Akhir yang telah dikerjakan baik dari sisi teori, rancangan, maupun implementasi sehingga memudahkan bagi pembaca dan juga pihak yang ingin mengembangkan lebih lanjut. Sistematika penulisan buku Tugas Akhir secara garis besar antara lain:

Bab I Pendahuluan

Bab ini berisi penjelasan latar belakang, rumusan masalah, batasan masalah dan tujuan pembuatan Tugas Akhir. Selain itu, metodologi pengerjaan dan sistematika penulisan laporan Tugas Akhir juga terdapat di dalamnya.

Bab II Dasar Teori

Bab ini berisi penjelasan secara detail mengenai dasar-dasar penunjang dan teori-teori yang digunakan untuk mendukung pembuatan Tugas Akhir ini.

Bab III Analisis dan Perancangan Sistem

Bab ini berisi penjelasan tentang rancangan dari sistem yang akan dibangun.

Bab IV Implementasi

Bab ini berisi penjelasan implementasi dari rancangan yang telah dibuat pada bab sebelumnya. Implementasi disajikan dalam bentuk *pseudocode* disertai dengan penjelasannya.

Bab V Pengujian dan Evaluasi

Bab ini berisi penjelasan mengenai data hasil percobaan dan pembahasan mengenai hasil percobaan yang telah dilakukan.

Bab VI Kesimpulan dan Saran

Bab ini merupakan bab terakhir yang menyampaikan kesimpulan dari hasil uji coba yang dilakukan dan saran untuk pengembangan perangkat lunak ke depannya.

[*Halaman ini sengaja dikosongkan*]

BAB II
DASAR TEORI

Pada bab ini akan dijelaskan mengenai dasar teori yang mejadi dasar pengerjaan Tugas Akhir ini.

2.1 Deskripsi Umum Permasalahan

Permasalahan yang diangkat pada Tugas Akhir ini adalah permasalahan untuk mencari sebuah bilangan diskrit pada rentang [1..n] yang diberikan dengan jumlah pertanyaan seminimal mungkin. dengan batasan bahwa untuk setiap pertanyaan yang diajukan penjawab dapat memberikan jawaban bohong maksimal satu kali untuk setiap permainan. Pertanyaan yang dapat diajukan berupa sebuah pertanyaan rentang [l,r], yang berarti "Apakah bilangan yang dimaksud berada diantara rentang l dan r, secara inklusif?". Penjawab selanjutnya dapat memberikan respon berupa "YES" jika benar bilangan yang dimaksud berada diantara rentang tersebut dan "NO" jika bilangan yang dimaksud tidak berada diantara rentang tersebut. Penjawab dapat berbohong seperti batasan yang telah diberikan diatas.

Permasalahan utama pada persoalan ini adalah mencari sebuah strategi untuk menemukan bilangan yang dicari dengan jumlah pertanyaan sesedikit mungkin. Yang dimaksud dengan *k* pertanyaan sesedikit mungkin adalah dipastikan tidak terdapat strategi yang dapat menemukan bilangan yang dimaksud dengan pertanyaan $<k$.

Permasalahan ini dapat diumpamakan menjadi sebuah permainan, dimana permainan ini dimenangkan oleh penanya jika jumlah pertanyaan yang ditanyakan tidak lebih dari batas pertanyaan yang diperbolehkan, yaitu k, sehingga dalam kasus ini penanya akan berusaha meminimalkan pertanyaan yang diajukan, sedangkan penjawab akan berusaha memaksimalkan pertanyaan yang diajukan.

2.2 Deskripsi Umum Struktur Data

2.2.1 *Pair*

Pair adalah sebuah struktur yang dapat menyimpan dua objek heterogen menjadi sebuah kesatuan[3]. *Pair* sama seperti sebuah tipe data *tuple*, hanya saja *pair* hanya menyimpan dua elemen.

2.2.2 *Set*

Set adalah sebuah *associative container* yang berisi himpunan objek unik yang terurut. Objek pada *set* diurutkan dengan menggunakan konsep *strict weak ordering*, yang berarti sebuah objek dikatakan ekuivalen dengan objek lain jika terdapat dua buah objek a dan b dan keduanya memenuhi !comp(a,b) dan !comp(b,a), dimana fungsi comp(a,b) akan mengembalikan nilai *true* jika a < b.

Pada *set*, operasi pencarian, penambahan, dan penghapusan elemen dapat dilakukan dengan kompleksitas logaritmik. Hal ini disebabkan karena *set* merupakan implementasi dari *balanced binary search tree*, yaitu red-black tree [4].

2.3 Strategi Penyelesaian Permasalahan

2.3.1 Definisi Domain Solusi

Dasar dari permasalahan diatas adalah permasalahan Ulam, dimana pertanyaannya adalah berapakah pertanyaan ya-tidak minimum yang diperlukan untuk mencari sebuah bilangan diantara 1 – 1.000.000, jika diperbolehkan bohong maksimal satu kali.

Dalam jurnal yang ditulis Andrzej Pelc[2], setiap saat penanya akan memberikan pertanyaan, permainan ini dapat dinyatakan dalam sebuah *state* yang dinyatakan dalam bentuk (a, b) yang merupakan bilangan asli. Bilangan a menyatakan isi dari *truth-set*. *Truth-set* didefinisikan sebagai himpunan bilangan {1,…, n} yang memenuhi semua jawaban sebelumnya, atau dengan kata lain, sebuah himpunan bilangan dengan asumsi penjawab belum pernah berbohong. Bilangan b menyatakan isi dari *lie-set*. *Lie-set* didefinisikan sebagai himpunan bilangan {1,…,

n} yang memenuhi semua jawaban sebelumnya kecuali satu jawaban, atau dengan kata lain, sebuah himpunan bilangan dengan asumsi penjawab telah berbohong sekali.

Dengan menggunakan ide dari Berlekamp[5], bobot dari *state* (a, b) dengan sisa j pertanyaan dapat didefinisikan sebagai

$$w_j(a, b) = a(j + 1) + b$$

Menurut Spencer[6], bobot diatas dapat diterjemahkan bahwa isi dari *truth-set* dapat memiliki $j + 1$ kemungkinan berbohong. Yaitu, berbohong satu kali dari j pertanyaan yang tersisa, atau tidak berbohong sama sekali, sedangkan isi dari *lie-set* hanya memiliki satu kemungkinan, yaitu terpaksa memberikan jawaban yang jujur hingga permainan berakhir.

Sebuah pertanyaan dari *state* (a, b) akan menghasilkan dua kemungkinan *state*, yaitu (a_1, b_1) dan (a_2, b_2), masing-masing untuk jawaban ya dan tidak. Dapat dilihat bahwa jika *state* (a, b) menghasilkan *state* (a_1, b_1) dan *state* (a_2, b_2), maka

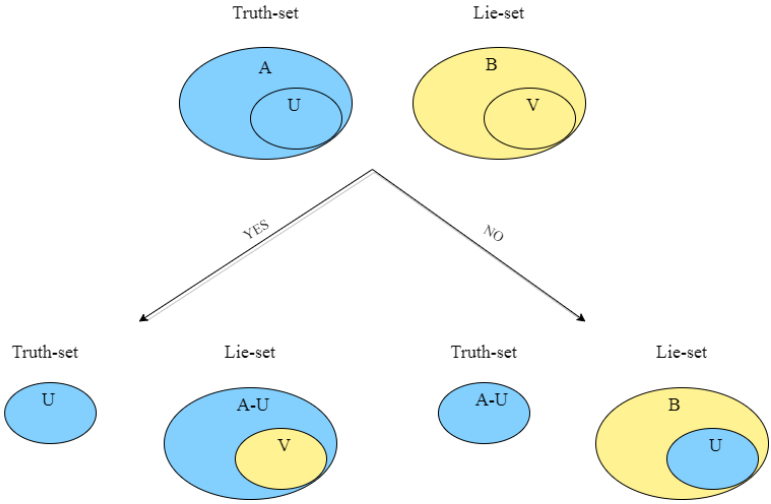
$$w_k(a, b) = w_{k-1}(a_1, b_1) + w_{k-1}(a_2, b_2)$$

Dari definisi *state* diatas, dapat dilihat bahwa bobot dari *state* $w_k(a, b)$ akan konvergen ke 0 pada setiap pertanyaan. Sehingga, dapat diasumsikan bahwa strategi optimal dari penjawab adalah dengan mencari *state* tujuan yang memiliki bobot lebih besar, atau dengan kata lain, $\max(w_{k-1}(a_1, b_1), w_{k-1}(a_2, b_2))$. Pembuktian untuk strategi pemenang penjawab akan disertakan kemudian.

Pada *state* (a, b), pertanyaan mengenai u elemen dari *truth-set* dan v elemen dari *lie-set* akan mendapat salah satu dari 2 respon, yaitu "ya" dan "tidak". Untuk setiap pertanyaan yang mendapat respon "ya", *state* akan berubah menjadi $(u, v + a - u)$. Sedangkan untuk setiap pertanyaan yang mendapat respon "tidak", *state* akan berubah menjadi $(a - u, b - v + u)$. Penjelasan proses ini dapat dilihat pada Gambar 2.1.

Untuk setiap *state* (a, b), didefinisikan karakteristik dari *state* tersebut dengan notasi

$$ch(a, b) = \min\{\mathrm{k}: w_k(a, b) \leq 2^k\}$$



Gambar 2.1 Perubahan Set terhadap Respons Penjawab

2.3.2 Analisis Penyelesaian Masalah

Pada bagian ini akan dijelaskan berbagai *lemma* dan formulasi dari permasalahan Ulam. Hal pertama yang harus dilakukan adalah mencari formulasi untuk mendapatkan k minimum untuk menyelesaikan permasalahan jika diberikan nilai n. Dengan kata lain, memberikan kondisi cukup bagi penanya untuk memenangkan permainan $[n, k]$ dengan k pertanyaan.

Lemma pertama akan menetapkan jumlah k agar penjawab dapat memenangkan permainan $[n, k]$ dengan strategi optimal penjawab. *Lemma* bagian 1 didapat dari Rivest[7] dan Spencer[6].

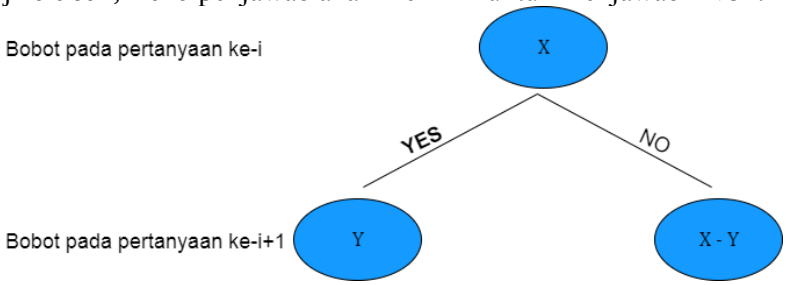
Lemma 1. (a) Untuk n genap, penjawab memenangkan permainan (n, k) *jika* $n(k+1) > 2^k$

(b) Untuk n ganjil, penjawab memenangkan permainan (n, k) *jika* $n(k+1) + k - 1 > 2^k$

(a) Pertama, didefinisikan sebuah strategi dari penjawab, yaitu penjawab selalu memilih *state* yang memiliki bobot lebih

besar diantara dua *state* yang dihasilkan dari pertanyaan penanya. Strategi ini dapat dinyatakan sebagai berikut:

Misalkan bobot permainan saat pertanyan ke-i adalah x. Selanjutnya, diajukan pertanyaan sedemikian sehingga bobot dari permainan jika dijawab dengan "YES" adalah y. Sebaliknya, bobot dari permainan jika dijawab dengan "NO" adalah $x - y$. Jika $y > x - y$, maka penjawab akan memilih untuk menjawab "YES" dan jika tidak, maka penjawab akan memilih untuk menjawab "NO".



Gambar 2.2 Strategi Penjawab untuk $y > x - y$

Untuk mengatasi strategi yang dimiliki oleh penjawab, didefinisikan strategi penanya, yaitu sebuah strategi untuk mengatasi strategi penjawab yang telah didefinisikan diatas. Strategi penanya dapat didefinisikan sebagai berikut:

Misalkan bobot permainan saat pertanyaan ke-i adalah x. Selanjutnya, diajukan pertanyaan sedemikian sehingga bobot dari permainan jika dijawab dengan "YES" adalah y. Sebaliknya, bobot dari permainan jika dijawab dengan "NO" adalah $x - y$. penanya harus memastikan bobot yang dipilih oleh penjawab sekecil mungkin, yaitu dengan meminimalkan nilai $max(y, x - y)$. Hal ini dapat dicapai jika $|y - (x - y)| \leq 1$. Sehingga, penanya harus memastikan bobot yang akan dituju bernilai berimbang.

Bobot awal dari permainan ini adalah $n(k + 1)$. Setiap pertanyaan yang diajukan akan menghasilkan bobot yang memiliki nilai minimal setengah dari bobot sebelumnya, yaitu saat $abs\left(w_{k-1}(a_1, b_1) - w_{k-1}(a_2, b_2)\right) \leq 1$. Karena bobot dari state awal adalah $n(k + 1) > 2^k$, maka:

$$w_k(n,0) = n(k+1) > 2^k$$
$$w_{k-1}(a_1,b_1) = a_1k + b_1 > 2^{k-1}$$
$$w_{k-2}(a_2,b_2) = a_2(k-1) + b_2 > 2^{k-2}$$
$$\dots$$
$$w_t(a_{k-t},b_{k-t}) = a_{k-t}(t+1) + b_{k-t} > 2^t \geq 2$$

setelah $t \leq k$ pertanyaan maka bobot dari *state* akhir pasti setidaknya 2. Pada saat ini, penanya harus sudah menyelesaikan permainan. Dari 2 kondisi menang penanya, yaitu $(1,0)$ dan $(0,1)$, hanya kondisi pertama yang dapat diraih oleh penanya. Hal ini disebabkan karena *state* kedua memiliki bobot 1, dimana bobot ini tidak mungkin dicapai sesuai telah dijelaskan. Selanjutnya, yang perlu dibuktikan adalah kondisi akhir dari permainan tidak mungkin $(1,0)$. *State* ini hanya dapat terjadi jika *state* sebelumnya adalah *state* $(1,c)$ dengan *t* pertanyaan tersisa dan pertanyaan yang diajukan adalah pertanyaan mengenai satu-satunya bilangan yang tersisa dari *truth-set*. Pertanyaan ini akan menghasilkan *state* $(1,0)$ dan $(0,c+1)$. Karena penjawab dipastikan akan memilih *state* dengan bobot yang tidak lebih kecil dan *state* yang akan dituju adalah *state* $(1,0)$, maka:

$$w_{t-1}(1,0) \geq w_{t-1}(0,c+1)$$

Yang berarti penjawab akan menjawab pertanyaan tersebut dengan "YES", berarti

$$1((t-1)+1)+0 \geq 0(t-1+1)+c+1$$
$$t \geq c+1$$

Karena state $(1,c)$ dicapai setelah $k-t$ pertanyaan, maka

$$w_t(1,c) = t+1+c > 2^k.2^{-(k-t)} = 2^t.$$

Dengan pertidaksamaan sebelumnya, didapatlah

$$t > 2^{t-1}$$

Dimana pasti tidak terpenuhi.

(b) Untuk n ganjil, pertanyaan pertama yang diajukan dari permainan $[n,k]$ akan menghasilkan (a_1,b_1) dan (a_2,b_2) sehingga

$$\max(w_{k-1}(a_1,b_1), w_{k-1}(a_2,b_2)) \geq \frac{n+1}{2}k + \frac{n-1}{2}.$$

Sehingga, berdasar strategi penjawab, *state* yang akan dituju setelah pertanyaan pertama pasti memiliki bobot minimal $((n+1)/2) + (n-1)/2$.

Jika

$$n(k+1) + (k-1) > 2^k$$

maka

$$\frac{n+1}{2}k + \frac{n-1}{2} > 2^{k-1}$$

Sehingga setelah $(k-1)$ pertanyaan selanjutnya, strategi ini akan menghasilkan *state* berbobot setidaknya 2. Dapat dibuktikan seperti pembuktian pertama bahwa *state* ini bukanlah *state* $(1,0)$, sehingga penjawab akan memenangkan permainan ini.

Selanjutnya, akan dianalisis kemungkinan menang bagi penanya

.

Lemma 2. Misalkan n adalah sebuah bilangan asli dan $k = ch(1,n)$. *Penanya menang dengan maksimal k pertanyaan dimulai dari state* $(1,n)$.

Pembuktian terhadap *Lemma 2* akan dilakukan dengan melakukan induksi terhadap n. Misal untuk setiap $m < n$ *Lemma 2* benar.

Jika $n < k$, maka pertanyaan mengenai satu-satunya bilangan dalam *truth-set* akan menghasilkan *state* $(1,0)$ dan $(0,n+1)$ dengan

$$w_{k-1}(1,0) \geq w_{k-1}(0,n+1)$$

State $(1,0)$ adalah *state* kemenangan penanya, dan

$$w_{k-1}(0,n+1) \leq 2^{k-1}$$

Sehingga, penanya akan memenangkan permainan setelah maksimal $k-1$ pertanyaan dimulai dari $(0,n+1)$ (karena *truth-set* sudah kosong, maka penjawab tidak dapat berbohong lagi). Sehingga untuk $n < k$ penanya menang dalam maksimal k pertanyaan dimulai dari $(1,n)$.

Jika $n \geq k$, maka misalkan

$$x = \left\lceil \frac{n-k+1}{2} \right\rceil$$

Pertanyaan mengenai satu-satunya bilangan dari *truth-set* dan x bilangan dari *lie-set* akan menghasilkan *state* $(1, x)$ dan $(0, n+1-x)$. Perhatikan bahwa

$$|w_{k-1}(1,x) - w_{k-1}(0,n+1-x)| = |(k+x)-(n+1-x)|$$
$$= |k-n-1+2x| \leq 1$$

Yang berarti $ch(1,x), ch(0,n+1-x) \leq k-1$.

Penanya menang dengan maksimal $k-1$ pertanyaan selanjutnya dari *state* $(0, n+1-x)$. Dengan menggunakan induksi, penanya juga akan memenangkan permainan dari *state* $(1, x)$. Sehingga untuk $n \geq k$, penanya akan memenangkan permainan dengan maksimal k pertanyaan dari *state* $(1, n)$.

Lemma 3. Misalkan terdapat state (a, b) *sedemikian sehingga* $b \geq a-1 \geq 1$, *maka terdapat pertanyaan yang menghasilkan state* (a_1, b_1) *dan state* (a_2, b_2) *sehingga:*

1. $\left\lfloor \frac{a}{2} \right\rfloor \leq a_1 \leq \left\lceil \frac{a+1}{2} \right\rceil, \left\lfloor \frac{a}{2} \right\rfloor \leq a_2 \leq \left\lceil \frac{a+1}{2} \right\rceil$;
2. $b_1 \geq a_1 - 1, b_2 \geq a_2 - 1$;
3. $ch(a_1, b_1), ch(a_2, b_2) \leq ch(a, b) - 1$

Pertama, akan dibuktikan *lemma* diatas untuk $b = a-1 \geq 1$. Misalkan $l = ch(a, b)$, terdapat 2 kasus, yaitu:

- a bilangan genap. Misalkan $a = 2c$, maka $b = 2c-1$. Pertanyaan mengenai c bilangan dari *truth-set* dan c bilangan dari *lie-set* akan menghasilkan *state* $(c, 2c)$ dan $(c, 2c-1)$. Kondisi 1 dan 2 telah terpenuhi. Untuk kondisi 3, cukup dibuktikan bahwa

$$w_{l-1}(c, 2c) \leq 2^{l-1}$$

Pertidaksamaan diatas benar, karena

$$w_l(2c, 2c-1) = 2c(l+1) + 2c - 1 = 2c(l+2) - 1 \leq 2^l$$

Karena $2c(l+2) - 1$ ganjil, maka

$$2c(l+2) \leq 2^l$$

Sehingga,

$$w_{l-1}(c, 2c) = cl + 2c = c(l+2) \leq 2^{l-1}$$

- a bilangan ganjil. Misalkan $a = 2c + 1$, maka $b = 2c$. Pertama, asumsikan $a > 5$. Pertanyaan mengenai $c + 1$ bilangan dari *truth-set* dan $c - \left\lfloor \frac{l}{2} \right\rfloor$ bilangan dari *lie-set* akan menghasilkan *state* $(c + 1, 2c - \left\lfloor \frac{l}{2} \right\rfloor)$ dan $(c, 2c + \left\lfloor \frac{l}{2} \right\rfloor + 1)$. Kondisi 1 telah terpenuhi. Untuk membuktikan kondisi 2, cukup dibuktikan bahwa $2c - [l/2] \geq c$ yang berarti $[l/2] \leq c$. Karena $a \geq 6$, maka

$$a(a + 1) + (a - 1) \leq 2^a$$

Sehingga,

$$l = ch(a, a - 1) \leq a$$

Yang berarti $[l/2] \leq c$

Selanjutnya, akan dibuktikan untuk kondisi 3

$$w_{l-1}\left(c + 1, 2c - \left\lfloor \frac{l}{2} \right\rfloor\right) = (c + 1)l + 2c - \left\lfloor \frac{l}{2} \right\rfloor = c(l + 2) + l - \left\lfloor \frac{l}{2} \right\rfloor$$

Dan

$$w_{l-1}\left(c, 2c + \left\lfloor \frac{l}{2} \right\rfloor + 1\right) = c(l + 2) + \left\lfloor \frac{l}{2} \right\rfloor + 1$$

Jika l genap, maka $w_l(2c + 1, 2c) = 2c(l + 2) + l + 1$ bernilai ganjil dan pertidaksamaan $w_l(2c + 1,2c) \leq 2^l$ mengakibatkan

$$2c(l + 2) + l + 2 \leq 2^l,$$

Sehingga

$$c(l + 2) + l - \left\lfloor \frac{l}{2} \right\rfloor = c(l + 2) + \frac{l}{2} \leq 2^{l-1}$$

Dan

$$c(l + 2) + \left\lfloor \frac{l}{2} \right\rfloor + 1 = c(l + 2) + \frac{l}{2} + 1 \leq 2^{l-1}$$

Jika l ganjil, maka

$$c(l + 2) + l - \left\lfloor \frac{l}{2} \right\rfloor = c(l + 2) + \left\lfloor \frac{l}{2} \right\rfloor + 1 = c(l + 2) + \frac{l}{2} + \frac{1}{2} \leq \frac{2^l}{2} = 2^{l-1}$$

Yang berarti

$$ch\left(c+1,2c-\left[\frac{l}{2}\right]\right), ch\left(c,2c+\left[\frac{l}{2}\right]+1\right) \leq l-1$$

Selanjutnya *Lemma 3* akan dibuktikan untuk $b = a-1 \geq 1, a$ ganjil dan $a \leq 5$, yaitu untuk *state* $(3,2)$ dan $(5,4)$.

Untuk kasus *state* $(3,2)$, maka $ch(3,2) = 5$. Pertanyaan mengenai 2 bilangan dari *truth-set* menghasilkan *state* $(2,1)$ dan $(1,4)$. Kondisi 1 dan 2 telah terpenuhi. Untuk kondisi 3, perhatikan bahwa $w_4(2,1) = 11$ dan $w_4(1,4) = 9$, keduanya bernilai tidak lebih dari 2^4.

Selanjutnya, untuk kasus *state* $(5,4)$, maka $ch(5,4) = 6$. Pertanyaan mengenai 3 bilangan dari *truth-set* menghasilkan *state* $(3,2)$ dan $(2,7)$. Kondisi 1 dan 2 telah tercapai. Untuk kondisi 3, $w_5(3,2) = 20$ dan $w_5(2,7) = 19$, keduanya bernilai tidak lebih dari 2^5.

Sehingga, *Lemma 3* telah terbukti untuk $b = a-1 \geq 1$.

Untuk $b > a-1 \geq 1$, misalkan $b = a-1+x, x > 0, m = ch(a,b)$, dan $l = ch(a,a-1)$. Dari pembuktian diatas, terdapat pertanyaan mengenai s bilangan dari *truth-set* dan t bilangan dari *lie-set*, dimana jika ditanyakan pada saat *state* $(a,a-1)$ akan menghasilkan *state* (a_1,b_1) dan (a_2,b_2) yang memenuhi *lemma 3*. Misalkan $v_1 = w_{m-1}(a_1,b_1)$ dan $v_2 = w_{m-1}(a_2,b_2)$. Dapat diketahui bahwa $v_1, v_2 \leq 2^{m-1}$. $w_m(a,a-1) = v_1 + v_2$ dan $w_m(a,b) = v_1 + v_2 + x$.

Misalkan $y = \min(x, 2^{m-1} - v_1)$. Pertidaksamaan $y \geq 0$ dan $x - y \geq 0$ terpenuhi. Perhatikan bahwa pertanyaan mengenai s bilangan dari *truth-set* dan $t+y$ bilangan dari *lie-set*, jika ditanyakan pada *state* (a,b) akan menghasilkan *state* (a_1,b_1+y) dan (a_2,b_2+x-y). Keduanya memenuhi kondisi 1 dan 2 dari *Lemma 3*. Selanjutnya,

$$w_{m-1}(a_1,b_1+y) = v_1 + y \leq v_1 + 2^{m-1} - v_1 = 2^{m-1}$$

Dan

$$w_{m-1}(a_2,b_2+x-y) = v_2 + x - y$$

Selanjutnya, jika $y = x$,
$w_{m-1}(a_2, b_2 + x - y) = v_2 + x - x = v_2 \leq 2^{m-1}$

Jika $y = 2^{m-1} - v_1$,
$w_{m-1}(a_2, b_2 + x - y) = v_1 + v_2 + x - 2^{m-1}$
$= w_m(a_2, b_2) - 2^{m-1} \leq 2^m - 2^{m-1} = 2^{m-1}$

Kedua *state* diatas memenuhi kondisi 3 dari *Lemma 3* yang membuktikan *Lemma 3*.

Dari *Lemma 1 – Lemma 3*, selanjutnya akan dibuat kondisi cukup dan kondisi perlu bagi penanya untuk memenangkan $[n, k]$

Teorema. (a) Untuk n genap, penanya memenangi permainan $[n, k]$ *jika dan hanya jika* $n(k+1) \leq 2^k$

(b) Untuk n ganjil, penanya memenangi permainan $[n, k]$ *jika dan hanya jika* $n(k+1) + (k-1) \leq 2^k$

Bagian "hanya jika" dari (a) dan (b) didapat dari *lemma 1*. Selanjutnya akan dibuktikan bagian "jika" dari teorema diatas.

(a) Misalkan $n = 2a$ dan $n(k+1) \leq 2^k$. Pertanyaan mengenai a bilangan dari *truth-set* dari *state* awal $(2a, 0)$ menghasilkan *state* (a, a) dan (a, a) dengan $ch(a, a) \leq k - 1$. Jika $a = 1$, menurut *Lemma 2*, penanya akan memenangi permainan dalam $k - 1$ pertanyaan. Jika $a > 1$, *state* (a, a) memenuhi *lemma 3*. Selanjutnya, gunakan *Lemma 3* hingga permainan mencapai *state* $(1, b)$. Perhatikan bahwa setiap kali menggunakan *Lemma 3*, kondisi 2 dari *Lemma 3* selalu tercapai. Selain itu, kondisi 1 dari *Lemma 3* menjamin bahwa *state* $(1, b)$ pasti tercapai setelah $t \leq [\lg a] + 1$ pertanyaan. Menurut kondisi 3, didapatkan $ch(1, b) \leq k - 1 - t$ untuk masing-masing *state*. Selanjutnya, dari *Lemma 2*, penanya dapat memenangkan permainan dalam $k - 1 - t$ pertanyaan dari *state* $(1, b)$. Karena penanya sudah menggunakan t+1 pertanyaan sebelumnya, maka penanya dapat menyelesaikan permainan $[n, k]$.

(b) Misalkan $n = 2a + 1$ dan $n(k+1) + (k-1) \leq 2^k$. Pertanyaan pertama mengenai $a + 1$ bilangan dari *truth-set* dari

state awal $(2a+1,0)$ menghasilkan *state* $(a+1,a)$ dan $(a,a+1)$. Karena $w_{k-1}(a,a+1) = ak+a+1 \leq w_{k-1}(a+1,a) = (a+1)k+a = \left(\frac{n+1}{2}\right)k + \frac{n-1}{2} = \frac{n(k+1)+(k-1)}{2} \leq 2^{k-1}$, sehingga $ch(a,a+1), ch(a+1,a) \leq k-1$.

Bukti untuk sisa permainan sama seperti bukti bagian (a)

BAB III
ANALISIS DAN PERANCANGAN

Pada bagian ini akan dijelaskan analisis dan desain sistem yang digunakan untuk menyelesaikan permasalahan pada Tugas Akhir ini.

3.1 Analisis Observasi Permasalahan

Pada bagian ini akan diberikan beberapa observasi dan analisis yang akan membantu jalannya implementasi.

3.1.1 Cara Penyimpanan Elemen dalam Set

Pada batasan permasalahan yang diberikan, permainan dapat dimulai hingga $n = 10^{18}$. Ruang yang dibutuhkan untuk menyimpan elemen-elemen untuk *truth-set* maupun *lie-set* menjadi sangat besar. Sebagai gambaran, untuk menyimpan elemen-elemen tersebut dibutuhkan ruang sebanyak n elemen, dimana masing-masing elemen memakan ukuran sebanyak 8 *bytes*, sehungga diperlukan kurang lebih $8 * 10^{18}$ *bytes*. Menyelesaikan permasalahan Tugas Akhir ini dengan batas tersebut tidak mungkin tercapai secara ukuran maupun secara waktu pemrosesan.

Solusi yang digunakan pada pengerjaan Tugas Akhir ini adalah dengan melakukan kompresi nilai terhadap elemen yang disimpan. Kompresi nilai dilakukan dengan cara menyimpan elemen-elemen yang berurutan berdasarkan bilangan pertama dan bilangan terakhir dari elemen yang terurut tersebut. Sehingga, memori yang digunakan untuk menyimpan elemen menjadi lebih kecil. Selanjutnya kedua bilangan tersebut diikat menjadi satu kesatuan dengan struktur *pair*. Contoh dari kompresi ini adalah sebagai berikut:

Misalkan elemen yang ingin disimpan adalah $\{1,2,3,\dots,1.000\}$. Jika elemen-elemen tersebut disimpan secara naif, maka diperlukan ruang untuk 1.000 elemen. Akan tetapi, dengan kompresi nilai, elemen yang disimpan hanya elemen pertama dan elemen terskhir dari urutan tersebut, yaitu 1 dan 1.000. Sehingga, hanya diperlukan ruang untuk 2 elemen saja.

Dalam melakukan pemrosesan selanjutnya akan banyak terjadi operasi penghapusan dan penambahan elemen. Penghapusan dan penambahan ini dilakukan seiring berkurangnya dan bertambahnya elemen dari *truth-set* maupun *lie-set*. Sementara itu, elemen-elemen yang terdapat dalam *truth-set* maupun *lie-set* harus tetap terjaga dalam bentuk terurut naik. Hal ini penting untuk memudahkan proses pencarian, penghapusan, dan penambahan elemen. Sehingga, Tugas Akhir ini menggunakan sebuah struktur data yang mampu menjaga sifat terurut tersebut, yaitu *set*.

Data-data berupa *pair* tersebut akan disimpan kedalam *set*, sehingga data-data tersebut akan tetap terurut dan dapat diakses, dihapus, maupun ditambahkan dengan mudah dan cepat.

3.1.2 Pemrosesan Set berdasarkan Respon Penjawab

Pada permainan ini, penjawab akan merespon pertanyaan dengan jawaban "YES" atau "NO". Setelah mengajukan pertanyaan a elemen dari *truth-set* dan b elemen dari *lie-set*, selanjutnya isi dari *truth-set* dan *lie-set* perlu diproses tergantung dari respon dari penjawab. Untuk setiap respon dari penjawab, *set* harus diproses dengan asumsi penjawab berbohong maupun tidak. Perlu diingat bahwa *lie-set* akan menyimpan semua bilangan dengan asumsi bahwa penjawab telah berbohong, sehingga semua jawaban yang diberikan oleh penjawab akan dianggap jawaban jujur. Sementara itu, *truth-set* akan menyimpan semua bilangan dengan asumsi bahwa penjawab belum berbohong, sehingga isi dari *truth-set* akan diproses berdasarkan kedua asumsi tersebut. Penjabaran pemrosesan *set* untuk setiap respon akan dijelaskan sebagai berikut:

1. Jika penjawab memberikan respon "YES", maka isi dari *truth-set* akan menjadi irisan dari semua elemen dari *truth-set* dan a, sedangkan isi dari *lie-set* akan menjadi irisan dari *lie-set* dan b, serta komplemen dari *truth-set* yang baru dengan *truth-set* yang lama.
2. Jika penjawab memberikan respon "NO", maka isi dari *truth-set* akan menjadi komplemen dari *truth-set* yang sekarang dengan a, sedangkan isi dari *lie-set* akan

menjadi komplemen dari *lie-set* dengan *b*, serta irisan dari *truth-set* dengan *a*.

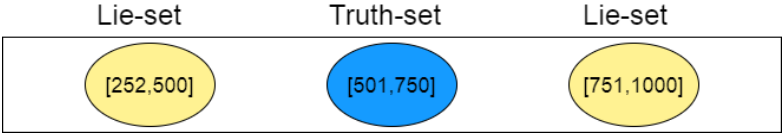
Dari kedua fakta diatas, maka dapat dipastikan bahwa isi dari *truth-set* akan berupa sebuah rentang bilangan. Hal ini disebabkan karena hanya terdapat pemotongan elemen, tidak terdapat penggabungan elemen dari *truth-set*. Karena pada awal permainan isi dari *truth-set* berupa sebuah rentang [1,n], maka pemotongan terhadap rentang ini tidak akan membuat pemisahan rentang seperti pada *lie-set*.

3.1.3 Pemilihan Elemen dalam Pemrosesan Set

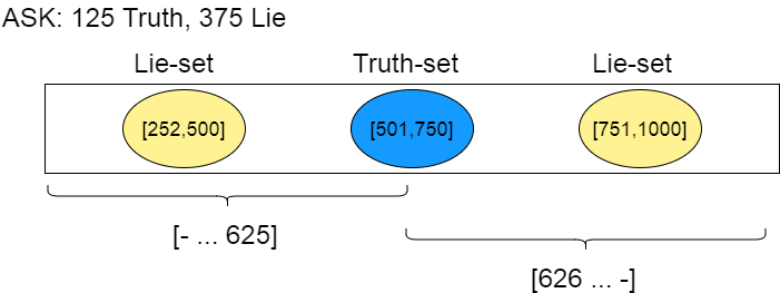
Pertanyaan yang diajukan dalam permainan ini adalah dalam bentuk rentang [l,r], sehingga rentang yang dipilih untuk mengajukan pertanyaan harus merupakan bilangan terurut yang tepat mengandung *a* elemen dari *truth-set* dan *b* elemen dari *lie-set*. Rentang tersebut hanya dapat ditemukan pada perbatasan dari elemen-elemen pada *truth-set* dan elemen-elemen pada *lie-set*. Sehingga, dalam melakukan pemilihan elemen untuk ditanyakan, elemen yang dipilih adalah *a* elemen terkecil dari *truth*-set dan *b* elemen terbesar dari *lie-set* yang lebih kecil dari elemen terkecil *truth-set*, atau *a* elemen terbesar dari *truth-set* dan *b* elemen terkecil dari *lie-set* yang lebih besar dari elemen terbesar dari *truth-set*.

3.1.4 Bentuk Ekuivalen Pertanyaan

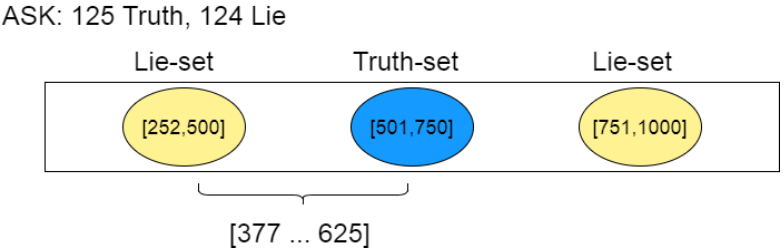
Pada saat mengajukan pertanyaan mengenai *a* elemen dari *truth-set* dan *b* elemen dari *lie-set*, terdapat kemungkinan tidak ditemukan rentang yang memenuhi. Salah satu contoh pertanyaan yang tidak dapat dipenuhi rentangnya terdapat saat nilai dari *b* melebihi dari setengah isi dari *lie-set*. Sebagai ilustrasi, perhatikan Gambar 3.1- Gambar 3.3.



Gambar 3.1 Kondisi *Truth-set* dan *Lie-set*



Gambar 3.2 Contoh Pertanyaan yang Tidak Dapat Dipenuhi



Gambar 3.3 Bentuk Ekuivalen Pertanyan

Pada ilustrasi diatas, pemilihan rentang dengan mengambil nilai terkecil dari *truth-set* maupun nilai terbesar dari *truth-set* tidak dapat dipenuhi. Untuk mengatasi hal tersebut, perlu dipahami bahwa pertayaan pada Gambar 3.2 memiliki bentuk ekuivalen. Pada ilustrasi diatas, *truth-set* memiliki 250 elemen dan *lie-set* memiliki 499 elemen, maka pertanyaan mengenai 125 elemen dari *truth-set* dan 375 elemen dari *lie-set* akan ekuivalen dengan pertanyaan mengenai 125 elemen dari *truth-set* dan 124 elemen dari *lie-set*. Hasil respon dan bobot dari *state* tujuan akan setara dengan pertanyaan sebelumnya. Sehingga, dapat dipastikan bahwa rentang pertanyaan ini akan ditemukan.

3.2 Perancangan Sistem

Pada bagian ini akan dijelaskan desain sistem dalam membantu proses implementasi.

3.2.1 Deskripsi Umum Sistem

Sistem akan menerima masukan berupa sebuah bilangan yang menyatakan jumlah permainan sebagai kasus uji sistem. Setiap memulai permainan, sistem akan memberikan sinyal untuk memulai permainan. Selanjutnya, sistem akan menerima masukan berupa dua buah bilangan, masing-masing adalah n dan k, yang menandakan permainan $[n, k]$. *Pseudocode* Fungsi Main dari sistem ditunjukkan pada Gambar 3.4.

```
1  T = Input() //total permainan
2  for t = 1 to T
3     INIT()
4     START_GAME()
5     SOLVE_GAME()
```

Gambar 3.4 Pseudocode Fungsi Main

3.2.2 Desain Fungsi INIT

Fungsi INIT akan berfungsi sebagai penginisialisasi dari data-data yang diperlukan saat permainan dimulai, yang terdiri dari *truth-set* dan *lie-set*. Sebelum permainan dimulai, kedua isi *set* tersebut harus dikosongkan sehingga nilai *set* dari permainan sebelumnya tidak terbawa saat akan memulai permainan baru. *Pseudocode* Fungsi INIT ditunjukkan pada Gambar 3.5.

```
1  lie-set = ∅
2  truth-set = ∅
3  truthsize = 0
4  liesize = 0
```

Gambar 3.5 Pseudocode Fungsi INIT

3.2.3 Desain Fungsi START_GAME

Fungsi START_GAME berfungsi untuk memberikan sinyal kepada *online judge* untuk memulai permainan. Selanjutnya, fungsi ini akan menerima n dan k, yang menyatakan deskripsi dari permainan, yaitu rentang atas dari permainan dan banyaknya pertanyaan yang dapat diajukan kepada penjawab. *Pseudocode* fungsi START_GAME ditunjukkan pada Gambar 3.6.

1	send START signal
2	n = Input()
3	k = Input()

Gambar 3.6 Pseudocode Fungsi START_GAME

3.2.4 Desain Fungsi SOLVE_GAME

Fungsi SOLVE_GAME merupakan fungsi yang berisi seluruh algoritma untuk menyelesaikan permainan. Fungsi ini akan melakukan pemrosesan mulai dari masukan n dan k yang dimasukkan pada fungsi START_GAME. Fungsi ini akan menghasilkan luaran berupa angka yang ingin dicari. Proses penyelesaian permainan dimulai dengan memberikan inisialisasi kepada *truth-set* dan *lie-set*. Jika elemen terakhir adalah *truth*-set, maka fungsi akan selesai lebih awal. Pseudocode fungsi SOLVE_GAME ditunjukkan pada Gambar 3.7 - Gambar 3.8.

1	truth-set = $\{1 \ldots n\}$
2	truthsize = n
2	liesize = 0
3	Question = SELECT($\left\lfloor \frac{n}{2} \right\rfloor$, 0)
4	ASK(Question)
5	RESULT(Question)
6	ch = COUNT_CHARACTERISTIC(truthsize,
	truthsize-1)
7	while $truthsize > 1$ do
8	if truthsize is even then
9	t = $\frac{truthsize}{2}$, l = $\frac{truthsize}{2}$
10	Else
11	t = $\frac{truthsize+1}{2}$
12	l = max($\frac{truthsize-1}{2} - \left\lfloor \frac{ch}{2} \right\rfloor$, 0)
13	weight = t * char + l + truthsize - t
14	adjust = min(liesize-(truthsize-
	1), 2^{char-1}-weight)
15	l = l + adjust

Gambar 3.7 Pseudocode Fungsi SOLVE_GAME(1)

16	Question = SELECT(t,l)
17	ASK(Question)
18	RESULT(Question,t,l,truthsize,liesize,
	char)
19	ch=COUNT_CHARACTERISTIC(truthsize,
	truthsize-1)
20	while truthsize==1 and liesize ≥char do
21	Question = SELECT(1,$\left\lfloor \frac{liesize-char+1}{2} \right\rfloor$)
22	ASK(Question)
23	RESULT(Question,1,$\left\lfloor \frac{liesize-char+1}{2} \right\rfloor$,truthsize,
	liesize,char)
24	if truthsize==1 and liesize > 0 then
25	Question = SELECT(1,0)
26	ASK(Question)
27	RESULT(Question,1,0,truthsize,liesize,
	char)
28	if truthsize==1
29	Answer = Truth //the only element of
	truth-set
30	Return
31	while liesize > 1 do
32	Question = SELECT(0,$\left\lfloor \frac{liesize}{2} \right\rfloor$)
33	ASK(Question)
34	RESULT(Question,0,$\left\lfloor \frac{liesize}{2} \right\rfloor$,truthsize,
	liesize,char)
35	Answer = Lie //the only element of lie-set
36	Return

Gambar 3.8 Pseudocode Fungsi SOLVE_GAME(2)

3.2.5 Desain Fungsi SELECT

Fungsi SELECT merupakan fungsi untuk mendapatkan rentang pertanyaan yang akan ditanyakan. Fungsi ini menerima dua buah masukan, yaitu *a* dan *b*, yang masing-masing menyatakan banyak elemen dari *truth-set* dan *lie-set* yang ingin

diambil. Selanjutnya, fungsi SELECT akan mencari elemen tersebut dari *truth-set* dan *lie-set* sesuai dengan penjelasan pada subbab 2.4.3.

3.2.6 Desain Fungsi ASK

Fungsi ASK merupakan fungsi untuk mengirimkan pertanyaan yang telah didapat dari fungsi SELECT. Fungsi ini mengirimkan sinyal pertanyaan kepada *online judge* dengan cara mencetak sebuah baris *string* seperti yang telah diminta pada *online judge* tersebut.

3.2.7 Desain Fungsi RESULT

Fungsi RESULT merupakan fungsi yang digunakan untuk menghapus dan menambahkan elemen-elemen dari *truth-set* maupun *lie-set*. Fungsi ini menerima respon dari *online judge* dari pertanyaan yang dikirimkan oleh fungsi ASK, lalu melakukan pemrosesan terhadap *set* seperti yang dijelaskan pada subbab 2.4.2. Selanjutnya, terdapat juga sebuah variabel *char* yang digunakan untuk menyimpan *characteristic* dari *state* tersebut yang selanjutnya akan digunakan dalam penghitungan dalam menentukan jumlah bilangan yang akan ditanyakan. *Pseudocode* dari fungsi RESULT ditunjukkan pada Gambar 3.9.

1	`if answer is "yes" then`
2	`  lie-set = (lie-set` $\cap b$ `) ∪`
	`  (truth\ truth` $\cap a$`)`
3	`  truth-set = truth` $\cap a$
4	`Else`
5	`  lie-set = (lie-set\ lie-set` $\cap b$ `) ∪`
	`  (truth-set` $\cap a$`)`
6	`  truth-set = truth-set\ truth-set` $\cap a$
7	`truthsize = card(truth)`
8	`liesize = card(lie)`
9	`char=COUNT_CHARACTERISTIC(truthsize,`
	`  liesize)`

Gambar 3.9 Pseudocode Fungsi RESULT

3.2.8 Desain Fungsi COUNT_CHARACTERISTIC

Fungsi COUNT_CHARACTERISTIC merupakan fungsi yang menerima dua masukan, yaitu banyaknya bilangan pada *truth-set* dan banyaknya bilangan pada *lie-set.* Fungsi ini digunakan untuk menghitung karakteristik dari sebuah *state*. Karakteristik ini akan menghitung perkiraan jumlah pertanyaan yang dibutuhkan untuk menyelesaikan permainan. Selanjutnya, hasil pengembalian dari fungsi ini akan digunakan untuk penghitungan dalam fungsi SOLVE_GAME. *Pseudocode* untuk fungsi COUNT_CHARACTERISTIC dapat dilihat pada Gambar 3.10.

1	`return` $\min\{j : a * (j+1) + (a-1) \leq 2^j\}$

Gambar 3.10 Pseudocode Fungsi COUNT_CHARACTERISTIC

[*Halaman ini sengaja dikosongkan*]

BAB IV
IMPLEMENTASI

Pada bab ini akan dijelaskan implementasi dari algoritma dan struktur data berdasarkan desain yang telah dilakukan.

4.1 Lingkungan Implementasi

Lingkungan implementasi menggunakan sebuah komputer dengan spesifikasi perangkat lunak dan perangkat keras seperti tercantum pada Tabel 4.1.

Tabel 4.1 Spesifikasi Lingkungan Implementasi

No.	Jenis Perangkat	Spesifikasi
1	Perangkat Keras	• *Processor* Intel Core i7-2670QM CPU @ 2.20GHz • *Memory* 8GB 1333MHz DDR3
2	Perangkat Lunak	• Sistem operasi Windows 8 Pro • *Integrated Development Environment* Dev-C++ 5.3.0.3

4.2 Pendefinisian *Preprocessor Directives*

Pada bagian ini akan dijelaskan beberapa pendefinisian dari *preprocessor directives* yang akan digunakan dalam program ini. Penggunaan *preprocessor directive* ditujukan untuk mempermudah dan mempersingkat implementasi. Implementasi dari *preprocessor directive* ditunjukkan pada Kode Sumber 4.1 .

```
1  #define ld long double
2  #define ii pair<ld, ld>
3  #define mp make_pair
4  #define fi first
5  #define sc second
```

Kode Sumber 4.1 Implementasi *Preprocessor Directive*

4.3 Implementasi Fungsi Main

Fungsi Main diimplementasikan sesuai *pseudocode* subbab 3.2.1. Pada awalnya sistem akan menetima masukan berupa

banyaknya kasus uji, yaitu banyaknya permainan yang akan dijalankan pada sekali eksekusi program. Selanjutnya, untuk setiap permainan, fungsi Main akan memanggil fungsi INIT, fungsi START_GAME, dan fungsi SOLVE_GAME. Implementasi dari fungsi Main dapat dilihat pada Kode Sumber 4.2.

```
1  int main() {
2     int test;
3     scanf("%d", &test);
4     while(test--) {
5        INIT();
1        START_GAME();
2        SOLVE_GAME();
3     }
4     return 0;
5  }
```

Kode Sumber 4.2 Implementasi Fungsi Main

4.4 Implementasi Variabel Global

Variabel dengan *scope* global dibutuhkan untuk kemudahan pengaksesan sebuah variabel oleh setiap fungsi yang ingin mengaksesnya. Penggunaan variabel global pada implementasi ini diterapkan pada variabel `truth_set` yang digunakan merepresentasikan isi dari *truth-set*, variabel `lie_set` yang digunakan untuk merepresentasikan isi dari *lie-set*, variabel `n` dan `k` yang digunakan untuk mendeskripsikan permainan, variabel `truthsize` dan `liesize` yang digunakan untuk menyimpan ukuran dari *truth-set* maupun *lie-set*, dan juga variabel bantu `characteristic` untuk membantu mengkomputasi banyak pertanyaan yang akan diajukan. Implementasi dari variabel global dapat dilihat pada Kode Sumber 4.3.

```
1  ii truth_set;
2  set<ii> lie_set;
3  long long n,k,characteristic;
4  long long truthsize, liesize;
```

Kode Sumber 4.3 Implementasi Variabel Global

4.5 Implementasi Fungsi INIT

Fungsi INIT berfungsi untuk menginisialisasi isi dari *set* seperti yang telah dijelaskan pada subbab 3.2.2. Implementasi dari fungsi init dapat dilihat pada Kode Sumber 4.4.

```
1  void INIT() {
2     lie_set.clear();
3     truth_set = mp(0,0);
4  }
```

Kode Sumber 4.4 Implementasi Fungsi INIT

Pada implementasi dari fungsi INIT, yang dilakukan adalah melakukan penghapusan terhadap isi dari `lie_set` dan `truth_set`. *Lie-set* yang diimplementasikan menggunakan struktur data *set* dibersihkan menggunakan fungsi STL clear(), sedangkan *truth-set* yang diimplementasikan menggunakan pair dibersihkan cukup dengan melakukan *reset* terhadap isi dari *truth-set*.

4.6 Implementasi fungsi START_GAME

Fungsi START_GAME melakukan pengiriman sinyal tanda permainan dimulai. Penggunaan fflush(stdout) ditujukan agar *online judge* dapat membaca sinyal yang diberikan oleh program. Selanjutnya, fungsi ini menerima masukan berupa dua bilangan `n` dan `k`. Implementasi dari fungsi START_GAME dapat dilihat pada Kode Sumber 4.5.

```
1  void START_GAME() {
2     printf("START GAME\n");
3     fflush(stdout);
4  cin>>n>>k;
5  }
```

Kode Sumber 4.5 Implementasi Fungsi START_GAME

4.7 Implementasi fungsi SOLVE_GAME

Fungsi SOLVE_GAME memulai permainan (n, k). Fungsi ini mulai melakukan inisialisasi dalam permainan (n, k) dan

menjalankan algoritma seperti pada subbab 3.2.4. Implementasi dari fungsi SOLVE_GAME dapat dilihat pada Kode Sumber 4.8 Implementasi Fungsi SOLVE_GAME(3) - Kode Sumber 4.8.

```
1   void SOLVE_GAME() {
2      ii question;
3      ld weight, adjust, ch;
4      long long t,l;
5      truth_set = mp((ld)1,(ld)n);
6      question = mp((ld)1,floor((ld)n/2));
7      ASK(question);
8      RESULT(question);
9      ch = count_characteristic(truthsize,
          truthsize-1);
10     while(truthsize>1) {
11        if(truthsize%2==0) {
12           t = truthsize/2;
13           l = truthsize/2;
14        }
15        else {
16           t = (truthsize+1)/2;
17           l = max((truthsize-1)/2 -
               floor(ch/2), (long double)0);
18        }
19        weight = (long double) (t) *
             characteristic + l + truthsize - t;
20        long double temp = 1;
21        for(int i = 1; i<=characteristic-1;
             i++) {
22           temp*=2;
23        }
24        temp-=weight;
25        adjust = min((long double)liesize-
             (truthsize-1), temp);
26        l+=adjust;
27        if(l > liesize-l) {
28           l = liesize-l;
```

Kode Sumber 4.6 Implementasi Fungsi SOLVE_GAME(1)

```
29          t = truthsize-t;
30        }
31        question = SELECT(t,l);
32        ASK(question);
33        RESULT(question);
34        ch = count_characteristic(
            truthsize, truthsize-1);
35      }
36      while(truthsize==1 &&
                liesize>=characteristic) {
37        question = SELECT(1,
          floor((liesize-characteristic+1)/2));
38        ASK(question);
39        RESULT(question);
40      }
41      if(truthsize==1 && liesize>0) {
42        question = SELECT(1,0);
43        ASK(question);
44        RESULT(question);
45      }
46      if(truthsize==1) {
47        printf("ANSWER %.0LF\n",
                    truth_set.fi);
48        fflush(stdout);
49      }
50      else {
51        while(liesize>1) {
52          question = SELECT(0,
                              floor(liesize/2));
53          ASK(question);
54          RESULT(question);
55        }
56        for(set<ii>::iterator it =
lie_set.begin(); it!=lie_set.end(); ++it){
57          ii element = *it;
58          if(element.fi==element.sc) {
```

Kode Sumber 4.7 Implementasi Fungsi SOLVE_GAME(2)

```
59              printf("ANSWER %.0LF\n",
                                    element.fi);
60              fflush(stdout);
61            }
62          }
63      }
64  }
```

Kode Sumber 4.8 Implementasi Fungsi SOLVE_GAME(3)

Pada fungsi ini algoritma utama untuk menyelesaikan permainan akan diimplementasikan. Setiap akan mengajukan pertanyaan, pertama dicari terlebih dahulu jumlah elemen dari *truth-set* maupun *lie-set*. Setelah itu, pertanyaan diajukan ke *online judge*. Setelah itu, program akan menunggu respon dari *online judge* lalu memrosesnya. Proses pada kode baris 10-35 akan mengarahkan permainan agar mencapai *lemma 2*. Setelah itu, kode baris 36-40 akan mengarahkan permainan dari *lemma 2* sehingga mencapai *state* $(0, m)$ atau $(1,0)$. Setelah itu, program akan menyelesaikan permainan jika telah mencapai *state* $(1,0)$ atau melanjutkan hingga mencapai $(0,1)$.

4.8 Implementasi Fungsi SELECT

Fungsi SELECT akan mengambil elemen dari *truth-set* dan *lie-set* sesuai dengan permintaan untuk ditanyakan. Fungsi ini akan mengembalikan nilai berupa *pair*, yaitu dua bilangan yang menyatakan rentang yang mengandung elemen dari *truth-set* dan *lie-set* yang diinginkan. Implementasi fungsi SELECT ditunjukkan pada Kode Sumber 4.9 - Kode Sumber 4.11.

```
1   ii SELECT(long double truth_ask,
                long double lie_ask) {
2     if(lie_ask == 0) {
3       return mp(truth_set.fi,
            truth_set.fi + truth_ask - 1);
      }
4     long double less = 0;
5     for(set<ii>::iterator it =
```

Kode Sumber 4.9 Implementasi Fungsi SELECT(1)

```
    lie_set.begin(); it!=lie_set.end(); ++it){
6       ii element  = *it;
7       if(element.sc < truth_set.fi) {
8         less += element.sc -
                    element.fi + 1;
9       }
10      if(element.fi > truth_set.sc) {
11        break;
12      }
13    }
14    long double start = 0;
15    if(less>=lie_ask) {
16      long double lie_found = 0;
17      for(set<ii>::reverse_iterator it =
lie_set.rbegin(); it!=lie_set.rend();
      ++it) {
18        ii element = *it;
19        if(element.sc <= truth_set.fi-1) {
20          start = 1;
21        }
22        if(start) {
23          if(lie_ask - lie_found >
element.sc-element.fi+1) {
24            lie_found += element.sc -
element.fi + 1;
25          }
26          else {
27            return mp(
element.sc - (lie_ask - lie_found) + 1,
truth_set.fi + truth_ask - 1
                        );
28          }
29        }
30      }
31    }
32    else {
```

Kode Sumber 4.10 Implementasi Fungsi SELECT(2)

```
33        long double lie_found = 0;
34        for(set<ii>::iterator it =
    lie_set.begin(); it!=lie_set.end(); ++it){
35          ii element = *it;
36          if(element.fi>=truth_set.sc+1) {
37            start = 1;
38          }
39          if(start) {
40            if(lie_ask - lie_found >
    element.sc-element.fi+1) {
41              lie_found += element.sc -
    element.fi + 1;
42            }
43            else {
44              return mp(
    truth_set.sc - truth_ask + 1,
    element.fi + lie_ask - lie_found - 1
                            );
45            }
46          }
47        }
48     }
49  }
```

Kode Sumber 4.11 Implementasi Fungsi SELECT(3)

Pada fungsi SELECT, apabila banyaknya *lie-set* yang diminta adalah 0, maka fungsi akan mengembalikan `truth_ask` elemen pertama dari *truth-set*. Kode tersebut diimplementasikan pada baris 2-4. Selanjutnya, pada kode baris 4-13, fungsi SELECT akan menghitung banyaknya elemen *lie-set* yang nilainya lebih kecil dan lebih besar dari elemen-elemen dari *truth-set*. Hasil penghitungan ini akan digunakan untuk menentukan elemen yang diambil untuk pertanyaan. Apabila banyaknya elemen *lie-set* yang lebih kecil dari elemen pada *truth-set* lebih banyak daripada elemen dari *lie-set* yang diminta, maka `lie_ask` elemen terbesar dari bagian tersebut akan diambil dan digabung dengan `truth_ask` elemen terkecil dari *truth-set*. Sebaliknya, jika

banyaknya elemen yang lebih besar dari elemen pada *truth-set* lebih banyak daripada elemen dari *lie-set* yang diminta, maka `lie_ask` elemen terkecil dari bagian tersebut akan diambil dan digabung dengan `truth_ask` elemen terbesar dari *lie-set*.

4.9 Implementasi Fungsi ASK

Fungsi ASK akan mengirimkan sinyal untuk mengajukan pertanyaan yang telah dicari pada fungsi SELECT. Fflush pada fungsi ini bertujuan untuk membersihkan *output stream* sehingga *online judge* dapat memproses pertanyaan yang dikirimkan. Implementasi fungsi ASK dapat dilihat pada Kode Sumber 4.12.

```
1  void ASK(ii question) {
2     printf("QUERY %.0LF %.0LF\n",
           question.fi, question.sc);
3     fflush(stdout);
4     return;
5  }
```

Kode Sumber 4.12 Implementasi Fungsi ASK

4.10 Implementasi Fungsi RESULT

Fungsi RESULT akan menerima masukan dari *online judge* mengenai respon dari pertanyaan yang telah diajukan. Selanjutnya, fungsi ini akan memproses *truth-set* dan *lie-set* seperti yang telah dijelaskan pada subbab 3.2.7. Implementasi dari fungsi RESULT dapat dilihat pada Kode Sumber 4.13 - Kode Sumber 4.16.

```
1  void RESULT(ii question) {
2     string response;
3     cin>>response;
4     if(response=="YES") {
5        ii intersection = mp(
   max(truth_set.fi, question.fi),
   min(truth_set.sc, question.sc)
                          );
6        set<ii> temp;
7        if(truth_set.fi<=truth_set.sc) {
```

Kode Sumber 4.13 Implementasi Fungsi RESULT(1)

```
8           if(truth_set.fi==intersection.fi){
9             temp.insert(mp(
intersection.sc+1,truth_set.sc
                                ));
10          }
11          else {
12            temp.insert(mp(
truth_set.fi,intersection.fi-1
                                ));
13          }
14          truth_set = intersection;
15        }
16        for(set<ii>::iterator it =
lie_set.begin(); it!=lie_set.end(); ++it){
17          ii element = *it;
18          if(max(element.fi, question.fi) <=
              min(element.sc, question.sc)) {
19            temp.insert(mp(
max(element.fi, question.fi),
min(element.sc, question.sc)
                                ));
20          }
21        }
22        lie_set = temp;
23      }
24      else {
25        ii intersection = mp(
max(truth_set.fi, question.fi),
min(truth_set.sc, question.sc)
                                 );
26        set<ii> temp;
27        if(truth_set.fi <= truth_set.sc) {
28          if(truth_set.fi==intersection.fi) {
29            truth_set = mp(
intersection.sc+1, truth_set.sc
                               );
```

Kode Sumber 4.14 Implementasi Fungsi RESULT(2)

```
30          }
31          else {
32             truth_set = mp(
truth_set.fi, intersection.fi-1
                              );
33          }
34          temp.insert(intersection);
35       }
36       for(set<ii>::iterator it =
lie_set.begin(); it!=lie_set.end(); ++it){
37          ii element = *it;
38          if(max(element.fi, question.fi) <=
               min(element.sc, question.sc)) {
39             if(element.fi == max(element.fi,
question.fi)) {
40                temp.insert(mp(
min(element.sc, question.sc)+1,
element.sc
                   ));
41             }
42             else {
43                temp.insert(mp(
element.fi,
max(element.fi, question.fi)-1
                                  ));
44             }
45          }
46          else {
47             temp.insert(element);
48          }
49       }
50       lie_set = temp;
51    }
52    truthsize = truth_set.sc-truth_set.fi+1;
53    liesize = 0;
54    for(set<ii>::iterator it =
```

Kode Sumber 4.15 Implementasi Fungsi RESULT(3)

```
    lie_set.begin(); it!=lie_set.end(); ++it){
55      ii element = *it;
56      liesize+= element.sc-element.fi+1;
57    }
58    Characteristic =
    count_characteristic(truthsize, liesize);
59    return ;
60  }
```

Kode Sumber 4.16 Implementasi Fungsi RESULT(4)

Pertama, fungsi RESULT akan menerima masukan berupa *pair* dari pertanyaan yang diajukan. Setelah itu, RESULT akan menerima respon dari *online judge*. Selanjutnya, RESULT akan mencari perpotongan dari pertanyaan yang diajukan dengan elemen dari *truth-set* maupun *lie-set* dan memprosesnya seperti pada subbab 3.2.7. Untuk mendapatkan isi dari *lie-set* yang baru, pertama dicari terlebih dahulu hasil perpotongan dan penggabungan yang dijelaskan pada subbab 3.2.7. Hasil operasi ini disimpan kedalam *set* baru untuk menampung sementara. Setelah itu, *set* ini akan digunakan untuk membaharui *lie-set*. Terakhir, variabel penampung ukuran dari *truth-set* dan *lie-set* diperbaharui sesuai dengan isi dari *set* yang baru.

4.11 Implementasi Fungsi count_characteristic

Fungsi count_characteristic akan menghitung karakteristik dari sebuah *state*, seperti telah dijelaskan pada subbab 2.2 dan 3.2.8. Implementasi dari fungsi count_characteristic dapat dilihat pada Kode Sumber 4.17 - Kode Sumber 4.18.

Fungsi count_characteristic akan mencari *j* terkecil yang memenuhi pertidaksamaan yang dijelaskan pada subbab 3.2.8. Nilai *j* didapatkan dengan cara mencari nilainya secara linear hingga pertidaksamaan pada subbab 3.2.8 dipenuhi. Setelah itu, fungsi akan berhenti dan mengembalikan nilai *j*.

```
1   long double count_characteristic(long
    double cardinality_truth, long double
```

Kode Sumber 4.17 Implementasi Fungsi count_characteristic(1)

```
    cardinality_lie) {
2       long double kanan = 1.0,kiri =
    cardinality_truth+cardinality_lie, j = 0;
3     while(kiri>kanan) {
4       j++;
5       kanan = kanan * 2;
6       kiri = kiri+cardinality_truth;
7     }
8     return j;
9   }
```

Kode Sumber 4.18 Implementasi Fungsi count_characteristic(2)

[*Halaman ini sengaja dikosongkan*]

BAB V
UJI COBA DAN EVALUASI

Pada bab ini akan dijelaskan tentang uji coba dan evaluasi dari implementasi sistem yang telah dilakukan pada bab 4.

5.1 LINGKUNGAN UJI COBA

Lingkungan uji coba menggunakan sebuah komputer dengan spesifikasi perangkat lunak dan perangkat keras seperti tercantum pada Tabel 5.1.

Tabel 5.1 Spesifikasi Lingkungan Uji Coba

No.	Jenis Perangkat	Spesifikasi
1	Perangkat Keras	• *Processor* Intel Core i7-2670QM CPU @ 2.20GHz • *Memory* 8GB 1333MHz DDR3
2	Perangkat Lunak	• Sistem operasi Windows 8 Pro • *Integrated Development Environment* Dev-C++ 5.3.0.3

5.2 UJI COBA

Uji coba dilakukan dengan analisis penyelesaian sebuah contoh kasus menggunakan pendekatan penyelesaian yang telah dijelaskan pada subbab 2.2 serta pengumpulan berkas kode sumber hasil implementasi ke situs sistem penilaian daring SPOJ. Permasalahan yang diselesaikan adalah permasalahan Guess The Number With Lies v3 dengan kode GUESSN3[8]. Uji coba keberanan akan dilakukan dalam dua tahap, yaitu uji coba lokal dan uji coba dengan menggunakan kasus uji dari daring SPOJ.

5.2.1 Uji Coba Lokal

Uji coba lokal dilakukan dengan menjalankan program dan mencari responden untuk menguji program dengan mensimulasikan permainan. Kasus uji yang ditetapkan adalah kasus uji untuk permainan (1000,14), $(999, 14)$, dan $(10^{18}, 66)$. Pemilihan $n = 1000$ dan $n = 999$ didasarkan dengan tujuan

untuk membuktikan bahwa algoritma yang diimplementasikan dapat menyelesaikan permasalahan untuk n genap dan ganjil sedangkan pemilihan $n = 10^{18}$ ditujukan untuk membuktikan bahwa algoritma yang diimplementasikan dapat menyelesaikan permawsalahan hingga batas atas n pada masalah SPOJ 17320 Guess The Number With Lies v3. Pemilihan k dalam permainan ini didasarkan dari *lemma* yang sudah ditulis pada subbab 2.2.

Skenario uji coba dilakukan dengan memberikan instruksi dan petunjuk pengujian kepada responden. Selanjutnya, responden diminta untuk memainkan permainan untuk $(1000, 14)$ dan $(999, 14)$. Pengujian yang dilakukan meliputi dua hal, yaitu pengujian ketepatan program dalam mencari bilangan yang dimaksud dan ketepatan penggunaan pertanyaan dalam mencari bilangan yang dimaksud.

5.2.1.1 Permainan (n,k) Untuk n Genap

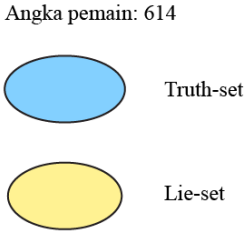
Uji coba dilakukan sebanyak 10 kali untuk permainan (1000,14). Hasil uji coba untuk permainan (1000,14) dapat dilihat pada Tabel 5.2.

Tabel 5.2 Hasil Uji Coba Permainan $(\mathbf{1000, 14})$

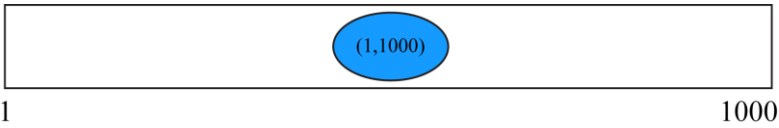
No.	Jumlah pertanyaan	Bilangan	Bohong
1	14	5	1
2	14	31	1
3	14	89	1
4	14	614	1
5	14	338	1
6	13	444	0
7	13	350	0
8	14	15	1
9	14	88	1
10	14	151	1

Dari hasil pengujian diatas, program mampu memenangkan permainan (n, k) pada seluruh percobaan untuk n yang bernilai 1000. Dalam skenario uji coba ini, seluruh angka dapat ditebak dan jumlah pertanyaan yang digunakan tidak melebihi batas k yang

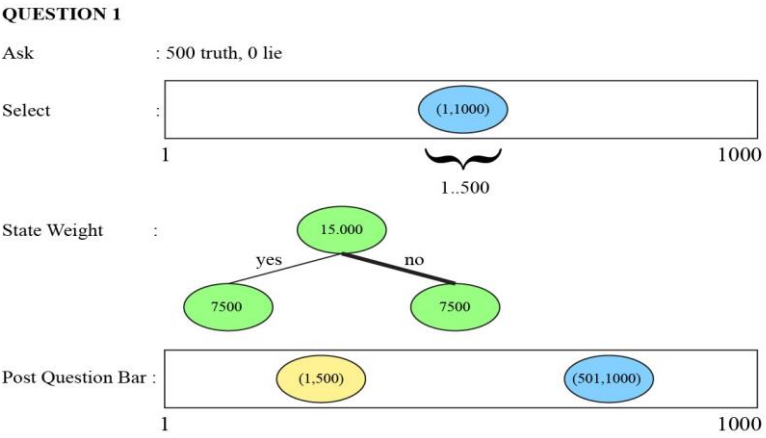
telah ditetapkan, yaitu 14. Contoh salah satu permainan dari uji coba diatas dapat dilihat pada Gambar 5.1 - Gambar 5.16.



Gambar 5.1 Keterangan dan Deskripsi Permainan $(1000, 14)$

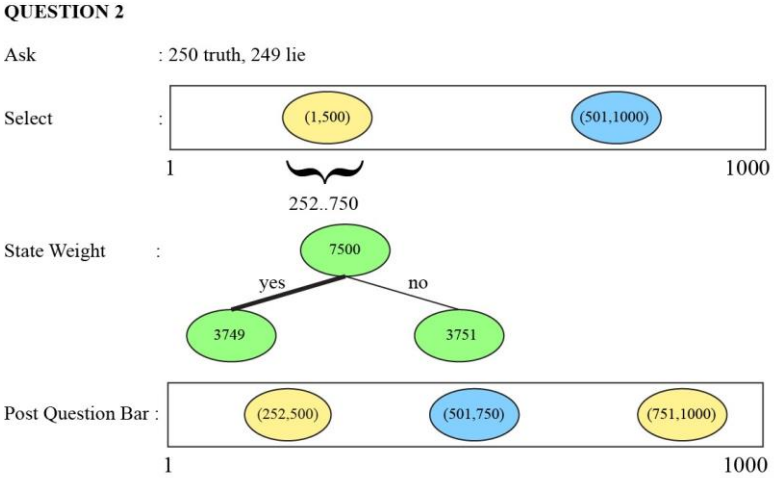


Gambar 5.2 Kondisi Awal Permainan $(1000, 14)$



Gambar 5.3 Kondisi Permainan (1000,14) Setelah Pertanyaan Pertama

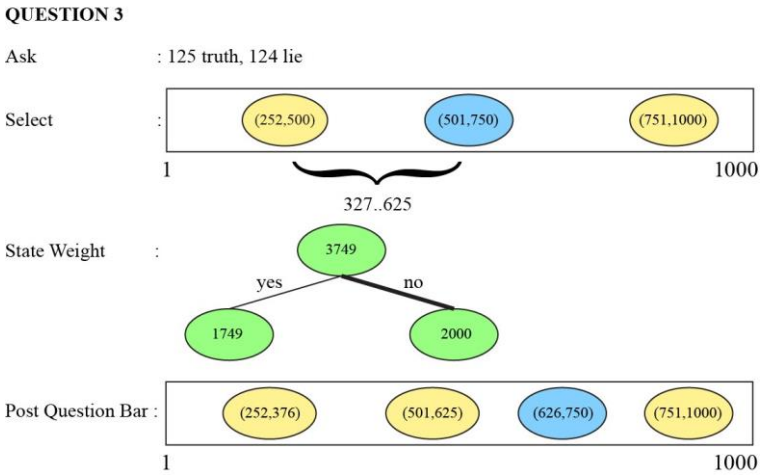
Pada pertanyaan pertama, program akan mencari sebuah rentang yang mengandung 500 elemen dari *truth-set* dan 0 elemen dari *lie-set*. Jumlah elemen tersebut dipilih karena akan menyebabkan *state* yang dituju berimbang. Setelah dilakukan pencarian, program menemukan bahwa rentang [1..500] memenuhi pencarian yang diinginkan. Selanjutnya, program menanyakan pertanyaan tersebut kepada pemain. Pemain memilih untuk menjawab dengan jujur sehingga menjawab pertanyaan dengan respon "NO". Setelah itu, *truth-set* dan *lie-set* diproses sedemikian sehingga menjadi *state* pada bagian *Post Question Bar* pada Gambar 5.3.



Gambar 5.4 Kondisi Permainan (1000,14) Setelah Pertanyaan Kedua

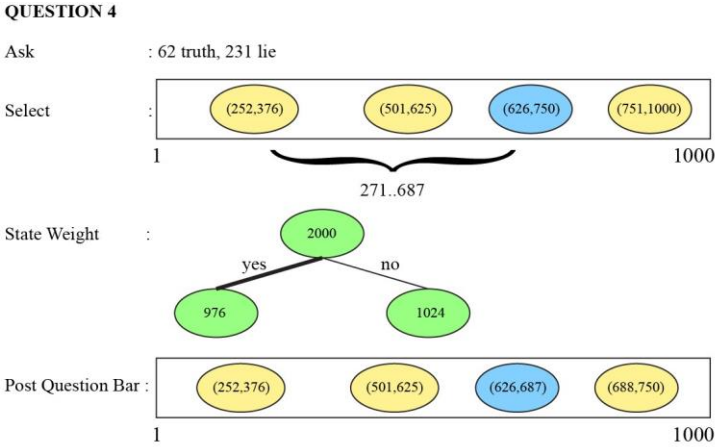
Pada pertanyaan kedua, program akan mencari sebuah rentang yang mengandung 250 elemen *truth-set* dan 249 elemen dari *lie-set*. Jumlah elemen tersebut dipilih karena akan menyebabkan *state* yang dituju berimbang. Setelah dilakukan pencarian, program menemukan bahwa rentang [252..750] memenuhi pencarian yang diinginkan. Selanjutnya, program

menanyakan pertanyaan tersebut kepada pemain. Pemain memilih untuk menjawab dengan jujur sehingga menjawab pertanyaan dengan "YES". Setelah itu, *truth-set* dan *lie-set* diproses sehingga menjadi *state* seperti ditunjukkan pada *Post Question Bar* pada Gambar 5.4.



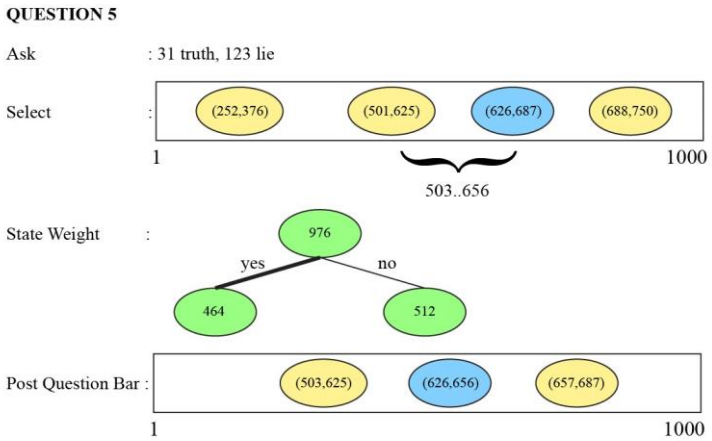
Gambar 5.5 Kondisi Permainan (1000,14) Setelah Pertanyaan Ketiga

Pada pertanyaan ketiga, program akan mencari sebuah rentang yang mengandung 125 elemen dari *truth-set* dan 124 elemen dari *lie-set*. Jumlah elemen tersebut dipilih karena akan menyebabkan *state* yang dituju berimbang. Setelah dilakukan pencarian, program menemukan bahwa rentang [327..625] memenuhi pencarian yang diinginkan. Selanjutnya, program menanyakan pertanyaan tersebut kepada pemain. Pemain memilih untuk menjawab dengan bohong sehingga menjawab pertanyaan dengan "NO". Perlu dicatat bahwa pemain tidak akan berbohong lagi setelah pertanyaan ketiga ini. Setelah itu, *truth-set* dan *lie-set* diproses sehingga menjadi *state* seperti ditunjukkan pada *Post Question Bar* pada Gambar 5.5.



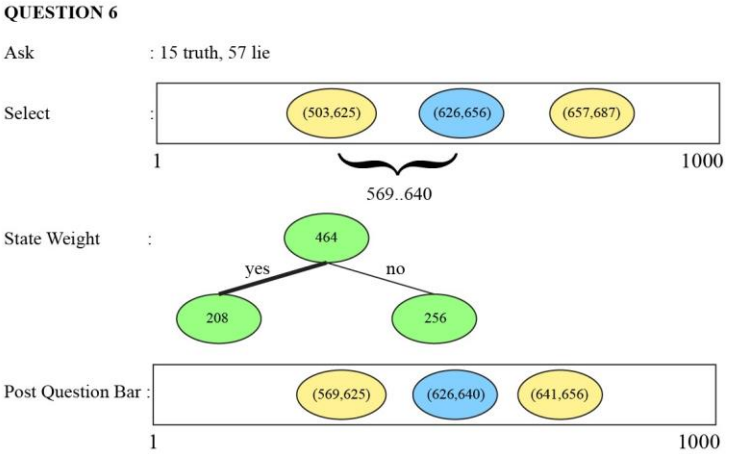
Gambar 5.6 Kondisi Permainan (1000,14) Setelah Pertanyaan Keempat

Pada pertanyaan keempat, program akan mencari sebuah rentang yang mengandung 62 elemen dari *truth-set* dan 231 elemen dari *lie-set*. Jumlah elemen tersebut dipilih karena akan menyebabkan *state* yang dituju berimbang. Setelah dilakukan pencarian, program menemukan bahwa rentang [271..687] memenuhi pencarian yang diinginkan. Selanjutnya, program menanyakan pertanyaan tersebut kepada pemain. Pemain harus menjawab dengan jujur sehingga menjawab pertanyaan dengan "YES". Setelah itu, *truth-set* dan *lie-set* diproses sehingga menjadi *state* seperti ditunjukkan pada *Post Question Bar* pada Gambar 5.6.



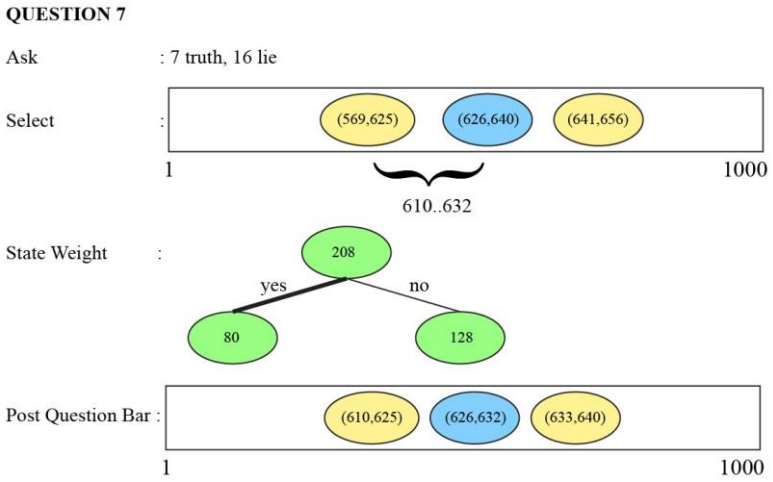
Gambar 5.7 Kondisi Permainan (1000,14) Setelah Pertanyaan Kelima

Pada pertanyaan kelima, program akan mencari sebuah rentang yang mengandung 31 elemen dari *truth-set* dan 123 elemen dari *lie-set*. Jumlah elemen tersebut dipilih karena akan menyebabkan *state* yang dituju berimbang. Setelah dilakukan pencarian, program menemukan bahwa rentang [503..656] memenuhi pencarian yang diinginkan. Selanjutnya, program menanyakan pertanyaan tersebut kepada pemain. Pemain harus menjawab dengan jujur sehingga menjawab pertanyaan dengan "YES". Setelah itu, *truth-set* dan *lie-set* diproses sehingga menjadi *state* seperti ditunjukkan pada *Post Question Bar* pada Gambar 5.7.



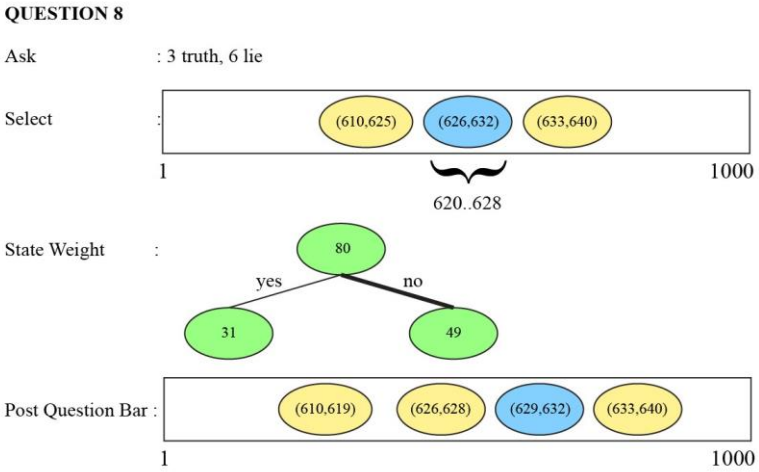
Gambar 5.8 Kondisi Permainan (1000,14) Setelah Pertanyaan Keenam

Pada pertanyaan keenam, program akan mencari sebuah rentang yang mengandung 15 elemen dari *truth-set* dan 57 elemen dari *lie-set*. Jumlah elemen tersebut dipilih karena akan menyebabkan *state* yang dituju berimbang. Setelah dilakukan pencarian, program menemukan bahwa rentang [569..640] memenuhi pencarian yang diinginkan. Selanjutnya, program menanyakan pertanyaan tersebut kepada pemain. Pemain harus menjawab dengan jujur sehingga menjawab pertanyaan dengan "YES". Setelah itu, *truth-set* dan *lie-set* diproses sehingga menjadi *state* seperti ditunjukkan pada *Post Question Bar* pada Gambar 5.8.



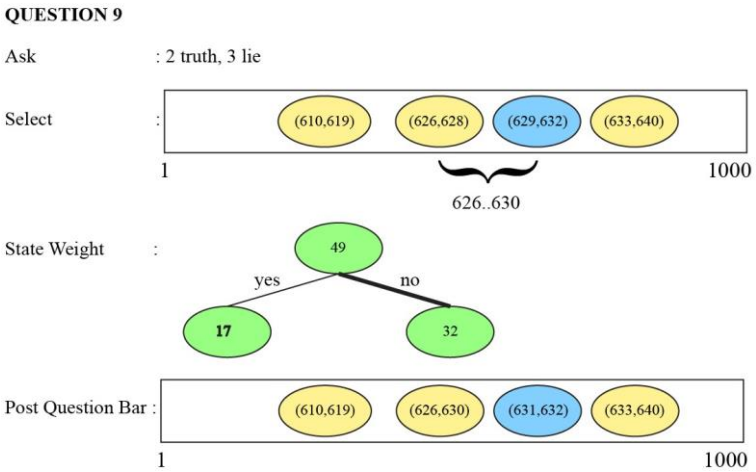
Gambar 5.9 Kondisi Permainan (1000,14) Setelah Pertanyaan Ketujuh

Pada pertanyaan ketujuh, program akan mencari sebuah rentang yang mengandung 7 elemen dari *truth-set* dan 16 elemen dari *lie-set*. Jumlah elemen tersebut dipilih karena akan menyebabkan *state* yang dituju berimbang. Setelah dilakukan pencarian, program menemukan bahwa rentang [610..632] memenuhi pencarian yang diinginkan. Selanjutnya, program menanyakan pertanyaan tersebut kepada pemain. Pemain harus menjawab dengan jujur sehingga menjawab pertanyaan dengan "YES". Setelah itu, *truth-set* dan *lie-set* diproses sehingga menjadi *state* seperti ditunjukkan pada *Post Question Bar* pada Gambar 5.9.



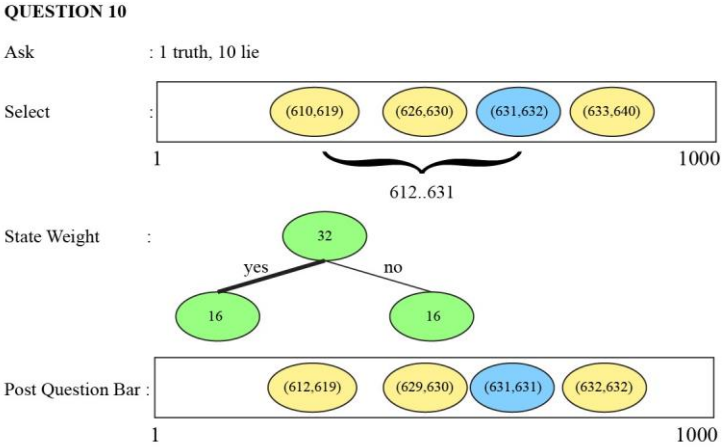
Gambar 5.10 Kondisi Permainan (1000,14) Setelah Pertanyaan Kedelapan

Pada pertanyaan kedelapan, program akan mencari sebuah rentang yang mengandung 3 elemen dari *truth-set* dan 6 elemen dari *lie-set*. Jumlah elemen tersebut dipilih karena akan menyebabkan *state* yang dituju berimbang. Setelah dilakukan pencarian, program menemukan bahwa rentang [620..628] memenuhi pencarian yang diinginkan. Selanjutnya, program menanyakan pertanyaan tersebut kepada pemain. Pemain harus menjawab dengan jujur sehingga menjawab pertanyaan dengan "NO". Setelah itu, *truth-set* dan *lie-set* diproses sehingga menjadi *state* seperti ditunjukkan pada *Post Question Bar* pada Gambar 5.10.



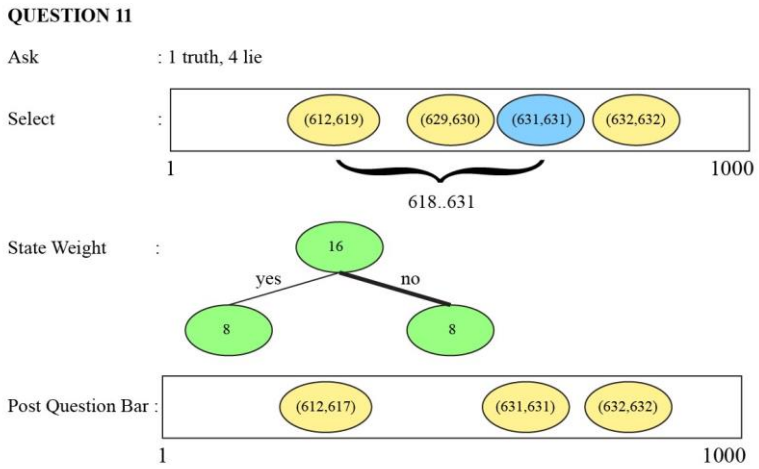
Gambar 5.11 Kondisi Permainan (1000,14) Setelah Pertanyaan Kesembilan

Pada pertanyaan kesembilan, program akan mencari sebuah rentang yang mengandung 2 elemen dari *truth-set* dan 3 elemen dari *lie-set*. Jumlah elemen tersebut dipilih karena akan menyebabkan *state* yang dituju berimbang. Setelah dilakukan pencarian, program menemukan bahwa rentang [626..630] memenuhi pencarian yang diinginkan. Selanjutnya, program menanyakan pertanyaan tersebut kepada pemain. Pemain harus menjawab dengan jujur sehingga menjawab pertanyaan dengan "NO". Setelah itu, *truth-set* dan *lie-set* diproses sehingga menjadi *state* seperti ditunjukkan pada *Post Question Bar* pada Gambar 5.11.



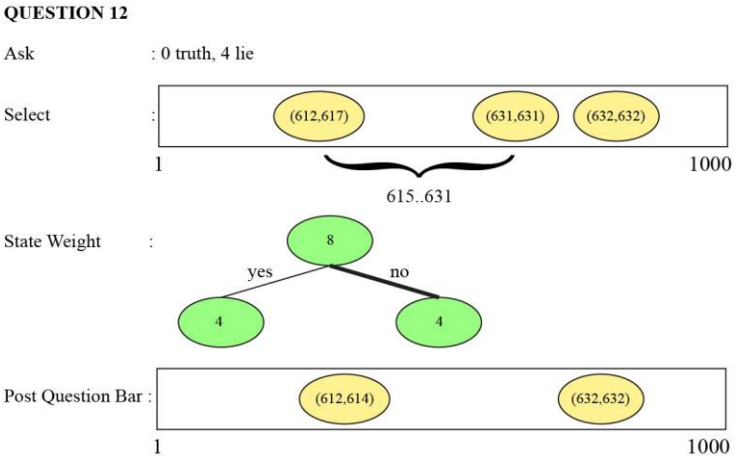
Gambar 5.12 Kondisi Permainan (1000,14) Setelah Pertanyaan Kesepuluh

Pada pertanyaan kesepuluh, program akan mencari sebuah rentang yang mengandung 1 elemen dari *truth-set* dan 10 elemen dari *lie-set*. Jumlah elemen tersebut dipilih karena akan menyebabkan *state* yang dituju berimbang. Setelah dilakukan pencarian, program menemukan bahwa rentang [612..631] memenuhi pencarian yang diinginkan. Selanjutnya, program menanyakan pertanyaan tersebut kepada pemain. Pemain harus menjawab dengan jujur sehingga menjawab pertanyaan dengan "YES". Setelah itu, *truth-set* dan *lie-set* diproses sehingga menjadi *state* seperti ditunjukkan pada *Post Question Bar* pada Gambar 5.12.



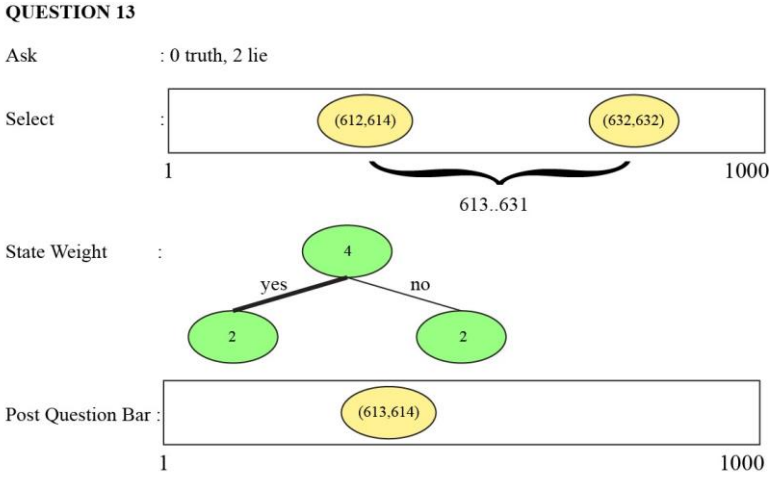
Gambar 5.13 Kondisi Permainan (1000,14) Setelah Pertanyaan Kesebelas

Pada pertanyaan kesebelas, program akan mencari sebuah rentang yang mengandung 1 elemen dari *truth-set* dan 4 elemen dari *lie-set*. Jumlah elemen tersebut dipilih karena akan menyebabkan *state* yang dituju berimbang. Setelah dilakukan pencarian, program menemukan bahwa rentang [618..631] memenuhi pencarian yang diinginkan. Selanjutnya, program menanyakan pertanyaan tersebut kepada pemain. Pemain harus menjawab dengan jujur sehingga menjawab pertanyaan dengan "NO". Setelah itu, *truth-set* dan *lie-set* diproses sehingga menjadi *state* seperti ditunjukkan pada *Post Question Bar* pada Gambar 5.13.



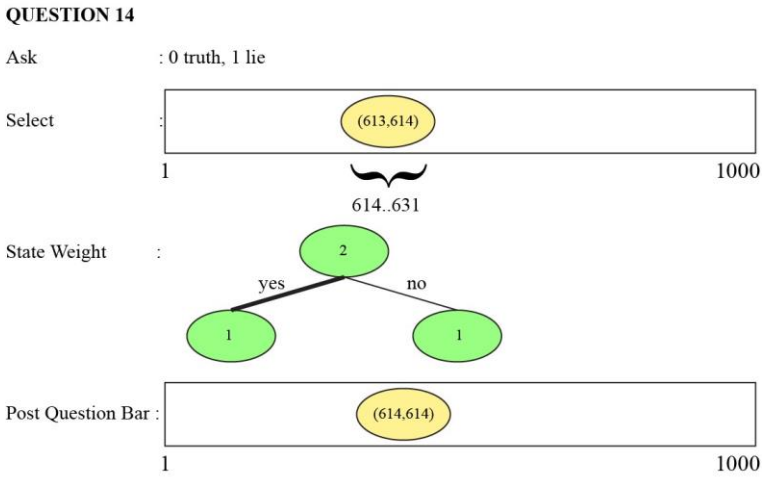
Gambar 5.14 Kondisi Permainan (1000,14) Setelah Pertanyaan Kedua belas

Pada pertanyaan kedua belas, program akan mencari sebuah rentang yang mengandung 0 elemen dari *truth-set* dan 4 elemen dari *lie-set*. Jumlah elemen tersebut dipilih karena akan menyebabkan *state* yang dituju berimbang. Setelah dilakukan pencarian, program menemukan bahwa rentang [615..631] memenuhi pencarian yang diinginkan. Selanjutnya, program menanyakan pertanyaan tersebut kepada pemain. Pemain harus menjawab dengan jujur sehingga menjawab pertanyaan dengan "NO". Setelah itu, *truth-set* dan *lie-set* diproses sehingga menjadi *state* seperti ditunjukkan pada *Post Question Bar* pada Gambar 5.14.



Gambar 5.15 Kondisi Permainan (1000,14) Setelah Pertanyaan Ketiga belas

Pada pertanyaan ketiga belas, program akan mencari sebuah rentang yang mengandung 0 elemen dari *truth-set* dan 2 elemen dari *lie-set*. Jumlah elemen tersebut dipilih karena akan menyebabkan *state* yang dituju berimbang. Setelah dilakukan pencarian, program menemukan bahwa rentang [613..631] memenuhi pencarian yang diinginkan. Selanjutnya, program menanyakan pertanyaan tersebut kepada pemain. Pemain harus menjawab dengan jujur sehingga menjawab pertanyaan dengan "YES". Setelah itu, *truth-set* dan *lie-set* diproses sehingga menjadi *state* seperti ditunjukkan pada *Post Question Bar* pada Gambar 5.15.



Gambar 5.16 Kondisi Permainan (1000,14) Setelah Pertanyaan Keempat belas

Pada pertanyaan keempat belas, program akan mencari sebuah rentang yang mengandung 0 elemen dari *truth-set* dan 1 elemen dari *lie-set*. Jumlah elemen tersebut dipilih karena akan menyebabkan *state* yang dituju berimbang. Setelah dilakukan pencarian, program menemukan bahwa rentang [614..631] memenuhi pencarian yang diinginkan. Selanjutnya, program menanyakan pertanyaan tersebut kepada pemain. Pemain harus menjawab dengan jujur sehingga menjawab pertanyaan dengan "YES". Setelah itu, *truth-set* dan *lie-set* diproses sehingga menjadi *state* seperti ditunjukkan pada *Post Question Bar* pada Gambar 5.16.

Setelah mencapai *Post Question Bar* pada Gambar 5.16, dapat dilihat bahwa setelah pertanyaan keempatbelas, hanya tersisa 1 elemen, yaitu elemen pada *lie-set*. Maka dari itu, program akan menghentikan pertanyaan dan menebak angka yang dimaksud oleh penjawab, yaitu 614. Pada salah satu contoh permainan ini, program dapat menyelesaikan permainan sesuai dengan batasan

yang telah ditentukan. Contoh berjalannya program untuk menyelesaikan permainan ini dapat dilihat pada Gambar 5.33.

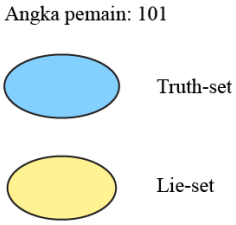
5.2.1.2 Permainan (n,k) Untuk n Ganjil

Uji coba dilakukan sebanyak 10 kali untuk permainan (999,14). Hasil uji coba untuk permainan (999,14) dapat dilihat pada Tabel 5.3.

Tabel 5.3 Hasil Uji Coba Permainan (999,14)

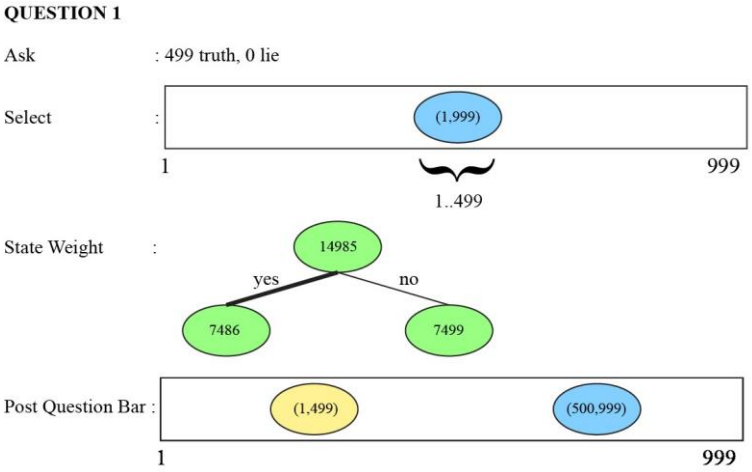
No.	Jumlah pertanyaan	Bilangan	Bohong
1	14	123	1
2	14	214	1
3	14	101	1
4	13	500	1
5	14	300	1
6	14	666	1
7	14	595	1
8	14	277	1
9	14	21	1
10	14	951	1

Dari hasil pengujian seperti pada Tabel 5.3, program mampu memenangkan permainan (n, k) pada seluruh percobaan untuk n yang bernilai 999. Dalam skenario uji coba ini, seluruh angka dapat ditebak dengan jumlah pertanyaan yang digunakan tidak melebihi batas k, yaitu 14. Contoh salah satu permainan dari skenario uji coba diatas dapat dilihat pada Gambar 5.17 - Gambar 5.32.



Gambar 5.17 Keterangan dan Deskripsi Permainan $(999, 14)$

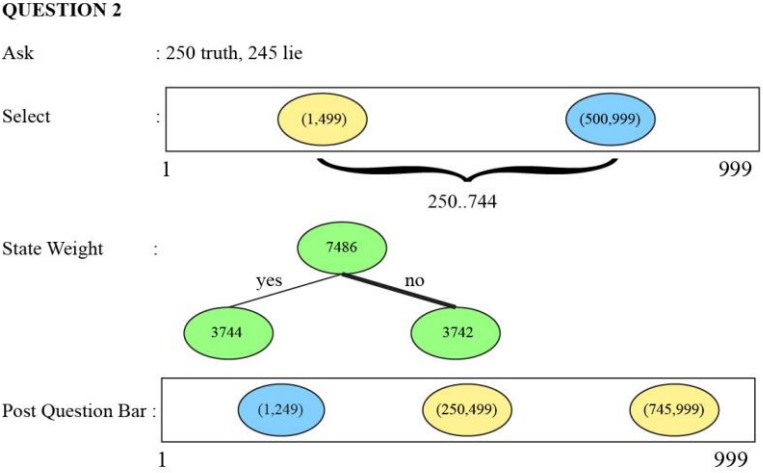
Gambar 5.18 Kondisi Awal Permainan $(999, 14)$



Gambar 5.19 Kondisi Permainan $(999, 14)$ Setelah Pertanyaan Pertama

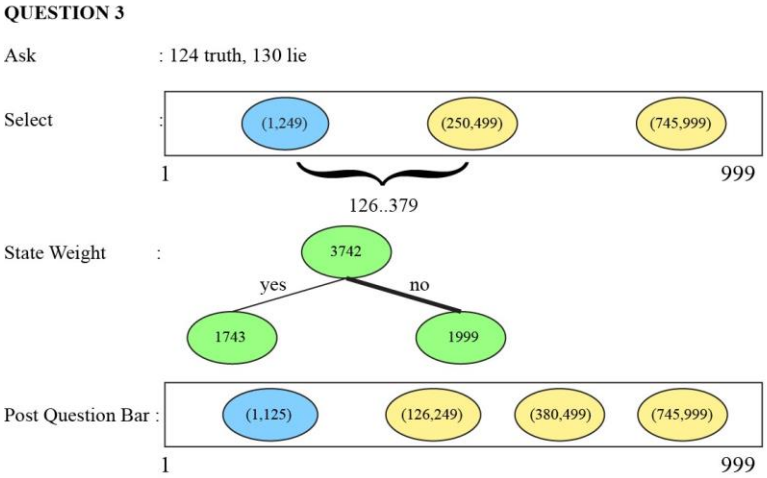
Pada pertanyaan pertama, program akan mencari sebuah rentang yang mengandung 499 elemen dari *truth-set* dan 0 elemen dari *lie-set*. Jumlah elemen tersebut dipilih karena akan menyebabkan *state* yang dituju berimbang. Setelah dilakukan pencarian, program menemukan bahwa rentang [1..499] memenuhi pencarian yang diinginkan. Selanjutnya, program menanyakan pertanyaan tersebut kepada pemain. Pemain memilih untuk menjawab dengan jujur sehingga menjawab pertanyaan dengan respon "YES". Setelah itu, *truth-set* dan *lie-set* diproses sedemikian sehingga menjadi *state* pada bagian *Post Question Bar* pada Gambar 5.19.

.



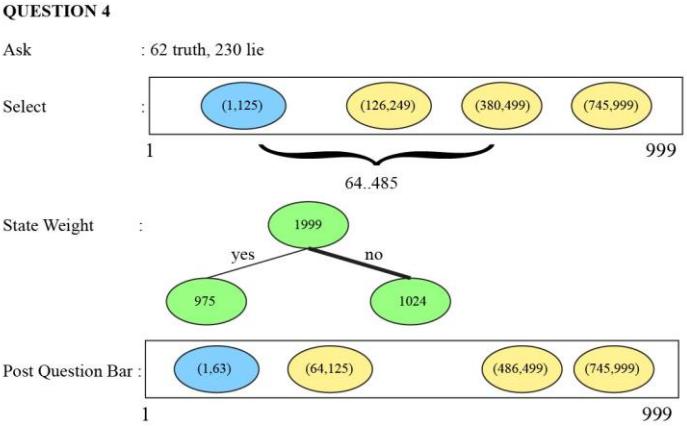
Gambar 5.20 Kondisi Permainan (999, 14) Setelah Pertanyaan Kedua

Pada pertanyaan kedua, program akan mencari sebuah rentang yang mengandung 250 elemen dari *truth-set* dan 245 elemen dari *lie-set*. Jumlah elemen tersebut dipilih karena akan menyebabkan *state* yang dituju berimbang. Setelah dilakukan pencarian, program menemukan bahwa rentang [250..744] memenuhi pencarian yang diinginkan. Selanjutnya, program menanyakan pertanyaan tersebut kepada pemain. Pemain memilih untuk menjawab dengan jujur sehingga menjawab pertanyaan dengan respon "NO". Setelah itu, *truth-set* dan *lie-set* diproses sedemikian sehingga menjadi *state* pada bagian *Post Question Bar* pada Gambar 5.20.



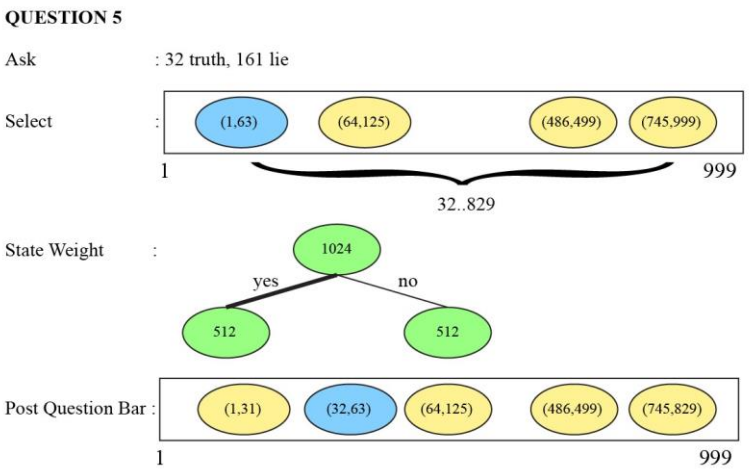
Gambar 5.21 Kondisi Permainan $(999, 14)$ Setelah Pertanyaan Ketiga

Pada pertanyaan ketiga, program akan mencari sebuah rentang yang mengandung 1124 elemen dari *truth-set* dan 130 elemen dari *lie-set*. Jumlah elemen tersebut dipilih karena akan menyebabkan *state* yang dituju berimbang. Setelah dilakukan pencarian, program menemukan bahwa rentang [126..379] memenuhi pencarian yang diinginkan. Selanjutnya, program menanyakan pertanyaan tersebut kepada pemain. Pemain memilih untuk menjawab dengan jujur sehingga menjawab pertanyaan dengan respon "NO". Setelah itu, *truth-set* dan *lie-set* diproses sedemikian sehingga menjadi *state* pada bagian *Post Question Bar* pada Gambar 5.21.



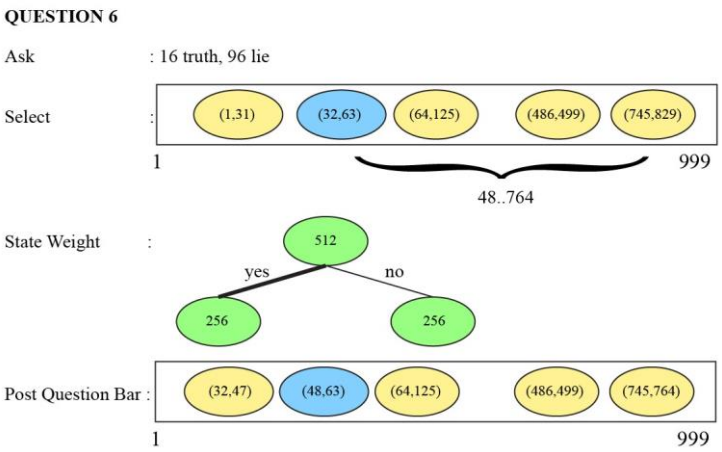
Gambar 5.22 Kondisi Permainan $(999, 14)$ Setelah Pertanyaan Keempat

Pada pertanyaan keempat, program akan mencari sebuah rentang yang mengandung 62 elemen dari *truth-set* dan 230 elemen dari *lie-set*. Jumlah elemen tersebut dipilih karena akan menyebabkan *state* yang dituju berimbang. Setelah dilakukan pencarian, program menemukan bahwa rentang [64..485] memenuhi pencarian yang diinginkan. Pemain memilih untuk berbohong sehingga menjawab pertanyaan dengan respon "NO". Perlu dicatat bahwa penjawab tidak dapat lagi berbohong setelah pertanyaan keempat. Setelah itu, *truth-set* dan *lie-set* diproses sedemikian sehingga menjadi *state* pada bagian *Post Question Bar* pada Gambar 5.22.



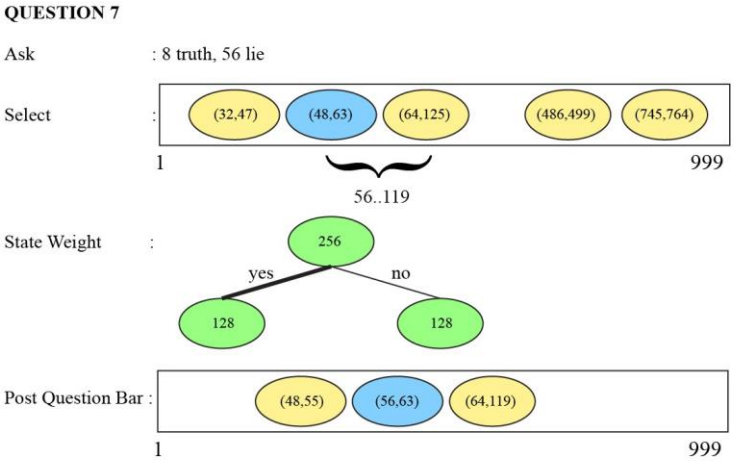
Gambar 5.23 Kondisi Permainan $(999, 14)$ Setelah Pertanyaan Kelima

Pada pertanyaan kelima, program akan mencari sebuah rentang yang mengandung 32 elemen dari *truth-set* dan 161 elemen dari *lie-set*. Jumlah elemen tersebut dipilih karena akan menyebabkan *state* yang dituju berimbang. Setelah dilakukan pencarian, program menemukan bahwa rentang [32..829] memenuhi pencarian yang diinginkan. Selanjutnya, program menanyakan pertanyaan tersebut kepada pemain. Pemain menjawab dengan jujur sehingga menjawab pertanyaan dengan respon "YES". Setelah itu, *truth-set* dan *lie-set* diproses sedemikian sehingga menjadi *state* pada bagian *Post Question Bar* pada Gambar 5.23.



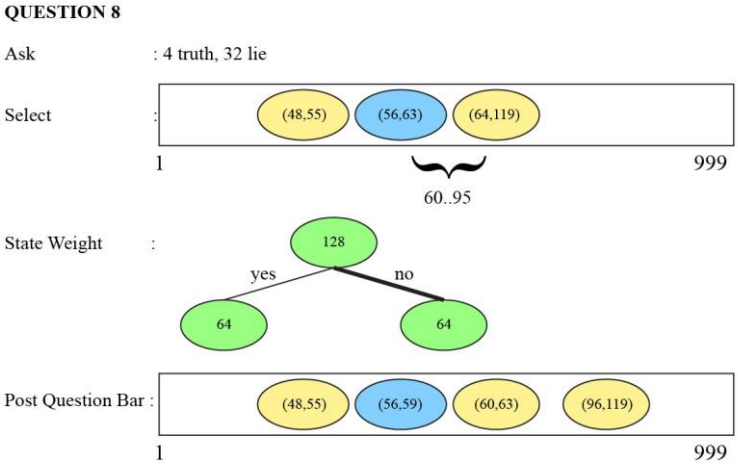
Gambar 5.24 Kondisi Permainan $(999, 14)$ Setelah Pertanyaan Keenam

Pada pertanyaan keenam, program akan mencari sebuah rentang yang mengandung 16 elemen dari *truth-set* dan 96 elemen dari *lie-set*. Jumlah elemen tersebut dipilih karena akan menyebabkan *state* yang dituju berimbang. Setelah dilakukan pencarian, program menemukan bahwa rentang [49..764] memenuhi pencarian yang diinginkan. Selanjutnya, program menanyakan pertanyaan tersebut kepada pemain. Pemain menjawab dengan jujur sehingga menjawab pertanyaan dengan respon "YES". Setelah itu, *truth-set* dan *lie-set* diproses sedemikian sehingga menjadi *state* pada bagian *Post Question Bar* pada Gambar 5.24.



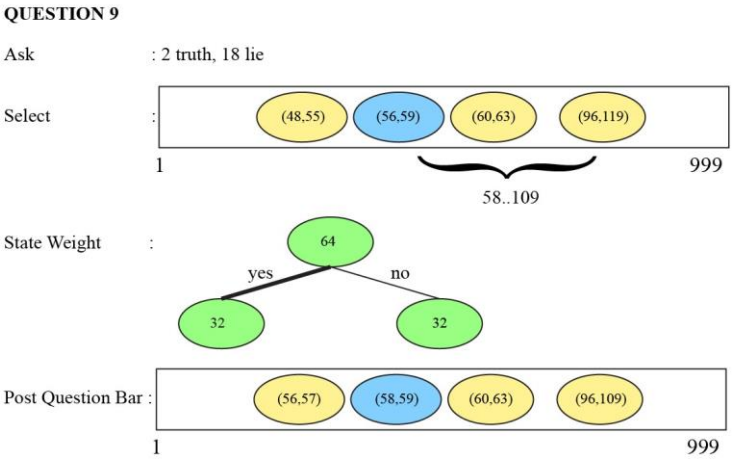
Gambar 5.25 Kondisi Permainan $(999, 14)$ Setelah Pertanyaan Ketujuh

Pada pertanyaan ketujuh, program akan mencari sebuah rentang yang mengandung 8 elemen dari *truth-set* dan 56 elemen dari *lie-set*. Jumlah elemen tersebut dipilih karena akan menyebabkan *state* yang dituju berimbang. Setelah dilakukan pencarian, program menemukan bahwa rentang [56..119] memenuhi pencarian yang diinginkan. Selanjutnya, program menanyakan pertanyaan tersebut kepada pemain Pemain menjawab dengan jujur sehingga menjawab pertanyaan dengan respon "YES". Setelah itu, *truth-set* dan *lie-set* diproses sedemikian sehingga menjadi *state* pada bagian *Post Question Bar* pada Gambar 5.25.



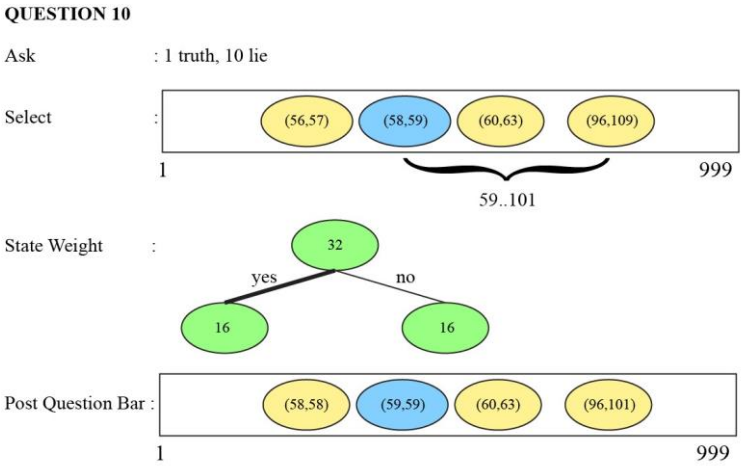
Gambar 5.26 Kondisi Permainan $(999, 14)$ Setelah Pertanyaan Kedelapan

Pada pertanyaan kedelapan, program akan mencari sebuah rentang yang mengandung 4 elemen dari *truth-set* dan 32 elemen dari *lie-set*. Jumlah elemen tersebut dipilih karena akan menyebabkan *state* yang dituju berimbang. Setelah dilakukan pencarian, program menemukan bahwa rentang [60..95] memenuhi pencarian yang diinginkan. Selanjutnya, program menanyakan pertanyaan tersebut kepada pemain. Pemain menjawab dengan jujur sehingga menjawab pertanyaan dengan respon "NO". Setelah itu, *truth-set* dan *lie-set* diproses sedemikian sehingga menjadi *state* pada bagian *Post Question Bar* pada Gambar 5.26.



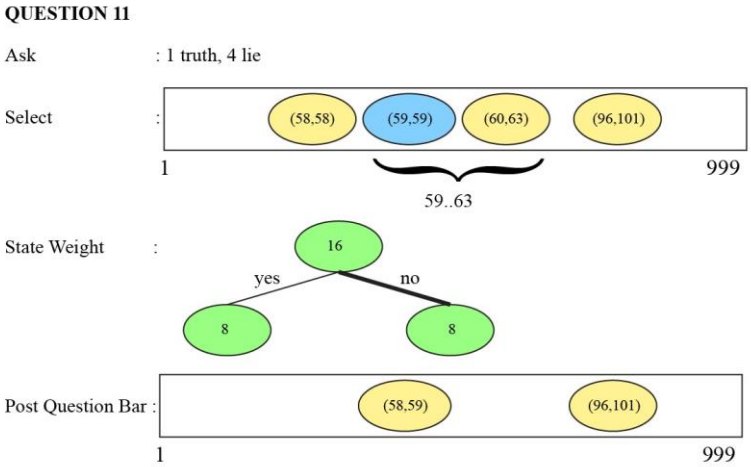
Gambar 5.27 Kondisi Pertanyaan (999, 14) Setelah Pertanyaan Kesembilan

Pada pertanyaan kesembilan, program akan mencari sebuah rentang yang mengandung 2 elemen dari *truth-set* dan 18 elemen dari *lie-set*. Jumlah elemen tersebut dipilih karena akan menyebabkan *state* yang dituju berimbang. Setelah dilakukan pencarian, program menemukan bahwa rentang [58..109] memenuhi pencarian yang diinginkan. Selanjutnya, program menanyakan pertanyaan tersebut kepada pemain. Pemain menjawab dengan jujur sehingga menjawab pertanyaan dengan respon "YES". Setelah itu, *truth-set* dan *lie-set* diproses sedemikian sehingga menjadi *state* pada bagian *Post Question Bar* pada Gambar 5.27.



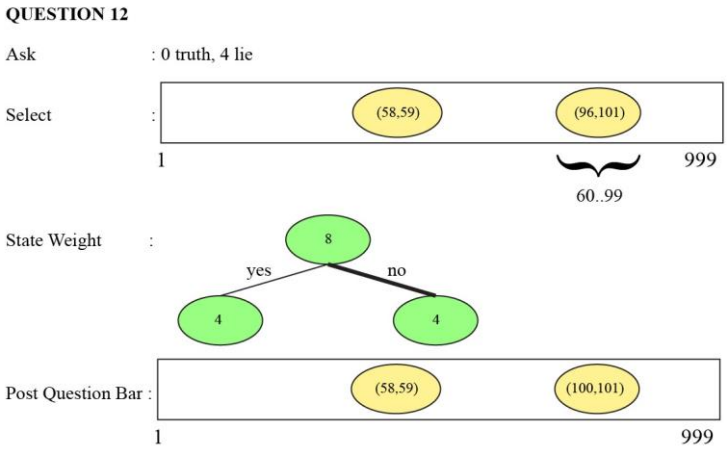
Gambar 5.28 Kondisi Permainan $(999, 14)$ Setelah Pertanyaan Kesepuluh

Pada pertanyaan kesepuluh, program akan mencari sebuah rentang yang mengandung 1 elemen dari *truth-set* dan 10 elemen dari *lie-set*. Jumlah elemen tersebut dipilih karena akan menyebabkan *state* yang dituju berimbang. Setelah dilakukan pencarian, program menemukan bahwa rentang [59..101] memenuhi pencarian yang diinginkan. Selanjutnya, program menanyakan pertanyaan tersebut kepada pemain. Pemain menjawab dengan jujur sehingga menjawab pertanyaan dengan respon "YES". Setelah itu, *truth-set* dan *lie-set* diproses sedemikian sehingga menjadi *state* pada bagian *Post Question Bar* pada Gambar 5.28.



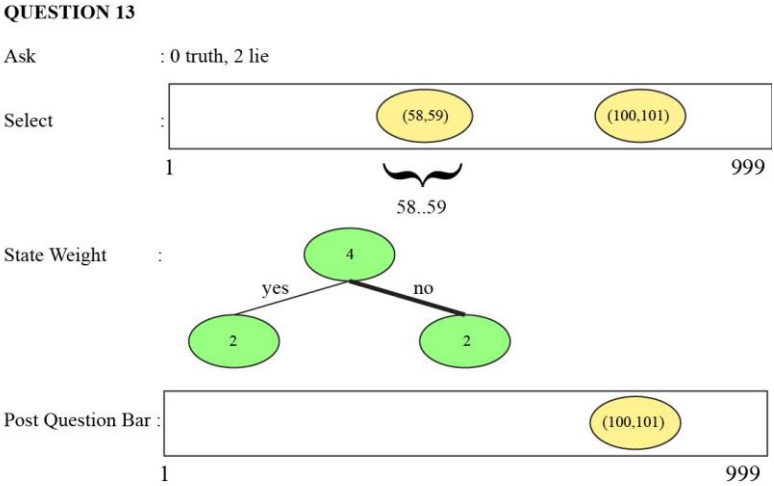
Gambar 5.29 Kondisi Permainan $(999, 14)$ Setelah Pertanyaan Kesebelas

Pada pertanyaan kesebelas, program akan mencari sebuah rentang yang mengandung 1 elemen dari *truth-set* dan 4 elemen dari *lie-set*. Jumlah elemen tersebut dipilih karena akan menyebabkan *state* yang dituju berimbang. Setelah dilakukan pencarian, program menemukan bahwa rentang [59..63] memenuhi pencarian yang diinginkan. Selanjutnya, program menanyakan pertanyaan tersebut kepada pemain. Pemain menjawab dengan jujur sehingga menjawab pertanyaan dengan respon "NO". Setelah itu, *truth-set* dan *lie-set* diproses sedemikian sehingga menjadi *state* pada bagian *Post Question Bar* pada Gambar 5.29.



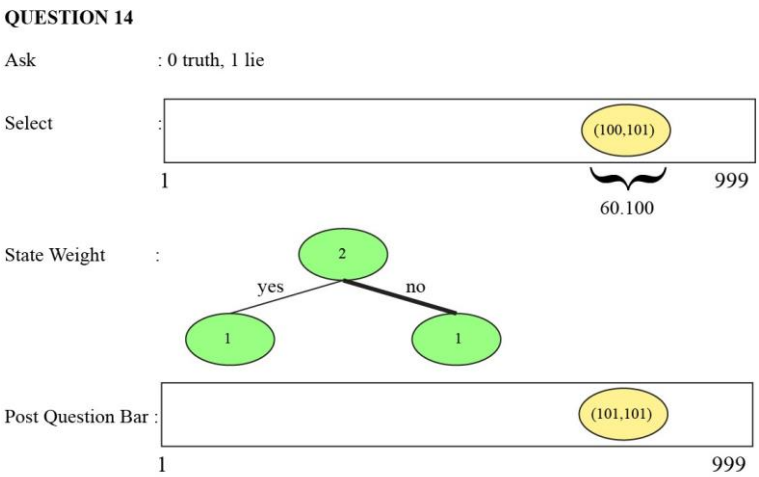
Gambar 5.30 Kondisi Pertanyaan $(999, 14)$ Setelah Pertanyaan Kedua belas

Pada pertanyaan kedua belas, program akan mencari sebuah rentang yang mengandung 0 elemen dari *truth-set* dan 4 elemen dari *lie-set*. Jumlah elemen tersebut dipilih karena akan menyebabkan *state* yang dituju berimbang. Setelah dilakukan pencarian, program menemukan bahwa rentang [60..99] memenuhi pencarian yang diinginkan. Selanjutnya, program menanyakan pertanyaan tersebut kepada pemain. Pemain menjawab dengan jujur sehingga menjawab pertanyaan dengan respon "NO". Setelah itu, *truth-set* dan *lie-set* diproses sedemikian sehingga menjadi *state* pada bagian *Post Question Bar* pada Gambar 5.30.



Gambar 5.31 Kondisi Permainan $(9999, 18)$ Setelah Pertanyaan Ketiga belas

Pada pertanyaan ketiga belas, program akan mencari sebuah rentang yang mengandung 0 elemen dari *truth-set* dan 2 elemen dari *lie-set*. Jumlah elemen tersebut dipilih karena akan menyebabkan *state* yang dituju berimbang. Setelah dilakukan pencarian, program menemukan bahwa rentang [58..59] memenuhi pencarian yang diinginkan. Selanjutnya, program menanyakan pertanyaan tersebut kepada pemain. Pemain menjawab dengan jujur sehingga menjawab pertanyaan dengan respon "NO". Setelah itu, *truth-set* dan *lie-set* diproses sedemikian sehingga menjadi *state* pada bagian *Post Question Bar* pada Gambar 5.31.



Gambar 5.32 Kondisi Permainan (999, 14) Setelah Pertanyaan Keempat belas

Pada pertanyaan keempat belas, program akan mencari sebuah rentang yang mengandung 0 elemen dari *truth-set* dan 1 elemen dari *lie-set*. Jumlah elemen tersebut dipilih karena akan menyebabkan *state* yang dituju berimbang. Setelah dilakukan pencarian, program menemukan bahwa rentang [60..100] memenuhi pencarian yang diinginkan. Selanjutnya, program menanyakan pertanyaan tersebut kepada pemain. Pemain menjawab dengan jujur sehingga menjawab pertanyaan dengan respon "NO". Setelah itu, *truth-set* dan *lie-set* diproses sedemikian sehingga menjadi *state* pada bagian *Post Question Bar* pada Gambar 5.32.

Setelah mencapai *Post Question Bar* pada Gambar 5.32, dapat dilihat bahwa setelah pertanyaan kedelapanbelas, hanya tersisa 1 elemen, yaitu elemen pada *lie-set*. Maka dari itu, program akan menghentikan pertanyaan dan menebak angka yang dimaksud oleh penjawab, yaitu 101. Pada salah satu contoh permainan ini, program dapat menyelesaikan permainan sesuai dengan batasan

yang telah ditentukan. Contoh berjalannya program untuk menyelesaikan permainan ini dapat dilihat pada Gambar 5.34 .

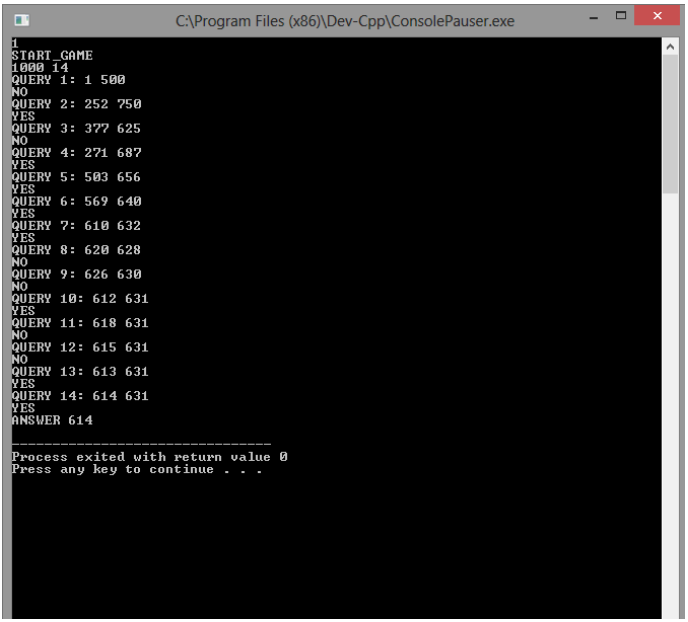
5.2.1.3 Permainan (n, k) Untuk n yang Bernilai Besar

Uji coba dilakukan sebanyak 10 kali untuk permainan $(10^{18}, 66)$. Hasil uji coba untuk permainan $(10^{18}, 66)$ dapat dilihat pada Tabel 5.4.

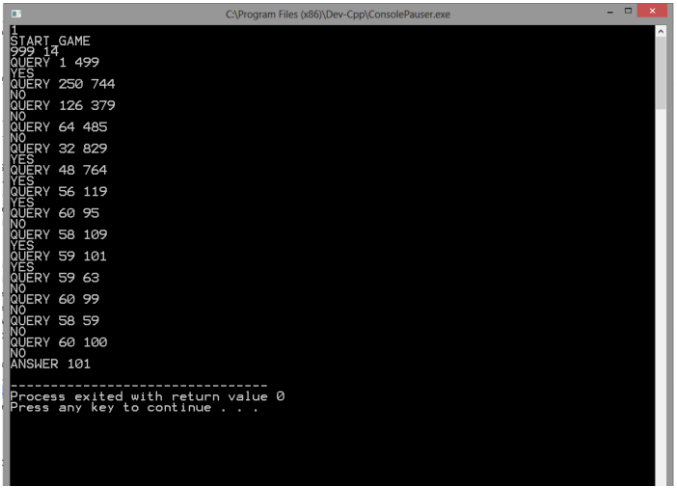
Tabel 5.4 Hasil Uji Coba Permainan $(10^{18}, 66)$

No.	Jumlah pertanyaan	Bilangan	Bohong
1	66	18375	1
2	66	9500	1
3	66	1234	1
4	66	878	1
5	66	1111	1
6	66	600	1
7	66	30000	1
8	66	15000	1
9	66	9876	1
10	66	1000000	1

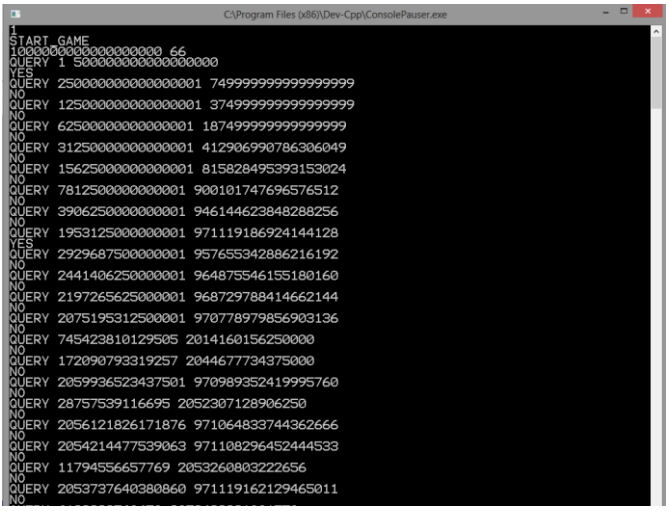
Dari hasil pengujian seperti pada Tabel 5.4, program mampu memenangkan permainan (n, k) pada seluruh percobaan untuk n yang bernilai 10^{18}. Dalam skenario uji coba ini, seluruh angka dapat ditebak dengan jumlah pertanyaan yang digunakan tidak melebihi batas k, yaitu 66. Contoh berjalannya program untuk menyelesaikan permainan ini dapat dilihat pada Gambar 5.35 - Gambar 5.38.



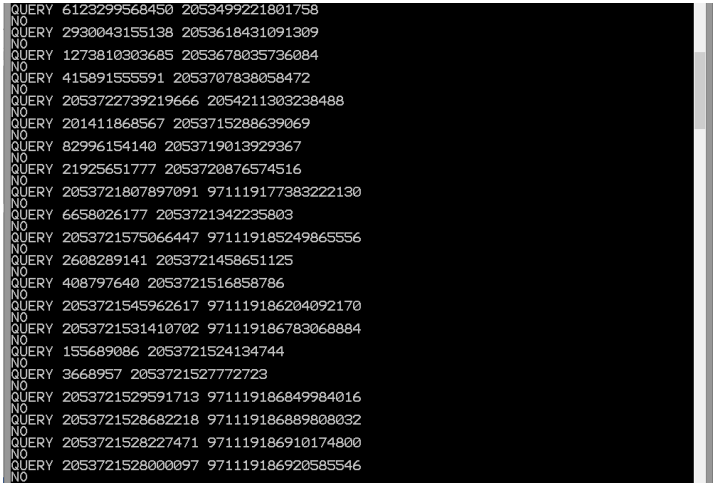
Gambar 5.33 Salah Satu Hasil Uji Coba Program (1000, 14)



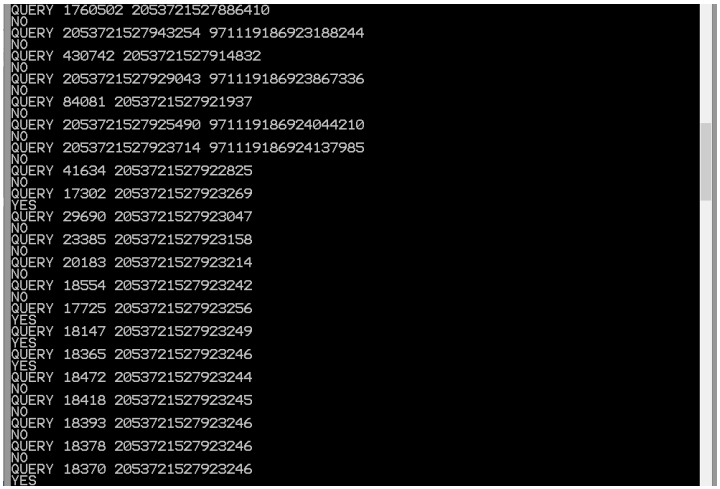
Gambar 5.34 Salah Satu Hasil Uji Coba Program (999, 14)



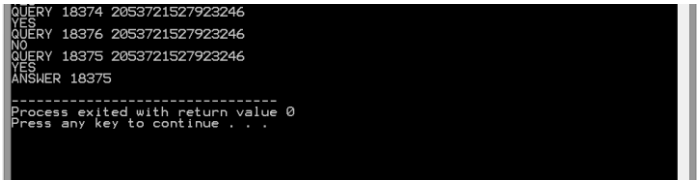
Gambar 5.35 Salah Satu Hasil Uji Coba Program $(10^{18}, 66)$(1)



Gambar 5.36 Salah Satu Hasil Uji Coba Program $(10^{18}, 66)$(2)



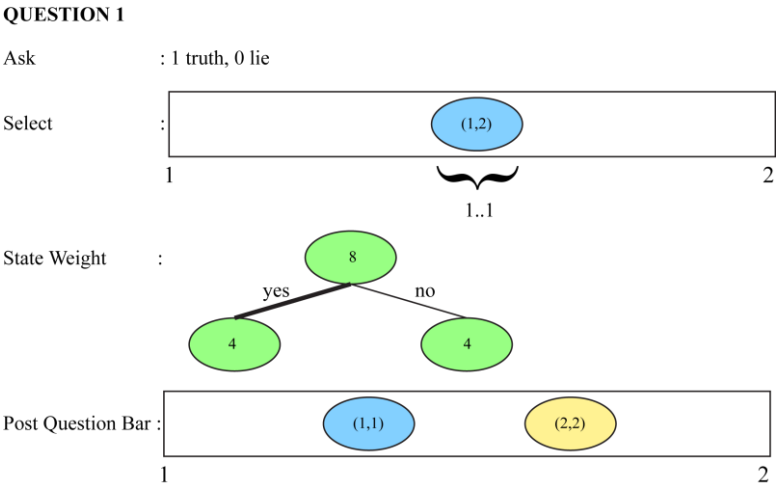
Gambar 5.37 Salah Satu Hasil Uji Coba Program $(10^{18}, 66)$(3)



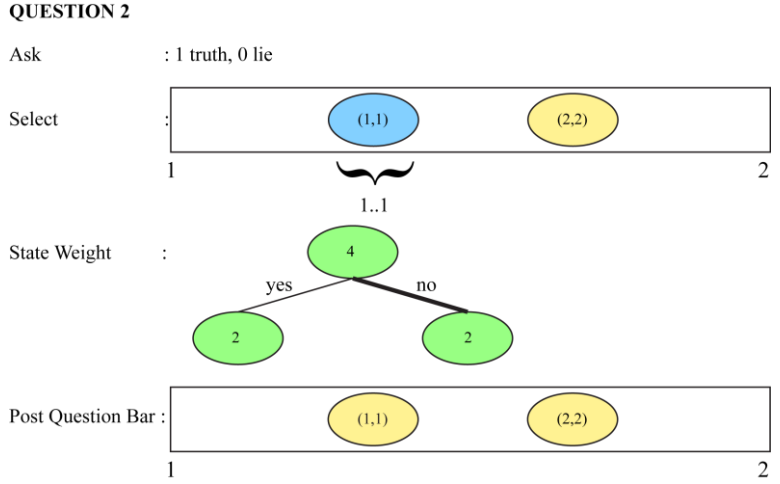
Gambar 5.38 Salah Satu Hasil Uji Coba Program $(10^{18}, 66)$(4)

5.2.2 Permasalahan Guess The Number With Lies v3

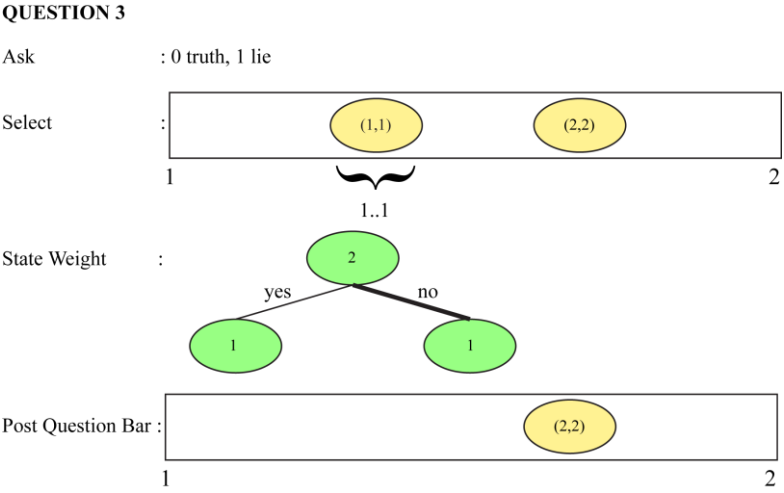
Uji coba kebenaran dilakukan dengan melakukan interaksi dengan *judge* yang terdapat pada daring SPOJ dengan kode permasalahan GUESSN3. Dalam melakukan uji coba, pertama dilakukan terlebih dahulu pengujian terhadap contoh kasus uji yang telah disertakan dalam permasalahan tersebut.



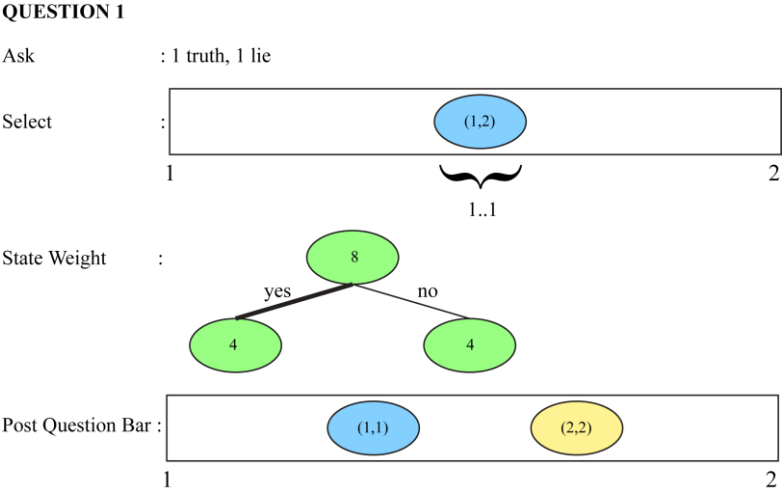
Gambar 5.39 Penyelesaian Permainan pada Kasus Uji 1 SPOJ(1)



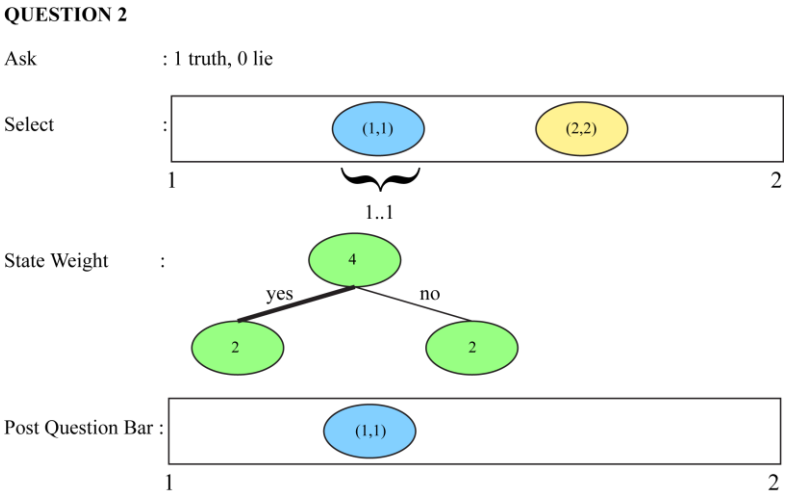
Gambar 5.40 Penyelesaian Permainan pada Kasus Uji 1 SPOJ(2)



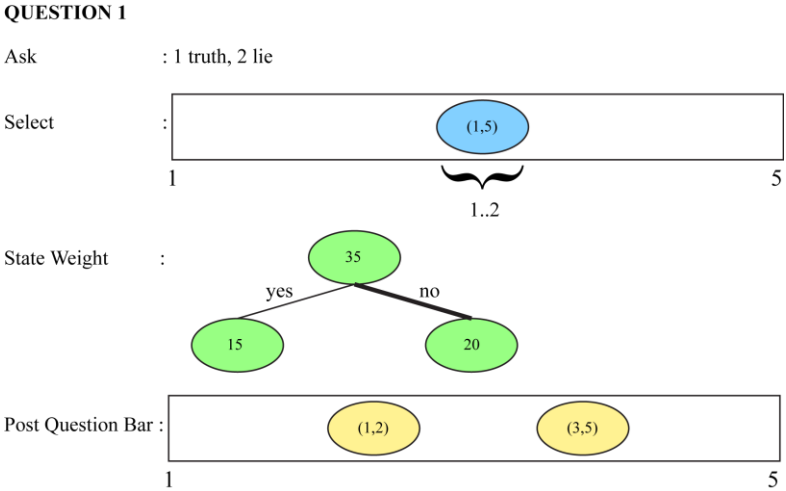
Gambar 5.41 Penyelesaian Permainan pada Kasus Uji 1 SPOJ(3)



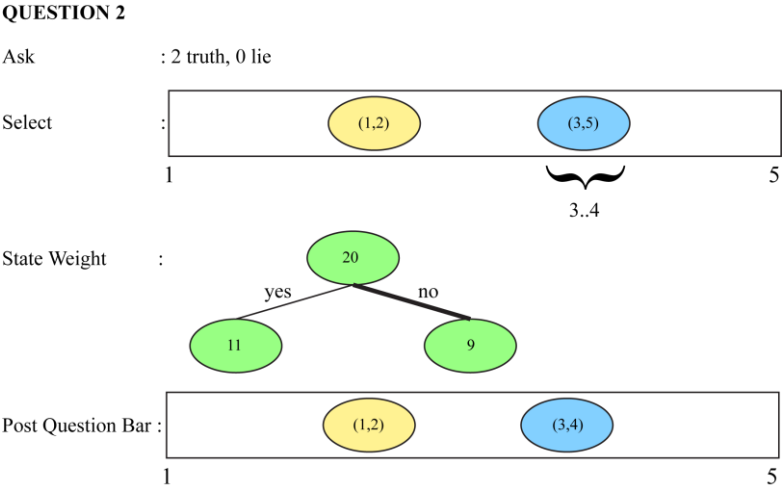
Gambar 5.42 Penyelesaian Permainan pada Kasus Uji 2 SPOJ(1)



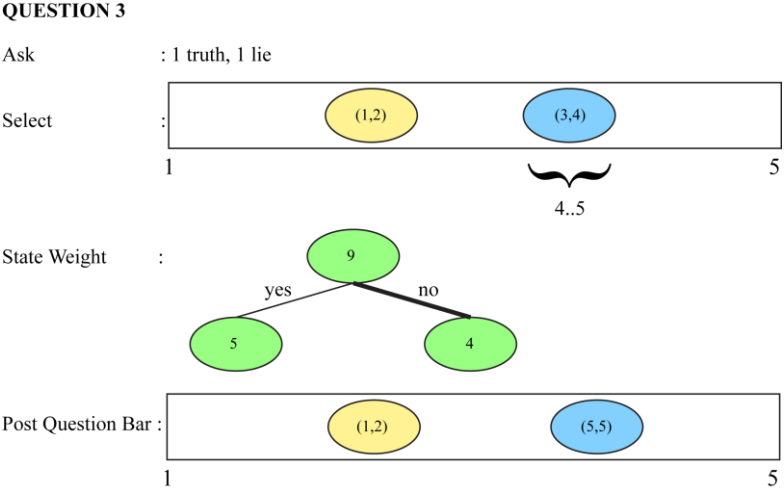
Gambar 5.43 Penyelesaian Permainan pada Kasus Uji 2 SPOJ(2)



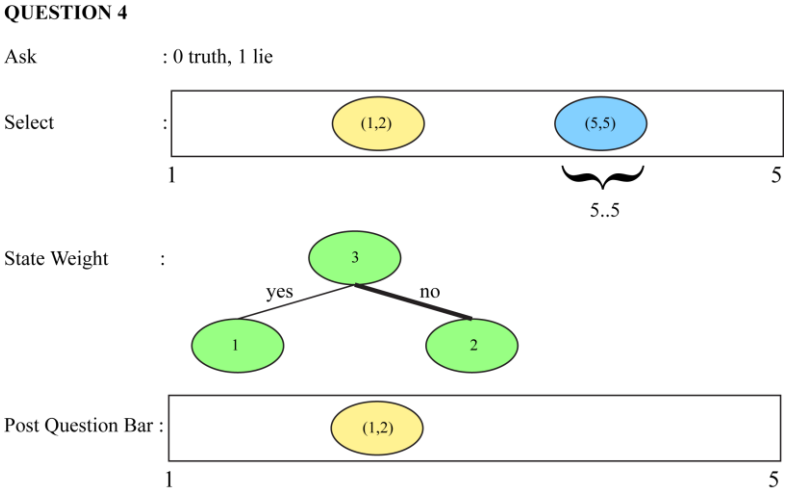
Gambar 5.44 Penyelesaian Permainan pada Kasus Uji 3 SPOJ(1)



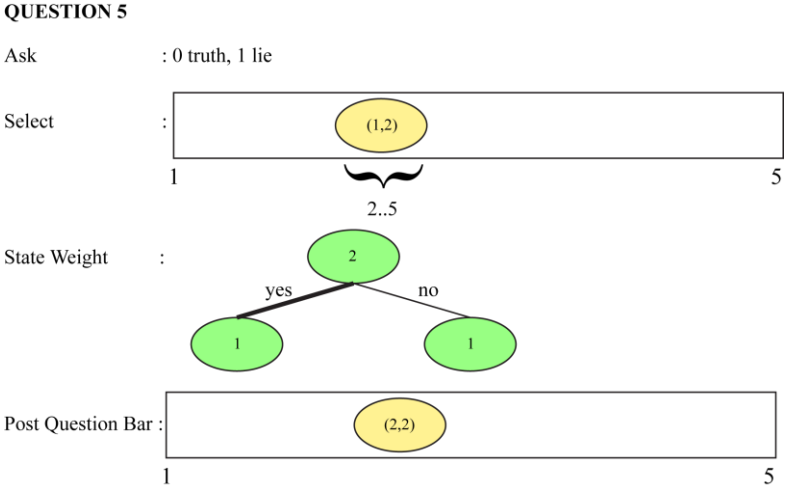
Gambar 5.45 Penyelesaian Permainan pada Kasus Uji 3 SPOJ(2)



Gambar 5.46 Penyelesaian Permainan pada Kasus Uji 3 SPOJ(3)



Gambar 5.47 Penyelesaian Permainan pada Kasus Uji 3 SPOJ(4)



Gambar 5.48 Penyelesaian Permainan pada Kasus Uji 3 SPOJ(5)

Pada kasus uji pertama pada Gambar 5.39 - Gambar 5.41, *judge* memilih angka 2 dan melakukan kebohongan sebanyak satu kali. Dalam kasus uji tersebut, program dapat menyelesaikan permainan dengan 3 pertanyaan.

Pada kasus uji kedua pada Gambar 5.42 - Gambar 5.43, *judge* memilih angka 1 dan tidak melakukan kebohongan. Dalam kasus uji tersebut, program dapat menyelesaikan permainan dengan 2 pertanyaan.

Pada kasus uji ketiga pada Gambar 5.44 - Gambar 5.48, *judge* memilih angka 2 dan melakuan kebohongan. Dalam kasus uji tersebut, program dapat menyelesaikan permainan dengan 5 pertanyaan.

Selanjutnya, dilakukan juga uji coba kebenaran dengan melakukan pengumpulan berkas program ke daring SPOJ. Umpan balik dari pengumpulan berkas ke daring SPOJ dapat dilihat pada Gambar 5.49.

Gambar 5.49 Hasil Uji Kebenaran GUESSN3 pada Situs SPOJ

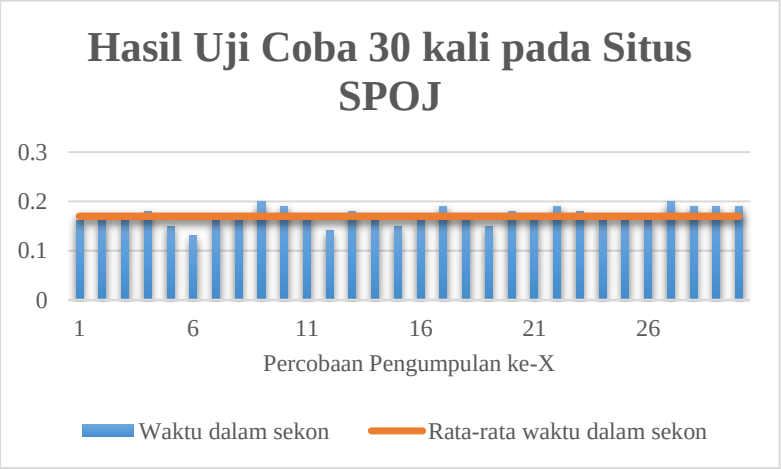
Dari hasil uji coba yang telah dilakukan, kode sumber program mendapat umpan balik *Accepted*. Waktu yang dibutuhkan program adalah 0.19 detik dan memori yang dibutuhkan adalah 2.8 MB. Hasil uji coba diatas membuktikan bahwa implementasi yang dilakukan telah berhasil menyelesaikan permasalahan klasik Ulam dengan batasan-batasan yang telah ditetapkan. Selanjutnya, dilakukan pembandingan kinerja implementasi yang telah dilakukan dibandingkan dengan implementasi lain. Perbandingan performa dapat dilihat pada Gambar 5.50.

Dari perbandingan yang telah dilakukan, implementasi yang dilakukan memiliki *running time* lebih baik dibanding yang lain. Setelah itu dilakukan pengiriman kode sumber sebanyak 30 kali untuk melihat variasi waktu dan memori yang dibutuhkan program.

Grafik hasil uji coba sebanyak 30 kali ditunjukkan pada Gambar 5.51.

RANK	DATE	USER	RESULT	TIME	MEM	LANG
1	2017-05-27 15:20:49	John Stephanus Peter	accepted	0.12	16M	CPP14
2	2017-01-27 21:10:36	Rully Soelaiman	accepted	0.16	3.4M	CPP
3	2014-09-11 06:53:55	Marcin Smulewicz	accepted	0.23	2.8M	C++ 4.3.2

Gambar 5.50 Peringkat Waktu Eksekusi Program *Guess The Number With Lies v3* pada SPOJ



Gambar 5.51 Grafik Hasil Pengumpulan Kode Berkas Sebanyak 30 Kali

Dari hasil pengumpulan kode sebanyak 30 kali, didapat waktu rata-rata program yaitu 0.17 detik dan penggunaan memori rata-rata yang dibutuhkan program yaitu 2.8 MB.

BAB VI
KESIMPULAN

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

6.1 Kesimpulan

Dari hasil uji coba yang telah dilakukan terhadap implementasi solusi untuk permasalahan Ulam dalam mencari sebuah bilangan dari rentang yang telah ditentukan dengan pertanyaan seminimum mungkin, dapat diambil kesimpulan sebagai berikut:

1. Permasalahan mencari bilangan diskrit dari rentang yang telah diberikan dengan jumlah pertanyaan seminimal mungkin telah berhasil diselesaikan dengan teori-teori dan *lemma* yang telah dicantumkan pada subbab 2.2.
2. Penggunaan struktur data *set* dan *pair* untuk menyimpan elemen-elemen pada *truth-set* dan *lie-set* dapat menyelesaikan permasalahan mencari bilangan dengan cukup efisien.
3. Waktu dan penggunaan memori rata-rata yang diperlukan untuk menyelesaikan permasalahan *Guess The Number With Lies v3* dari 30 kali uji coba pada daring SPOJ adalah 0.17 detik dan penggunaan memori sebesar 2.8 MB.
4. Rata-rata waktu yang diperlukan untuk menyelesaikan kasus uji pada daring SPOJ merupakan yang terbaik dibandingkan dengan implementasi lain.

6.2 Saran

Saran yang diberikan untuk implementasi tugas akhir ini adalah sebagai berikut:

1. Penggunaan struktur data array. Pada tugas akhir ini, struktur data yang digunakan adalah struktur data *set* sehingga setiap operasi memerlukan waktu logaritmik.

Struktur data array dapat dipertimbangkan karena elemen dari *set* yang digunakan relatif kecil sehingga operasi terhadap *set* dapat memperlambat proses.

2. Melakukan pemotongan *lie-set* tanpa menyimpan pada struktur data penampung sementara. Apabila pemotongan *lie-set* dilakukan secara langsung tanpa menggunakan struktur data penampung sementara, maka penggunaan memori akan lebih efisien dengan *trade off* pada kompleksitas kode.

DAFTAR PUSTAKA

[1] S. M. Ulam, D. Hirsch, W. G. Matthews, F. Ulam, dan J. Mycielski, Adventures of a Mathematician. Berkeley: University of California Press, 1991.

[2] A. Pelc, "Solution of Ulam's problem on searching with a lie," Journal of Combinatorial Theory, Series A, vol. 44, no. 1, hal. 129–140, Jan 1987.

[3] "std::pair - cppreference.com." [Daring]. Tersedia pada: http://en.cppreference.com/w/cpp/utility/pair. [Diakses: 28-Jun-2017].

[4] "std::set - cppreference.com." [Daring]. Tersedia pada: http://en.cppreference.com/w/cpp/container/set. [Diakses: 05-Jun-2017].

[5] H. B. Mann, Error Correcting Codes: Proceedings of a Symposium. Wiley, New York, 1968.

[6] J. Spencer, "Guess a Number-with Lying," Mathematics Magazine, vol. 57, no. 2, hal. 105–108, 1984.

[7] R. L. Rivest, A. R. Meyer, D. J. Kleitman, K. Winklmann, dan J. Spencer, "Coping with errors in binary search procedures," Journal of Computer and System Sciences, vol. 20, no. 3, hal. 396–404, Jun 1980.

[8] "SPOJ.com - Problem GUESSN3," spoj.com. [Daring]. Tersedia pada: http://www.spoj.com/problems/GUESSN3/. [Diakses: 05-Jun-2017].

[*Halaman ini sengaja dikosongkan*]

LAMPIRAN A
HASIL UJI COBA PADA SITUS SPOJ SEBANYAK 30 KALI

Berikut merupakan lampiran hasil uji coba pengumpulan berkas kode solusi sebanyak 30 kali dari grafik yang ditampilkan pada gambar 5.44.

19545980	2017-06-03 14:45:04	Guess The Number With Lies v3	accepted	0.19	2.8M	C++ 4.3.2
19545979	2017-06-03 14:45:03	Guess The Number With Lies v3	accepted	0.20	2.8M	C++ 4.3.2
19545978	2017-06-03 14:45:01	Guess The Number With Lies v3	accepted	0.17	2.8M	C++ 4.3.2
19545974	2017-06-03 14:44:32	Guess The Number With Lies v3	accepted	0.16	2.8M	C++ 4.3.2
19545973	2017-06-03 14:44:19	Guess The Number With Lies v3	accepted	0.13	2.8M	C++ 4.3.2
19545971	2017-06-03 14:44:11	Guess The Number With Lies v3	accepted	0.15	2.8M	C++ 4.3.2
19545970	2017-06-03 14:44:03	Guess The Number With Lies v3	accepted	0.18	2.8M	C++ 4.3.2
19545967	2017-06-03 14:43:55	Guess The Number With Lies v3	accepted	0.16	2.8M	C++ 4.3.2
19545966	2017-06-03 14:43:42	Guess The Number With Lies v3	accepted	0.16	2.8M	C++ 4.3.2
19545903	2017-06-03 14:26:17	Guess The Number With Lies v3	accepted	0.16	2.8M	C++ 4.3.2

Gambar A.1 Hasil Pengumpulan Kode Sumber Sebanyak 30 Kali(1)

19545999	2017-06-03 14:48:45	Guess The Number With Lies v3	accepted	0.18	2.8M	C++ 4.3.2
19545998	2017-06-03 14:48:41	Guess The Number With Lies v3	accepted	0.15	2.8M	C++ 4.3.2
19545996	2017-06-03 14:48:31	Guess The Number With Lies v3	accepted	0.17	2.8M	C++ 4.3.2
19545995	2017-06-03 14:48:18	Guess The Number With Lies v3	accepted	0.19	2.8M	C++ 4.3.2
19545994	2017-06-03 14:48:17	Guess The Number With Lies v3	accepted	0.16	2.8M	C++ 4.3.2
19545988	2017-06-03 14:46:29	Guess The Number With Lies v3	accepted	0.15	2.8M	C++ 4.3.2
19545987	2017-06-03 14:46:27	Guess The Number With Lies v3	accepted	0.16	2.8M	C++ 4.3.2
19545986	2017-06-03 14:46:11	Guess The Number With Lies v3	accepted	0.18	2.8M	C++ 4.3.2
19545982	2017-06-03 14:45:38	Guess The Number With Lies v3	accepted	0.14	2.8M	C++ 4.3.2
19545981	2017-06-03 14:45:15	Guess The Number With Lies v3	accepted	0.17	2.8M	C++ 4.3.2

Gambar A.2 Hasil Pengumpulan Kode Sumber Sebanyak 30 Kali(2)

19546012	2017-06-03 14:49:42	Guess The Number With Lies v3	**accepted**	0.19	2.8M	C++ 4.3.2
19546011	2017-06-03 14:49:41	Guess The Number With Lies v3	**accepted**	0.19	2.8M	C++ 4.3.2
19546010	2017-06-03 14:49:40	Guess The Number With Lies v3	**accepted**	0.19	2.8M	C++ 4.3.2
19546009	2017-06-03 14:49:39	Guess The Number With Lies v3	**accepted**	0.20	2.8M	C++ 4.3.2
19546008	2017-06-03 14:49:38	Guess The Number With Lies v3	**accepted**	0.16	2.8M	C++ 4.3.2
19546007	2017-06-03 14:49:37	Guess The Number With Lies v3	**accepted**	0.16	2.8M	C++ 4.3.2
19546006	2017-06-03 14:49:35	Guess The Number With Lies v3	**accepted**	0.16	2.8M	C++ 4.3.2
19546004	2017-06-03 14:49:24	Guess The Number With Lies v3	**accepted**	0.18	2.8M	C++ 4.3.2
19546003	2017-06-03 14:49:14	Guess The Number With Lies v3	**accepted**	0.19	2.8M	C++ 4.3.2
19546000	2017-06-03 14:49:00	Guess The Number With Lies v3	**accepted**	0.17	2.8M	C++ 4.3.2

Gambar A.3 Hasil Pengumpulan Kode Sumber Sebanyak 30 Kali(3)

[*Halaman ini sengaja dikosongkan*]

BIODATA PENULIS

John Stephanus Peter, lahir di Mojokerto tanggal 25 Juli 1995. Penulis merupakan anak kedua dari 4 bersaudara. Penulis telah menempuh pendidikan formal TK Taruna Nusa Harapan Mojokerto, SD Taruna Nusa Harapan Mojokerto (2001-2007), SMP Taruna Nusa Harapan Mojokerto (2007-2010) , SMAK St. Louis 1 Surabaya (2010-2013). Penulis melanjutkan studi kuliah program sarjana di Departemen Teknik Informatika ITS.

Selama kuliah di Teknik Informatika ITS, penulis mengambil bidang minat Algoritma Pemrograman (AP). Penulis pernah menjadi asisten dosen dan praktikum untuk mata kuliah Dasar Pemrograman (2014, 2015, dan 2016) dan Struktur Data (2015 dan 2016). Selama menempuh perkuliahan, penulis juga aktif mengikuti kompetisi pemrograman tingkat nasional dan menjadi Juara 1 kategori pemrograman pada lomba Techphoria UNSRI 2016, Juara 2 kategori pemrograman pada lomba Petra Informatics Competition 2016, Juara 2 kategori pemrograman pada lomba Ideafuse 2017, serta menjadi finalis kategori pemrograman pada lomba COMPFEST UI (2015 dan 2016), INC (2014, 2015, dan 2016), dan ACM ICPC Regional Asia-Jakarta (2014 dan 2015. Selain itu, penulis juga pernah menjadi asisten Pelatnas 2 TOKI (2015 dan 2016), serta menjadi instruktur Pelatnas 2 TOKI (2017). Penulis dapat dihubungi melalui surel di `johnstephanus@ymail.com`.