TUGAS AKHIR - IF184802

PENERAPAN ALGORITMA MO UNTUK PEMROSESAN QUERY PADA TREE PADA PENYELESAIAN PERMASALAHAN SPOJ ADAFTBLL - ADA AND FOOTBALL

RIMBA AZHARA
05111640000108

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

Dosen Pembimbing II :
Ir. F.X. Arunanto, M.Sc.

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

TUGAS AKHIR – IF184802

PENERAPAN *ALGORITMA MO* UNTUK PEMROSESAN KUERI PADA *TREE* PADA PENYELESAIAN PERMASALAHAN *SPOJ ADAFTBLL* - ADA AND FOOTBALL

RIMBA AZHARA
05111640000118

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

Dosen Pembimbing II :
Ir. F.X. Arunanto, M.Sc.

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

[*Halaman ini sengaja dikosongkan*]

UNDERGRADUATE THESIS – IF184802

IMPLEMENTATION OF MO'S ALGORITHM FOR QUERY PROCESSING ON TREE TO SOLVE SPOJ ADAFTBLL - ADA AND FOOTBALL

RIMBA AZHARA
05111640000118

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

Supervisor II
Ir. F.X. Arunanto, M.Sc.

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

[*Halaman ini sengaja dikosongkan*]

LEMBAR PENGESAHAN

PENERAPAN *ALGORITMA MO* UNTUK PEMROSESSAN KUERI PADA *TREE* PADA PENYELESAIAN PERMASALAHAN *SPOJ ADAFTBLL* - ADA AND FOOTBALL

TUGAS AKHIR

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

Oleh:
RIMBA AZHARA
NRP: 051116 40000 118

Disetujui oleh Dosen Pembimbing Tugas Akhir:

Rully Soelaiman, S.Kom., M.Kom.
NIP. 197002131994021001

(Pembimbing 1)

Ir. F.X. ARUNANTO, M.Sc.
NIP. 195701011983031004

(Pembimbing 2)

SURABAYA
2020

[*Halaman ini sengaja dikosongkan*]

PENERAPAN *ALGORITMA MO* UNTUK PEMROSESSAN KUERI PADA *TREE* PADA PENYELESAIAN PERMASALAHAN *SPOJ ADAFTBLL* - ADA AND FOOTBALL

Nama Mahasiswa	:	Rimba Azhara
NRP	:	051116 40000 118
Departemen	:	Teknik Informatika FTEIC - ITS
DosenPembimbing 1	:	Rully Soelaiman, S.Kom., M.Kom.
DosenPembimbing 2	:	Ir. F.X. Arunanto, M.Sc.

Abstrak

Permasalahan tugas akhir ini bermula dari adanya permasalahan ADAFTBLL – Ada and Football pada Sphere Online Judge (SPOJ). Permasalahan tersebut menggambarkan sebuah tree yang akan menerima beberapa kueri untuk melakukan penelusuran atau perubahan nilai pada tree. Jumlah vertex dan kueri yang cukup tinggi dan batasan waktu yang singkat membuat permasalahan harus diselesaikan dengan metode yang efisien. Permasalahan ini dapat diselesaikan dengan metode pengerjaan kueri yang tepat. Selain itu untuk memproses data Tree dengan algoritma kueri maka diperlukan perubahan data tree menjadi Array linear. Pendekatan menggunakan Algorimta Mo dipilih sebagai solusi utama permasalahan ini di mana algoritma ini mengurutkan pengerjaan kueri sehingga memperoleh waktu pengerjaan yang baik. Solusi yang telah dibuat cukup efisien dengan rata-rata waktu penyelesaian 9,336 detik.

Kata Kunci: Algoritma Mo, Kueri, Tree

[*Halaman ini sengaja dikosongkan*]

IMPLEMENTATION OF MO'S ALGORITHM FOR QUERY PROCESSING ON TREE TO SOLVE SPOJ ADAFTBLL - ADA AND FOOTBALL

Student Name	: Rimba Azhara
Registration Number	: 051116 40000 118
Department	: Informatics Engineering Department Faculty of Intelligent Electrical and Informatics Technology – ITS
First Supervisor	: Rully Soelaiman, S.Kom., M.Kom.
Second Supervisor	: Ir. F.X. Arunanto, M.Sc.

Abstract

This thesis start with ADAFTBLL – Ada and Football problem in Sphere Online Judge. The problem is about a tree that will receive numbers of queries. The queries are either make a traversal, or just simply modify the data on tree. The high number of vertex and queries in problems, also the limited time and memory constraints making this problem needs to be solved with efficient method. This problem can be solved with right method to handle the queries. Furthermore, to process a tree into queries, it is needed to convert the trees into linear arrays. Approach of Mo's Algorithm is selected to be the main solution as the algorithm method is to reorder queries solving to be the most efficient. The solution provided by this thesis has efficient result, averaging at 9,336 seconds.

Keywords: Mo's Algorithm, Queries, Tree

[*Halaman ini sengaja dikosongkan*]

KATA PENGANTAR

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

Desain dan Analisa Algoritma *Lowest Common Ancestor* dengan Metode Range Minimum kueri pada Permasalahan *SPOJ Ada and Football*

Pengerjaan Tugas Akhir ini dilakukan untuk memenuhi salah satu syarat meraih gelar Sarjana di Departemen Teknik Informatika Fakultas Teknologi Elektro dan Informatika Cerdas 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. Terimakasih kepada Allah SWT, di mana penulis masih diberi kesempatan, kesehatan dan umur untuk menempuh kuliah disini dan menjalani hidup dengan baik.
2. Ayah, Ibu, dan Adik penulis yang selalu memberikan perhatian, dorongan, secara fisik maupun psikis selama menjalani masa perkuliahan.
3. Bapak Rully Soelaiman, S.Kom., M.Kom. selaku Dosen Pembimbing yang telah membimbing saya selama masa kuliah maupun selama penyelesaian Tugas Akhir ini, Dosen yang paling perhatian dan berpengaruh kepada saya dalam memberi ilmu, nasihat, dan motivasi selama berada menempuh kuliah di Departemen teknik informatika ITS.

4. Bapak Ir. F. X. Arunanto M.Sc. Selaku dosen pembimbing yang telah memberikan ilmu, dan masukan kepada penulis.
5. Yoshima dan Zevi yang telah membantu meneliti tata tulis buku ini.
6. Denny R, Jeremia R M, dan M Farras R yang selama ini sudah menghabiskan banyak waktu dan kesibukan bersama penulis di masa perkuliahan.
7. Teman-teman *FoA* yang telah memberikan tempat dan hiburan selama masa perkuliahan.
8. Teman-teman angkatan 2016 jurusan Informatika ITS yang telah menemani perjuangan penulis selama 4 tahun masa perkuliahan.
9. 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, 2020

Rimba Azhara

DAFTAR ISI

[Halaman ini sengaja dikosongkan]

DAFTAR GAMBAR

[Halaman ini sengaja dikosongkan]

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 buku tugas akhir ini.

1.1 Latar Belakang

Tree adalah sebuah struktur data non-linier yang terdiri dari *vertex* dan *edges* yang tersusun secara hierarki di mana nilai *root*, dan *sub-tree* yang terdiri dari *children* dan *parent vertex* direpresantasikan dalam rangkaian *linked lists*.

Topik Tugas Akhir ini mengacu pada permasalahan dalam *tree* pada *Sphere Online Judge* (*SPOJ*) dengan kode ADAFTBALL[1]. Pada permasalahan ini diberikan sebuah *tree* yang *vertex* dan *edge*-nya didapatkan dari masukan *test case*. Setiap *vertex* memiliki nilai. Pada *tree* ini akan dilakukan penelusuran dari satu *vertex* ke satu *vertex* yang lain. Setelah penelusuran selesai, akan didapatkan sebuah nilai yang merupakan penjumlahan dari semua nilai Si di mana $Si = (n-1)+(n-2)+(n-2)+\ldots\ldots+[n-(n-1)]$ di mana n= jumlah *vertex* dengan nilai i yang telah dilalui selama proses penelusuran. Dalam permasalahan ini terdapat kueri untuk mengganti nilai *vertex* tertentu.

Hasil Tugas Akhir ini diharapkan dapat memberi gambaran mengenai algoritma untuk menyelesaikan permasalahan di atas secara optimal dan diharapkan dapat memberikan kontribusi pada perkembangan ilmu pengetahuan dan teknologi informasi.

1.2 Rumusan Permasalahan

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

1. Bagaimana menganalisis serta menentukan desain dan algoritma yang tepat dan optimal untuk

menyelesaikan soal *SPOJ Ada and Football* (ADAFTBALL)?

2. Bagaimana mengimplementasikan algoritma yang tepat untuk menyelesaikan permasalahan *SPOJ Ada and Football*?
3. Bagaimana hasil dari kinerja algoritma yang dibuat dalam menyelesaikan permasalahan *SPOJ Ada and Football*?

1.3 Batasan Permasalahan

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

1. Pembuktian kebenaran didasarkan pada hasil *submission* di sistem penilaian daring *SPOJ*.
2. Implementasi menggunakan bahasa pemrograman C++.

Batasan – batasan dari *SPOJ* adalah sebagai berikut:

1. *N* sebagai jumlah *vertex*s merupakan bilangan bulat dengan rentang 1 sampai dengan 100000
2. *Q* sebagai jumlah kueri merupakan bilangan bulat dengan rentang 1 sampai dengan 100000
3. *A* sebagai tim yang disupport merupakan bilangan bulat dengan rentang 1 sampai dengan 100000
4. Batas waktu perangkat lunak berjalan adalah 3 detik
5. Batas memori perangkat lunak adalah 1536 MB

1.4 Tujuan Pembuatan Tugas Akhir

Tujuan dari Tugas Akhir ini adalah sebagai berikut:

1. Melakukan analisis dan mendesain *algoritma Mo* untuk menemukan solusi dari permasalahan *Ada and Football* secara optimal dari soal *SPOJ ADAFTBLL – Ada and Football*.
2. Melakukan implementasi algoritma yang tepat untuk menyelesaikan permasalahan *SPOJ Ada and Football*.

3. Melakukan uji coba untuk mengevaluasi implementasi algoritma yang dibuat untuk menyelesaikan permasalahan *SPOJ Ada and Football.*

1.5 Manfaat Tugas Akhir

Tugas Akhir ini dapat membantu memahami penggunaan *algoritma Mo* dan penerapannya untuk menemukan solusi dari permasalahan *Ada and Football* secara optimal dari soal *SPOJ ADAFTBLL – Ada and Football*.

1.6 Metodologi

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

1.6.1 Penyusunan Proposal Tugas Akhir

Tahap awal yang dilakukan untuk memulai pengerjaan Tugas Akhir adalah dengan menyusun proposal Tugas Akhir. Dalam proposal, penulis mengajukan gagasan yang akan digunakan untuk menyelesaikan permasalahan *Ada and Football* secara optimal dari soal *SPOJ ADAFTBLL – Ada and Football*..

1.6.2 Studi Literatur

Tahap kedua adalah mencari informasi dan studi literatur yang relevan untuk dijadikan referensi dalam mengerjakan Tugas Akhir. Informasi dan studi literatur didapat dari buku, *scientific paper*, artikel internet, dan materi kuliah yang berhubungan.

1.6.3 Studi Literatur

Pada tahap ini dilakukan desain rancangan algoritma yang digunakan dalam solusi untuk pemecahan masalah *Ada and Football* pada *Sphere Online Judge(SPOJ)*.

1.6.4 Implementasi Algoritma

Tahap implementasi meliputi implementasi algoritma pada perangkat lunak yang telah didukung oleh hasil analisis dan

desain pada tahap sebelumnya. Implementasi akan dilakukan dengan menggunakan bahasa pemrograman C++.

1.6.5 Pengujian dan Evaluasi

Tahap pengujian dan evaluasi dilakukan dengan menggunakan dataset *SPOJ ADAFTBLL – Ada and Football* untuk mengetahui hasil dan performa metode yang telah dibangun. Evaluasi dan perbaikan akan dilakukan hingga perangkat lunak yang diuji menghasilkan hasil performa yang baik.

1.6.6 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 penjelasanlatar 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 Desain dan Analisis 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 menjelaskan kesimpulan dari hasil uji coba yang dilakukan dan saran untuk pengembangan perangkat lunak kedepannya.

[*Halaman ini sengaja dikosongkan*]

BAB II
DASAR TEORI

Pada bab ini akan dijelaskan mengenai dasar-dasar teori yang akan digunakan guna menyelesaikan permasalahan ini. Dasar teori yang digunakan meliputi *deep-first search*, *local common ancestor*, *algoritma Mo* serta penjabaran penggunaan teori dan landasan pemilihan struktur data yang akan digunakan sebagai solusi pada permasalahan tersebut.

2.1 Deskripsi Umum Permasalahan *SPOJ ADAFTBLL*

ADAFTBLL- Ada and Football, merupakan suatu permasalahan yang terdapat pada situs penilaian *Sphere Online Judge* (*SPOJ*) dengan deskripsi soal dari sumber asli menggunakan bahasa Polandia yang diterjemahkan kedalam bahasa Inggris, lihat Gambar 2.1.

ADAFTBLL - Ada and Football

#sorting #tree #algorithm #sweep

Ada the Ladybug has many friends who live on a tree. Each of them is fan of a football team. Ada sometime go on a trip from one friend to another. If she meets a friend she tells him/her about all previously visited friends who are fans of the same team. The visited friend will gain **+1** happiness for each friend Ada told him/her about.

Ada is wondering (for each trip) what will be the total gaines happiness.

Also note that sometime a friend changes his/her mind and start to support different team.

Input

The first line two integers $1 \le N$ $Q \le 10^5$, the number of friends (**N**) and the number of queries.

The next line will contain **N** integers $0 \le A_i \le 10^5$, the team which i^{th} friend supports.

The next **N-1** lines will contain two integers $0 \le a, b < N$, the friends which will be connected by a branch (edge).

The next **Q** lines will be of two kinds:

1 x y ($0 \le x < N$, $0 \le y \le 10^5$), meaning that x^{th} friend will start supporting team **y** (instead of the old one).

2 a b ($0 \le a, b < N$), meaning that Ada will travel from friend **a** to friend **b** and wants to know the happiness.

Output

For each query of second kind, output the gained happiness.

Gambar 2.1 Deskripsi Soal *SPOJ ADAFTBLL – Ada and Football*

Permasalahan *SPOJ ADAFTBLL – Ada and Football* menceritakan tentang seekor kumbang bernama Ada yang memiliki teman-teman yang hidup di *tree*. Setiap teman ada merupakan penggemar dari suatu tim sepak bola. Dalam beberapa kesempatan Ada akan melakukan perjalanan mengunjungi satu teman ke teman yang lain. Ketika Ada mengunjungi seorang teman, Ada akan memberitahu teman tersebut berapa teman pendukung tim yang sama yang telah dikunjungi sebelumnya. Untuk setiap teman yang diceritakan Ada, teman tersebut akan mendapat +1 *happiness point*. Ada ingin mengetahui berapa banyak *happiness point* yang didapat dalam satu kali perjalanan. Terkadang, teman Ada dapat mengganti tim mana yang mereka dukung.

Masukan yang diberikan adalah *N* merepresentasikan jumlah teman ada dan *Q* merepresentasikan kueri yang akan dikerjakan. Kemudian diikuti bilangan bulat *Ai* sejumlah *N* yang merepresentasikan tim yang didukung teman-teman ada, secara berurutan. Setelah itu diikuti oleh pasangan bilangan bulat *a b* sebanyak *N-1* yang merepresentasikan bahwa rumah teman *a* dan *b* terhubung oleh ranting. Kemudian dilanjutkan oleh masukan sejumlah *Q* berupa kueri yang akan dikerjakan, dalam permasalahan ini terdapat dua jenis kueri, yaitu:

1. 1 *x y* yang berarti teman Ada yang bernomor *x* akan berganti mendukung tim *y*
2. 2 *a b* yang berarti Ada akan melakukan perjalanan dimulai dari teman *a* menuju teman *b*.

Keluaran dari permasalahan ini adalah untuk setiap kueri jenis kedua akan mengeluarkan jumlah *happiness point* yang telah didapat. Pada permasalahan ini terdapat batasan-batasan sebagai berikut:

1. *N*, *Q*, dan *Ai* merupakan bilangan bulat dengan rentang dari 1 sampai dengan 100000.

2. *x*, *a*, dan *b* merupakan bilangan bulat yang mewakilkan teman ada, sehingga rentangnya adalah 1 sampai dengan nilai *N*.

2.2 Penjelasan Permasalahan *SPOJ ADAFTBLL*

Permasalahan *SPOJ ADAFTBLL* meminta kita untuk memberikan nilai *happiness point* yang didapat untuk setiap kueri yang diawali dengan angka 2 atau dengan kata lain kueri di mana Ada mengunjungi temannya. *Happiness Point* akan dihitung ketika Ada menceritakan kepada temannya berapa teman yang mendukung tim yang sama yang sudah dikunjungi oleh Ada.

Misalkan pada perjalanan Ada, teman yang dikunjungi Ada adalah teman 1, 2, 3, 4, dan 5, yang secara berurutan mendukung tim nomor 1, 2, 2, 1, 2 maka cara menghitung *Happiness Point* adalah:

- Pada saat di teman 1 mendapat 0 poin, karena baru pertama kali mengunjungi pendukung tim 1. Kumulatif poin = 0.
- Pada saat di teman 2 mendapat 0 poin, karena baru pertama kali mengunjungi pendukung tim 1. Kumulatif poin=0.
- Pada saat di teman 3 mendapat 1 poin karena sudah mengunjungi 1 orang teman pendukung tim 2 sebelumnya, yaitu teman 2. Kumulatif poin: 1.
- Pada saat di teman 4 mendapat 1 poin karena sudah mengunjungi 1 orang pendukug tim 1 sebelumnya, yaitu teman 1. Kumulatif Poin: 2.
- Pada saat di teman 5 mendapat 2 poin, karena sudah mengunjungi 2 orang pendukung tim 2 sebelumnya, yaitu teman 2 dan 3. Kumulatif Poin: 4.

Sehingga untuk penelusuran tersebut keluaran yang diberikan adalah 4 yang merupakan kumulatif poin selama perjalanan Ada.

2.3 Deskripsi Umum Teori

Pada subbab ini, akan dijelaskan berbagai landasan teori secara umum yang digunakan untuk melakukan pendekatan terhadap penyelesaian permasalahan

2.3.1 *Tree*

Dalam teori graf, sebuah *tree[2]* adalah sebuah graf tidak terarah dimana dua *vertex* dihubungkan dengan tepat satu *edge*, atau graf asiklik tidak terarah.

Sebuah *tree* adalah graf G yang memenuhi salah satu dari kondisi berikut:

- G terhubung dan asiklik.
- G asiklik dan sebuah sirkuit akan terbentuk bila ditambahkan *edge* apapun.
- G terhubung, tapi menjadi tidak terhubung bila *edge* manapun dihapus.
- Dua *vertex* manapun pada G dapat dihubungkan dengan jalan sederhana yang unik.

2.3.2 *Deep First Search*

Deep First Search[3] adalah algoritma penelusuran pada *tree*. Algoritma ini dimulai dari salah satu *vertex* sebagai *root*. Penelusuran dilakukan sejauh/ sedalam mungkin, setelah itu akan dilakukan *backtracking*.

2.3.3 *Lowest Common Ancestor*

Lowest Common Ancestor(LCA)[4] dari dua buah *vertex v* dan *w* adalah dalam sebuah *tree* adalah *vertex* paling rendah (paling dalam) yang memiliki *v* dan *w* sebagai *descendant*-nya. *LCA* dari v dan w adalah *ancestor* yang sama dari *v* dan *w* yang paling jauh dari *vertex root*.

2.3.4 *Mo's* Algorithm

Algoritma *Mo*[5] adalah algoritma untuk memproses permasalahan struktur data. Ide utama dari algoritma ini adalah melakukan preproses semua kueri, sehingga hasil dari satu kueri dapat digunakan untuk kueri setelahnya.

[*Halaman ini sengaja dikosongkan*]

BAB III
DESAIN DAN ANALISIS

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

3.1 Analisis Permasalahan dan Strategi Penyelesaian

Inti dari permasalahan ini adalah bagaimana memproses seluruh kueri dan bagaimna melakukan penelusuran pada *tree* sehingga membentuk *path* yang benar serta menghitung *happiness point* yang didapat. Permasalahan tersebut dapat diselesaikan dengan menggunakan dasar algoritma *Mo*. Algoritma *Mo* merupakan algoritma yang digunakan untuk memproses kueri dengan mengurutkan seluruh kueri terlebih dahulu agar supaya pengerjaan kueri menjadi efisien secara komputasi. Permasalahan lain yang muncul dari penggunaan *algoritma MO* adalah *algoritma MO* digunakan untuk memproses sebuah *Array* linier, sedangkan untuk permasalahan ini yang akan diproses adalah sebuah *Tree* sehingga harus dilakukan perubahan atau penyesuaian data.

3.2 Deskripsi Umum Sistem

Sistem yang dirancang berupa sebuah aplikasi berbasis konsol, menggunakan masukan standar sebagai masukan sistem. Sistem akan menerima masukan jumlah *vertex* dan jumlah kueri pada baris pertama, lalu diikuti dengan masukan berupa tim pada tiap *vertex* secara berurutan, masukan ini akan dijadikan sebuah *array* untuk membangun data kueri. Setalah itu masukan diikuti oleh dua buah *vertex* untuk membangun *edge* di Antara kedua *vertex* tadi. Sistem akan membangun representasi *tree* pada langkah ini, diikuti oleh fungsi *deep first search* untuk mengubah *tree* menjadi beberapa *array* linier untuk selanjutnya digunakan oleh fungsi yang lain. Setelah itu sistem akan menerima masukan berupa kueri yang teridri dari dua jenis yang masing-masing oleh sistem akan dibangun struktur datanya serta dilakukan pengurutan pemrosesan kueri. Untuk setiap kueri tipe awalan 2 yang diproses,

sistem akan memberikan keluaran berupa bilangan bulat yang merepresentasikan *happiness point* yang didapat. *Pseudocode* fungsi ini dapat dilihat pada Gambar 3.1

```
Main()
N← masukan
Q← masukan
For i← 1 to n
   Team[i]←masukan
For i← 0 to n-1
   A←masukan
   B←masukan
   Adj[a].push_back(b)
   Adj[b].push_back(a)
Dfs(1,0)
Generate_lca()
For i←1 to q
   Aa←masukan
   Bb←masukan
   Cc←masukan
   If(a==2)
     Build_travel_query(b,c)
   Else
     Build_update_query(b,c)
Sort(travel_query)
For i←1 to travel_query.size()
   Execute_Mo()
   Execute_update()
For i←1 to travel_query.size()
   Print(answer[i]).
```

Gambar 3.1 *Pseudocode* Fungsi *Main*

3.3 Desain Algoritma

Pada bagian ini akan dijelaskan beberapa desain struktur data dan algoritma yang akan digunakan untuk menyelesaikan permasalahan.

3.3.1 Desain Fungsi *deep first search/dfs*

Fungsi *deep first search* adalah fungsi yang digunakan untuk menelusuri *tree*. Fungsi *dfs* pada permasalahan ini akan dimodifikasi untuk mengubah *tree* menjadi sebuah *array* linear

yang dapat diproses menggunakan algoritma *Mo*. Pada fungsi *dfs* ini akan dilakukan penyimpanan berbagai *array* berdasar penelusuran, yaitu *array Mo* yang terdiri dari kemunculan pertama dan terakhir suatu *vertex*, *array depth* untuk setiap *vertex*, serta inisiasi untuk *LCA*. *Pseudocode* fungsi ini dapat dilihat pada Gambar 3.2.

```
dfs(now,parent)
1.  Awal[now] = ++idx //awal kemunculan vertex
2.  Mo[idx] = make_pair(team[now],now)
3.  For i←0 to adj[now].size()-1
4.     Current = adj[now][i]
5.     If(current!=parent)
6.       Lca[current][0]=now
7.       Depth[current]=depth[now]+1
8.       Dfs(current,now)
9.  Akhir[now]=++idx
10. Mo[idx]=make_pair(team[now],now)
```

Gambar 3.2 ***Pseudocode*** **Fungsi** ***deep first search***

3.3.2 Desain Fungsi *generate_LCA*

Fungsi generate *LCA* adalah fungsi yang digunakan untuk membuat *Lookup table LCA*. Memanfaatkan inisasi *LCA* yang didapatkan dari fungsi *dfs* maka akan dilakukan *assignment LCA* sesuai depth *vertex* tersebut. *Pseudocode* fungsi ini dapat dilihat pada Gambar 3.3

```
Generate_LCA()
1.  For i←1 to 17
2.     For j←1 to n
3.       Lca[j][i] = lca[lca[j][i-1]][i-1]
```

Gambar 3.3 *Pseudocode* Fungsi Generate *LCA*

3.3.3 Desain Fungsi *find_LCA*

Fungsi ini digunakan untuk mencari *LCA* dari kedua *vertex*. Untuk mencari *LCA* dari dua *vertex* akan dilakukan iterasi penelusuran *depth* terlebih dahulu. Apabila pada *depth* yang sama *LCA* dari keduanya tidak sama, maka variabel penelusuran akan dipindahkan menjadi *LCA* tersebut. Pengulangan akan dilakukan sampai kedua *LCA* adalah sama, sehingga menjadi *LCA* dari kedua *vertex*. *Pseudocode* fungsi ini dapat dilihat pada Gambar 3.4

find_lca(u,v)

```
1. For i← 17 to 1
2.     If(lca[u][i]!=lca[v][i])
3.        U= lca[u][i]
4.        V= lca[v][i]
5. Return lca[u][0]
```

Gambar 3.4 *Pseudocode* Fungsi *find_lca*

3.3.4 Desain Fungsi *build_travel_query*

Setelah data dari *dfs* dan *LCA* selesai dibuat maka akan dilakukan pembangunan struktur data *travel_query* melalui fungsi ini. *Vertex* masukan kueri akan dicari *LCA*-nya menggunakan fungsi find *LCA*. Berdasarkan hasil *LCA* yang didapat, maka data yg dimasukan pada *travel_query* akan berbeda.

- *LCA* kedua *vertex* sama dengan *vertex* dengan *depth* lebih sedikit.

  Pada kasus ini maka path kedua *vertex* didapatkan dari kemunculan pertama kedua *vertex* pada penelusuran *dfs*, dinotasikan dalam *start[u]* dan *start[v]*. Sehingga data yang dimasukan adalah:
  Start Block: didapat dari *start[u]*/konstanta *block*
  End Block: didapat dari *start[v]*/konstanta *block*
  Update: jumlah update kueri yang sudah dilakukan

Query: Indeks kueri pada masukan sistem
Start Range: start[u]
End Range: start[v]
Lca: -1

- *LCA* kedua *vertex* berbeda dengan *vertex* dengan *depth* lebih sedikit.

  Pada kasus ini *path* antara kedua *vertex* didapatkan dari kemunculan terakhir *vertex* u dinotasikan *end[u]* sampai kemunculan pertama *vertex* v dinotasikan dengan *start[v]* + *LCA* dari *u* dan *v*. Sehingga data yang dimasukan adalah:
  Start Block: didapat dari *end[u]/*konstanta *block*
  End Block: didapat dari *start[v]/*konstanta *block*
  Update: jumlah update kueri yang sudah dilakukan
  Query: Indeks kueri pada masukan sistem
  Start Range: end[u]
  End Range: start[v]
  Lca: *LCA*(u,v)

Setelah stuktur data *travel_query* dibuat untuk semua kueri masukan, maka akan dilakukan *sorting*. *Pseudocode* fungsi ini dapat dilihat pada Gambar 3.5

Build_travel_query(b,c)

```
1.  If(depth[b]<depth[c]) swap(b,c)
2.  Depth_diff= depth[b]-depth[c]
3.  Current=b
4.  For i←1 to 17
5.     If(isodd(depth_diff))
6.       Current=lca[current][i]
7.     Depth_diff= depth_diff/2
8.  If(current==c)
9.     If(awal[b]>awal[c])swap(b,c)
10.    Travel_query.push_back(
   start_block,end_block,update,query,awal[b],awal[c],-1)
11. Else
12.    Current2 = c
13.    True_lca = find_lca(current,current2)
14.    Travel_query.push_back(
   start_block,end_block,update,query,end[b],awal[c],true_
   lca)
```

Gambar 3.5 *Pseudocode* Fungsi *build_travel_query*

3.3.5 Desain Fungsi *build_update_query*

Struktur data kedua yang dibutuhkan untuk menyelesaikan permasalahan ini adalah *update_query* yaitu perubahan pada nilai tim tiap *vertex*. Pada saat *sistem* menerima masukan untuk value tim, nilai tersebut disimpan dalam *array* tim. Data yang dimasukan sebagai *update_query* adalah:

Vertex: *Vertex* yang mengalami *update*, didapat dari kueri masukan

Nilai baru: Nilai tim setelah terjadi *update*, didapat dari kueri

Nilai sebelumnya: Nilai tim sebelum terjadi *update*, didapatkan dari *array* tim.

Setelah *update_query* terbentuk maka nilai tim *vertex* pada *array* tim akan diupdate dengan nilai yang baru agar dapat digunakan pada update kueri selanjutnya. *Pseudocode* fungsi ini dapat dilihat pada Gambar 3.6

Build_update_query(*vertex*,new_value)
1. *Update_query*.push_back(*vertex*,new_value,team[*vertex*]) 2. Team[*vertex*]=new_value

Gambar 3.6 Fungsi *build_update_query*

3.3.6 Desain Algoritma Modifikasi Mo

Untuk mendapatkan *path* penelusuran serta menghitung *happiness point* maka akan menggunakan *algoritma mo* yang dimodifikasi. *Algoritma Mo* menggunakan dua buah nilai pointer, yaitu pointer kiri dan pointer kanan. Untuk setiap kueri pointer kiri akan digeser sampai *start* kueri dan kueri kanan akan digeser sampai *end* kueri. Variabel *happiness_point* digunakan untuk menyimpan nilai *happiness point* yang didapat sealama proses eksekusi *Mo*. Dalam setiap pergeseran pointer maka yang akan dilakukan adalah:

1. Jika *vertex* yang ditunjuk pointer dalam kondisi *visited* akan diubah statusnya menjadi tidak *visited*, lalu mengurangi jumlah kemunculan tim dari *vertex* tersebut. Setelah itu mengurangi nilai *happiness_point* dengan nilai kemunculan tim tersebut.
2. Jika *vertex* yang ditunjuk pointer tidak dalam kondisi *visited* maka diubah menjadi *visited*. Lalu menambahkan nilai *happiness_point* dengan kemunculan nilai tim pada *vertex* yang ditunjuk pointer. Setelah itu menambah nilai kemunculan tim tersebut

Setelah proses *Mo* selesai akan dilihat pada data *travel_query* berapa nilai *LCA*, jika nilai *LCA* adalah -1 maka nilai *happiness point* akan digunakan sebagai jawaban. Sedangkan jika nilai *LCA* adalah sebuah *vertex*, maka nilai *happiness*

point akan ditambahkan dengan kemunculan tim *vertex* tersebut, baru digunakan sebagai jawaban. *Pseudocode* fungsi ini dapat dilihat pada Gambar 3.7 dan Gambar 3.8.

```
Execute_Mo(start_query,end_query)
1.  While(end_query<right_ptr)
2.     MO(right_ptr)
3.     Right_ptr - -
4.  While(end_query > right_ptr)
5.     Right_ptr++
6.     Mo(right_ptr)
7.  While(start_query<left_ptr)
8.     Left_ptr - -
9.     Mo(left_ptr)
10. While(start_query>left_ptr)
11.    Mo(left_ptr)
12.    Left_ptr++
```

Gambar 3.7 *Pseudocode* Fungsi Execute Mo

```
Mo(now)
1.  If(visited[Mo[now].vertex]=true
2.     Visited[Mo[now].vertex]=false
3.     Team_occurence[Mo[now].team] - -
4.     happiness_point -= team_occurence[Mo[now].team]
5.  If(visited[Mo[now].vertex]=false
6.     Visited[Mo[now].vertex]=true
7.     happiness_point += team_occurence[Mo[now].team]
8.     Team_occurence[Mo[now].team]++
```

Gambar 3.8 *Pseudocode* Fungsi Mo

3.3.7 Desain Fungsi *execute_update*

Setelah kueri *travel* selesai diproses maka kueri *update* akan diproses. Selama pointer *updateNow* tidak sama dengan data *update* pada *travel_query* maka akan dilakukan penyamaan. Apabila *updateNow* mendahului data *update* maka akan

dilakukan *decrement* di setiap awal proses sedangkan jika belum mendahului, akan dilakukan *increment* di setiap akhir proses. Proses yang dilakukan adalah mengambil data pada *update_query* sesuai *index updateNow*, data yang diambil berupa *vertex* dan nilai tim, baik yang terupdate ataupun yang lama. Setelah itu dilakukan pengecekan apakah *vertex* tersebut tepat muncul 1 kali di antara *path* yang sudah diproses, jika tidak maka *happiness point* tidak akan diproses karena kemunculan nol atau dua kali berarti *vertex* tersebut tidak di dalam *path* yang diminta. Apabila *vertex* tersebut di dalam *path* maka nilai kemunculan tim yang didukung *vertex* sebelum *update* akan dikurangi lalu *happiness point* juga akan dikurangi. Setelah itu *happiness point* akan ditambahkan dengan kemunculan nilai yang baru setelah *update*. Setelah itu mengubah nilai tim yang didukung *vertex* tersebut pada *Array Mo* sesuai nilai yang baru. *Pseudocode* fungsi ini dapat dilihat pada Gambar 3.9

```
Execute_update
1.  While(updateNow<update)
2.     Update_vertex= update_query[updateNow].vertex
3.     Value= update_query[updateNow].new_value
4.     Muncul=0
5.     Inpath=0
6.     if(awal[update_vertex]>=left_ptr             &&
    awal[update_vertex<= right_ptr) muncul++, inpath=
    awal[update_vertex]
7.
    if(akhir[update_vertex>=left_ptr&&akhir[update_ve
    rtex<=right_ptr)
    muncul++,inpath=akhir[update_vertex]
8.  If(muncul==1)
9.     Team_occurence[Mo[inpath].team] - -
10.    Happiness_point                              -=
    Team_occurence[Mo[inpath].team]
11.    Happiness_point += team_occurence[value]
12.    Mo[awal[update_vertex].team = value
13.    Mo[akhir[update_vertex].tem = value
14.    updateNow++
15. While(updateNow>update)
16.    updateNow--
17.    Update_vertex = update_query[updateNow].vertex
18.    Value= update_query[updateNow].old_value
19.    Muncul=0
20.    Inpath=0
21.    if(awal[update_vertex]>=left_ptr             &&
    awal[update_vertex<= right_ptr) muncul++, inpath=
    awal[update_vertex]
22.
    if(akhir[update_vertex>=left_ptr&&akhir[update_ve
    rtex<=right_ptr)
    muncul++,inpath=akhir[update_vertex]
23. If(muncul==1)
24.    Team_occurence[Mo[inpath].team] - -
25.    Happiness_point                              -=
    Team_occurence[Mo[inpath].team]
26.    Happiness_point += team_occurence[value]
27.    Mo[awal[update_vertex].team = value
28.    Mo[akhir[update_vertex].tem = value
```

Gambar 3.9 *Pseudocode* Fungsi *execute_update*

BAB IV
IMPLEMENTASI

Pada bab ini akan dijelaskan tentang implementasi yang dilakukan berdasarkan algoritma yang telah dirancang pada bab 3.

4.1 Lingkungan Implementasi

Lingkungan implementasi dan pengembangan yang dilakukan adalah sebagai berikut.

1. Perangkat Keras

   (a) ProcessorIntel(R) Core(TM) i5-8250U CPU @1.60GHz 1.80GHz

   (b) Random Access Memory: 8GB

2. Perangkat Lunak

   (a) Sistem Operasi: Windows 10 Home

   (b) Text Editor: DevC++

   (c) Bahasa Pemrograman: C++

4.2 Rancangan Data

Pada subbab ini dijelaskan mengenai desain data masukan yang diperlukan untuk melakukan proses algoritma serta data keluaran yang dihasilkan oleh program.

4.2.1 Data Masukan

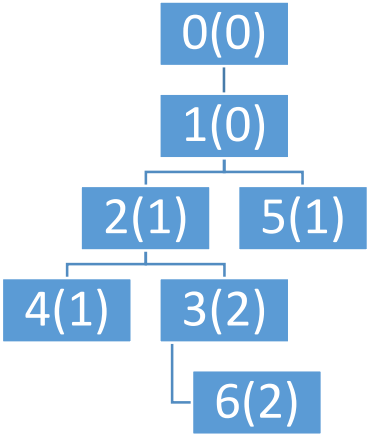
Data masukan adalah data yang akan diproses oleh program sebagai masukan dengan menggunakan algoritma dan struktur data yang telah dirancang dalam penyelesaian permasalahan *Ada and Football* ini. Data masukan untuk permasalahan ini adalah sebagai berikut:

1. Masukan berupa *n* dan *q* dengan rentang 1<= n q <=10^5. Dengan *n* merepresantsikan jumlah *vertex* dengan penomoran *vertex* dari 0 sampai dengan *n*-1. Sedangkan *q* merepresentasikan jumlah kueri yang akan diproses.
2. Baris kedua diikuti sejumlah *n* bilangan bulat setiap bilangan merepresentasikan nilai tim yang didukung setiap *vertex* secara berurutan.
3. Pada *n-1* baris berikutnya sepasang bilangan bulat *a* dan *b* yang menunjukkan adanya *edge* Antara *vertex* a dan b.
4. Diikuti dengan sejumlah *q* baris masukan berupa 3 bilangan bulat 1 *x y* atau 2 *x y* yang merepresentasikan kueri untuk diproses, dengan awalan 1 adalah *update* kueri dan awalan 2 adalah *travel* kueri.

Contoh data masukan dapat dilihat pada Gambar 4.1 dan representasi *tree* nya dapat dilihat pada Gambar 4.2.

```
7 8
0 0 1 2 1 1 2
0 1
1 5
1 2
2 4
2 3
3 6
2 4 5
2 0 6
2 1 3
1 1 2
1 2 2
2 4 5
2 0 6
2 1 3
```

Gambar 4.1 Contoh Masukan dari *SPOJ*



Gambar 4.2 Representasi *Tree* dari data masukan

4.2.2 Data Keluaran

Keluaran program adalah beberapa baris bilangan bulat untuk setiap *travel* kueri yang merupakan berapa *happiness point* yang didapat pada *travel* kueri tersebut. Contoh data keluaran dapat dilihat pada Gambar 4.3

```
3
2
0
2
6
3
```

Gambar 4.3 Contoh Keluaran Program

4.3 Penggunaan Library, Konstanta, dan Variabel Global

Pada subbab ini dijelaskan mengenai *library*, *template*, konstanta, dan variabel global yang digunakan dalam program. Pada Kode Sumber 4.1, terdapat lima *library* yang digunakan yaitu *vector*, *utility*, *iostream*, *cstring*, dan *algorithm*. Didefinisikan *f* sebagai *first*, *s* sebagai *second*, *mp* sebagai *make_pair*, dan *pb* sebagai *push_back* dengan tujuan meringkas dan memudahkan pembuatan kode. Selanjutnya didefinisikan konstanta *MAXN* yang bernilai 10^5. Tabel 4.1 menampilkan variabel-variabel global yang akan digunakan dalam implementasi program. Kode sumber untuk bagian ini dapat dilihat pada Kode Sumber 4.1

```
1.  #include <vector>
2.  #include <utility>
3.  #include <iostream>
4.  #include <cstring>
5.  #include <algorithm>
6.  #define f first
7.  #define s second
8.  #define mp make_pair
9.  #define pb push_back
10. using namespace std;
11. const int MAXN = 100005;
12.
13. int n;
14. int q;
15. int awal[MAXN];
16. int akhir[MAXN];
17. int happiness_point;
18. vector<ll> adj[MAXN];
19. int idx;
20. int block;
21. int update;
22. int updateNow;
23. int query;
24. int left_ptr;
25. int right_ptr;
26. int answer[MAXN];
27. int team[MAXN];
28. int team_occurence[MAXN];
```

```
int lca[MAXN][25];
int depth[MAXN];
bool visited[MAXN];
pair<int,int> Mo[200005];
vector<pair<pair<int,int>,pair<pair<int,int>
   ,pair<int,pair<int,int> > > > >travel_query;
vector<pair<int,pair<int,int> > >update_query;
```

Kode Sumber 4.1 Potongan Kode penggunaan library, konstanta, dan variabel global

Tabel 4.1 Penjelasan variabel global

No	Nama Variabel	Tipe	keterangan
1	*N*	Integer	Jumlah *Vertex*
2	*Q*	Integer	Jumlah Kueri
3	Awal	Integer[]	*Array* untuk menyimpan kemunculan pertama *vertex* pada penelusuran
4	Akhir	Integer[]	*Array* untuk menyimpan kemunculan terakhir *vertex* pada penelusuran
5	*happiness_point*	Integer	Nilai *happiness point* yang didapat saat menjalankan *travel* kueri
6	*Adj*	Vector<integer>[]	*Adjacency matrix* representasi untuk *tree*
7	*idx*	Integer	Nilai pada urutan berapa *vertex* diproses pada *deep first search*
8	*Block*	Integer	Jumlah *block* pembagian kueri untuk algoritma *Mo*
9	*Update*	Integer	indeks *update* kueri
10	*updateNow*	Integer	Pointer *update* kueri yang sudah dilalui proses update
11	*Query*	Integer	Indeks *travel* kueri

12	*left_ptr*	Integer	Pointer kiri untuk algoritma Mo
13	*right_ptr*	Integer	Pointer kanan untuk algoritma Mo
14	*Answer*	Integer[]	Menyimpan keluaran untuk setiap travel kueri
15	*Team*	Integer[]	Menyimpan nilai tim yang didukung untuk setiap *vertex*
16	*team_occurence*	Integer[]	Menyimpan jumlah kemunculan setiap tim pada proses treavel kueri
17	*Lca*	Integer[][]	Menyimpan *lowest common ancestor*
18	*Depth*	Integer[]	Menyimpan depth pada *tree* untuk setiap *vertex*
19	*Visited*	boolean	Menyimpan status *vertex* apakah sudah divisit saat proses Mo
20	*Mo*	Pair<integer, integer>	Menyimpan data *tree* yang sudah dilenearkan berupa urutan kemunculan pertama dan terakhir setiap *vertex*, serta tim yang didukung setiap *vertex*
21	*travel_query*	vector<pair<pair<integer,integer>,pair<pair<integer,integer> ,pair<integer,pair<integer,integer> > > > >	Kueri travel yang akan diproses dengan detail secara berurutan: start block, end block, indeks update, indeks kueri, start *vertex*, end *vertex*, *LCA*
22	*update_query*	Vector<pair<integer,<pair<integer, integer> >	Kueri untuk update yang akan diproses, secara berurutan data di dalamnya

			adalah: *vertex* yang diupdate, nilai tim baru, nilai tim lama.

4.4 Implementasi Fungsi Main

Fungsi *Main* nantinya menjadi fungsi yang dijalankan pertama kali oleh sistem. Fungsi ini berisikan fungsi-fungsi lainnya. Untuk implementasi Fungsi main dapat dilihat pada Kode Sumber 4.2 di bawah.

```
1.  int main(){
2.     int aa,bb;
3.     aa = io<int>(); bb = io<int>();
4.     n = (int) aa;   q = (int) bb;
5.     for(int x = 1 ; x <= n ; x++){
6.        aa=io<int>();
7.        team[x] = (int) aa;
8.     }
9.     for(int x=0;x<n-1;x++){
10.      int a,b;
11.      aa = io<int>();
12.      bb = io<int>();
13.      int a = (int) aa;
14.      int b = (int) bb;
15.      a++;
16.      b++;
17.      adj[a].pb(b);
18.      adj[b].pb(a);
19.    }
20.    idx = 0;
21.    depth[1] = 0;
22.    dfs(1,0);
23.    Gen_LCA();
24.    memset(visited,false,sizeof visited);
25.    block = 4021;
26.    update=0;
27.    query=1;
28.    for(int x = 1;x <= q;x++){
29.     int a,b,c;
30.     aa = io<int>();bb = io<int>(); cc = io<int>();
31.     a = (int) aa;  b = (int) bb; c = (int) cc;
```

```
32.     if(a == 2){
33.        b++;  c++;
34.        build_travel_query(b,c);
35.        query++;
36.     }
37.     else{
38.      b++;
39.      build_update_query(b,c);
40.      update++;
41.     }
42.  }
43.    sort(travel_query.begin(),travel_query.end());
44.    int p = travel_query.size();
45.    left_ptr = 1;
46.    right_ptr = 0;
47.    updateNow = 0;
48.    happiness_point= 0;
49.    for(int x = 0;x<p;x++){
50.      update = travel_query[x].s.f.f;
51.      int index = travel_query[x].s.f.s;
52.      int start_query = travel_query[x].s.s.f;
53.      int end_query = travel_query[x].s.s.s.f;
54.      int sama = travel_query[x].s.s.s.s;
55.      execute_MO(start_query,end_query);
56.      execute_update();
57.      if(sama     !=     -1)     answer[index]     =
   happiness_point+ team_occurence[Mo[awal[sama]].f]
   ;
58.      else answer[index] = happiness_point;
59.     }

60.      for(int x=1;x<=p;x++){
61.      printf("%d\n",answer[x]);
62.      }

63.    return 0;
64.    }
```

Kode Sumber 4.2 Implementasi Fungsi Main

4.5 Implementasi *Fast IO(Input Output)*

Pada permasalahan ini untuk mengoptimalkan waktu komputasi digunakan metode *Fast IO* yang merupakan metode

untuk mempercepat proses pembacaan masukkan oleh program. Metode *Fast IO* yang digunakan dapat dilihat pada Kode Sumber 4.3

```
template <typename T>
T io(){
   T res=0;
   int sign=1;
   char c;
   while(1){
     c=getchar_unlocked();
     if (c==' '||c=='\t') continue;
     else break;
   }
   res=c-'0';
   while(1){
     c=getchar_unlocked();
     if (c>='0' && c<='9') res=10*res+c-'0';
     else break;
   }
   return res;
}
```

Kode Sumber 4.3 Implementasi *Fast IO*

4.6 Implementasi Algoritma

Pada bagian ini akan dijelaskan mengenai implementasi dari subbab 3.3 yang dibagi menjadi beberapa subbab sebagai berikut

4.6.1 Impelementasi Fungsi deep_first_search

Fungsi *deep first search* digunakan untuk menelusuri *tree* dan menyiapkan data yang akan dibutuhkan untuk menjalankan algoritma Mo. Implementasi fungsi ini dapat dilihat pada Kode Sumber 4.4

```
1.  void dfs(now,int par)
2.  {
3.     awal[now] = ++idx;
4.     Mo[idx] = mp(team[now],now);
5.     int p = adj[now].size();
6.     for(int x=0;x<p;x++)
7.     {
8.       int cur=adj[now][x];
9.       if(cur!=par)
10.      {
11.        lca[cur][0] = now;
12.        depth[cur] = depth[now] + 1;
13.        dfs(cur,now);
14.      }
15.    }
16.    akhir[now] = ++idx;
17.    Mo[idx] = mp(team[now],now);
18.}
```

Kode Sumber 4.4 Implementasi Fungsi *Deep First Search*

4.6.2 Implementasi Fungsi *generate_lca*

Fungsi ini digunakan untuk membangun tabel *Lowest Common Ancestor* Implementasi fungsi ini dapat dilihat pada Kode Sumber 4.5

```
1.  void Gen_LCA()
2.  {
3.     for(int x=1;x<=17;x++)
4.     {
5.       for(int y=1;y<=n;y++)
6.       {
a.         lca[y][x] = lca[lca[y][x-1]][x-1];
7.       }
8.     }
9.  }
```

Kode Sumber 4.5 Implementasi fungsi *generate_lca*

4.6.3 Impelementasi fungsi *find_lca*

Fungsi ini digunakan untuk mencari *Lowest Common Ancestor* antara 2 *vertex*. Implementasi fungsi ini dapat dilihat pada Kode Sumber 4.6

```
int findLCA(int u,int v){
    for(int y=17;y>=0;y--)
    {
      if(lca[u][y]!=lca[v][y])
      {
        u = lca[u][y];
        v = lca[v][y];
      }
    }
    return lca[u][0];
}
```

Kode Sumber 4.6 Implementasi Fungsi *find_lca*

4.6.4 Implementasi Fungsi *build_travel_query*

Fungsi ini digunakan untuk membuat *travel_query* yang siap diproses. Implementasi fungsi ini dapat dilihat pada Kode Sumber 4.7

```
void build_travel_query(int b,int c){
    if(depth[b] < depth[c]) swap(b,c);
    int sel = depth[b] - depth[c];
    int cur= b;
    for(int x=0;x<=17;x++)
    {
      if((sel&1)==1)
      {
        cur= lca[cur][x];
      }
      sel=sel/2;
    }
    if(cur == c)
    {
      if(awal[b] > awal[c]) swap(b,c);
travel_query.pb(mp(mp(awal[b]/block,awal[c]/block
),mp(mp(update,query),mp(awal[b],mp(awal[c],-
1)))));
    }
    else
```

```
19.    {
20.      int cur2 = c;
21.      int true_lca = findLCA(cur,cur2);

22.      if(akhir[b]>awal[c]) swap(b,c);
23. travel_query.pb(mp(mp(akhir[b]/block,awal[c]/bloc
    k),mp(mp(update,query),mp(akhir[b],mp(awal[c],tru
    e_lca)))));
24.    }
25. }
```

Kode Sumber 4.7 Implementasi fungsi *build_travel_query*

4.6.5 Implementasi Fungsi *build_update_query*

Fungsi ini akan membangun data *update_query* yang siap diproses. Implementasi fungsi ini dapat dilihat pada Kode Sumber 4.8

```
1. void      build_update_query(int      vertex,int
   new_value){
2. update_query.pb(mp(vertex,mp(new_value,team[verte
   x])));
3.    team[vertex] = new_value;
4. }
```

Kode Sumber 4.8 Implementasi fungsi *build_update_query*

4.6.6 Implementasi Algoritma Modifikasi Mo

Algoritma untuk memproses path dan *happiness point*, algoritma ini terdiri dari dua fungsi yaitu *execute_Mo* yang berfungsi untuk menggeser pointer dan fungsi *Mo* yang memproses *vertex* yang sedang ditunjuk oleh pointer. Implementasi fungsi ini dapat dilihat pada Kode Sumber 4.9 dan Kode Sumber 4.10

```
1.  void execute_MO(int i,int j){
2.     while(j < right_ptr)
3.     {
4.       MO(right_ptr);
5.      right_ptr--;
6.     }
7.     while(j > right_ptr)
8.     {
9.       right_ptr++;
10.      MO(right_ptr);
11.    }
12.    while(i < left_ptr)
13.    {
14.      left_ptr--;
15.      MO(left_ptr);
16.    }
17.    while(i > left_ptr)
18.    {
19.      MO(left_ptr);
20.      left_ptr++;
21.    }
22. }
```

Kode Sumber 4.9 Implementasi Fungsi *execute_Mo*

```
1.  void MO(int now)
2.  {
3.     if(visited[Mo[now].s]==true)
4.     {
5.       visited[Mo[now].s]=false;
6.       team_occurence[Mo[now].f]--;
7.       happiness_point=happiness_point-
    team_occurence[Mo[now].f];
8.     }
9.     else
10.    {
11.      visited[Mo[now].s]=true;
12.      happiness_point=happiness_point+
    team_occurence[Mo[now].f];
13.      team_occurence[Mo[now].f]++;
14.    }
15. }
```

Kode Sumber 4.10 Implementasi fungsi *Mo*

4.6.7 Implementasi Fungsi proses update

Fungsi ini digunakan untuk memproses update kueri setelah *algoritma Mo* pada *travel_query* selesai dilakukan. Implementasi fungsi ini dapat dilihat pada Kode Sumber 4.11

```
1. void execute_update(){
2.    while(updateNow < update)
3.    {
4.      int indx = update_query[updateNow].f;
5.      int value = update_query[updateNow].s.f;
6.      int muncul = 0;
7.      int inpath;
8.      if(awal[indx] >= left_ptr && awal[indx] <=
   right_ptr) muncul++,inpath = awal[indx];
9.      if(akhir[indx] >= left_ptr && akhir[indx] <=
   right_ptr) muncul++,inpath = akhir[indx];
10.     if(muncul == 1)
11.     {
12.       team_occurence[Mo[inpath].f]--;
13.       happiness_point=happiness_point-
   team_occurence[Mo[inpath].f];
14.       happiness_point=happiness_point+
   team_occurence[value];
15.       team_occurence[value]++;
16.     }
17.     Mo[awal[indx]].f = value;
18.     Mo[akhir[indx]].f = value;
19.     updateNow++;
20.   }
21.   while(updateNow > update)
22.   {
23.    updateNow--;
24.    int indx = update_query[updateNow].f;
25.    int                              value_before=
   update_query[updateNow].s.s;
26.    int muncul = 0;
27.    int inpath;
28.    if(awal[indx]>=left_ptr&&
   awal[indx]<=right_ptr)muncul++,inpath=awal[indx];
29.    if(akhir[indx]>=left_ptr&&
   akhir[indx]<=right_ptr)muncul++,inpath=akhir[indx
   ];
30.   if(muncul == 1)
```

```
31.    {
32.      team_occurence[Mo[inpath].f]--;
33.      happiness_point=happiness_point-
   team_occurence[Mo[inpath].f];
34.      happiness_point=happiness_point+
   team_occurence[value_before];
35.      team_occurence[value_before]++;
36.     }
37.    Mo[awal[indx]].f = value_before;
38.    Mo[akhir[indx]].f = value_before;
39.    }
40.}
```

Kode Sumber 4.11 Implementasi fungsi *execute_update*

[*Halaman ini sengaja dikosongkan*]

BAB V
UJICOBA 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

Digunakan lingkungan uji coba dengan spesifikasi perangkat lunak dan perangkat keras seperti terlihat pada Tabel 5.1

Tabel 5.1 Lingkungan Uji Coba

No	Jenis Perangkat	Spesifikasi
1	Perangkat Keras	• ProcessorIntel(R) Core(TM) i5-8250U CPU @1.60GHz 1.80GHz • Random Access Memory: 8GB
2	Perangkat Lunak	• Sistem Operasi: Windows 10 Home • Text Editor: DevC++ • Bahasa Pemrograman: C++

5.2 Skenario Uji Coba

Pada bagian ini akan dijelaskan skenario yang akan digunakan untuk melakukan pengujian terhadap implementasi yang dibuat untuk permasalahan *ADAFTBLL*. Uji coba kebenaran akan dilakukan dengan melihat umpan balik yang diberikan oleh *SPOJ* setelah sumber kode dikirimkan. *SPOJ* akan mengecek kebenaran dari sumber kode yang dikirim dengan memasukan kasus uji dengan batasan yang sudah dijabarkan pada subbab 1.3. Uji coba kinerja akan dilakukan dengan mengirimkan kode sumber hasil implementasi program ke situs penilaian *SPOJ* sebanyak 10 kali kemudian menganalisa performa dari umpan balik yang diberikan.

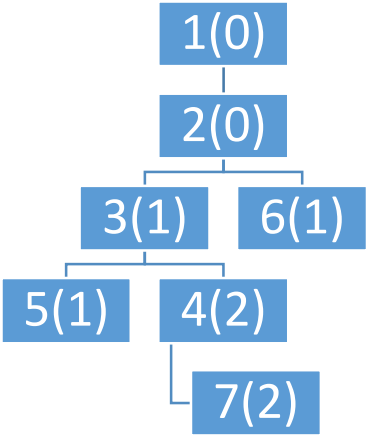
5.2.1 Evaluasi Kebenaran

Evaluasi dilakukan dengan mengecek masukan yang diberikan dengan keluaran yang dihasilkan dari implementasi program yang sudah dibuat sama dengan contoh keluaran yang ada pada permasalahan SPOJ - Ada and Football. Kasus uji dapat dilihat pada Tabel 5.2.

Tabel 5.2 Tabel Kasus Uji

Masukan	Keluaran
7 8 0 0 1 2 1 1 2 0 1 1 5 1 2 2 4 2 3 3 6 2 4 5 2 0 6 2 1 3 1 1 2 1 2 2 2 4 5 2 0 6 2 1 3	3 2 0 2 6 3

Setelah delapan baris pertama, akan direpresentasikan dalam tree yang telah diincrement seperti pada Gambar



Gambar 5.1 Representasi *Tree* data Masukan

Setelah representasi Tree dibuat, akan dilakukan penelusuran dengan fungsi *dfs*, dimulai dari vertex 1 sebagai *root* yang menghasilkan beberapa Array yang akan digunakan untuk membantu pemrosesan selanjutnya. Array tersebut dapat dilihat pada Tabel 5.3, Tabel 5.4 dan Tabel 5.5

Tabel 5.3 Tabel *Array Awal*

Indeks	1	2	3	4	5	6	7
Nilai	1	2	5	8	6	3	9

Tabel 5.4 Tabel *Array Akhir*

Indeks	1	2	3	4	5	6	7
Nilai	14	13	12	11	7	4	10

Tabel 5.5 Tabel *Array Mo*

Indeks	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Vertex	1	2	6	6	3	5	5	4	7	7	4	3	2	1
Tim	0	0	1	1	1	1	1	2	2	2	2	1	0	0

Dapat dilihat dari Tabel 5.3, Tabel 5.4, dan Tabel 5.5 bahwa terdapat kesesuaian antara *array awal*, *array akhir* dengan data pada *Array Mo*.

Setelah proses *dfs* selesai maka akan dilanjutkan dengan pembangunan kueri, baik kueri *travel* maupun kueri *update*. Pembuatan kueri ini berdasarkan kueri pada masukan, yaitu baris ke-9 dan seterusnya. Hasil kueri yang terbentuk dapat dilihat pada Tabel 5.6 dan Tabel 5.7

Tabel 5.6 Tabel Kueri *travel*

No.	Kueri	Start Block	End Block	Update	Kueri	Start Range	End Range	LCA
1	2 4 5	0	0	0	1	4	6	2
2	2 0 6	0	0	0	2	1	9	-1
3	2 1 3	0	0	0	3	2	8	-1
4	2 4 5	0	0	2	4	4	6	2
5	2 0 6	0	0	2	5	1	9	-1
6	2 1 3	0	0	2	6	2	8	-1

Tabel 5.7 Tabel Kueri *update*

No	Kueri	Vertex	Nilai Baru	Nilai Lama
1	1 1 2	2	2	0
2	1 2 2	3	2	1

Setelah kedua jenis kueri siap maka dilanjutkan dengan pemorsesan menggunakan algoritma utama, algoritma Mo. Proses yang terjadi dapat dilihat pada Tabel 5.8

Tabel 5.8 Tabel Pemrosesan Kueri

No	Kueri	Akhir Ptr Kiri	Akhir Ptr Kanan	Perpindahan Ptr Kiri	Perpindahan Ptr Kanan	Poin setelah Mo	Poin setalah Update	Occurrence tim LCA	Happiness Point akhir
0	Init	1	0	-	-	-	-	-	-
1	2 4 5	4	6	3	6	3	3	0	3
2	2 0 6	1	9	3	3	2	2	0	2
3	2 1 3	2	8	1	1	0	0	0	0
4	2 4 5	4	6	2	2	3	1	1	2
5	2 0 6	1	9	3	3	6	6	0	6
6	2 1 3	2	8	1	1	3	3	0	3

Dari Tabel 5.8 dapat dilihat bahwa *pointer* kiri dan *pointer* kanan akan berhenti pada *start range* dan *end range* kueri tersebut. *Start range* dan *end range* pada kueri menunjuk ke indek pada *Array Mo*. Untuk pemrosesan *Mo* misalnya untuk nomor 2 menjadi nomor 3, yang terjadi adalah sebagai berikut:

1. *Pointer* kiri berada pada indeks 1, sedangkan *start* kueri berawal dari 2, sehingga *pointer* kiri akan bergaser ke kanan 1 indeks
2. *Pointer* kiri menunjuk indeks 2, pada *Array Mo*(Tabel 5.5) indeks 2 memiliki nilai *vertex* 2
3. Karena *vertex* 2 termasuk di dalam *range path* pada proses nomor 2, maka kondisinya *visited*
4. Karena kondisi visited, maka nilai *happiness point* dari proses 2 akan dikurangi sejumlah *occurrence* tim yang sedang didukung *vertex* 2, yaitu tim 0 dengan occurrence 1 menghasilkan nilai 1
5. Untuk pointer kanan hal serupa terjadi, sehingga dilakukan pengurangan juga, sehingga hasil akhir *happiness point* adalah 0

Pada Tabel 5.8 dapat dilihat bahwa *happiness point* akhir adalah keluaran yang sama dengan contoh pada soal, sehingga

dapat dikatakan program berhasil menyelesaikan permasalahan dengan benar. Selain itu kolom perpindahan pointer menunjukkan efisiensi pengerjaan ketika kueri sudah digolongkan menjadi block.

5.2.2 Uji Coba Kebenaran

Uji coba kebenaran dilakukan dengan mengirimkan kode hasil implementasi program ke daring *SPOJ*. Permasalahan yang diselesaikan adalah *ADAFTBLL*. Setelah mengirimkan kode sumber maka akan mendapatkan umpan balik dari *SPOJ* seperti yang ada pada Gambar 5.2

26296840	2020-07-18 08:20:19	Ada and Football	**accepted** edit	9.37	20M	CPP14-CLANG

Gambar 5.2 Hasil Uji Coba Kebenaran

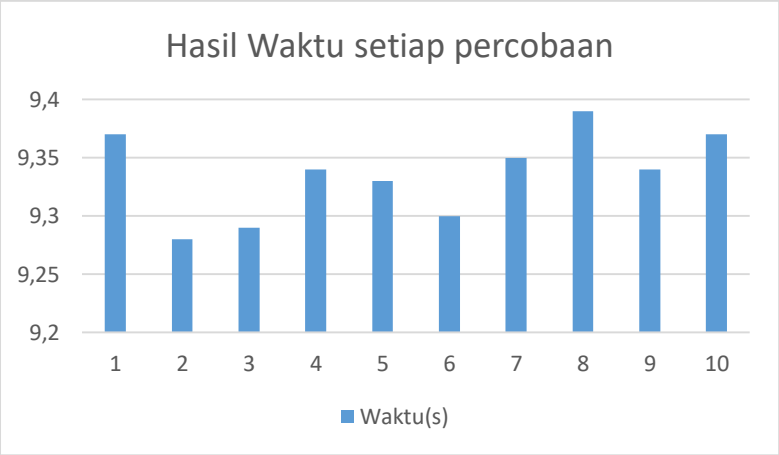
Dari hasil uji coba yang dilakukan kode sumber mendapatkan umpan balik *Accepted*. Waktu yang diperlukan program adalah 9,37 detik dan memori yang dibutuhkan program adalah 20 MB.

5.2.3 Uji Coba Kinerja

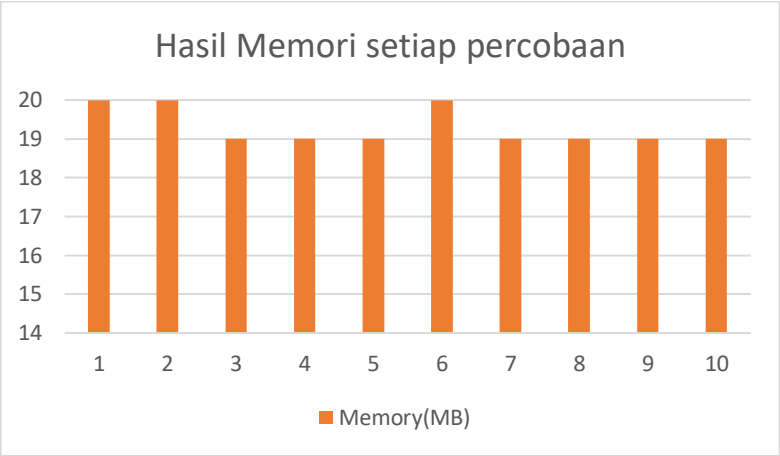
Setelah itu kode sumber yang sama akan dikirimkan sampai 10 kali untuk melihat variasi waktu dan memori yang dibutuhkan. Hasil uji coba dengan mengirimkan kode sumber sebanyak 10 kali, dapat dilihat pada Gambar 5.3, Gambar 5.4 dan Gambar 5.5

ID	DATE	PROBLEM	RESULT	TIME	MEM	LANG
26310868	2020-07-21 10:48:34	Ada and Football	**accepted** edit ideone it	9.37	19M	CPP14-CLANG
26310839	2020-07-21 10:41:03	Ada and Football	**accepted** edit ideone it	9.34	19M	CPP14-CLANG
26310833	2020-07-21 10:39:53	Ada and Football	**accepted** edit ideone it	9.39	19M	CPP14-CLANG
26310819	2020-07-21 10:35:39	Ada and Football	**accepted** edit ideone it	9.35	19M	CPP14-CLANG
26310806	2020-07-21 10:32:17	Ada and Football	**accepted** edit ideone it	9.30	20M	CPP14-CLANG
26310798	2020-07-21 10:30:25	Ada and Football	**accepted** edit ideone it	9.33	19M	CPP14-CLANG
26310786	2020-07-21 10:27:39	Ada and Football	**accepted** edit ideone it	9.34	19M	CPP14-CLANG
26310769	2020-07-21 10:23:45	Ada and Football	**accepted** edit ideone it	9.29	19M	CPP14-CLANG
26310761	2020-07-21 10:22:21	Ada and Football	**accepted** edit ideone it	9.28	20M	CPP14-CLANG
26310750	2020-07-21 10:19:59	Ada and Football	**accepted** edit ideone it	9.37	20M	CPP14-CLANG

Gambar 5.3 Hasil Uji Coba Kinerja



Gambar 5.4 Perbandingan Waktu Hasil Uji Coba



Gambar 5.5 Perbandingan Memori Hasil Uji Coba

Pada pengujian waktu, rata-rata waktu yang didapatkan dari 10x pengujian adalah 9,336 detik untuk keseluruhan kasus uji. Sedangkan untuk memori dari 10 pengujian tidak menunjukkan perbedaan, seluruh pengujian menghasilkan rata-rata 19,4 MB memori.

5.2.4 Uji Coba Konstanta *Block*

Pada pemrosesan kueri dengan *algoritma Mo* dibutuhkan pembagian kueri menjadi beberapa *block*. Pembagian ini akan menentukan bagaimana kinerja program berjalan dikarenakan pembanding utama dalam pengurutan kueri sendiri adalah *start block* dan *end block*. Dalam pengujian ini akan dicari nilai yang memiliki kinerja paling baik. Berikut Tabel 5.9 yang menunjukkan beberapa nilai yang telah diuji dan hasil umpan balik yang didapatkan.

Tabel 5.9 Tabel Uji Coba Konstanta *Block*

No	Nilai	Status	Waktu(s)	Memory(MB)
1	1000	Time Limit Exceeded	-	19
2	2000	Accepted	13,79	20
3	3000	Accepted	10,02	20
4	4000	Accepted	9,38	19
5	5000	Accepted	9,80	19
6	6000	Accepted	10,70	19
7	7000	Accepted	11,82	19
8	8000	Accepted	13,01	20
9	9000	Accepted	14,63	20
10	10000	Time Limit Exceeded	-	20

Dari data pengujian didapatkan bahwa hasil yang cukup baik dan mengalami peningkatan adalah pada rentang 3000 sampai 5000, sehingga dilakukan pengujian lagi pada rentang itu dengan interval sebesar 200. Hasil pengujian dapat dilihat pada Tabel 5.10

Tabel 5.10 Tabel Uji Coba Konstanta *Block* 2

No	Nilai	Status	Waktu(s)	Memory(MB)
1	3000	Accepted	10,02	20
2	3200	Accepted	9,68	19
3	3400	Accepted	9,55	19
4	3600	Accepted	9,42	20
5	3800	Accepted	9,44	19
6	4000	Accepted	9,38	19
7	4200	Accepted	9,61	20
8	4400	Accepted	9,84	20
9	4600	Accepted	9,56	19
10	4800	Accepted	11,32	19
11	5000	Accepted	9,80	20

Berdasarkan Tabel dapat dilihat bahwa rentang yang cukup baik adalah 3800 sampai dengan 4200 sehingga akan dilakukan pengujian lagi dengan nilai interval sebesar 50. Hasil pengujian dapat dilihat pada tabel 5.11

Tabel 5.11 Table Uji Coba Konstanta *Block* 3

No	Nilai	Status	Waktu(s)	Memory(MB)
1	3800	Accepted	9,44	20
2	3850	Accepted	9,44	19
3	3900	Accepted	9,38	19
4	3950	Accepted	9,38	19
5	4000	Accepted	9,38	20
6	4050	Accepted	9,35	19
7	4100	Accepted	9,44	19
8	4150	Accepted	9,55	20
9	4200	Accepted	9,61	20

Rentang berikutnya adalah antara 3950 sampai 4050, akan diuji dengan rentang 25. Hasil pengujian dapat dilihat di tabel 5.12

Tabel 5.12 Tabel Pengujian Konstanta *Bloack* 4

No	Nilai	Status	Waktu(s)	Memory(MB)
1	3950	Accepted	9,38	20
2	3975	Accepted	9,44	19
3	4000	Accepted	9,38	19
4	4025	Accepted	9,32	19
5	4050	Accepted	9,35	20

Selanjutnya dipilih rentang 4000 sampai 4050 dengan interval 10. Hasil pengujian dapat dilihat pada table 5.13

Tabel 5.13 Tabel Pengujian Konstanta *Block* 5

No	Nilai	Status	Waktu(s)	Memory(MB)
1	4000	Accepted	9,38	20
2	4010	Accepted	9,37	19
3	4020	Accepted	9,33	19
4	4025	Accepted	9,32	20
5	4030	Accepted	9,33	19
6	4040	Accepted	9,34	20
7	4050	Accepted	9,35	20

Berikutnya dipilih rentang 4020 sampai 4030 dengan interval 1, sebagai pengujian terakhir. Hasil pengujian dapat dilihat pada tabel 5.14

Tabel 5.14 Tabel Hasil Pengujian Konstanta *Block* 5

No	Nilai	Status	Waktu(s)	Memory(MB)
1	4020	Accepted	9,33	20
2	4021	Accepted	9,28	19
3	4022	Accepted	9,30	19
4	4023	Accepted	9,29	20
5	4024	Accepted	9,30	19
6	4025	Accepted	9,32	20
7	4026	Accepted	9,32	20
8	4027	Accepted	9,32	20
9	4028	Accepted	9,34	19
10	4029	Accepted	9,33	20
11	4030	Accepted	9,33	19

Dari keseluruhan hasil pengujian, didapat bahwa nilai konstanta *block* yang optimal digunakan pada permasalahan ini adalah 4021.

[*Halaman ini sengaja dikosongkan*]

BAB VI
KESIMPULAN

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

6.1 Kesimpulan

Dari analisis dan uji coba yang dilakukan terhadap implementasi algoritma untuk menyelesaikan permasalahan *ADAFTBLL*, dapat diambil kesimpulan sebagai berikut:

1. Permasalahan *ADAFTBLL* pada situs *SPOJ* dapat diselesaikan dengan mengubah *Tree* menjadi representasi *array* linier.
2. *Algoritma Mo* dapat menyelesaikan model dari permasalahan *ADAFTBLL* yang telah diubah menjadi representasi *Array* linier.
3. Rata-rata waktu dari implementasi dari *algoritma Mo* kurang dari batas waktu yang diberikan oleh soal dengan waktu rata-rata 9,336 detik untuk seluruh kasus uji dan memori rata-rata yang digunakan 19,4 MB.

6.2 Saran

Pada tugas akhir ini tentunya terdapat kekurangan serta nilai-nilai yang dapat penulis ambil. Berikut adalah saran-saran yang dapat diambil melalui tugas akhir ini:

1. Untuk ke depannya, materi yang ada pada Tugas Akhir ini dapat menjadi bahan riset untuk mencari optimasi yang lebih lanjut.
2. Peluang optimasi pada permasalahan ini di antaranya adalah pada tahap-tahap preproses sebelum algoritma Mo.

[*Halaman ini sengaja dikosongkan*]

DAFTAR PUSTAKA

[1] Morras, "Sphere Onlinde Judge," 8 10 2017. [Online]. Available: https://www.spoj.com/problems/ADAFTBLL/. [Accessed 16 2 2020].

[2] T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein, Introduction to Algorithms Third Edition, London: The MIT Press, 2009.

[3] Geeks for Geeks, "Deep First Search - Geeks for Geeks," [Online]. Available: https://www.geeksforgeeks.org/depth-first-search-of-dfs-for-a-graph. [Accessed 11 3 2020].

[4] M. A. e. a. Bender, "Lowest Common Ancestors in Trees and Directed Acyclic Graphs," *Journal of Algorithms*, 2005.

[5] Anudeep2011, "Mo's Algorithm," 28 12 2014. [Online]. Available: https://blog.anudeep2011.com/mos-algorithm/. [Accessed 11 3 2020].

[*Halaman ini sengaja dikosongkan*]

BIODATA PENULIS

Rimba Azhara, lahir di Banyuwangi pada tanggal 10 Oktober 1997, penulis telah menempuh pendidikan di SDN 3 Rejoagung Srono Banyuwangi, SMPN 1 Genteng Banyuwangi, dan SMAN 1 Genteng Banyuwangi. Penulis melanjutkan studi kuliah program sarjana di Jurusan Informatika ITS. Selama kuliah di Informatika ITS, penulis mengambil bidang minat Algoritma dan pemrograman (AP). Selama menempuh perkuliahan penulis aktif di kegiatan organisasi seperti staf kepanitiaan PSDM di ITS Foreign Language Society (2017 dan 2018) . Penulis dapat dihubungi melalui surel berikut: rimbaazhara55@gmail.com