Institut
Teknologi
Sepuluh Nopember

TUGAS AKHIR - KI141502

DESAIN DAN ANALISIS ALGORITMA LOCAL SEARCH DENGAN METODE HILL CLIMBING PADA PERMASALAHAN GRACEFUL LABELLING PADA GENERAL TREE

WILLIAM ALBERTUS DEMBO
NRP 05111540000075

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

Dosen Pembimbing 2
Wijayanti Nurul Khotimah, S.Kom., M.Sc.

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

[*Halaman ini sengaja dikosongkan*]

TUGAS AKHIR - IF184802

DESAIN DAN ANALISIS ALGORITMA LOCAL SEARCH DENGAN METODE HILL CLIMBING PADA PERMASALAHAN GRACEFUL LABELLING PADA GENERAL TREE

WILLIAM ALBERTUS DEMBO
NRP 05111540000075

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

Dosen Pembimbing 2
Wijayanti Nurul Khotimah, S.Kom., M.Sc.

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

[*Halaman ini sengaja dikosongkan*]

UNDERGRADUATE THESES - IF184802

DESIGN AND ANALYSIS OF LOCAL SEARCH ALGORITHM WITH HILL CLIMBING METHOD FOR GRACEFUL LABELLING PROBLEMS IN GENERAL TREE

WILLIAM ALBERTUS DEMBO
NRP 05111540000075

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

Supervisor 2
Wijayanti Nurul Khotimah, S.Kom., M.Sc.

INFORMATICS DEPARTMENT
Faculty of Information Technology and Communication
Institut Teknologi Sepuluh Nopember
Surabaya, 2018

[*Halaman ini sengaja dikosongkan*]

LEMBAR PENGESAHAN

DESAIN DAN ANALISIS ALGORITMA LOCAL SEARCH DENGAN METODE HILL CLIMBING PADA PERMASALAHAN GRACEFUL LABELLING PADA GENERAL TREE

TUGAS AKHIR

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

Oleh:

William Albertus Dembo
NRP. 05111540000075

Disetujui oleh Dosen Pembimbing Tugas Akhir:

Rully Sulaiman, S.Kom., M.Kom. .....

NIP. 19700213194021001 (Pembimbing 1)

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

NIP. 198603122012122004 (Pembimbing 2)

SURABAYA
JANUARI, 2019

[*Halaman ini sengaja dikosongkan*]

ABSTRAK

DESAIN DAN ANALISIS ALGORITMA LOCAL SEARCH DENGAN METODE HILL CLIMBING PADA PERMASALAHAN GRACEFUL LABELLING PADA GENERAL TREE

Nama	:	William Albertus Dembo
NRP	:	05111540000075
Departemen	:	Departemen Informatika, Fakultas Teknologi Informasi dan Komunikasi, ITS
Pembimbing I	:	Rully Sulaiman, S.Kom., M.Kom.
Pembimbing II	:	Wijayanti Nurul Khotimah, S.Kom., M.Sc.

Abstrak

Graceful labelling dari suatu graf G dengan N node adalah injeksi $f : V(G) \rightarrow 0, 1, 2, ..., N - 1$ *sehingga ketika setiap edge* $xy \in E(G)$ *diberi label* $|f(x) - f(y)|$ *maka semua label pada edge berbeda. Hal ini pertama kali dicetuskan oleh Rosa pada tahun 1967.*

Pada Tugas Akhir ini akan dirancang penyelesaian permasalahan graceful labelling pada general tree dengan batasan maksimal jumlah node pada tree adalah 27. Permasalahan ini dapat diselesaikan dengan menggunakan algoritma local search dengan metode hill climbing dan dibantu dengan batasan tabu. Hal yang harus perhatikan adalah bagaimana menghasilkan ruang pencarian solusi dari permasalahan ini sehingga dapat ditelusuri solusinya menggunakan teknik heuristik.

Hasil dari tugas akhir ini telah berhasil menyelesaikan permasalahan di atas dengan cukup efisien, dengan waktu penyelesa-

ian terbaik 0.17 detik dan penggunaan memori 16 MB.

Kata Kunci: graceful labelling; local search; hill climbing; heuristik;

ABSTRACT

DESIGN AND ANALYSIS OF LOCAL SEARCH ALGORITHM WITH HILL CLIMBING METHOD FOR GRACEFUL LABELLING PROBLEMS IN GENERAL TREE

Name	:	William Albertus Dembo
Student ID	:	05111540000075
Department	:	Informatics Department, Faculty of Information Technology and Communication, ITS
Supervisor I	:	Rully Sulaiman, S.Kom., M.Kom.
Supervisor II	:	Wijayanti Nurul Khotimah, S.Kom., M.Sc.

Abstract

A graceful labelling of a graph G with N nodes is an injection $f : V(G) \rightarrow 0, 1, 2, ..., N - 1$ *such that when each edge* $xy \in E(G)$ *is assigned the label,* $|f(x) - f(y)|$*, all of the edge labels are distinct. This idea was introduced by Rosa in 1967.*

This undergraduate thesis will design the problem solving of graceful labelling in general tree with nodes in tree limited to 27. This problem can be solved using local search algorithm with hill climbing method and tabu limit. The difficult thing from this method is the correct way to generate the solution search space so it can be traversed by heuristic method.

The result shows that graceful labelling problem in general tree is succesfully solved efficiently with best time of 0.17 seconds and memory usage of 16 MB.

Keywords: graceful labelling; local search; hill climbing; heuristic;

[*Halaman ini sengaja dikosongkan*]

KATA PENGANTAR

Puji syukur penulis panjatkan kepada Tuhan Yang Maha Esa atas penyertaan dan karunia-Nya sehingga penulis dapat menyelesaikan tugas akhir dan laporan akhir dalam bentuk buku ini. Penelitian tugas akhir ini dilakukan untuk mengeksplorasi topik yang menarik perhatian penulis serta memenuhi salah satu syarat dalam mendapatkan gelar Sarjana Komputer di Departemen Informatika Fakultas Teknologi Informasi dan Komunikasi Institut Teknologi Sepuluh Nopember. Penulis memiliki harapan bahwa apa yang penulis kerjakan dapat membawa manfaat bagi perkembangan ilmu pengetahuan di bidang komputer dan bagi penulis sendiri selaku peneliti.

Penulis ingin mengucapkan terima kasih kepada semua pihak yang telah membimbing dan memberi dukungan baik secara langsung maupun tidak langsung selama proses pengerjaan tugas akhir ini maupun selama menempuh masa studi. Pihak tersebut antara lain:

1. Keluarga besar penulis yang selalu memberikan doa dan semangat sehingga penulis dapat menyelesaikan tugas akhir dalam rentang waktu yang diharapkan.
2. Bapak Rully Soelaiman S.Kom., M.Kom., selaku pembimbing penulis yang memberikan dukungan baik berupa didikan dan ajaran, maupun semangat dan nasihat selama menempuh masa studi dan pengerjaan tugas akhir. Berkat bimbingan dan ketersediaannya untuk berdiskusi, penulis dapat menyelesaikan tugas akhir dalam rentang waktu yang diharapkan.
3. Ibu Wijayanti Nurul Khotimah, S.Kom., M.Sc., selaku pembimbing penulis yang memberikan dukungan berupa ilmu, arahan, dan semangat selama pengerjaan tugas akhir.
4. Seluruh dosen dan karyawan Departemen Informatika Fakul-

tas Teknologi Informasi dan Komunikasi Institut Teknologi Sepuluh Nopember yang telah memberi ilmu dan waktunya untuk mempersiapkan penulis agar siap untuk masuk ke dalam dunia kerja.

5. Teman-teman, kakak-kakak, dan adik-adik mahasiswa Departemen Informatika Fakultas Teknologi Informasi dan Komunikasi Institut Teknologi Sepuluh Nopember yang senantiasa membantu, menemani, memberi semangat dan kenangan selama kurang lebih 3,5 tahun masa studi.
6. Keluarga *administrator* Laboratorium Pemrograman 2 Departemen Informatika Fakultas Teknologi Informasi dan Komunikasi Institut Teknologi Sepuluh Nopember yang selalu menemani dan membantu penulis ketika ada kesusahan selama masa studi.
7. Teman-teman kelompok GLBK yang sudah menemani, berjuang dan menorehkan banyak kenangan selama masa studi.
8. Teman-teman kelompok Petualang yang sudah menemani penulis menelusuri berbagai hobi berpetualang selama masa studi.

Penulis mohon maaf jika masih ada kekurangan pada tugas akhir ini. Penulis juga berharap tugas akhir ini dapat memberikan kontribusi dan manfaat bagi pembaca.

Surabaya, Desember 2018

William Albertus Dembo

DAFTAR ISI

[*Halaman ini sengaja dikosongkan*]

DAFTAR GAMBAR

DAFTAR TABEL

[*Halaman ini sengaja dikosongkan*]

DAFTAR KODE SUMBER

[*Halaman ini sengaja dikosongkan*]

BAB I
PENDAHULUAN

Bab ini menjelaskan konteks tugas akhir yang akan dikerjakan, termasuk latar belakang, rumusan masalah, batasan masalah, tujuan, manfaat, metodologi, dan sistematika penulisan.

1.1. Latar Belakang

Pendekatan terhadap permasalahan *graceful labelling* telah banyak dilakukan oleh peneliti di berbagai belahan dunia seperti dirangkum oleh Joseph A. Gallian dalam risetnya yang berjudul *A Dynamic Survey of Graph Labeling*[1] yang terus diperbaharui hingga edisi ke 20 pada 2017 lalu. Pendekatan permasalahan *graceful labelling* khususnya pada *tree* terbagi menjadi dua arah penyelesaian yaitu penyelesaian pada *tree* dengan kelas khusus dan *tree* secara umum. Pendekatan penyelesaian *graceful labelling* pada *tree* kelas khusus cenderung menggunakan properti khusus pada kelas tersebut sedangkan pada *tree* secara umum maka hal ini akan sulit dilakukan. Salah satu terobosan penyelesaian *graceful labelling* pada *tree* secara umum dilakukan oleh Aldred dan Mckay [2] yang memastikan bahwa semua *tree* hingga 27 vertex memiliki *graceful labelling*.

Topik Tugas Akhir ini mengacu pada permasalahan klasik SPOJ dengan kode PT07G. Pada permasalahan ini terdapat sebuah tree yang memiliki N vertex dan $N-1$ edge. Setiap vertex pada tree tersebut harus diberikan sebuah nilai diantara 0 hingga $N-1$ supaya setiap vertex memiliki nilai yang unik, sedangkan nilai dari edge yaitu selisih dari nilai vertex yang dihubungkan. Dengan ketentuan yang diberikan, tentukan konfigurasi penyusunan nilai pada vertex sehingga semua nilai pada edge berbeda. Permasalahan ini sesuai dengan pelabelan graf *β-labelings* yang dicetuskan oleh Alexander Rosa tahun 1967 pada risetnya tentang pelabelan graf[3]

yang sekarang lebih populer dengan sebutan *graceful labelling*.

Untuk menyelesaikan permasalahan pada tugas akhir ini, penulis akan menggunakan pendekatan solusi dengan teknik *hill climbing* yang mengacu pada riset Aldred dan McKay[2]. Selain dapat menjawab pertanyaan dengan benar, waktu juga menjadi salah satu faktor penting untuk memberikan gambaran tentang peforma dari algoritma yang dirancang.

Hasil Tugas Akhir ini diharapkan dapat memberi gambaran mengenai performa dari algoritma *local search* dengan metode *hill climbing* untuk menyelesaikan permasalahan *graceful labelling* secara optimal dan diharapkan dapat memberikan kontribusi pada perkembangan ilmu pengetahuan dan teknologi informasi.

1.2. Rumusan Masalah

Permasalahan yang akan diselesaikan pada tugas akhir ini adalah sebagai berikut:

1. Bagaimana membawa permasalahan *graceful labelling* pada *general tree* ke dalam domain heuristik?
2. Bagaimana menyelesaikan permasalahan *graceful labelling* pada *general tree* menggunakan *local search*?
3. Bagaimana performa penyelesaian permasalahan *graceful labelling* pada *general tree* menggunakan *local search*?

1.3. Batasan Masalah

Batasan dari masalah yang akan diselesaikan adalah sebagai berikut:

1. Implementasi dilakukan menggunakan bahasa pemrograman C++.
2. Batas maksimum *node* adalah 27.
3. Dataset yang digunakan adalah dataset pada SPOJ Colorful Lights Party (PT07G) .

1.4. Tujuan

Tujuan tugas akhir ini adalah sebagai berikut:

1. Melakukan analisis permasalahan *graceful labelling* pada *general tree* di dalam domain heuristik.
2. Melakukan analisis dan mendesain algoritma untuk menyelesaikan permasalahan *graceful labelling* pada *general tree* menggunakan *local search*.
3. Melakukan analisis performa penyelesaian permasalahan *graceful labelling* pada *general tree* menggunakan *local search*.

1.5. Manfaat

Manfaat tugas akhir ini adalah sebagai berikut:

1. Membantu memahami permasalahan *graceful labelling* pada *general tree* di dalam domain heuristik.
2. Membantu memahami penggunaan algoritma *local search* pada permasalahan *graceful labelling* pada *general tree*.
3. Mengetahui performa penyelesaian permasalahan *graceful labelling* pada *general tree* dengan menggunakan *local search*.

1.6. Metodologi

Metodologi pengerjaan yang digunakan pada tugas akhir ini memiliki beberapa tahapan. Tahapan-tahapan tersebut adalah sebagai berikut:

1. Penyusunan proposal tugas akhir
   Pada tahapan ini, penulis akan menyusun rencana dan langkah-langkah yang akan dilakukan dalam proses pembuatan tugas akhir.
2. Studi literatur
   Pada tahapan ini, penulis mengumpulkan referensi yang diperlukan guna mendukung pengerjaan tugas akhir. Referensi

yang digunakan dapat berupa hasil penelitian yang sudah pernah dilakukan, buku, artikel internet, atau sumber lain yang bisa dipertanggungjawabkan.

3. Desain
   Pada tahapan ini, penulis melakukan desain rancangan algoritma yang akan digunakan untuk proses pengenalan karakter pada SPOJ Colorful Lights Party (PT07G) .
4. Implementasi algoritma
   Pada tahapan ini, penulis mulai mengembangkan algoritma yang telah didukung oleh hasil rancangan desain pada tahapan sebelumnya. Implementasi ini dilakukan dengan menggunakan bahasa pemrograman C++.
5. Pengujian dan evaluasi
   Pada tahapan ini, penulis melakukan pengujian dan evaluasi menggunakan dataset SPOJ Colorful Lights Party (PT07G) pada sistem penilaian daring SPOJ untuk mengetahui nilai dan performa algoritma yang telah dibangun.
6. Penyusunan buku
   Pada tahapan ini, penulis menyusun laporan yang menjelaskan teori dan metode yang benar-benar digunakan dalam menyelesaikan studi kasus SPOJ Colorful Lights Party (PT07G) serta hasil dari implementasi algoritma yang telah dibuat dalam bentuk buku tugas akhir.

1.7. Sistematika Penulisan

Sistematika laporan tugas akhir yang akan digunakan adalah sebagai berikut:

1. Bab 1 : PENDAHULUAN
   Bab ini menjelaskan konteks tugas akhir yang akan dikerjakan, termasuk latar belakang, rumusan masalah, batasan masalah, tujuan, manfaat, metodologi, dan sistematika penulisan.

2. Bab 2 : DASAR TEORI
   Bab ini menjelaskan dasar teori yang akan digunakan dalam proses pengerjaan tugas akhir.
3. Bab 3 : DESAIN
   Bab ini menjelaskan desain algoritma yang akan dibangun berdasarkan dasar teori yang dijelaskan pada bab 2.
4. Bab 4 : IMPLEMENTASI
   Bab ini menjelaskan implementasi desain algoritma yang dijelaskan pada bab 3.
5. Bab 5 : PENGUJIAN DAN EVALUASI
   Bab ini menjelaskan hasil pengujian dan evaluasi algoritma yang telah diimplementasikan pada bab 4.
6. Bab 6 : PENUTUP
   Bab ini berisi kesimpulan yang telah didapat dari hasil pengujian dan evaluasi yang telah dilakukan.

[*Halaman ini sengaja dikosongkan*]

BAB II
DASAR TEORI

Bab ini menjelaskan dasar teori yang penulis gunakan sebagai landasan pengerjaan tugas akhir. Pertama, bab ini menjelaskan deskripsi permasalahan pada SPOJ Colorful Lights Party (PT07G) , dilanjutkan dengan menjelaskan beberapa metode penyelesaian yang dapat digunakan, dilanjutkan dengan deskripsi umum yang terdapat pada tugas akhir ini.

2.1. Deskripsi Permasalahan

Permasalahan pada tugas akhir ini mengacu pada SPOJ Colorful Lights Party (PT07G) [4]. Terdapat sebuah aula dengan sistem pencahayaan elektronik yang terdiri dari N lampu yang saling terhubung menggunakan $N - 1$ kabel, setiap kabel hanya dapat menghubungkan dua buah lampu yang berbeda. Susunan lampu pada aula membentuk sebuah topologi *tree* untuk mengurangi jumlah kabel yang berlebihan.

Masing-masing lampu memiliki nomor pengenal dari 1 hingga N agar mudah untuk di identifikasi. Setiap lampu akan diberikan masing-masing warna yang unik, setiap warna akan diberikan nomor pengenal dari 0 hingga $N - 1$. Setiap kabel yang menghubungkan dua kabel akan memiliki warna dengan nomor pengenal bernilai selisih dari nomor pengenal warna lampu yang dihubungkan. Contoh jika kabel menghubungkan lampu dengan warna 1 dan 7 maka warna dari kabel itu adalah 6 (7 - 1). Permasalahan yang perlu diselesaikan yaitu menentukan konfigurasi pemberian warna lampu sehingga setiap warna lampu memiliki warna yang unik dan begitu juga untuk setiap kabel memiliki warna yang unik.

2.2. Deskripsi Umum

Subbab ini menjelaskan definisi, deskripsi dan landasan yang akan digunakan pada keseluruhan penyelesaian masalah.

2.2.1. General Tree dan Tree dengan Kelas

Dalam teori graf, sebuah tree adalah graf yang tidak berarah dimana setiap dua vertex terhubung dengan tepat satu jalur. Definisi diatas merupakan definisi tree secara umum, sedangkan tree dengan kelas khusus merupakan kelas dari tree yang telah terbukti memiliki *graceful labelling* berdasarkan hasil riset. Beberapa tree dengan kelas khusus yang diteliti memiliki *graceful labelling* yang dirangkum pada [5] yaitu:

1. *Caterpillars*: *Tree* yang semua vertex-nya memiliki jarak maksimal satu dari *path* utama[6]. Rosa[3] membuktikan bahwa semua *caterpillars* memiliki *graceful labelling*.
2. *Symmetrical trees*: Sebuah *tree* dengan *root* dimana setiap level memiliki vertex dengan derajat yang sama. Bermond dan Schonheim[7] membuktikan bahwa semua *symmetrical tree* memiliki *graceful labelling*.
3. *Spider tree*: *Tree* yang memiliki maksimal satu buah vertex dengan derajat lebih dari dua[8]. Pada [9] telah dibuktikan bahwa semua *spider tree* memiliki *graceful labelling*.
4. *Lobster tree*: *Tree* yang semua vertex-nya memiliki jarak maksimal dua dari *path* utama[10].

2.2.2. Local Search

Dalam ilmu komputer, *local search*[11] adalah algoritma yang bertujuan mencari state terbaik dalam *state-space* berdasarkan fungsi objektif yang telah ditentukan. Algoritma ini berfokus mencari hasil akhir tanpa memperdulikan langkah yang ditempuh untuk mendapatkan hasil tersebut. *Local search* beroperasi menggunakan

satu buah *node* sebagai penanda *state* yang sedang dievaluasi dan secara umum hanya akan bergerak ke *node* yang menjadi tetangga dari *node* sekarang.

2.2.3. Hill Climbing

Pada buku *"Artificial Intelligence: A Modern Approach"*[11] dijelaskan bahwa *hill climbing* secara sederhana adalah perulangan yang terus menerus bergerak ke arah yang memiliki nilai evaluasi lebih tinggi. Algoritma ini tidak melakukan penyimpanan terhadap ruang pencarian sehingga data yang diperlukan hanya kondisi sekarang dan tetangga terdekatnya saja. Ketika terdapat lebih dari satu arah yang memiliki nilai evaluasi yang sama maka dapat dipilih yang mana saja. Terdapat beberapa kelemahan pada algoritma ini yaitu :

- **Local Maxima**: sebuah kondisi puncak dimana tidak ada nilai yang lebih baik pada kondisi tetangganya namun nilainya lebih rendah dari nilai tertinggi pada ruang pencarian. *Hill climbing* akan mengehentikan pencarian pada *local maxima* meskipun solusi sesungguhnya belum ditemukan.

- **Plateaux**: Sebuah daerah berbentuk dataran pada ruang pencarian sehingga evaluasi kondisi sekarang dan semua tetangganya bernilai sama, pada saat ini maka *hill climbing* akan melakukan pergerakan secara acak.

2.2.4. Graceful Labelling

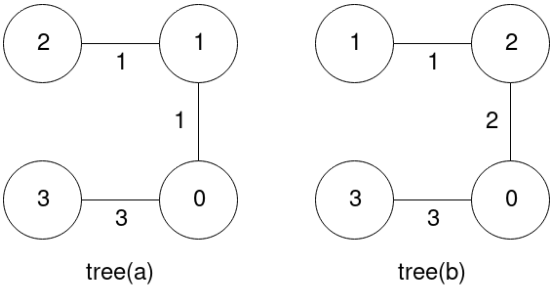
Graceful labelling pertama kali dicetuskan oleh Rosa pada [3] dengan nama *"β-valuation"*. *Graceful labelling* dari suatu graf G dengan N *node* adalah injeksi $f : V(G) \rightarrow 0, 1, 2, ..., N-1$ sehingga ketika setiap *edge* $xy \in E(G)$ diberi label $|f(x) - f(y)|$ maka semua label pada *edge* berbeda.

2.2.5. Tabu Search

Hill climbing mencari solusi potensial dari permasalahan kemudian mencari solusi lain pada tetangga terdekatnya dengan harapan menemukan solusi yang lebih baik. Metode *local search* memiliki kecenderungan untuk berhenti di daerah suboptimal (*local maxima*).*Tabu search* merupakan metode meta-heuristik yang dipasangkan pada metode heuristik lainnnya, metode ini dicetuskan oleh Glover pada [12]. *Tabu Search* dapat meningkatkan kinerja *hill climbing* menghadapi masalah *local maxima* menggunakan strategi pembatasan langkah dan melonggarkan aturan dasar *hill climbing*. Pertama, setiap langkah yang memburuk dapat diterima apabila tidak ada langkah yang menghasilkan nilai evaluasi yang lebih baik(seperti pada *local maxima*). Selanjutnya diperkenalkan batasan baru atau disebut tabu untuk mencegah pencarian kembali melalui langkah yang sudah dilewati dalam waktu dekat.

2.3. Abstraksi Permasalahan

Permasalahan pada SPOJ Colorful Lights Party (PT07G) merupakan permasalahan pencarian *graceful labelling* pada *general tree* dimana aula dapat direpresentasikan sebagai sebuah *general tree* yang terdiri dari lampu sebagai *node* dan kabel sebagai *edge*. Warna pada lampu dan kabel dapat direpresentasikan sebagai sebuah nilai dari *node* dam *edge*. Mendapatkan semua *edge* bernilai unik pada *tree* dengan n *node* sama halnya dengan mendapatkan $n - 1$ nilai yang unik pada *edge*, ilustrasi dapat dilihat pada Gambar 2.1 dimana *tree(a)* yang memiliki tiga *edge* dengan nilai masing-masing 1,1, dan 3 memiliki dua nilai yang unik sedangkan pada *tree(b)* yang juga memiliki tiga *edge* dengan nilai masing-masing 1,2, dan 3 memiliki tiga nilai yang unik.



Gambar 2.1: Ilustrasi korelasi jumlah nilai *edge* yang unik dengan semua *edge* bernilai unik

[*Halaman ini sengaja dikosongkan*]

BAB III
ANALISIS DAN PERANCANGAN

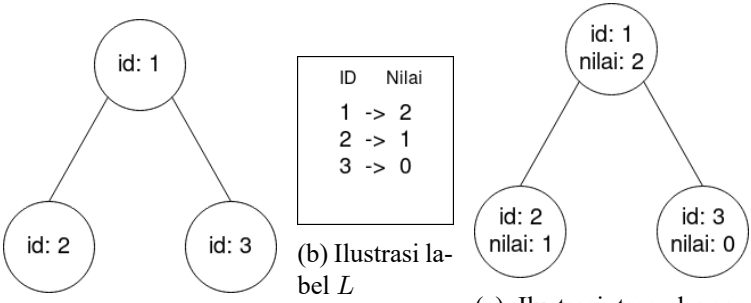
Bab ini akan menjelaskan analisis dan desain sistem yang digunakan untuk menyelesaikan permasalahan pada tugas akhir ini.

3.1. Analisis Permasalahan

Pada bagian ini akan diberikan analisis permasalahan yang akan digunakan untuk membantu pembuatan sistem.

3.1.1. Definisi Label

Sebuah label L dari *tree* T dengan *node* berjumlah n adalah sebuah fungsi injeksi yang memetakan setiap *node* dari T terhadap kumpulan nilai antara 0 hingga $n-1$. Ilustrasi tree T dengan *node* berjumlah 3 dapat dilihat pada Gambar 3.1a, ilustrasi label L untuk T dapat dilihat pada Gambar 3.1b, dan ilustrasi untuk tree T dengan label L dapat dilihat pada Gambar 3.1c.



(a) Ilustrasi tree T

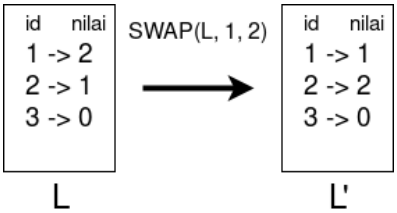
(b) Ilustrasi label L

(c) Ilustrasi tree dengan label

Gambar 3.1: Ilustrasi definisi label

3.1.2. Fungsi SWAP

Fungsi SWAP ini merupakan fungsi yang merupakan interpretasi dari notasi S_w pada [2]. Fungsi $SWAP(L, i, j)$ merupakan fungsi yang menukar nilai dari node dengan id i dengan nilai dari node dengan id j pada label L yang akan menghasilkan L'. Ilustrasi fungsi swap dapat dilihat pada Gambar 3.2.



Gambar 3.2: Ilustrasi fungsi swap

3.1.3. Graf dari Kondisi Label

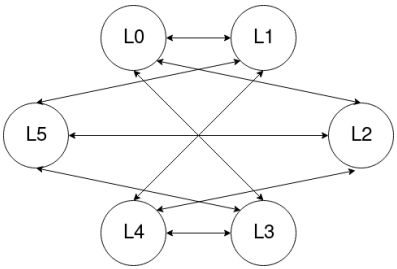
Sebuah label L akan menghasilkan beberapa L' melalui proses $SWAP(L, i, j)$ dimana i dan j merupakan semua kemungkinan anggota L. Hal serupa dapat dilakukan pada L' untuk menghasilkan L'' dan berlaku seterusnya. Hal ini dapat menghasilkan sebuah graf dimana *node* dari graf tersebut merupakan kondisi (*state*) dari L. Graf ini memiliki *node* berjumlah $n!$, hal ini sesuai dengan total seluruh kemungkinan kondisi L. Setiap *node* pada graf ini memiliki $(n*(n-1))/2$ *edge* dimana nilai tersebut merupakan kemungkinan nilai i dan j pada L yang menghasilkan L' yang berbeda. Perlu diketahui bahwa graf ini belum terikat terhadap bentuk *tree* yang akan diberi label. Data seluruh transisi L dengan jumlah *node* 3 dapat dilihat pada Tabel 3.1 dan ilustrasi graf dari kondisi L menggunakan data dari Tabel 3.1 dapat dilihat pada Gambar 3.3.

L	SWAP(L,i,j)	L'
L0(0,1,2)	SWAP(L0,1,2)	L1(1,0,2)
L0(0,1,2)	SWAP(L0,1,3)	L2(2,1,0)
L0(0,1,2)	SWAP(L0,2,3)	L3(0,2,1)
L1(1,0,2)	SWAP(L1,1,2)	L0(0,1,2)
L1(1,0,2)	SWAP(L1,1,3)	L4(2,0,1)
L1(1,0,2)	SWAP(L1,2,3)	L5(1,2,0)
L2(2,1,0)	SWAP(L2,1,2)	L5(1,2,0)
L2(2,1,0)	SWAP(L2,1,3)	L0(0,1,2)
L2(2,1,0)	SWAP(L2,2,3)	L4(2,0,1)
L3(0,2,1)	SWAP(L3,1,2)	L4(2,0,1)
L3(0,2,1)	SWAP(L3,1,3)	L5(1,2,0)
L3(0,2,1)	SWAP(L3,2,3)	L0(0,1,2)
L4(2,0,1)	SWAP(L4,1,2)	L3(0,2,1)
L4(2,0,1)	SWAP(L4,1,3)	L1(1,0,2)
L4(2,0,1)	SWAP(L4,2,3)	L2(2,1,0)
L5(1,2,0)	SWAP(L5,1,2)	L2(2,1,0)
L5(1,2,0)	SWAP(L5,1,3)	L3(0,2,1)
L5(1,2,0)	SWAP(L5,2,3)	L1(1,0,2)

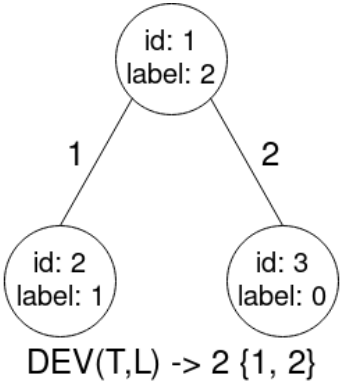
Tabel 3.1: Tabel data transisi L dengan 3 *node*

3.1.4. Fungsi DistinctEdgeValue

Fungsi $DistinctEdgeValue(T, L)$ atau $DEV(T, L)$ merupakan fungsi yang menghitung jumlah nilai *edge* yang unik dari *tree* T dengan label L dimana nilai *edge* adalah selisih dari nilai *node* yang dihubungkan. Fungsi ini merupakan interpretasi dari notasi $z(T, L)$ pada [2]. Ilustrasi fungsi $DEV(T, L)$ dengan tiga *node* dapat dilihat pada Gambar 3.4.



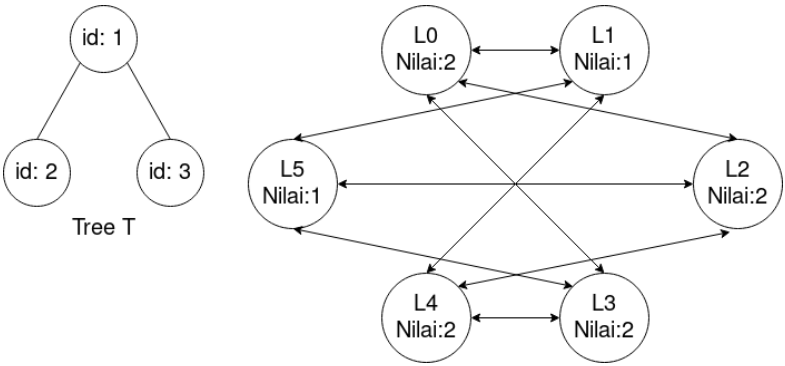
Gambar 3.3: Ilustrasi graf kondisi L dari Tabel 3.1



Gambar 3.4: Ilustrasi $DEV(T, L)$ dengan tiga *node*

3.1.5. Ruang Pencarian Solusi

Graf G merupakan graf hasil dari perubahan kondisi L melalui fungsi $SWAP(L, i, j)$ dimana *node* dari G merupakan kondisi dari L. *Node* pada G dengan *tree* T dapat diberi bobot nilai menggunakan fungsi $DEV(T, L)$. Ilustrasi untuk graf pada Gambar 3.3 setelah diberi bobot dengan fungsi $DEV(T, L)$ dapat dilihat pada Gambar 3.5. Graf G yang telah diberi bobot nilai oleh fungsi $DEV(T, L)$ ini digunakan sebagai ruang pencarian solusi dari permasalahan *graceful labelling* pada *general tree* dengan tujuan akhir mencari *node* dengan bobot bernilai $n - 1$.



Gambar 3.5: Ilustrasi graf pada Gambar 3.3 setelah diberi bobot menggunakan $DEV(T, L)$

3.1.6. Algoritma Solusi

Berdasarkan analisis yang telah dilakukan dan penelitian dari Aldred dan McKay[2] menghasilkan algoritma yang dapat digunakan untuk mencari *graceful labelling* pada *general tree*. Algoritma ini menggunakan metode *hill climbing* yang dibantu dengan metode *tabu search* untuk menelusuri ruang pencarian solusi hingga mendapatkan kondisi label yang diingingkan.

1. Mulai dengan L sembarang.
2. Jika DEV(T, L) == n-1, berhenti.
3. Untuk setiap kemungkinan i, j, tukar L dengan L' dari SWAP(L, i, j) apabila DEV(T, L') > DEV(T, L).
4. Jika langkah 3 berakhir tanpa ada perubahan L, tukar L dengan L' dari SWAP(L, i, j), dimana i, j dipilih secara acak dari semua kemungkinan i, j sedemikian rupa sehingga
   (a) i, j belum dipilih selama M pemilihan sebelumnya.
   (b) DEV(T, SWAP(L, i, j)) maksimal setelah memenuhi (a).
5. Ulangi dari langkah 2

3.2. Perancangan Sistem

3.2.1. Deskripsi Umum Sistem

Sistem akan menerima masukan yang terdiri dari jumlah lampu pada aula dan detail pemasangan lampu pada aula. Sistem akan melakukan proses pencarian solusi dari *tree* yang dibangun dari data aula. Kemudian, sistem akan mencetak hasil solusi yang telah didapatkan dari proses sebelumnya.

3.2.2. Desain Fungsi Utama

Fungsi ini menerima masukan berupa bilangan N yang menyatakan jumlah lampu pada aula. Kemudian, sistem akan membuat struktur *tree* awal yang terdiri dari N *node*. Sistem lalu menerima masukan yang berisi detail koneksi antar *node* pada *tree*, melakukan pencarian solusi dari *tree* kemudian mencetak solusi konfigurasi. *Pseuodocode* fungsi utama dapat dilihat pada Gambar 3.6

```
1   N = input()
2   tree = new TreeWithLabel(N)
3   inputTree(tree)
4   searchAnswer(tree)
5   tree.print()
```

Gambar 3.6: *Pseudocode* fungsi utama

3.2.3. Desain Fungsi Masukan Tree

Fungsi ini menerima masukan berupa $N - 1$ baris bilangan yang merepresentasikan data dari *edge* pada *tree*. Setiap baris masukan terdiri dari bilangan u dan v yang menyatakan bahwa *edge* tersebut menghubungkan *node* u dan *node* v. Fungsi ini akan menghubungkan setiap *node* u dan v pada *tree* yang telah dibuat. *Pseudocode* fungsi masukan *tree* dapat dilihat pada Gambar 3.7.

```
    inputTree(tree: TreeWithlabel): void
1   for i = 1 to tree->getNumberOfNode() - 1:
2     u = input()
3     v = input()
4     tree.connectNode(u, v)
```

Gambar 3.7: *Pseudocode* fungsi masukan *tree*

3.2.4. Desain Fungsi Pencarian Jawaban

Fungsi pencarian jawaban merupakan fungsi yang menjabarkan algoritma pada 3.1.6 dalam sistem. Fungsi ini akan menghasilkan label yang sesuai dengan ketentuan. Fungsi ini menggunakan M pada algoritma penyelesaian dengan nilai 15. *Pseudocode* fungsi pencarian jawaban dapat dilihat pada Gambar 3.8

```
    searchAnswer(tree: TreeWithlabel): void
1   tabu = new TabuLimit(15)
2   currentDistinctEdgeValue =
        tree->getDistinctEdgeValue()
3   while currentDistinctEdgeValue !=
        tree->getNumberOfNode() - 1:
4     maxDistinctEdgeValue = 0
5     foundBetterConfiguration = false
6     lastNode = tree->getNumberOfNode() - 1
7     for i = 0 to lastNode:
8       for j = i + 1 to lastNode:
9         actionId = (i * 100) + j
10        if (tabu->isAllowed == false):
11          continue
12        tree->swapNodeValue(i, j)
13        possibleDistinctValue =
              tree->getDistinctEdgeValue()
```

```
12          if possibleDistinctValue >
                currentDistinctEdgeValue:
13            foundBetterConfiguration = true
14            currentDistinctEdgeValue =
                  possibleDistinctValue
15            tabu.choose(actionId)
16            continue
17          if foundBetterConfiguration == false and
                possibleDistinctValue >
                maxDistinctEdgeValue:
18            maxDistinctEdgeValue =
                  possibleDistinctValue
19            possibleSwap = {i ,j}
20          tree->swapNodeValue(i, j)
21      if foundBetterConfiguration == false:
22        tree->swapNodeValue(possibleSwap[0],
              possibleSwap[1])
23        currentDistinctEdgeValue =
              tree->getDistinctEdgeValue()
24        tabu.choose(possibleSwap[0] * 100 +
              possibleSwap[1])
```

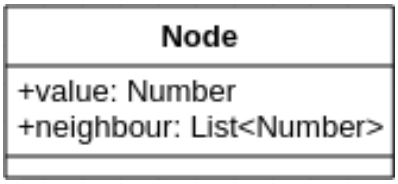
Gambar 3.8: *Pseudocode* Fungsi Pencarian Jawaban

3.2.5. Desain Kelas Node

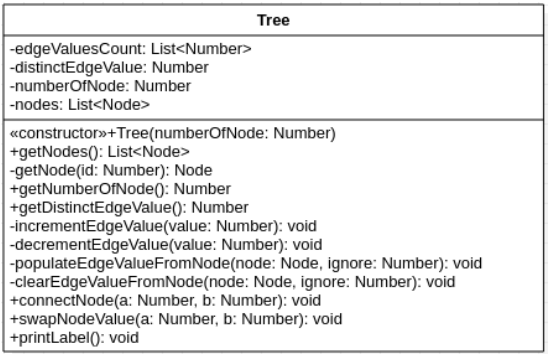
Kelas *node* merupkan kelas yang merepresentasikan data dari sebuah node. Sebuah node memiliki nilai label dan *node* lain yang terhubung dengannya. Desain kelas *node* dapat dilihat pada Gambar 3.9.

3.2.6. Desain Kelas Tree

Kelas *tree* ini merupakan kelas utama yang merupakan representasi dari sistem pencahayaan aula. Desain kelas *tree* dapat dilihat pada Gambar 3.10.



Gambar 3.9: Desain kelas *node*



Gambar 3.10: Desain kelas *tree*

3.2.6.1. Desain Konstruktor Kelas Tree

Konstruktor untuk kelas *tree* berfungsi menginisialisasi data yang diperlukan objek *tree*. Pertama yaitu mengisi jumlah *node* sesuai ukuran *tree*. Selanjutnya mengisi nilai awal *node* sesuai urutan pemasangan. *Pseudocode* dari konstruktor kelas *tree* dapat dilihat pada Gambar 3.11.

3.2.6.2. Desain Metode Mendapatkan Semua Node

Metode ini berfungsi untuk mendapatkan daftar *node* pada tree. *Pseudocode* dari metode ini dapat dilihat pada Gambar 3.12.

```
    Tree(numberOfNodeInput)
1   this->numberOfNode = numberOfNodeInput
2   this->distinctEdgeValue = 0
3   //Perulangan unuk inisialisasi data
        masing-masing node
4   for i = 0 to numberOfNodeInput - 1
5     this->edgeValuesCount[i] = 0
6     this->nodes[i]->value = i
```

Gambar 3.11: *Pseudocode* konstruktor kelas *tree*

```
    getNodes(): List of Node
1   return this->nodes
```

Gambar 3.12: *Pseudocode* metode mendapatkan semua *node*

3.2.6.3. Desain Metode Mendapatkan Sebuah Node

Metode ini berfungsi untuk mendapatkan *node* pada *tree* di posisi tertentu. *Pseudocode* dari metode ini dapat dilihat pada Gambar 3.13.

```
    getNode(i: Number): Node
1   return this->nodes[i]
```

Gambar 3.13: *Pseudocode* metode mendapatkan sebuah *node*

3.2.6.4. Desain Metode Mendapatkan Jumlah Node

Metode ini berfungsi untuk mendapatkan jumlah *node* pada *tree*. *Pseudocode* dari metode ini dapat dilihat pada Gambar 3.14.

```
    getNumberOfNode(): Number
1   return this->numberOfNode
```

Gambar 3.14: *Pseudocode* metode mendapatkan jumlah *node*

3.2.6.5. Desain Metode Mendapatkan Jumlah Nilai Edge yang Unik

Metode ini berfungsi untuk mendapatkan jumlah nilai *edge* yang unik. *Pseudocode* dari metode ini dapat dilihat pada Gambar 3.15.

```
    getDistinctEdgeValue(): Number
1   return this->distinctEdgeValue
```

Gambar 3.15: *Pseudocode* metode mendapatkan jumlah nilai *edge* yang unik

3.2.6.6. Desain Metode Menambah Jumlah Kemunculan Nilai Edge

Metode ini berfungsi untuk menambah data jumlah kemunculan nilai dari *edge*. Ketika menambah jumlah kemunculan sebuah nilai maka perlu dilakukan pengecekan apakah sebelumnya nilai ini sudah pernah muncul, apabila belum pernah muncul maka jumlah nilai *edge* yang unik juga ditambah. *Pseudocode* dari metode ini dapat dilihat pada Gambar 3.16.

3.2.6.7. Desain Metode Mengurangi Jumlah Kemunculan Nilai Edge

Metode ini berfungsi untuk mengurangi data jumlah kemunculan nilai dari *edge*. Setelah data kemunculan nilai dikurangi maka

```
    incrementEdgeValue(value: Number): void
1   \\ Jika nilai belum pernah muncul
2   if (this->edgeValuesCount[value] == 0):
3     \\ Tambahkan jumlah nilai unik
4     this->distinctEdgeValue++
5   this->edgeValuesCount[value]++
6   return
```

Gambar 3.16: *Pseudocode* metode menambah jumlah kemunculan nilai *edge*

dilakukan pengecekan apakah data kemunculan sekarang kosong, apabila kosong maka jumlah nilai *edge* yang unik juga dikurangi. *Pseudocode* dari metode ini dapat dilihat pada Gambar 3.17.

```
    decrementEdgeValue(value: Number): void
1   this->edgeValuesCount[value]--
2   \\ Apabila ini ini merupakan nilai terakhir
3   if (this->edgeValuesCount[value] == 0):
4     \\ Kurangi jumlah nilai yang unik
5     this->distinctEdgeValue--
```

Gambar 3.17: *Pseudocode* metode mengurangi jumlah kemunculan nilai *edge*

3.2.6.8. Desain Metode Mengisi Nilai Edge dari Node

Metode ini mengisi nilai dari semua *edge* yang terhubung dari sebuah *node* ke semua *node* tetangganya kecuali *node* yang dimasukan pada pengecualian. *Pseudocode* dari metode ini dapat dilihat pada Gambar 3.18.

```
    populateEdgeValueFromNode(node: Node, ignore:
        Number): void
1   \\ Perulangan untuk semua tetangga dari node
2   foreach(node.neighbour as neighbourId):
3     \\ Abaikan untuk tetangga dengan id bernilai
          sama dengan variabel ignore
4     if (neighbourId != ignore):
5       neighbour = this->getNode(neighbourId)
6       edgeValue = abs(node.value -
            neighbour.value)
7       this->incrementEdgeValue(edgeValue)
```

Gambar 3.18: *Pseudocode* metode mengisi nilai *edge* dari *node*

3.2.6.9. Desain Metode Menghapus Nilai Edge dari Node

Metode ini menghapus nilai dari semua *edge* yang terhubung dari sebuah *node* ke semua *node* tetangganya kecuali *node* yang dimasukan pada pengecualian. *Pseudocode* dari metode ini dapat dilihat pada Gambar 3.19.

```
    clearEdgeValueFromNode(node: Node, ignore:
        Number): void
1   \\ Perulangan untuk semua tetangga dari node
2   foreach(node.neighbour as neighbourId):
3     \\ Abaikan untuk tetangga dengan id bernilai
          sama dengan variabel ignore
4     if (neighbourId != ignore):
5       neighbour = this->getNode(neighbourId)
6       edgeValue = abs(node.value -
            neighbour.value)
7       this->decrementEdgeValue(edgeValue)
```

Gambar 3.19: *Pseudocode* metode menghapus nilai *edge* dari *node*

3.2.6.10. Desain Metode Menghubungkan Dua Node

Metode ini berfungsi untuk menghubungkan dua buah *node* dengan edge. Setelah kedua *node* telah terhubung maka akan dicari nilai dari *edge* kemudian data jumlah kemunculan nilai *edge* tersebut akan di ditambah. *Pseudocode* dari metode ini dapat dilihat pada Gambar 3.20.

```
    connectNode(a: Number, b: Number): void
1   \\ Mendapatkan node a dan b
2   nodeA = this->getNode(a)
3   nodeB = this->getNode(b)
4   \\ Mendaftarkan node a sebagai tetangga b dan
        sebaliknya
5   nodeA.neighbour.insert(b)
6   nodeB.neighbour.insert(a)
7   \\ Menghitung nilai dari edge
8   edgeValue = abs(nodeA.value - nodeB.value)
9   this->incrementEdgeValue(edgeValue)
```

Gambar 3.20: *Pseudocode* metode menghubungkan dua *node*

3.2.6.11. Desain Metode Menukar Nilai dari Dua Node

Metode ini akan menghapus nilai *edge* yang terhubung pada kedua *node* sebelum menukar nilai dari dua *node*. Setelah nilai dua *node* tersebut ditukar, selanjutnya metode ini akan mengisi kembali nilai *edge* yang terhubung pada kedua *node*. *Pseudocode* dari metode ini dapat dilihat pada Gambar 3.21.

3.2.6.12. Desain Metode Mencetak Label pada Tree

Metode ini berfungsi untuk mencetak label pada *tree* secara terurut berdasarkan urutan *node* sesuai format yang ditentukan. *Pseudocode* dari metode ini dapat dilihat pada Gambar 3.22.

```
    swapNodeValue(a: Number, b: Number): void
 1  \\ Mendapatkan node a dan b
 2  nodeA = this->getNode(a)
 3  nodeB = this->getNode(b)
 4  \\ Menghapus semua nilai edge yang terhubung
        pada node a dan node b
 5  this->clearEdgeValueFromNode(nodeA, -1)
 6  this->clearEdgeValueFromNode(nodeB, a)
 7  \\ Menukar nilai node a dan node b
 8  swap(nodeA->value, nodeB->value)
 9  \\ Mengisi nilai edge yang terhubung pada node
        a dan node b
10  this->populateEdgeValueFromNode(nodeA, -1)
11  this->populateEdgeValueFromNode(nodeB, a)
```

Gambar 3.21: *Pseudocode* metode menukar nilai dari dua *node*

3.2.7. Desain Kelas Batasan Tabu

Kelas batasan tabu digunakan untuk memudahkan penyimpanan data tabu berupa jumlah langkah, batasan tabu, dan aksi yang dibatasi tabu. Kelas ini memiliki dua metode yaitu memilih aksi dan mengecek kebolehan aksi. Desain kelas batasan tabu dapat dilihat pada Gambar 3.23.

3.2.7.1. Desain Konstruktor Batasan Tabu

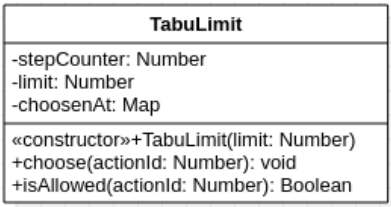
Konstruktor untuk kelas batasan tabu berfungsi untuk menginisialisasi nilai batasan dan jumlah langkah awal agar kelas ini dapat digunakan. *Pseudocode* dari konstruktor kelas batasan tabu dapat dilihat pada Gambar 3.24.

3.2.7.2. Desain Metode Memilih Aksi

Metode ini berfungsi untuk mencatat langkah saat aksi tersebut dilakukan, selanjutnya nilai langkah akan ditambah. *Pseudo-*

```
    printLabel(): void
1   \\ Perulangan untuk semua node
2   for (i from 0 to this->getNumberOfNode() - 1):
3     \\ Apabila bukan node pertama
4     if (i != 0):
5       \\ Cetak spasi antar node
6       print(' ')
7     print(this->getNode(i).value)
8   print('\n')
```

Gambar 3.22: *Pseudocode* metode mencetak label pada *tree*



Gambar 3.23: Desain kelas batasan tabu

code untuk metode ini dapat dilihat pada Gambar 3.25.

3.2.7.3. Desain Mengecek Kebolehan Aksi

Metode ini mengecek kebolehan suatu aksi dengan cara menghitung selisih dari nilai langkah sekarang dengan langkah saat aksi tersebut dipilih, jika selisihnya kurang dari limit maka aksi tersebut dibolehkan. *Pseudocode* dari metode ini dapat dilihat pada Gambar 3.26.

```
    TabuLimit(limit)
1   this->limit = limit
2   this->stepCounter = limit + 1
```

Gambar 3.24: *Pseudocode* konstruktor kelas batasan tabu

```
    choose(actionId: Number): void
1   choosenAt[actionId] = this->stepCounter
2   this->stepCounter++
```

Gambar 3.25: *Pseudocode* metode memilih aksi

```
    isAllowed(actionId: Number): Boolean
1   stepDifference = this->stepCounter -
        this->choosenAt[actionId]
2   return (stepCounter > this->limit)
```

Gambar 3.26: *Pseudocode* metode mengecek kebolehan aksi

[*Halaman ini sengaja dikosongkan*]

BAB IV
IMPLEMENTASI

4.1. Lingkungan Implementasi

Lingkungan implementasi dan pengembangan menggunakan sebuah komputer dengan spesifikasi perangkat lunak dan perangkat keras sebagai berikut.

1. Perangkat Keras
   (a) Processor Intel® Core™ i5-3320M CPU @ 2.60GHz
   (b) Random Access Memory 4GB
2. Perangkat Lunak
   (a) Sistem Operasi Manjaro Illyria 18.0 64-bit
   (b) Text Editor Visual Studio Code 1.29.1
   (c) Bahasa Pemrograman C++
   (d) g++ (GCC) 8.2.1 20181127

4.2. Implementasi Kelas Node

Subbab ini menjabarkan implementasi dari kelas *Node* yang dijelaskan pada subbab 3.2.5. Implementasi kelas *Node* dapat dilihat pada Kode Sumber 4.1.

```
    class Node
1     public:
2         int value;
3         vector<int> neighbour;
```

Kode Sumber 4.1: Implementasi Kelas *Node*

4.3. Implementasi Kelas Tree

Subbab ini menjabarkan implementasi dari kelas *Tree* yang dijelaskan pada subbab 3.2.6. Kelas ini menyimpan berbagai data dan operasi yang digunakan dalam sistem. Implementasi kelas *Tree* dapat dilihat pada Kode Sumber 4.2.

```
     class Tree
 1   {
 2     private:
 3     int edgeValuesCount[30];
 4     int distinctEdgeValue;
 5     int numberOfNode;
 6     vector<Node *> *nodes;
 7
 8     void incrementEdgeValue(int value)
 9     {
10       if (!this->edgeValuesCount[value]) {
11         ++this->distinctEdgeValue;
12       }
13       ++this->edgeValuesCount[value];
14     }
15
16     void decrementEdgeValue(int value)
17     {
18       this->edgeValuesCount[value]--;
19       if (!this->edgeValuesCount[value])
20       {
21         this->distinctEdgeValue--;
22         return;
23       }
24     }
25
26     Node *getNode(int id)
27     {
28       return this->nodes->at(id);
29     }
```

```
32      void populateEdgeValueFromNode(Node *node,
            int ignore)
33      {
34        for (vector<int>::iterator it =
              node->neighbour.begin(); it !=
              node->neighbour.end(); ++it)
35        {
36          if (*it == ignore)
37            continue;
38          Node *otherNode = this->getNode(*it);
39          this->incrementEdgeValue(getEdgeValue(node,
                otherNode));
40        }
41      }
42
43      void clearEdgeValueFromNode(Node *node, int
            ignore)
44      {
45        for (vector<int>::iterator it =
              node->neighbour.begin(); it !=
              node->neighbour.end(); ++it)
46        {
47          if (*it == ignore)
48            continue;
49          Node *otherNode = this->getNode(*it);
50          this->decrementEdgeValue(getEdgeValue(node,
                otherNode));
51        }
52      }
53
54      public:
55      Tree(int numberOfNodeInp){
56        this->numberOfNode = numberOfNodeInp;
57        memset(this->edgeValuesCount, 0,
              sizeof(this->edgeValuesCount));
58        distinctEdgeValue = 0;
59        this->nodes = new vector<Node
              *>(numberOfNodeInp, nullptr);
60        for (int i = 0; i < numberOfNodeInp; ++i)
```

```
62        {
63          Node *newNode = new Node();
64          newNode->value = i;
65          newNode->neighbour = vector<int>();
66          nodes->at(i) = newNode;
67        }
68      }
69
70      vector<Node*>* getNodes(){
71        return this->nodes;
72      }
73
74      int getNumberOfNode(){
75        return this->numberOfNode;
76      }
77
78      void connectNode(int a, int b)
79      {
80        Node *nodeA, *nodeB;
81        nodeA = this->getNode(a);
82        nodeB = this->getNode(b);
83        nodeA->neighbour.push_back(b);
84        nodeB->neighbour.push_back(a);
85        int edgeValue = getEdgeValue(nodeA, nodeB);
86        this->incrementEdgeValue(edgeValue);
87      }
88
89      int countDistinctEdgeValue()
90      {
91        return this->distinctEdgeValue;
92      }
93
94      void swapNodeValue(int a, int b)
95      {
96        Node *nodeA, *nodeB;
97        nodeA = this->nodes->at(a);
98        nodeB = this->nodes->at(b);
99        this->clearEdgeValueFromNode(nodeA, -1);
100       this->clearEdgeValueFromNode(nodeB, a);
```

```
102       int temp = nodeA->value;
103       nodeA->value = nodeB->value;
104       nodeB->value = temp;
105       this->populateEdgeValueFromNode(nodeA, -1);
106       this->populateEdgeValueFromNode(nodeB, a);
107       return;
108     }
109
110     void print(){
111       for (int i = 0; i < numberOfNode; ++i)
112       {
113         if (i)
114           putchar('␣');
115         printf("%d", nodes->at(i)->value);
116       }
117       putchar('\n');
118     }
119   };
```

Kode Sumber 4.2: Implementasi Kelas *Tree*

4.4. Implementasi Kelas Tabu

Subbab ini menjabarkan implementasi dari kelas batasan tabu yang dijelaskan pada subbab 3.2.7. Kelas ini berfungsi untuk mencatat data langkah tabu dan aksi yang dibatasi oleh tabu. Implementasi kelas batasan tabu dapat dilihat pada Kode Sumber 4.3.

```
    class TabuLimit
1     private:
2         int stepCounter;
3         int limit;
4         int choosenAt[2800];
```

```
 6      public:
 7          TabuLimit(int limit): limit(limit),
                stepCounter(limit + 1)
 8          {
 9              memset(choosenAt, 0,
                    sizeof(choosenAt));
10          }
11
12          void choose(int actionId) {
13              choosenAt[actionId] = stepCounter;
14              ++stepCounter;
15          }
16
17          bool isAllowed(int actionId) {
18              return (stepCounter -
                    choosenAt[actionId]) > limit;
19          }
```

Kode Sumber 4.3: Implementasi Kelas Batasan Tabu

4.5. Implementasi Fungsi Utama

Subbab ini menjabarkan implementasi dari fungsi utama yang dijelaskan pada subbab 3.2.2. Implementasi fungsi ini dapat dilihat pada Kode Sumber 4.4.

```
     void solution()
 1   int numberOfNode;
 2   scanf("%d", &numberOfNode);
 3   Tree tree(numberOfNode);
 4   inputTree(&tree);
 5   searchAnswer(&tree);
 6   tree.print();
 7   return;
```

Kode Sumber 4.4: Implementasi Fungsi Utama

4.6. Implementasi Fungsi Masukan Tree

Subbab ini menjabarkan implementasi dari fungsi masukan *tree* yang dijelaskan pada subbab 3.2.3. Implementasi fungsi ini dapat dilihat pada Kode Sumber 4.5.

```
    void inputTree(Tree *tree)
1     for (int i = 1; i < tree->getNumberOfNode();
          ++i)
2     {
3         int u, v;
4         scanf("%d%d", &u, &v);
5         --u;
6         --v;
7         tree->connectNode(u, v);
8     }
```

Kode Sumber 4.5: Implementasi Fungsi Masukan *Tree*

4.7. Implementasi Fungsi Pencarian Jawaban

Subbab ini menjabarkan implementasi dari fungsi pencarian jawaban yang dijelaskan pada subbab 3.2.4. Implementasi fungsi ini dapat dilihat pada Kode Sumber 4.6.

```
    void searchAnswer(Tree *tree)
1     TabuLimit tabu(15);
2     int currentDistinctEdgeValue =
          tree->countDistinctEdgeValue();
3     while (currentDistinctEdgeValue <
          tree->getNumberOfNode() - 1)
4     {
5         pair<int, int> possibleSwap;
6         int maxDistinctEdgeValue = 0;
7     bool foundBetterConfiguration = false;
```

```
 8          for (int i = 0; i <
                tree->getNumberOfNode(); ++i)
 9          {
10              for (int j = i + 1; j <
                    tree->getNumberOfNode(); ++j)
11              {
12                  int actionId = (i * 100) + j;
13                  if (!tabu.isAllowed(actionId))
14                  {
15                      continue;
16                  }
17                  tree->swapNodeValue(i, j);
18                  int possibleDistinctEdgeValue =
                        tree->countDistinctEdgeValue();
19                  if (possibleDistinctEdgeValue >
                        currentDistinctEdgeValue)
20                  {
21                      foundBetterConfiguration =
                            true;
22                      currentDistinctEdgeValue =
                            possibleDistinctEdgeValue;
23                      tabu.choose(actionId);
24                      continue;
25          }
26                  if (
27                      (!foundBetterConfiguration) &&
28                      possibleDistinctEdgeValue >
                            maxDistinctEdgeValue)
29                  {
30                      maxDistinctEdgeValue =
                            possibleDistinctEdgeValue;
31                      possibleSwap = make_pair(i,
                            j);
32                  }
33                  tree->swapNodeValue(i, j);
34              }
35      }
```

```
36          if (!foundBetterConfiguration)
37          {
38              pair<int, int> choosenSwap =
                    possibleSwap;
39              tree->swapNodeValue(choosenSwap.first,
                    choosenSwap.second);
40              currentDistinctEdgeValue =
                    tree->countDistinctEdgeValue();
41              tabu.choose((choosenSwap.first * 100)
                    + choosenSwap.second);
42          }
43      }
```

Kode Sumber 4.6: Implementasi Fungsi Masukan *Tree*

[*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 keras dan perangkat lunak sebagai berikut.

1. Perangkat Keras
   (a) Processor Intel® Core™ i5-3320M CPU @ 2.60GHz
   (b) Random Access Memory 4GB
2. Perangkat Lunak
   (a) Sistem Operasi Manjaro Illyria 18.0 64-bit
   (b) Text Editor Visual Studio Code 1.29.1
   (c) Bahasa Pemrograman C++
   (d) g++ (GCC) 8.2.1 20181127

5.2. Uji Coba Kebenaran

Subbab ini menjelaskan pengujian program untuk penyelesaian permasalahan SPOJ Colorful Lights Party (PT07G) . Uji coba dilakukan untuk mendapatkan status *Accepted* pada situs penilaian daring SPOJ dengan cara mengirim program ke SPOJ Colorful Lights Party (PT07G) [4].

5.2.1. Uji Coba Kebenaran Lokal

Uji coba lokal dilakukan dengan menjalankan program untuk menguji apakah program dapat berjalan tanpa *error* dan menghasilkan *graceful labelling* sesuai data masukan yang diberikan.

5.2.1.1. Data Uji Coba Kebenaran Lokal

Data yang digunakan untuk uji coba kebenaran lokal adalah data *general tree* yang dibuat pada situs *SPOJToolkit*[13]. Terdapat 27 data yang masing-masing berisi data *general tree* mulai dari satu *node* hingga 27 *node*.

5.2.1.2. Skenario Uji Coba Kebenaran Lokal

Uji coba dilakukan dengan menjalankan program dan melihat apakah program dapat berjalan tanpa *error* dan dapat menghasilkan *graceful labelling* dari data masukan.

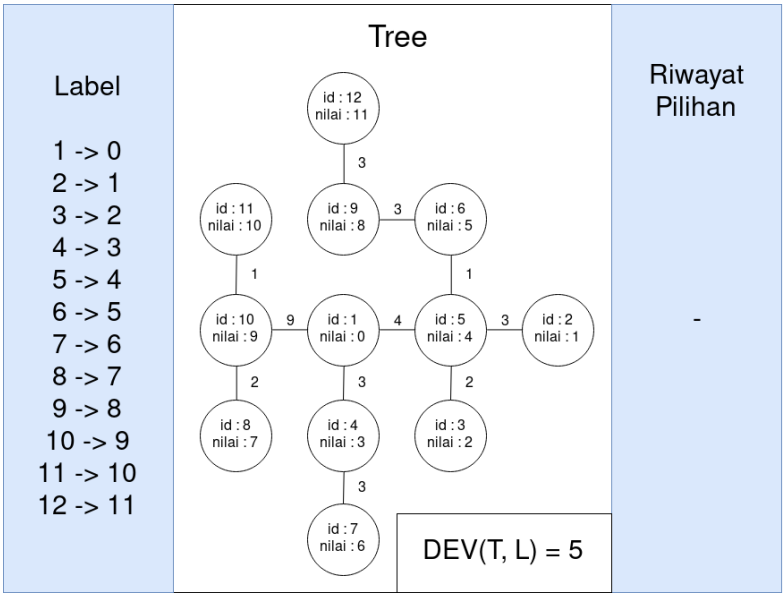
5.2.1.3. Evaluasi Kebenaran Uji Coba Lokal

Evaluasi dilakukan dengan mengecek label yang didapat dari hasil program. Keluaran program akan dianggap benar apabila label yang dihasilkan program sesuai dengan ketentuan yang diberikan yaitu bersifat *graceful* terhadap *tree* masukan. Hasil uji coba dengan 27 masukan dapat dilihat pada Lampiran A.

Dari hasil pengujian diatas membuktikan bahwa program mampu menghasilkan *graceful labelling* untuk *tree* masukan. Contoh salah satu proses pencarian solusi untuk input dengan jumlah *node* 12 dapat dilihat pada Gambar 5.1 - Gambar 5.11.

Gambar 5.1 menunjukan kondisi awal *program* yaitu nilai label awal berupa nilai *node* dikurangi satu dan belum ada langkah pada riwayat langkah. Pada contoh ini jumlah nilai *edge* awal yang unik bernilai 5.

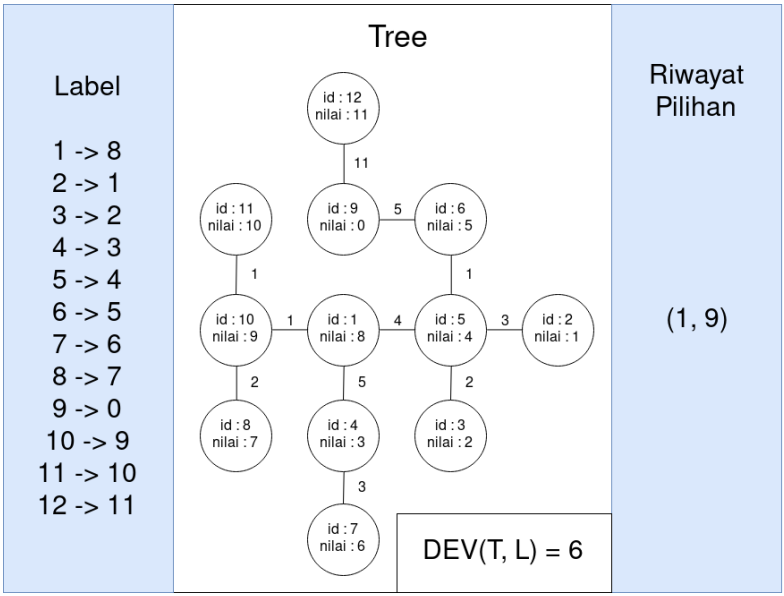
Proses pencarian kondisi tetangga selanjutnya menemukan bahwa dengan melakukan proses $SWAP(L, 1, 9)$ akan memiliki nilai yang lebih baik dari sebelumnya yaitu 6, maka program menukar nilai dari *node* dengan id 1 dan *node* dengan id 9. Program juga mencatat bahwa langkah penukaran antara *node* dengan id 1 dan *node* dengan id 9 telah dipilih saat ini. Kondisi setelah proses penukaran pertama dapat dilihat pada Gambar 5.2.



Gambar 5.1: Kondisi awal program

Proses pencarian kondisi tetangga selanjutnya menemukan bahwa dengan melakukan proses $SWAP(L, 1, 11)$ akan memiliki nilai yang lebih baik dari sebelumnya yaitu 7, maka program menukar nilai dari *node* dengan id 1 dan *node* dengan id 11. Program juga mencatat bahwa langkah penukaran antara *node* dengan id 1 dan *node* dengan id 11 telah dipilih saat ini. Kondisi setelah proses penukaran kedua dapat dilihat pada Gambar 5.3.

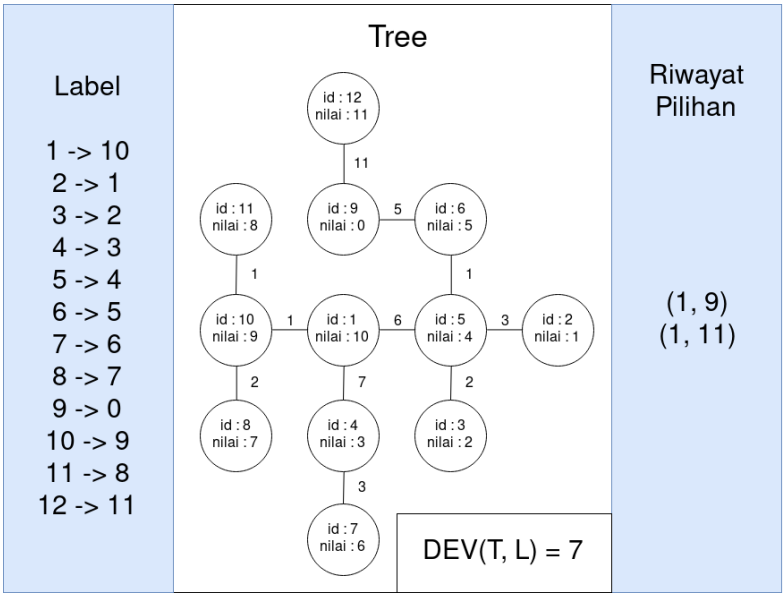
Proses pencarian kondisi tetangga selanjutnya menemukan bahwa dengan melakukan proses $SWAP(L, 2, 5)$ akan memiliki nilai yang lebih baik dari sebelumnya yaitu 8, maka program menukar nilai dari *node* dengan id 2 dan *node* dengan id 5. Program juga mencatat bahwa langkah penukaran antara *node* dengan id 2 dan *node* dengan id 5 telah dipilih saat ini. Kondisi setelah proses penukaran ketiga dapat dilihat pada Gambar 5.4.



Gambar 5.2: Kondisi setelah penukaran pertama

Proses pencarian kondisi tetangga selanjutnya menemukan bahwa dengan melakukan proses $SWAP(L, 2, 10)$ akan memiliki nilai yang lebih baik dari sebelumnya yaitu 9, maka program menukar nilai dari *node* dengan id 2 dan *node* dengan id 10. Program juga mencatat bahwa langkah penukaran antara *node* dengan id 2 dan *node* dengan id 10 telah dipilih saat ini. Kondisi setelah proses penukaran keempat dapat dilihat pada Gambar 5.5.

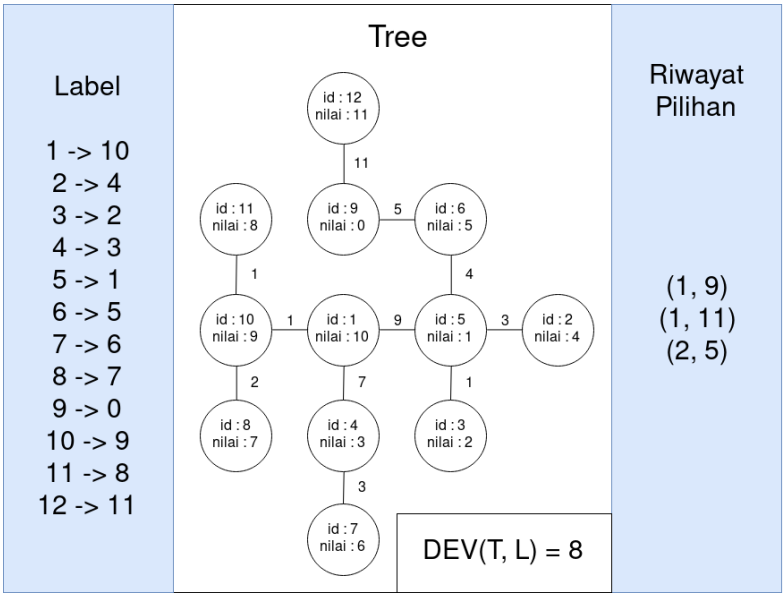
Proses pencarian kondisi tetangga selanjutnya menemukan bahwa dengan melakukan proses $SWAP(L, 6, 7)$ akan memiliki nilai yang lebih baik dari sebelumnya yaitu 10, maka program menukar nilai dari *node* dengan id 6 dan *node* dengan id 7. Program juga mencatat bahwa langkah penukaran antara *node* dengan id 6 dan *node* dengan id 7 telah dipilih saat ini. Kondisi setelah proses penukaran kelima dapat dilihat pada Gambar 5.6.



Gambar 5.3: Kondisi setelah penukaran kedua

Proses pencarian kondisi tetangga selanjutnya tidak menemukan kondisi yang memiliki nilai lebih baik dari kondisi sekarang. Program memilih menukar nilai dari *node* dengan id 2 dan *node* dengan id 3 dikarenakan nilai kondisi hasil dari $SWAP(L, 2, 3)$ menghasilkan nilai yang maksimal diantara semua kemungkinan tetangga kondisi sekarang yaitu 10 dan langkah $(2, 3)$ belum dipilih dalam 15 langkah sebelumnya. Program juga mencatat bahwa langkah penukaran antara *node* dengan id 2 dan *node* dengan id 3 telah dipilih saat ini. Kondisi setelah proses penukaran keenam dapat dilihat pada Gambar 5.7.

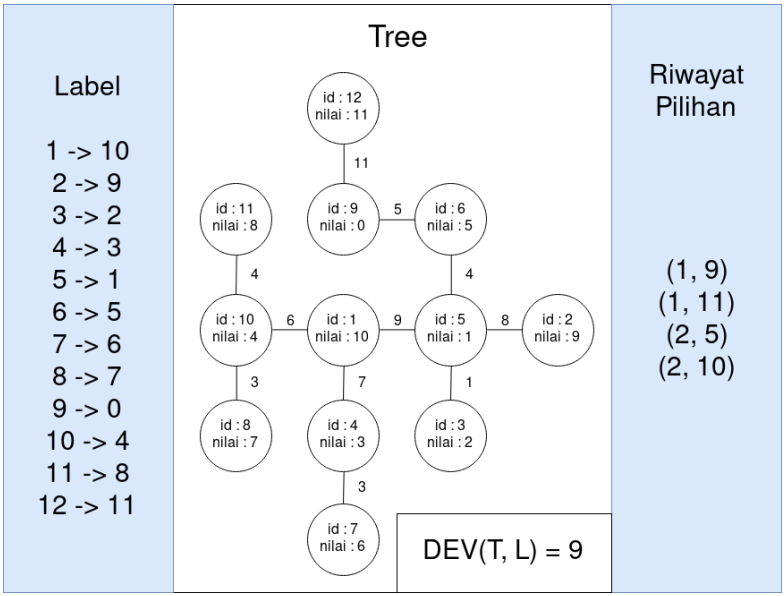
Proses pencarian kondisi tetangga selanjutnya tidak menemukan kondisi yang memiliki nilai lebih baik dari kondisi sekarang. Program memilih menukar nilai dari *node* dengan id 2 dan *node* dengan id 6 dikarenakan nilai kondisi hasil dari $SWAP(L, 2, 6)$



Gambar 5.4: Kondisi setelah penukaran ketiga

menghasilkan nilai yang maksimal diantara semua kemungkinan tetangga kondisi sekarang yaitu 10 dan langkah $(2, 6)$ belum dipilih dalam 15 langkah sebelumnya. Program juga mencatat bahwa langkah penukaran antara *node* dengan id 2 dan *node* dengan id 6 telah dipilih saat ini. Kondisi setelah proses penukaran ketujuh dapat dilihat pada Gambar 5.8.

Proses pencarian kondisi tetangga selanjutnya tidak menemukan kondisi yang memiliki nilai lebih baik dari kondisi sekarang. Program memilih menukar nilai dari *node* dengan id 3 dan *node* dengan id 6 dikarenakan nilai kondisi hasil dari $SWAP(L, 3, 6)$ menghasilkan nilai yang maksimal diantara semua kemungkinan tetangga kondisi sekarang yaitu 10 dan langkah $(3, 6)$ belum dipilih dalam 15 langkah sebelumnya. Program juga mencatat bahwa langkah penukaran antara *node* dengan id 3 dan *node* dengan id 6 telah

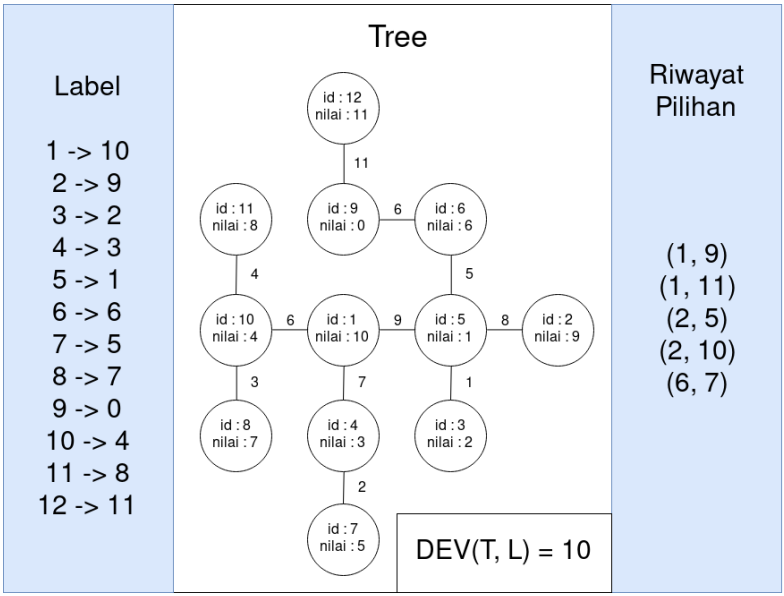


Gambar 5.5: Kondisi setelah penukaran keempat

dipilih saat ini. Kondisi setelah proses penukaran kedelapan dapat dilihat pada Gambar 5.9.

Proses pencarian kondisi tetangga selanjutnya tidak menemukan kondisi yang memiliki nilai lebih baik dari kondisi sekarang. Program memilih menukar nilai dari *node* dengan id 1 dan *node* dengan id 6 dikarenakan nilai kondisi hasil dari $SWAP(L, 1, 6)$ menghasilkan nilai yang maksimal diantara semua kemungkinan tetangga kondisi sekarang yaitu 10 dan langkah $(1, 6)$ belum dipilih dalam 15 langkah sebelumnya. Program juga mencatat bahwa langkah penukaran antara *node* dengan id 1 dan *node* dengan id 6 telah dipilih saat ini. Kondisi setelah proses penukaran kesembilan dapat dilihat pada Gambar 5.10.

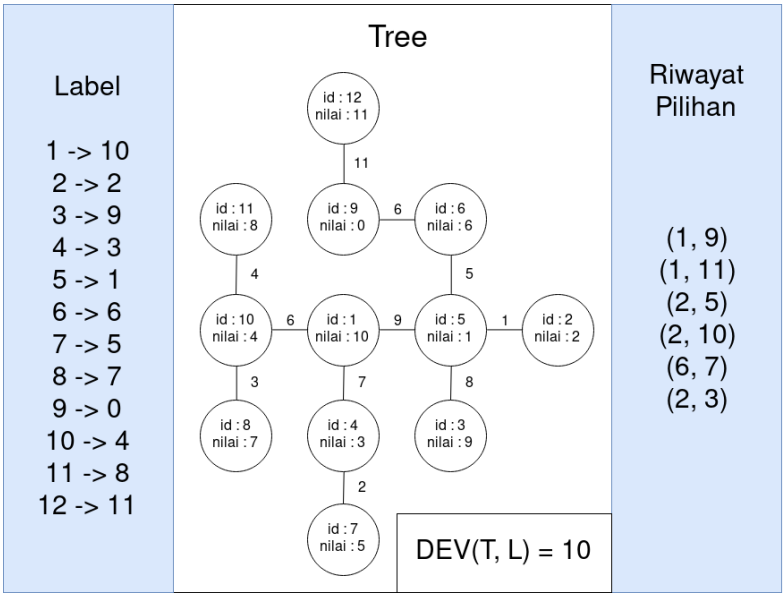
Proses pencarian kondisi tetangga selanjutnya tidak menemukan kondisi yang memiliki nilai lebih baik dari kondisi sekarang.



Gambar 5.6: Kondisi setelah penukaran kelima

Program memilih menukar nilai dari *node* dengan id 2 dan *node* dengan id 11 dikarenakan nilai kondisi hasil dari $SWAP(L, 2, 11)$ menghasilkan nilai yang maksimal diantara semua kemungkinan tetangga kondisi sekarang yaitu 10 dan langkah $(2, 11)$ belum dipilih dalam 15 langkah sebelumnya. Program juga mencatat bahwa langkah penukaran antara *node* dengan id 2 dan *node* dengan id 11 telah dipilih saat ini. Kondisi setelah proses penukaran kesepuluh dapat dilihat pada Gambar 5.11.

Proses pencarian kondisi tetangga selanjutnya menemukan bahwa dengan melakukan proses $SWAP(L, 3, 7)$ akan memiliki nilai yang lebih baik dari sebelumnya yaitu 11, maka program menukar nilai dari *node* dengan id 3 dan *node* dengan id 7. Program juga mencatat bahwa langkah penukaran antara *node* dengan id 3 dan *node* dengan id 7 telah dipilih saat ini. Kondisi setelah proses



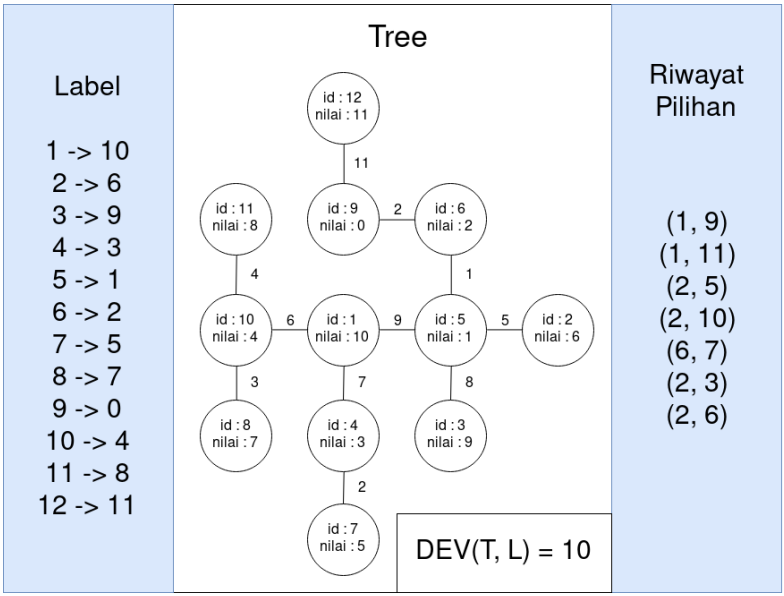
Gambar 5.7: Kondisi setelah penukaran keenam

penukaran kesebelas dapat dilihat pada Gambar 5.12.

Setelah proses penukaran kesebelas kondisi sekarang telah memiliki 11 nilai *edge* yang unik maka kondisi label sekarang telah memenuhi ketentuan *graceful labelling* yang diinginkan. Hal ini menunjukan program berhasil menyelesaikan permasalahan khususnya pada masukan dengan *tree* dengan *node* berjumlah 12.

5.2.2. Uji Coba Kebenaran Situs Penilaian Daring SPOJ

Uji coba pada situs penilaian daring SPOJ dilakukan dengan mengirim kode sumber ke situs SPOJ Colorful Lights Party (PT07G) [4].



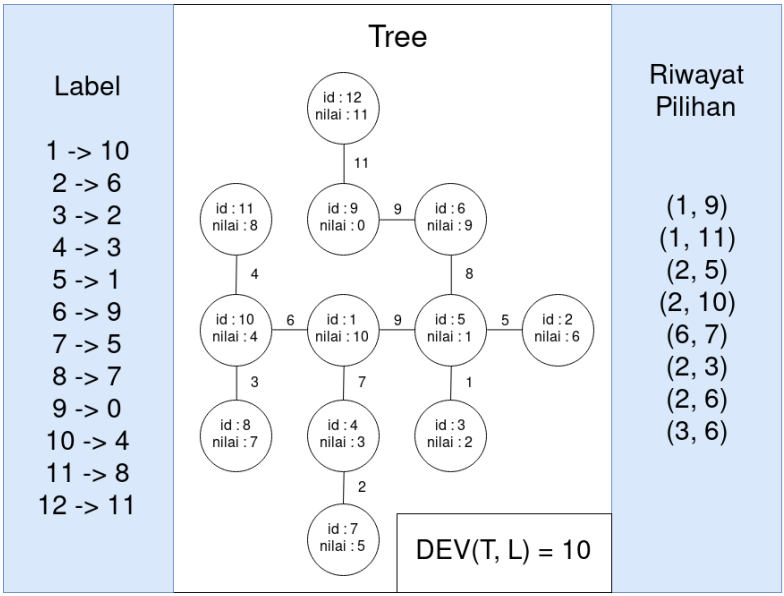
Gambar 5.8: Kondisi setelah penukaran ketujuh

5.2.2.1. Data Uji Coba Kebenaran Situs Penilaian Daring SPOJ

Data yang digunakan pada uji coba ini adalah data yang ada pada situs SPOJ Colorful Lights Party (PT07G) [4] sehingga data masukan yang digunakan untuk pengujian tidak dapat diketahui.

5.2.2.2. Evaluasi Uji Coba Kebenaran Situs Penilaian Daring SPOJ

Evaluasi dilakukan dengan melihat status pengumpulan yang didapatkan pada SPOJ Colorful Lights Party (PT07G) [4]. Hasil uji kebenaran situs penilaian daring SPOJ dapat dilihat pada Gambar 5.13



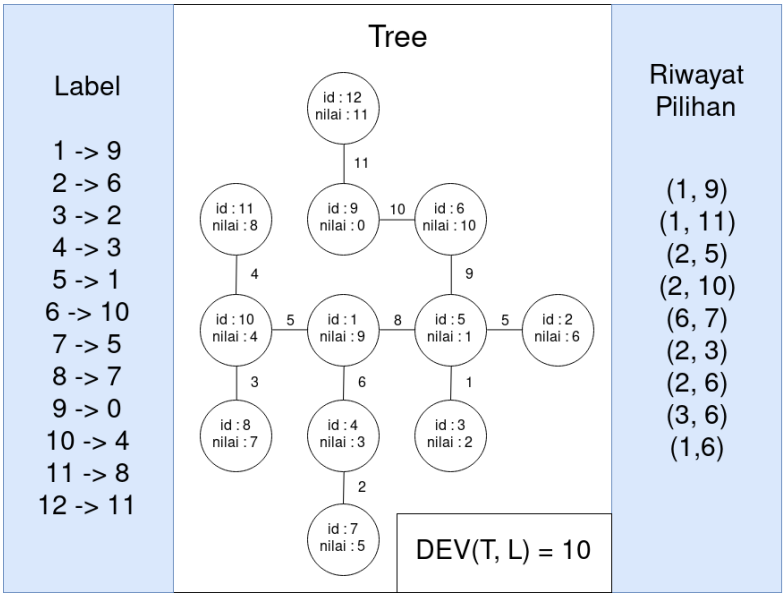
Gambar 5.9: Kondisi setelah penukaran kedelapan

5.3. Uji Coba Kinerja

Subbab ini menjelaskan pengujian kinerja program pada Tugas Akhir ini untuk menyelesaikan permasalahan *graceful labelling* pada *general tree*. Uji coba kinerja dilakukan untuk mengetahui gambaran kinerja program pada tugas akhir ini dan mendapatkan peringkat terbaik pada situs penilaian daring SPOJ.

5.3.1. Uji Coba Kinerja Lokal

Uji coba kinerja lokal dilakukan dengan menjalankan program pada tugas akhir ini dan program yang menggunakan algoritma naif untuk mendapatkan perbandingan kinerja diantara kedua program.



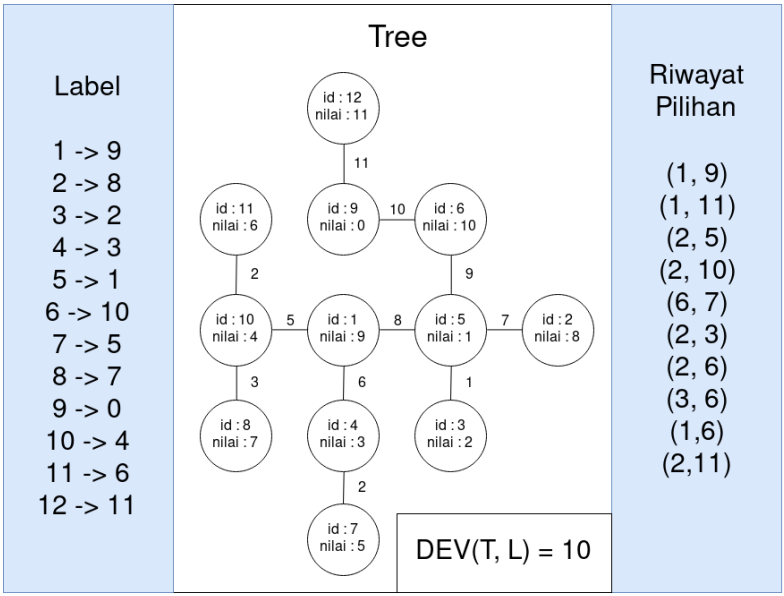
Gambar 5.10: Kondisi setelah penukaran kesembilan

5.3.1.1. Data Uji Coba Kinerja Lokal

Data yang digunakan untuk uji coba kinerja adalah data *general tree* yang dibuat pada situs SPOJ Colorful Lights Party (PT07G) [13]. Terdapat 100 data masing-masing 10 data untuk *general tree* dengan jumlah *node* satu hingga 10.

5.3.1.2. Skenario Uji Coba Kinerja Lokal

Uji coba dilakukan dengan menjalankan program pada Tugas Akhir ini dan program dengan algoritma naif kemudian dilakukan pencatatan waktu yang dibutuhkan program untuk menyelesaikan permasalahan.



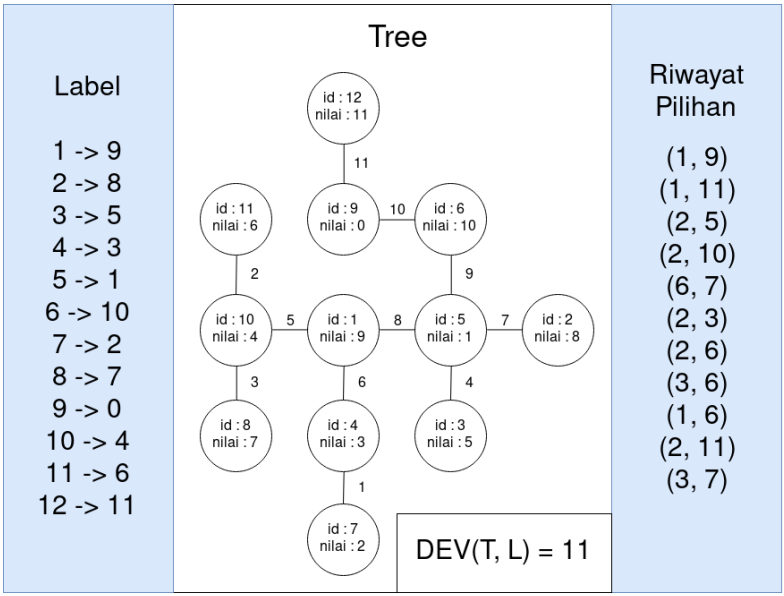
Gambar 5.11: Kondisi setelah penukaran kesepuluh

5.3.1.3. Evaluasi Uji Coba Kinerja Lokal

Evaluasi dilakukan dengan membandingkan waktu yang dibutuhkan masing-masing program untuk menyelesaikan permasalahan. Waktu yang ditampilkan merupakan rata-rata waktu yang dibutuhkan untuk *general tree* dengan masing-masing jumlah *node* dari satu hingga 10. Data hasil uji coba kinerja dapat dilihat pada Tabel 5.1 dan grafik hasil uji coba kinerja dapat dilihat pada Gambar 5.14.

5.3.2. Uji Coba Kinerja Situs Penilaian Daring SPOJ

Uji coba kinerja pada situs penilaian daring SPOJ dilakukan dengan mengirim program ke situs SPOJ SPOJ Colorful Lights Party (PT07G) [4]. Beberapa program yang dikirimkan pada si-



Gambar 5.12: Kondisi setelah penukaran kesebelas

Gambar 5.13: Hasil uji coba kebenaran situs penilaian daring SPOJ

tus penilaian daring SPOJ merupakan program yang masing-masing variabel bebasnya memiliki nilai yang berbeda.

5.3.2.1. Data Uji Coba Kinerja Situs Penilaian Daring SPOJ

Data yang digunakan pada uji coba ini adalah data yang ada pada situs SPOJ Colorful Lights Party (PT07G) [4] sehingga data masukan yang digunakan untuk pengujian tidak dapat diketahui.

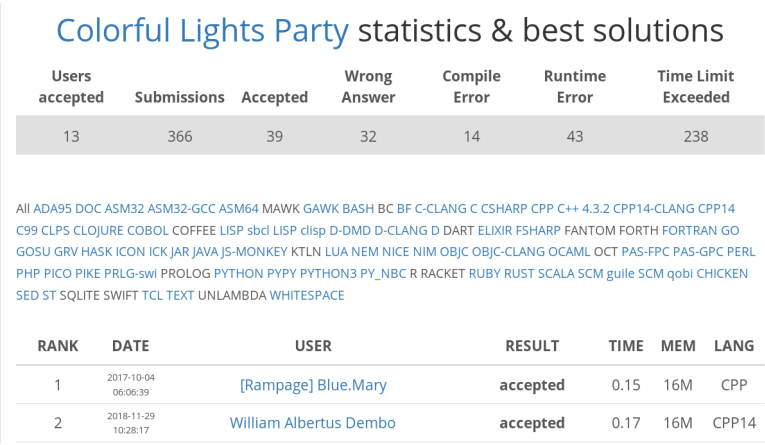
Gambar 5.14: Grafik uji coba kinerja lokal dalam logaritmik

5.3.2.2. Skenario Uji Coba Kinerja Situs Penilaian Daring SPOJ

Uji coba dilakukan dengan mengubah-ubah nilai dari komponen nilai batasan tabu atau nilai M pada Bab 3.1.6 dan mengirim program ke situs SPOJ Colorful Lights Party (PT07G) [4].

5.3.2.3. Evaluasi Uji Coba Kinerja Situs Penilaian Daring SPOJ

Evaluasi dilakukan dengan melihat waktu yang dibutuhkan program pada situs SPOJ Colorful Lights Party (PT07G) [4]. Hasil uji kinerja pada program menggunakan variabel bebas M pada situs penilaian daring SPOJ dapat dilihat pada Tabel 5.2. Pringkat kinerja waktu pada situs penilaian daring dapat dilihat pada Gambar 5.15.



Colorful Lights Party statistics & best solutions

Users accepted	Submissions	Accepted	Wrong Answer	Compile Error	Runtime Error	Time Limit Exceeded
13	366	39	32	14	43	238

All ADA95 DOC ASM32 ASM32-GCC ASM64 MAWK GAWK BASH BC BF C-CLANG C CSHARP CPP C++ 4.3.2 CPP14-CLANG CPP14 C99 CLPS CLOJURE COBOL COFFEE LISP sbcl LISP clisp D-DMD D-CLANG D DART ELIXIR FSHARP FANTOM FORTH FORTRAN GO GOSU GRV HASK ICON ICK JAR JAVA JS-MONKEY KTLN LUA NEM NICE NIM OBJC OBJC-CLANG OCAML OCT PAS-FPC PAS-GPC PERL PHP PICO PIKE PRLG-swi PROLOG PYTHON PYPY PYTHON3 PY_NBC R RACKET RUBY RUST SCALA SCM guile SCM qobi CHICKEN SED ST SQLITE SWIFT TCL TEXT UNLAMBDA WHITESPACE

RANK	DATE	USER	RESULT	TIME	MEM	LANG
1	2017-10-04 06:06:39	[Rampage] Blue.Mary	accepted	0.15	16M	CPP
2	2018-11-29 10:28:17	William Albertus Dembo	accepted	0.17	16M	CPP14

Gambar 5.15: Peringkat kinerja pada situs penilaian daring SPOJ

Jumlah Node	Waktu (mikrodetik)	
	Naive	Tugas Akhir
1	9	7
2	14	6
3	9	11
4	19	18
5	50	27
6	354	37
7	1193	88
8	8026	104
9	42386	514
10	434387	574
11	-	1769
12	-	1850
13	-	3764
14	-	3311
15	-	5459
16	-	7568
17	-	10495
18	-	11045
19	-	15168
20	-	24742
21	-	40080
22	-	62929
23	-	100159
24	-	99732
25	-	121648
26	-	152602
27	-	212134

Tabel 5.1: Tabel data hasil uji coba kinerja lokal

Nilai M	Waktu Kinerja (detik)
5	TLE (>1)
10	TLE (>1)
15	0.17
20	0.22
25	0.24
30	0.26

Tabel 5.2: Tabel data hasil uji coba kinerja situs penilaian daring SPOJ

BAB VI
KESIMPULAN DAN SARAN

Pada bab ini akan diberikan kesimpulan yang diambil selama pengerjaan Tugas Akhir serta saran-saran tentang pengembangan yang dapat dilakukan terhadap Tugas Akhir ini di masa yang akan datang.

6.1. Kesimpulan

Berdasarkan penjabaran di bab-bab sebelumnya, dapat disimpulkan beberapa poin terkait penyelesaian permasalahan SPOJ Colorful Lights Party (PT07G) .

1. Sistem yang diajukan dapat menyelesaikan permasalahan *graceful labelling* pada *general tree* dengan benar dan cepat.
2. Permasalahan *graceful labelling* pada *general tree* dapat diselesaikan dengan metode heuristik menggunakan teknik *local search*.
3. Sistem yang diajukan paling cepat membutuhkan waktu 0.17 detik dan memori 16 MB pada situs penilaian daring SPOJ untuk permasalahan SPOJ Colorful Lights Party (PT07G) .
4. Sistem yang diajukan memiliki kinerja yang lebih baik dari sistem yang menggunakan algoritma naif untuk menyelesaikan permasalahan *graceful labelling* pada *general tree*.

6.2. Saran

Berikut merupakan beberapa saran untuk pengembangan sistem di masa yang akan datang. Saran-saran ini didasarkan pada hasil perancangan, implementasi, dan uji coba yang telah dilakukan.

1. Desain dan implementasi sistem dapat dikembangkan lebih efisien lagi terutama bagian pencarian tetangga yang memi-

liki nilai yang lebih baik karena sekarang masih mencoba semua kemungkinan yang ada.
2. Pemilihan label awal dapat dilakukan lebih baik lagi untuk menghasilkan titik mulai yang lebih baik.
3. Metode pencarian ruang solusi pada Tugas Akhir ini dapat digunakan untuk permasalahan lain yang belum bisa diselesaikan.

DAFTAR PUSTAKA

[1] J. Gallian, "A Dynamic Survey of Graph Labeling", *Electron J Combin DS6*, vol. 20, Dec. 2017.

[2] R. Aldred and B. D. Mckay, "Graceful and Harmonious Labellings of Trees", *Bulletin ICA*, vol. 23, May 2000.

[3] A. Rosa, "On certain valuations of the vertices of a graph", *Journal of Graph Theory - JGT*, Jan. 1967.

[4] (). Colorful Lights Party, [Online]. Available: https://www.spoj.com/problems/PT07G.

[5] M. Forhad, T. Faequa, M. Kaykobad, and M. M. A. Aziz, "Graceful labeling of trees: Methods and Applications", *Journal of Computer Information Systems*, pp. 92–95, Dec. 2014. DOI: 10.1109/ICCITechn.2014.7073154.

[6] F. Harary and A. J. Schwenk, "The number of caterpillars", *Discrete Mathematics*, vol. 6, no. 4, pp. 359 –365, 1973, ISSN: 0012-365X. DOI: https://doi.org/10.1016/0012-365X(73)90067-8. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0012365X73900678.

[7] J. Bermond and J Schönheim, "G-decomposition of Kn, where G has four vertices or less", *Discrete Mathematics*, vol. 19, pp. 113–120, Dec. 1977. DOI: 10.1016/0012-365X(77)90027-9.

[8] E. W. Weisstein. (). Spider Graph. From MathWorld—A Wolfram Web Resource, [Online]. Available: http://mathworld.wolfram.com/SpiderGraph.html.

[9] P. Bahls, S. Lake, and A. Wertheim, "Gracefulness of families of spiders", *Involve*, vol. 3, Oct. 2010. DOI: 10.2140/involve.2010.3.241.

[10] E. W. Weisstein. (). Lobster. From MathWorld—A Wolfram Web Resource, [Online]. Available: http://mathworld.wolfram.com/Lobster.html.

[11] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd. Upper Saddle River, NJ, USA: Prentice Hall Press, 2009, ISBN: 0136042597, 9780136042594.

[12] F. Glover, "Tabu Search—Part I", *ORSA Journal on Computing*, vol. 1, no. 3, pp. 190–206, 1989. DOI: 10.1287/ijoc.1.3.190. eprint: https://doi.org/10.1287/ijoc.1.3.190. [Online]. Available: https://doi.org/10.1287/ijoc.1.3.190.

[13] (). SPOJ Toolkit - Test Case Generator, [Online]. Available: http://www.spojtoolkit.com/TestCaseGenerator.

LAMPIRAN A: Hasil Evaluasi Kebenaran Uji Coba Lokal

Jumlah Node	Data Masukan	Data Keluaran
1	1	0
2	2 2 1	0 1
3	3 2 1 1 3	0 1 2
4	4 2 1 4 3 2 4	2 1 0 3
5	5 2 1 5 2 1 3 3 4	0 4 2 3 1
6	6 2 1 4 2 4 3 6 5 4 6	4 3 2 0 1 5

Jumlah Node	Data Masukan	Data Keluaran
7	7 4 1 5 2 2 3 5 4 4 6 2 7	4 6 2 3 0 5 1
8	8 1 2 7 3 1 4 4 5 4 6 1 7 3 8	1 3 0 5 4 2 6 7
9	9 7 1 3 2 9 3 3 4 2 5 3 6 4 7 9 8	6 2 7 3 5 1 4 8 0

Jumlah Node	Data Masukan	Data Keluaran
10	10 1 2 4 3 1 4 6 5 2 6 2 7 1 8 6 9 5 10	9 1 7 0 4 5 6 3 8 2
11	11 9 1 1 2 9 3 10 4 7 5 9 6 8 7 6 8 8 10 7 11	8 4 9 5 7 10 1 3 0 2 6

Jumlah Node	Data Masukan	Data Keluaran
12	12 5 2 5 3 1 4 1 5 5 6 4 7 10 8 6 9 1 10 10 11 9 12	9 8 5 3 1 10 2 7 0 4 6 11
13	13 7 1 7 2 10 3 10 4 7 5 8 6 11 8 7 9 13 10 7 11 4 12 2 13	9 8 5 4 10 2 0 1 6 7 12 11 3

Jumlah Node	Data Masukan	Data Keluaran
14	14 10 1 1 2 4 3 10 4 10 5 11 6 14 8 14 9 14 10 10 11 14 12 7 13 7 14	6 7 9 3 11 12 0 2 10 1 5 4 8 13
15	15 5 1 6 2 9 3 15 4 11 6 13 7 9 8 1 9 5 10 10 11 13 12 6 13 6 14 11 15	9 13 12 10 8 2 5 4 7 1 14 3 11 6 0

Jumlah Node	Data Masukan	Data Keluaran
16	16 11 1 11 2 6 3 15 4 15 5 9 6 10 7 11 8 11 9 4 10 9 12 1 13 15 14 8 15 2 16	0 6 10 3 7 5 12 2 1 9 15 11 8 13 14 4

Jumlah Node	Data Masukan	Data Keluaran
17	17 3 1 1 2 15 3 9 4 12 5 16 6 1 8 12 9 7 10 7 11 3 12 2 13 3 14 7 15 3 16 1 17	16 0 2 7 10 9 8 1 3 6 14 11 5 13 15 12 4

Jumlah Node	Data Masukan	Data Keluaran
18	18 4 1 17 2 9 3 4 5 13 6 1 7 5 8 5 9 1 10 5 11 8 12 4 13 8 14 5 15 15 16 8 17 12 18	1 3 7 15 17 5 9 0 2 13 6 10 12 4 8 14 16 11

Jumlah Node	Data Masukan	Data Keluaran
19	19 7 1 17 2 1 3 1 4 17 5 4 6 7 8 18 9 8 10 19 11 7 12 12 13 7 14 18 15 10 16 11 17 19 18 7 19	13 12 5 3 2 15 0 11 10 14 1 7 6 9 8 16 17 4 18

Jumlah Node	Data Masukan	Data Keluaran
20	20 5 1 9 2 11 3 9 4 18 5 15 6 12 7 14 8 17 9 4 10 12 11 3 13 12 14 12 15 14 16 12 17 12 18 17 19 2 20	2 15 11 18 12 8 13 16 14 7 4 19 6 0 5 17 1 10 9 3

Jumlah Node	Data Masukan	Data Keluaran
21	21 5 1 1 2 13 3 2 4 12 5 10 6 19 7 17 8 20 9 12 10 15 11 20 12 8 13 17 14 20 15 17 16 6 18 12 19 17 20 13 21	7 15 8 11 14 16 12 5 17 6 1 9 19 3 20 2 18 4 10 0 13

Jumlah Node	Data Masukan	Data Keluaran
22	22 4 1 10 2 6 3 18 4 11 5 18 6 17 7 10 8 22 9 12 10 12 11 5 13 19 14 5 15 12 16 3 17 10 18 10 19 18 20 19 21 16 22	9 13 11 15 17 14 12 0 18 21 3 1 8 5 10 20 7 4 6 16 19 2

Jumlah Node	Data Masukan	Data Keluaran
23	23 6 1 6 2 3 4 13 5 3 6 2 7 17 8 22 9 12 10 17 11 3 12 4 13 4 14 3 15 18 16 13 17 3 18 2 19 11 20 22 21 17 22 3 23	8 18 22 3 12 1 13 17 5 2 19 4 11 14 16 20 7 0 15 10 6 21 9

Jumlah Node	Data Masukan	Data Keluaran
24	24 6 1 18 2 1 3 20 4 11 5 22 6 1 7 5 8 11 9 16 10 16 11 21 13 6 14 19 15 22 16 6 17 12 18 17 19 1 20 20 21 12 22 22 23 7 24	23 15 5 22 4 0 1 16 20 17 10 7 2 13 18 12 11 8 14 3 19 21 6 9

Jumlah Node	Data Masukan	Data Keluaran
25	25 20 1 5 2 12 3 25 4 24 5 12 7 9 8 10 9 12 10 18 11 24 12 25 13 7 14 7 15 12 16 16 17 12 18 16 19 11 20 20 21 4 22 6 23 6 24 12 25	18 10 22 7 9 2 15 19 3 5 6 1 0 4 12 23 13 14 8 11 17 16 20 21 24

Jumlah Node	Data Masukan	Data Keluaran
26	26 3 1 25 2 12 3 19 4 22 5 2 6 2 7 7 8 11 9 22 10 4 12 19 13 25 14 19 15 5 16 12 17 2 18 11 19 14 20 16 21 25 22 7 23 11 24 4 25 11 26	4 3 10 1 20 15 22 6 11 17 14 21 23 0 5 19 13 12 18 25 9 2 8 7 24 16

Jumlah Node	Data Masukan	Data Keluaran
27	27 12 1 11 2 5 3 1 4 12 5 5 6 24 7 10 8 14 9 24 10 1 11 24 13 2 14 9 15 9 16 12 17 23 18 23 19 2 20 11 21 4 22 13 23 2 24 1 25 2 26 13 27	8 1 10 13 0 25 11 6 15 14 24 26 3 22 17 12 2 21 18 5 9 7 4 23 19 20 16

Tabel 6.17: Tabel data hasil uji coba kebenaran lokal

[*Halaman ini sengaja dikosongkan*]

BIODATA PENULIS

Penulis bernama William Albertus Dembo, lahir di Sidoarjo pada tanggal 02 Februari 1997. Penulis telah menjalani masa pendidikan di Sekolah Dasar Katolik St Yustinus De Yacobis Krian pada tahun 2003 hingga 2009, Sekolah Menengah Pertama Katolik St Yustinus De Yacobis Krian pada tahung 2009 hingga 2012, dan Sekolah Menengah Atas Negeri 1 Krian pada tahun 2012 hingga 2015. Pada masa penulisan, penulis sedang menempuh masa studi S1 tahun ketiga di Departemen Informatika, Fakultas Teknologi Informasi dan Komunikasi, Institut Teknologi Sepuluh Nopember.

Selama masa studi S1, penulis memiliki ketertarikan yang dalam mengenai Algoritma dan Pemrograman, basis data, dan rancang bangun aplikasi situs web. Penulis pernah menjadi asisten dosen pada mata kuliah Dasar Pemrograman, Stuktur Data, dan Pemrograman Web. Dalam mengembangkan kemampuan yang dimiliki, penulis pernah menerima beberapa proyek untuk rancang bangun aplikasi web. Selain itu, penulis memiliki pengalaman magang di PT. Dwi Cermat Indonesia dan PT. Digital Otomotif Indonesia.

Di luar kesibukan akademik, penulis cukup aktif dalam organisasi baik dari dalam maupun luar jurusan. Penulis juga berkontribusi dalam kepanitiaan acara nasional selama 2 tahun. Selain itu, penulis pernah membantu kegiatan pelatihan nasional bagi peserta Olimpiade Komputer Indonesia pada tahun 2018 dan 2019. Penulis dapat dihubungi melalui surel di w.albertusd@gmail.com.