AUTOGRADER NOTASI ALGORITMIK BERBASIS *CONTROL FLOW GRAPH*

Laporan Tugas Akhir – Capstone

Advances in Automated Grading for Programming Exercise

Disusun sebagai syarat kelulusan tingkat sarjana

Oleh

Muhammad Hasan

NIM: 13518012

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO & INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

Agustus 2022

AUTOGRADER NOTASI ALGORITMIK BERBASIS *CONTROL FLOW GRAPH*

Laporan Tugas Akhir – Capstone

Advances in Automated Grading for Programming Exercise

Oleh

Muhammad Hasan

NIM : 13518012

Program Studi Teknik Informatika

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung

Telah disetujui dan disahkan sebagai Laporan Tugas Akhir

di Bandung, pada tanggal 12 Agustus 2022

Pembimbing I,	Pembimbing II,
Institut Teknologi Bandung OU=STEI, O=ITB, CN=Dr.Techn. Muhammad Zuhri Catur Candra, ST,MT, E=catur@stei.itb.ac.id Persetujuan Buku TA 13518012 Muhammad Hasan 2022-08-12 21:00:06	
Muhammad Zuhri Catur Candra, S.T, M.T.	Satrio Adi Rukmono, S.T., M.T.
NIP. 19770921 201012 1 002	NIP. 19880927 201903 1 007

LEMBAR IDENTITAS

TUGAS AKHIR CAPSTONE

Judul Proyek TA : *Advances in Automated Grading for Programming Exercise*

Anggota Tim dan Pembagian Peran:

No.	NIM	Nama	Peran
1	13518012	Muhammad Hasan	*Autograder* Notasi Algoritmik berbasis *Control Flow Graph*
2	13518069	Dimas Wahyu Langkawi	Pengembangan Sistem Integrasi *Autograder* Abstrak dengan Moodle dan GitLab
3	13518093	Morgen Sudyanto	*White Box Autograder* Berbasis Semantik
4	13518113	Muhammad Kamal Shafi	*White Box Autograder* Berbasis Kesamaan Struktur *Control Flow Graph*

Bandung, 12 Agustus 2022

Mengetahui,

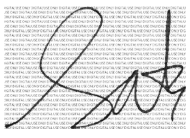
Pembimbing I,

Institut Teknologi Bandung
OU=STEI, O=ITB,
CN=Dr.Techn.
Muhammad Zuhri Catur
Candra, ST,MT,
E=catur@stei.itb.ac.id
Persetujuan Buku TA
13518012 - Muhammad
Hasan
2022-08-12 21:01:42

Muhammad Zuhri Catur Candra, S.T, M.T.

NIP. 19770921 201012 1 002

Pembimbing II,



Satrio Adi Rukmono, S.T., M.T.

NIP. 19880927 201903 1 007

LEMBAR PERNYATAAN

Dengan ini saya menyatakan bahwa:

1. Pengerjaan dan penulisan Laporan Tugas Akhir ini dilakukan tanpa menggunakan bantuan yang tidak dibenarkan.
2. Segala bentuk kutipan dan acuan terhadap tulisan orang lain yang digunakan di dalam penyusunan laporan tugas akhir ini telah dituliskan dengan baik dan benar.
3. Laporan Tugas Akhir ini belum pernah diajukan pada program pendidikan di perguruan tinggi mana pun.

Jika terbukti melanggar hal-hal di atas, saya bersedia dikenakan sanksi sesuai dengan Peraturan Akademik dan Kemahasiswaan Institut Teknologi Bandung bagian Penegakan Norma Akademik dan Kemahasiswaan khususnya Pasal 2.1 dan Pasal 2.2.

Bandung, 12 Agustus 2022

Muhammad Hasan

NIM 13518012

ABSTRAK

AUTOGRADER NOTASI ALGORITMIK BERBASIS *CONTROL FLOW GRAPH*

Oleh

Muhammad Hasan

NIM: 13518012

Mahasiswa Teknik Informatika ITB perlu melakukan pembelajaran pemrograman dengan mendesain dan menuliskan algoritma. Notasi yang digunakan dalam menulis algoritma tersebut disebut sebagai Notasi Algoritmik yang juga digunakan sebagai standar penulisan algoritma dalam berbagai ujian dan penilaian. Permasalahan terjadi ketika pendidik melaksanakan penilaian manual terhadap ujian Notasi Algoritmik. Hal tersebut membutuhkan waktu yang lama dan dapat menimbulkan turunnya objektivitas penilaian. Salah satu cara untuk mengatasi permasalahan tersebut adalah dengan membuat sebuah sistem *autograder* Notasi Algoritmik. Saat ini, belum tersedia sistem *autograder* tersebut, namun sudah ada beberapa penelitian mengenai sistem penilaian otomatis berbasis *control flow graph* yang dapat membantu dalam membuat sistem *autograder* tersebut. Penelitian ini merupakan hasil Tugas Akhir beberapa mahasiswa ITB tahun ajaran 2020/2021. Sistem penelitian yang sudah ada masih terpisah-terpisah dan dapat dikembangkan lebih lanjut untuk membuat sistem *autograder* Notasi Algoritmik.

Atas dasar hal tersebut, pada tugas akhir ini dilakukan penelitian dalam pembuatan sistem *autograder* Notasi Algoritmik berbasis *control flow graph* dengan menggunakan penelitian *autograder* yang sudah ada. Setelah itu, dilakukan pengujian terhadap data ujian untuk mengukur akurasi sistem *autograder* dalam melakukan penilaian otomatis terhadap hasil penilaian manual pendidik.

Berdasarkan pengujian yang dilaksanakan, hasil penilaian dari sistem *autograder* Notasi Algoritmik memiliki akurasi yang lebih baik untuk kasus uji persoalan yang lebih sederhana dan hasil juga cenderung memiliki nilai yang lebih besar dibandingkan dengan hasil penilaian manual pendidik. Selain itu, hasil penilaian *autograder* secara umum memiliki keterkaitan cukup positif dengan nilai manual yang dilakukan pendidik, namun korelasi antara kedua hasil tersebut masih lemah.

Kata Kunci: Notasi Algoritmik, *autograder*, *control flow graph*

KATA PENGANTAR

Puji syukur penulis panjatkan ke hadirat Tuhan Yang Maha Esa atas berkat dan rahmat-Nya, penulis dapat menyelesaikan Tugas Akhir yang berjudul "*Autograder* Notasi Algoritmik berbasis *Control Flow Graph*". Penulis ingin mengucapkan terima kasih kepada pihak-pihak yang telah membantu penulis selama pengerjaan tugas akhir:

1. Bapak Muhammad Zuhri Catur Candra, S.T, M.T. dan bapak Satrio Adi Rukmono, S.T., M.T. selaku dosen pembimbing yang telah memberikan banyak arahan, saran, dan dukungan selama pengerjaan tugas akhir.
2. Bapak Fitra Arifiansyah, S.Kom., M.T. selaku dosen wali penulis yang selalu memberikan arahan, bimbingan, dan motivasi selama menempuh pendidikan di Prodi Teknik Informatika ITB.
3. Staf pengajar Prodi Teknik Informatika ITB yang telah banyak memberikan ilmu dan wawasan kepada penulis selama menempuh pendidikan di Prodi Teknik Informatika ITB.
4. Tim pembimbing tugas akhir dan staf tata usaha Prodi Teknik Informatika ITB yang telah memberikan arahan dan membantu proses administrasi pengerjaan tugas akhir.
5. Orang tua dan keluarga penulis yang telah banyak memberikan dukungan baik moril maupun materiil selama pengerjaan tugas akhir.

6. Dimas, Kamal, dan Morgen selaku teman-teman satu *capstone* tugas akhir penulis yang telah menjadi teman untuk diskusi, bertanya dan pemberi motivasi antar satu sama yang lain.
7. Naufal Prima, Taufiq Husada, dan Naufal Dean selaku teman penulis yang telah sama-sama memberikan dukungan dan saling membantu dalam proses pengerjaan tugas akhir.
8. Teman-teman Prodi Teknik Informatika ITB yang tidak dapat penulis sebutkan satu-satu yang telah menemani penulis selama menempuh pendidikan di Prodi Teknik Informatika ITB.

Penulis berharap tugas akhir ini tidak hanya bermanfaat bagi penulis namun juga untuk pembaca tugas akhir ini.

Bandung, 12 Agustus 2022

Penulis

DAFTAR ISI

DAFTAR LAMPIRAN

DAFTAR GAMBAR

DAFTAR TABEL

DAFTAR KODE

BAB I
PENDAHULUAN

I.1 Latar Belakang

Pembelajaran pemrograman kini menjadi hal yang penting bagi orang yang ingin menekuni bidang teknologi, tidak terkecuali mahasiswa Teknik Informatika ITB. Pada Program Studi Teknik Informatika ITB, terdapat beberapa cara dalam melaksanakan pembelajaran pemrograman, salah satunya adalah dengan menggunakan pemrograman untuk menyelesaikan suatu permasalahan.

Mahasiswa Teknik Informatika ITB perlu melakukan penulisan pemrograman dengan mendesain dan menuliskan algoritma guna menyelesaikan permasalahan tertentu dalam beberapa ujian penilaian seperti kuis, UTS, dan UAS. Dalam menuliskan algoritma tersebut, diperlukan notasi standar yang disepakati dan dipahami oleh berbagai pihak yang terkait. Notasi standar tersebut kemudian disebut sebagai Notasi Algoritmik.

Notasi Algoritmik di lingkungan Teknik Informatika ITB merupakan sebuah standar khusus dalam menuliskan teks algoritma. Notasi ini dianggap perlu untuk menjembatani berbagai keragaman dan kompleksitas bahasa sehingga mahasiswa mampu melakukan “abstraksi”.

Notasi Algoritmik menjadi media komunikasi dosen dalam menjelaskan algoritma kepada mahasiswa pada mata kuliah terkait. Notasi Algoritmik juga digunakan sebagai standar penulisan algoritma dalam berbagai ujian dan penilaian seperti kuis, UTS, dan UAS.

Permasalahan terjadi ketika pendidik melaksanakan penilaian terhadap beberapa ujian Notasi Algoritmik peserta didik. Penilaian tersebut masih dilakukan secara manual, hal ini membutuhkan waktu yang lama, selain itu ketika jumlah peserta didik yang mengambil mata kuliah berhubungan dengan Notasi Algoritmik semakin meningkat. Pendidik melakukan aktivitas yang repetitif dan tentunya

beban pendidik dalam melakukan aktivitas penilaian perkuliahan menjadi meningkat.

Salah satu cara untuk mengatasi permasalahan tersebut adalah membuat sebuah sistem *autograder* atau penilaian otomatis sehingga penilai cukup menggunakan sistem tersebut dalam melaksanakan penilaian. Saat ini, sudah ada penelitian mengenai sistem penilaian otomatis atau *autograder*, yang juga merupakan hasil Tugas Akhir beberapa mahasiswa Institut Teknologi Bandung yang dilakukan pada tahun ajaran 2020/2021. Tabel I.1 menunjukkan hasil penelitian-penelitian tersebut.

Tabel I.1 Hasil Penelitian Sistem *Autograder* dari beberapa Mahasiswa ITB Tahun Ajaran 2020 / 2021

NIM	Nama	Penelitian
13517023	Kevin Sendjaja	Evaluasi Tugas Pemrograman dengan Pendekatan *Control Flow Graph*
13517078	Irfan Sofyana	Pembangkitan *Abstract Syntax Tree* (AST) dan *Control Flow Graph* (CFG) Notasi Algoritmik
13517081	M. Rifky I. Bariansyah	Penilaian Otomatis Tugas Pemrograman dengan Pendekatan Semantik
13517089	Bram Musuko Panjaitan	Integrasi *Learning Management System* dengan Gitlab dan Layanan *Autograder*

Namun, hasil sistem yang sudah ada masih terpisah-pisah, oleh karena itu proyek tugas akhir *capstone* ini bertujuan untuk mengembangkan empat buah subsistem sehingga dapat digunakan sebagai sebuah satu sistem *autograder* utuh. Selain itu, beberapa penelitian ini masih dapat dikembankan lebih lanjut lagi untuk meningkatkan utilitas dan kinerja subsistem. Tabel I.2 menunjukkan pembagian peran dari setiap subsistem yang akan dikembangkan.

Tabel I.2 Pembagian Peran Subsistem dari Sistem *Autograder*

NIM	Nama	Subsistem
13518012	Muhammad Hasan	*Autograder* Notasi Algoritmik berbasis *Control Flow Graph*
13518069	Dimas Wahyu Langkawi	Pengembangan Sistem Integrasi *Autograder* Abstrak dengan Moodle dan GitLab
13518093	Morgen Sudyanto	*White Box Autograder* Berbasis Semantik
13518113	Muhammad Kamal Shafi	*White Box Autograder* Berbasis Kesamaan Struktur *Control Flow Graph*

Proyek *capstone* ini pada intinya terbagi menjadi dua peran utama, yakni peran dalam pembuatan Integrasi *Autograder* dengan Moodle dan GitLab, serta peran dalam pembuatan beberapa sistem *autograder*. Kedua peran tersebut dikerjakan terpisah, namun akhirnya dapat disatukan dengan menggunakan kontrak *web service* yang disepakati, yang mana *autograder* dibuat menjadi *web service* sehingga dapat dilakukan integrasi dengan Moodle dan GitLab. Penjelasan lebih detail proyek *capstone* bisa ditemukan pada Lampiran A.

Sebagaimana yang disebutkan pada Tabel I.2, mengenai pembagian peran pengerjaan subsistem *autograder*, fokus tugas akhir ini adalah pada pembuatan *Autograder* Notasi Algoritmik Berbasis *Control Flow Graph*.

Pada hasil penelitian Sofyana dkk. (2021), sudah dihasilkan sistem yang dapat mengubah tulisan Notasi Algoritmik menjadi *Control Flow Graph* (CFG), yakni suatu representasi *graph* yang menyatakan alur eksekusi suatu kode program. Pada penelitian tersebut, hasil CFG yang didapat sebetulnya dapat digunakan untuk melakukan penilaian, namun ternyata belum dilaksanakan pada penelitian tersebut. Oleh karena itu, perlu ada pengembangan lanjut untuk melakukan penilaian otomatis dari hasil CFG tulisan Notasi Algoritmik.

Kemudian, pada hasil penelitian Sendjaja dkk. (2021), sudah dihasilkan sistem yang dapat menilai tugas pemrograman berbasis CFG. Penilaian yang dilaksanakan tersebut mendekati pendekatan penilaian *white box*, yakni teknik pengujian dengan membuat penilaian berdasarkan informasi dari sumber kode. Pada sistem ini, penilaian didapat dari hasil perbandingan CFG kode pengajar dengan CFG kode pelajar.

Penelitian Sofyana dkk. (2021) dan penelitian Sendjaja dkk. (2021) ini dapat diintegrasikan menjadi satu sistem *autograder* yang dapat melakukan penilaian terhadap tulisan Notasi Algoritmik. Namun, sampai saat ini belum dilakukan integrasi tersebut karena beberapa permasalahan di antaranya disebabkan pengerjaan penelitian yang dilakukan secara terpisah dan adanya perbedaan struktur CFG yang digunakan oleh masing-masing penelitian. Oleh karena itu, pada tugas akhir ini akan diselesaikan permasalahan terkait integrasi tersebut.

Sistem hasil penelitian Sofyana dkk. (2021) dan sistem hasil penelitian Sendjaja dkk. (2021) yang dilakukan integrasi menjadi satu sistem *autograder* kemudian dapat digunakan untuk menilai data ujian Notasi Algoritmik secara otomatis.

Sistem Notasi Algoritmik *autograder* kemudian perlu dilakukan pengukuran akurasi penilaian dibandingkan dengan nilai manual oleh pendidik. Hal ini dapat dilakukan dengan melakukan perbandingan antara hasil penilaian *autograder* dengan hasil penilaian manual yang dilakukan pendidik.

Sistem *autograder* untuk tulisan Notasi Algoritmik ini kemudian perlu dibuat menjadi *web service* agar subsistem Integrasi *Autograder* Abstrak dengan Moodle dan GitLab dapat memanfaatkan hasil penilaiannya. Hal tersebut dilaksanakan untuk memenuhi tujuan tugas akhir *capstone*.

I.2 Rumusan Masalah

Berdasarkan uraian latar belakang, terdapat 4 buah rumusan masalah dari tugas akhir ini:

1. Bagaimana cara melakukan integrasi hasil penelitian Sofyana dkk. (2021) dengan hasil penelitian Sendjaja dkk. (2021) untuk membuat sistem *autograder* Notasi Algoritmik berbasis *Control Flow Graph*?
2. Seberapa akurat hasil penilaian data ujian Notasi Algoritmik peserta didik menggunakan sistem *autograder* Notasi Algoritmik berbasis *Control Flow Graph*?

I.3 Tujuan

Tujuan utama yang akan dicapai dalam tugas akhir ini adalah:

1. Membuat sistem *autograder* Notasi Algoritmik berbasis *Control Flow Graph* dengan melakukan integrasi hasil penelitian Sofyana dkk. (2021) dan penelitian Sendjaja dkk. (2021).
2. Mengukur akurasi penilaian dari sistem *autograder* Notasi Algoritmik berbasis *Control Flow Graph* terhadap hasil penilaian manual pendidik.

I.4 Batasan Masalah

Batasan masalah untuk tugas akhir ini adalah sebagai berikut:

1. Sistem untuk membangkitkan CFG dari tulisan Notasi Algoritmik didapat dengan melanjutkan pengembangan penelitian Sofyana dkk. (2021).
2. Penilaian tugas pemrograman berbasis CFG yang dilakukan didapat dengan melanjutkan pengembangan penelitian Sendjaja dkk. (2021).
3. Notasi Algoritmik yang digunakan pada tugas akhir ini merupakan Notasi Algoritmik untuk pemrograman berparadigma prosedural yang digunakan di Program Studi Teknik Informatika ITB.
4. Segala format, keperluan, dan cara penggunaan *web service* penilaian pemrograman berbasis CFG dari tulisan notasi algoritmik, akan mengikuti keperluan subsistem lain dari sistem *autograder* yang akan dibuat.

I.5 Metodologi

Terdapat beberapa tahapan dalam melaksanakan tugas akhir ini, tahapan-tahapan tersebut adalah sebagai berikut:

1. Analisis Persoalan

   Pada tahap ini dilakukan identifikasi masalah dalam pembuatan *Autograder* Notasi Algoritmik berbasis *Control Flow Graph*. Saat melakukan identifikasi, dilakukan analisis mengenai permasalahan apa saja yang dapat timbul. Perumusan masalah menjadi dasar dari pengerjaan solusi yang akan dibuat.

2. Analisis dan Perancangan Solusi

   Pada tahap ini dilakukan perancangan solusi dari setiap permasalahan yang ditemukan saat melakukan analisis persoalan. Kemudian, dilakukan juga analisis terhadap beberapa alternatif solusi yang ditemukan selama pengerjaan tugas akhir. Untuk setiap alternatif solusi yang ditemukan, akan dibuat rancangan awal sebagai dasar dalam melakukan implementasi tugas akhir. Selain itu, dibuat beberapa pertimbangan dan perbandingan antar alternatif solusi, untuk mendapatkan gambaran tentang kelebihan dan kekurangan setiap alternatif solusi. Dari berbagai alternatif tersebut, akan dipilih satu solusi terbaik untuk dipakai.

3. Implementasi Solusi

   Pada tahap ini dilakukan implementasi rancangan solusi yang sudah dibuat saat melakukan analisis dan perancangan solusi. Berdasarkan rancangan solusi yang sudah dibuat, akan dibangun implementasi untuk melakukan pembuatan Sistem *Autograder* Notasi Algoritmik berbasis *Control Flow Graph*.

4. Pengujian dan Evaluasi Solusi

   Pada tahap ini dilakukan pengujian serta evaluasi terhadap solusi yang telah dilakukan implementasi dan integrasi. Pengujian ini berfungsi untuk mengetahui hasil serta kinerja dari implementasi solusi yang sudah dibuat.

I.6 Sistematika Pembahasan

Sistematika pembahasan pada tugas akhir ini adalah sebagai berikut:

1. BAB I Pendahuluan

   Bab ini berisi arahan dasar dalam pengerjaan tugas akhir ini. Bab ini akan memaparkan mengenai latar belakang, rumusan masalah, tujuan, batasan masalah, metodologi, dan sistematika pembahasan dalam tugas akhir ini.

2. Bab II Studi Literatur

   Bab ini berisi seluruh teori maupun kakas yang digunakan pada tugas akhir. Isi dari bab ini menjadi landasan dalam merancang solusi tugas akhir.

3. Bab III Analisis dan Perancangan Sistem

   Bab ini berisi analisis mengenai persoalan utama yang menjadi fokus tugas akhir ini, serta rancangan solusi yang diajukan untuk menyelesaikan persoalan tersebut.

4. Bab IV Implementasi

   Bab ini berisi tentang implementasi solusi untuk menyelesaikan permasalahan tugas akhir.

5. Bab V Pengujian

   Bab ini berisi pengujian terkait solusi yang telah diajukan, serta analisis terhadap hasil pengujian yang dilakukan.

6. Bab VI Kesimpulan dan Saran

   Bab ini berisi kesimpulan yang didapatkan berdasarkan eksperimen dan analisis terhadap hasil yang didapatkan dari hasil sistem penguji, serta saran untuk pengembangan lebih lanjut.

BAB II
STUDI LITERATUR

II.1 Notasi Algoritmik di Lingkungan Prodi Teknik Informatika ITB

Penjelasan subbab ini diambil berdasarkan Draft Diktat Kuliah Dasar Pemrograman (Bagian Pemrograman Prosedural) oleh Inggriani Liem (2007). Notasi algoritmik di lingkungan Teknik Informatika ITB merupakan sebuah standar khusus dalam menuliskan teks algoritma. Notasi ini dianggap perlu untuk menjembatani berbagai keragaman dan kompleksitas bahasa sehingga mahasiswa mampu melakukan "abstraksi". Notasi ini lebih berorientasi kepada *detail design* dibandingkan *coding*.

II.1.1 Struktur Notasi Algoritmik

Teks dalam notasi algoritmik selalu terdiri dari tiga bagian, yaitu judul (*header*), kamus, dan algoritma. Judul ini berupa deklarasi penamaan program, kamus berisi variabel yang digunakan, dan algoritma berisi logika dan isi inti algoritma dari suatu teks Notasi Algoritmik. Contoh Notasi Algoritmik ditunjukkan pada Gambar II.1 sebagai berikut.

```
PROGRAM ganjil_genap
{program untuk memeriksa suatu bilangan ganjil atau genap}

KAMUS
   n: integer

ALGORITMA
   input(n)
   if (n mod 2 = 1) then
       output("ganjil")
   else
       output("genap")
```

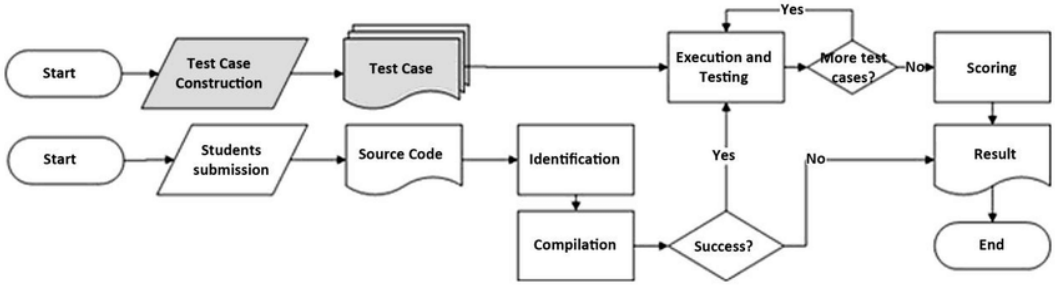
Gambar II.1 Contoh Notasi Algoritmik

II.2 Sistem Penilaian Otomatis

Sistem penilaian otomatis merupakan sistem yang disediakan untuk menilai hasil pekerjaan secara otomatis dan dapat memberikan respon dengan cepat sehingga dapat digunakan dengan interaktif. Penggunaan penilaian otomatis akan memberikan kemudahan dalam penilaian dan juga bisa meningkatkan kualitas pembelajaran. Berdasarkan yang dipaparkan oleh Sherman dkk. (2013) pada *paper*-nya, terdapat indikasi dampak positif yang dialami peserta didik mata pelajaran pemrograman dengan diterapkannya penggunaan penilaian otomatis. Penilaian otomatis akan meningkatkan pengalaman peserta didik dalam *hands-on* pembelajaran pemrograman. Untuk melakukan penilaian terhadap suatu kode program tertentu, perlu dilakukan pengujian terhadap program tersebut. Secara umum, terdapat dua cara pengujian yaitu *black box testing* dan juga *white box testing*.

II.2.1 *Black Box Testing*

Black box testing merupakan sebuah teknik pengujian yang dilakukan untuk mengetahui apakah sebuah program bekerja sesuai dengan yang diinginkan. Teknik *black box testing*, biasa disebut juga sebagai *functional testing*, melakukan pengujian dengan menggunakan beberapa kasus uji berupa masukkan maupun konfigurasi dari program. Teknik ini kemudian akan melakukan penilaian kode program berdasarkan keluaran yang diberikan program. Penilaian kode program dengan menggunakan pendekatan *black box* memiliki proses sesuai dengan diagram alur pada Gambar II.2.



Gambar II.2 Diagram *Flow Black Box Testing* (Danutama, K., & Liem, I., 2013)

Penilaian pada metode *black box*, tidak memerdulikan bagaimana isi dari program yang sedang diujikan. Pengujian seperti ini berfokus hanya terhadap kemampuan sebuah program untuk melakukan aksi sesuai dengan yang diinginkan. Pada praktiknya, untuk dapat melakukan *black box testing*, seorang penguji hanya perlu mengetahui spesifikasi dari program yang diuji. Dari pengetahuan mengenai spesifikasi program, penguji dapat membuat kasus uji yang sesuai dengan batasan spesifikasi dan juga penguji dapat mengetahui hasil yang seharusnya dihasilkan oleh kode program yang benar.

Berdasarkan *paper* yang ditulis oleh Ihantola dkk. (2010), penilaian otomatis dengan *functional testing* terhadap pengerjaan pelajar merupakan pendekatan yang paling sering digunakan. Saat ini, sudah banyak kakas yang dapat digunakan untuk membantu melakukan *functional testing*. Contoh-contoh jenis kakas yang digunakan pada bidang industri adalah XUnit, *acceptance testing frameworks*, dan *web testing frameworks*.

Meskipun penilaian otomatis dengan pendekatan seperti ini banyak digunakan, berdasarkan artikel Naudé dkk. (2010), penilaian otomatis yang dilakukan hanya dengan menguji keluaran program memiliki masalah berikut:

1. Memberikan informasi penilaian yang terbatas.
2. Mempunyai batasan hal yang dapat dinilai.
3. Sensitif terhadap kesalahan kecil.

Akibat dari masalah ketiga, penilaian dengan *black box testing* dapat memberikan hasil yang tidak stabil. Hal ini kemudian dapat menyebabkan kondisi yang kurang ideal pada penilaian kode program pengerjaan pelajar.

II.2.2 *White Box Testing*

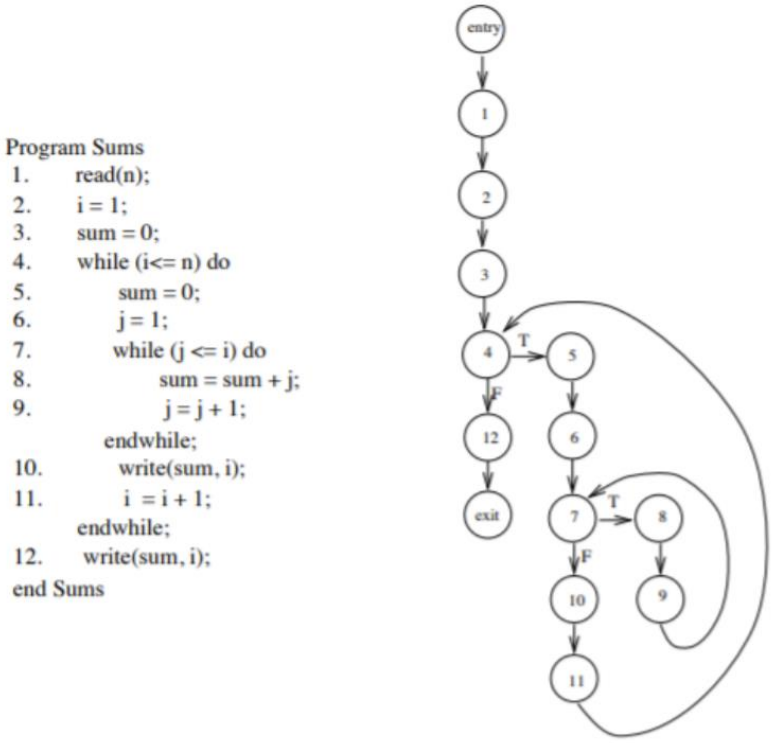
Berbeda dengan *black box testing*, metode *white box testing* melakukan pengujian berdasarkan informasi yang diperoleh dari kode program. *White box testing*, disebut juga sebagai *structural testing*, adalah sebuah metode yang dapat diterapkan untuk menguji mekanisme, struktur, maupun alur sebuah program. Fokus utama dari *white box testing* adalah pengujian terhadap *control flow* maupun *data flow* dari sebuah program. Pengujian dengan white box testing dapat dilakukan dengan menganalisis struktur program saja maupun dengan juga membuat kasus uji menyesuaikan dengan struktur program yang diketahui. Salah satu metode *white box* yang bisa digunakan untuk melakukan penilaian kode program adalah dengan menggunakan *static analysis.*

II.3 *Control Flow Graph*

Control Flow Graph adalah representasi, menggunakan notasi grafik, dari semua kemungkinan jalur yang mungkin dilalui selama suatu eksekusi program. *Control Flow Graph* ditemukan oleh Frances E. Allen pada tahun 1970, yang mencatat bahwa Reese T. Prosser menggunakan matriks konektivitas boolean untuk analisis aliran sebelumnya (Reese T. Prosser, 1959).

Dalam *control flow graph*, setiap simpul dalam grafik mewakili blok dasar, yaitu potongan kode dalam satu baris tanpa *jumps* atau *jumps target*; *jumps target* memulai blok, dan *jumps* mengakhiri blok. *Directed edges* digunakan untuk mewakili *jumps* dalam aliran kontrol. Dalam kebanyakan presentasi, terdapat dua blok yang ditunjuk secara khusus: *entry block*, yang di mana kontrol masuk melalui *flow graph*, dan *exit block*, di mana semua aliran *control flow* keluar (Yousefi, Javad, 2015).

Gambar II.3 menunjukkan *control flow graph* dari suatu *program sums*. Pada gambar tersebut, setiap baris direpresentasikan oleh sebuah *node*, kemudian adanya *directed edges* menunjukan aliran kontrol yang pada program tersebut.

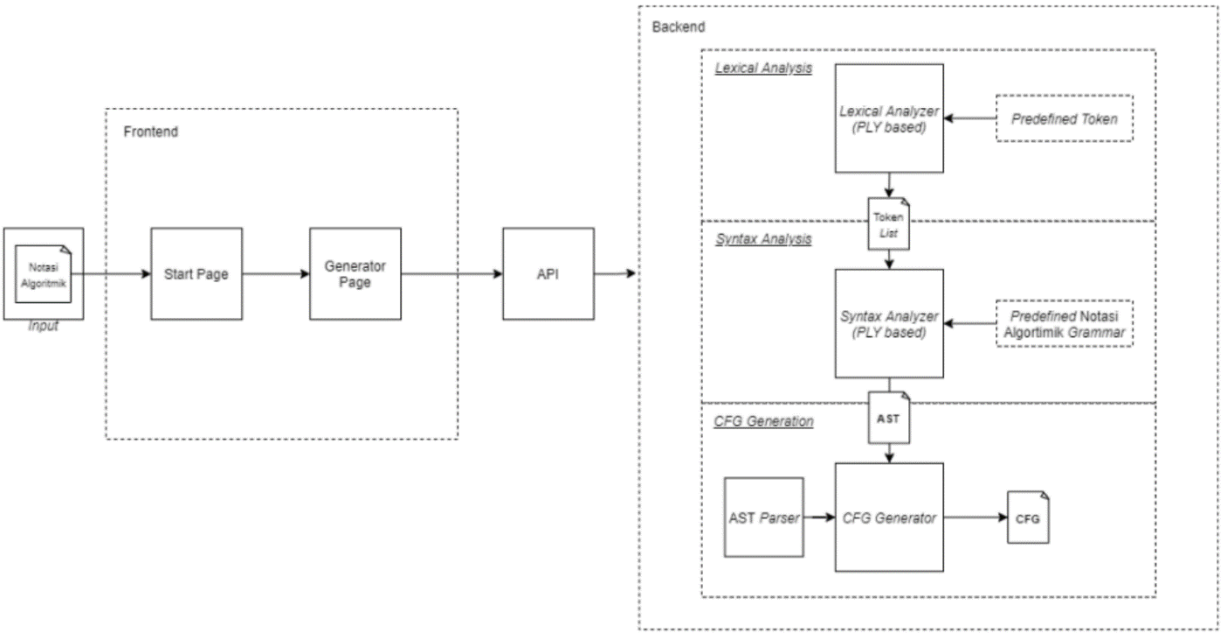


Gambar II.3 *Control Flow Graph* Dari Program Sums

II.4 Sistem Pembangkit AST dan CFG dari Notasi Algoritmik

Sistem pembangkit AST (*Abstract Syntax Tree*) dan CFG (*Control Flow Graph*) dari Notasi Algoritmik merupakan hasil penelitian tugas akhir Irfan Sofyana, mahasiswa ITB tahun ajaran 2020 / 2021. Sistem ini dibuat dengan Python dan dapat menerima tulisan Notasi Algoritmik dan dapat mengubahnya menjadi AST ataupun CFG.

Arsitektur dari sistem ini ditunjukkan pada Gambar II.4.



Gambar II.4 Arsitektur Sistem Pembangkit AST dan CFG dari Notasi Algoritmik

Secara umum, sistem terdiri dari dua modul besar yaitu modul *frontend*, yang digunakan untuk membantu pengguna dalam menggunakan kakas dan modul *backend*, modul yang digunakan untuk membangkitkan AST dan CFG dari suatu teks Notasi Algoritmik. Dua buah modul ini dihubungkan oleh sebuah komponen yang diberi nama API.

Penjelasan detail dari arsitektur kakas tersebut adalah sebagai berikut:

1. Komponen *StartPage*

   Komponen *StartPage* adalah sebuah komponen yang menjadi halaman antarmuka pertama yang dilihat oleh pengguna kakas. Pada komponen ini, pengguna diberikan dua buah pilihan, yaitu dapat membangkitkan AST dan CFG dari sebuah teks *file* Notasi Algoritmik atau mengetikkan Notasi Algoritmik pada kakas. Setelah pengguna memberikan pilihan, maka pengguna dialihkan ke komponen *GeneratorPage*.
2. Komponen *GeneratorPage*

Komponen *GeneratorPage* adalah komponen antarmuka yang digunakan oleh pengguna kakas untuk membangkitkan AST dan CFG. Pada komponen ini, pengguna dapat melakukan beberapa hal seperti melihat masukan Notasi Algoritmik yang diberikan, membangkitkan AST dan melihat hasilnya dalam bentuk JSON, membangkitkan CFG dan melihat representasi *graph* yang dihasilkan, memvisualisasikan AST dalam bentuk gambar, dan juga memvisualisasikan CFG dalam bentuk gambar.

3. *Predefined Token*

   Predefined token adalah daftar token yang terdefinisi pada Notasi Algoritmik. Mendefinisikan daftar token ini menjadi prasyarat untuk proses *lexical analysis*. Token ini menjadi satuan terkecil dari suatu teks Notasi Algoritmik. Token direpresentasikan dengan sebuah aturan *regular expression*. Token yang ada pada Notasi Algoritmik dikelompokkan menjadi *token symbol*, *token reserve word*, *token literal*, *token identifier*, *token whitespace*, *token*, *comment*, *token indent*, dan *token dedent*. Adapun jenis pengkodean karakter yang digunakan adalah UTF-8.

4. Komponen *Lexical Analyzer*

   Komponen ini ada pada proses *lexical analysis*. *Lexical analyzer* menganalisis Notasi Algoritmik yang diberikan dan kemudian mengubahnya menjadi rangkaian token berdasarkan *predefined token* yang didefinisikan pada Notasi Algoritmik. *Lexical analyzer* dibangun di atas modul lex pada pustaka PLY.

5. *Predefined Grammar*

   Predefined grammar adalah *grammar* yang digunakan untuk mendefinisikan tata bahasa dari Notasi Algoritmik. Mendefinisikan *grammar* dari Notasi Algoritmik menjadi prasyarat untuk melakukan proses *syntax analysis*. Dengan menetapkan *grammar* tersebut, teks Notasi Algoritmik yang ingin dibangkitkan AST dan CFG nya harus memenuhi aturan tersebut. Apabila tidak memenuhi aturan tersebut, maka *syntax error* diberikan oleh kakas.

6. Komponen *Syntax Analyzer*

Komponen ini ada pada proses *syntax analysis*. Rangkaian token yang dihasilkan oleh *lexical analyzer* diterima oleh *syntax analyzer*. *Syntax analyzer* kemudian menganalisis rangkaian token tersebut terhadap predefined grammar pada Notasi Algoritmik. *Syntax analyzer* dibangun di atas modul *yacc* pada pustaka PLY. Dengan menggunakan modul *yacc* tersebut, pengguna *syntax analyzer* dapat melakukan kustomisasi pada proses *parsing* sehingga dapat dihasilkan sebuah AST.

7. Komponen AST *Parser*

   AST *parser* adalah komponen yang digunakan untuk menguraikan informasi Notasi Algoritmik yang telah direpresentasikan dalam bentuk AST. Contoh pemanfaatan AST *Parser* adalah mendapatkan teks Notasi Algoritmik dari suatu node pada AST. Komponen ini digunakan oleh komponen CFG *generator* untuk mendapatkan informasi *statement* dari teks Notasi Algoritmik yang menjadi informasi pada suatu *node* dari CFG yang dibangkitkan.

8. Komponen CFG *Generator*

   AST yang dihasilkan kemudian digunakan pada proses pembangkitan CFG oleh CFG *generator*. AST digunakan sebagai struktur data yang berperan untuk merepresentasikan Notasi Algoritmik. Saat membangkitkan CFG, komponen CFG *generator* dibantu oleh komponen AST *Parser*.

9. Komponen API

   Komponen API adalah komponen yang berisi sekumpulan fungsi-fungsi yang digunakan untuk menghubungkan masukan pengguna berupa Notasi Algoritmik dari komponen *frontend* dengan komponen *backend*. Fungsi-fungsi tersebut diantaranya adalah fungsi untuk membangkitkan AST dan mengembalikan AST dalam bentuk JSON, fungsi untuk membangkitkan CFG, dan fungsi-fungsi untuk melakukan visualisasi AST dan CFG yang dibangkitkan.

II.5 Algoritma Hu untuk Melakukan Pengukur Kemiripan CFG

Algoritma Hu merupakan salah satu algoritma yang dapat digunakan untuk melakukan pengukuran kemiripan dari dua buah CFG. Algoritma ini didesain oleh Hu dkk. (2009).

Fokus utama dari algoritma ini adalah mencari jumlah perubahan yang minimum untuk mentransformasi sebuah graf menjadi graf lain, atau dapat dikatakan juga sebagai selisih *cost* antara dua buah graf. Kunci dari algoritma ini adalah pembangunan sebuah *cost matrix* yang akan digunakan untuk perhitungan *total cost*.

Proses ini diawali dengan membentuk sebuah *cost matrix* dari kedua graf tersebut. Untuk 2 buah graf G_1 dan G_2, dengan V_1 dan V_2 merupakan kumpulan *node* dari masing-masing G_1 dan G_2, sejumlah $|V_2|$ *dummy node* akan ditambahkan ke V_1 dan $|V_1|$ *dummy node* ditambahkan juga ke V_2, sehingga *cost matrix* yang terbentuk akan berukuran $(|V_1| + |V_2|) \times (|V_1| + |V_2|)$. Matriks ini dapat dibagi menjadi 4 bagian. Sub-matriks pada bagian kiri atas seukuran $|V_1| \times |V_2|$ akan diisi dengan cost dari pencocokan antara node asli pada G_1 dan G_2. Nilai dari masing-masing cost tersebut adalah:

$$a_{ij} = \text{relabeling cost} + (|ON_i| + |ON_j| - 2 \times |ON_i \cap ON_j| + (|IN_i| + |IN_j| - 2 \times |INi \cap INj|) \quad \text{(II-1)}$$

Pada rumus di atas, *relabeling cost* melambangkan *cost* untuk menyamakan instruksi atau kode antara 2 buah *node*, yang dapat diabaikan apabila pengukuran hanya berdasarkan topologi graf. IN melambangkan *in-neighbour*, yaitu *node-node* yang terhubung dengan panah menuju *node* yang diamati, dan ON melambangkan *out-neighbour*, yaitu *node-node* yang terhubung dengan panah keluar dari *node* yang diamati. Sub-matriks pada bagian kanan bawah seukuran $|V_2| \times |V_1|$ merupakan akan diisi dengan nilai 0, yang melambangkan pencocokan antara *node dummy* pada G_1 dan G_2. Sub-matriks pada bagian kanan atas seukuran $|V_1| \times |V_1|$ berisi *cost* dari pencocokan antara *node* asli pada G_1 dan *dummy node*, yang sama

dengan *cost* untuk menghapus *node* tersebut. Nilai dari masing-masing *cost* tersebut adalah:

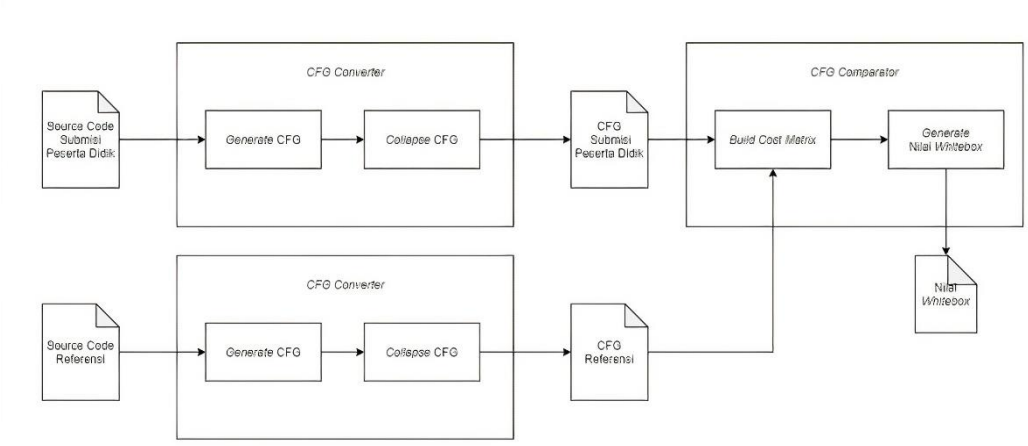
$$a_{ij} = 1 + |IE_i| + |OE_i| \quad \text{(II-2)}$$

Pada rumus di atas, IE melambangkan jumlah panah yang menuju *node* yang diamati, sementara OE melambangkan jumlah panah yang keluar dari *node* yang diamati.

Setelah *cost matrix* tersebut telah terdefinisi, langkah selanjutnya adalah mencocokkan seluruh *node* pada kedua graf dengan *cost* yang minimal. Proses ini dapat dicapai menggunakan Algoritma Hungaria (*Hungarian Algorithm*). Hasil dari proses tersebut adalah *edit distance* atau *cost* antara kedua graf, yang nantinya dapat dipakai dalam perhitungan untuk menghitung kemiripan antara kedua graf.

II.6 Sistem *White Box Autograder* berbasis Control Flow Graph

Sistem *White Box Autograder* berbasis *Control Flow Graph* merupakan hasil penelitian tugas akhir Kevin Sendjaja, mahasiswa ITB tahun ajaran 2020 / 2021. Sistem ini dibuat dengan Python dan dapat melakukan penilaian kemiripan antara dua CFG. Sistem ini dapat menerima kode program Python pengajar dan kode program Python pelajar, kemudian masing-masing kode program tersebut diubah menjadi CFG dengan *library* PyCFG, setelah itu kedua CFG dinilai kemiripannya dengan Algoritma Hu dkk. (2009).



Gambar II.5 Rancangan Sistem *White Box Autograder* berbasis *Control Flow Graph*

Rancangan dari sistem ini digambarkan pada Gambar II.5 Keluaran utama dari sistem yang dirancang adalah nilai *whitebox* yang merupakan hasil perbandingan kemiripan *Control Flow Graph*.

II.6.1 *CFG Converter*

Komponen ini berfungsi untuk membangkitkan representasi *control flow graph* dari kode program. Kode yang akan dikonversi adalah kode referensi yang diberikan oleh pengajar dan kode hasil submisi pelajar. Keluaran dari komponen ini adalah suatu representasi *control flow graph* dari kode-kode yang telah dimasukkan. Representasi ini nantinya yang akan digunakan untuk proses pemberian nilai berdasarkan kemiripan *control flow graph* yang akan dilakukan pada tahapan setelahnya.

Sebelum dilanjutkan ke proses selanjutnya, representasi *control flow graph* yang telah dibangkitkan akan melalui proses *collapse* terlebih dahulu. Fungsi ini akan menggabungkan seluruh *node* yang terletak pada satu *flow* yang sama menjadi satu *node*.

II.6.2 *CFG Comparator*

Komponen ini akan melakukan perbandingan terhadap *control flow graph* yang telah dibangun oleh komponen sebelumnya. Hasil perbandingan berupa nilai yang merepresentasikan tingkat kemiripan antara 2 buah *control flow graph* dengan skala dari 0 hingga 100. Komponen ini akan menggunakan Algoritma Hu dkk. (2009) untuk membandingkan representasi *control flow graph*.

Representasi *control flow graph* yang telah dibuat akan digunakan untuk membuat sebuah *cost matrix* yang menggambarkan *cost* memetakan sebuah *node* menjadi *node* lain antara dua buah *control flow graph*, serta *cost* untuk menghapus sebuah *node*. Deskripsi dari masing masing submatriks pada *cost matrix* adalah:

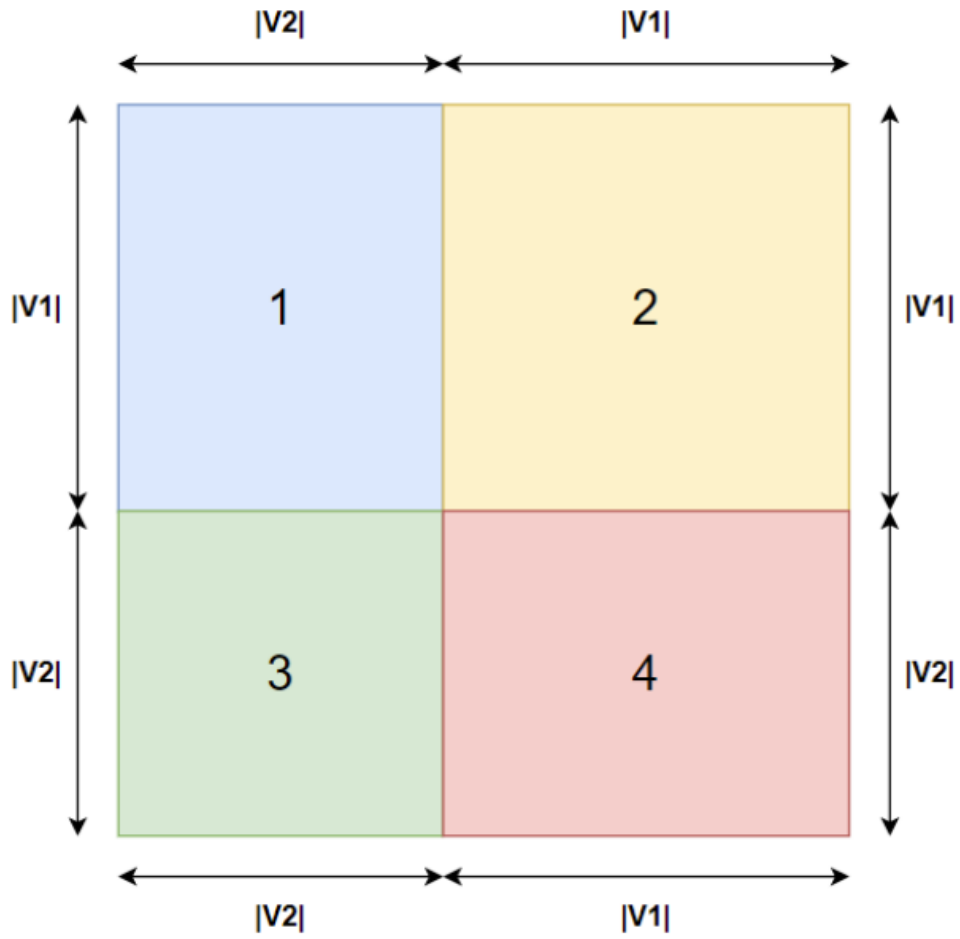
1. Submatriks kiri atas merupakan pemetaan *node* pada G_1 ke *node* pada G_2. Nilai dari setiap elemen adalah *cost* yang dibutuhkan untuk mengubah *node* indeks i pada G_1 menjadi *node* indeks j pada G_2.
2. Submatriks kanan atas merupakan pemetaan *node* pada G_1 ke *dummy node*. Nilai dari setiap elemen dapat diartikan sebagai *cost* yang dibutuhkan untuk menghapus node indeks i pada G_1.
3. Submatriks kiri bawah merupakan pemetaan *node* pada G_2 ke *dummy node*. Nilai dari setiap elemen dapat diartikan sebagai *cost* yang dibutuhkan untuk menghapus node indeks j pada G_2
4. Submatriks kanan bawah merupakan pemetaan antar *dummy node*, dan pada prosesnya hanya digunakan untuk memenuhi struktur *cost matrix*.

Langkah-langkah pembangunan *cost matrix* adalah sebagai berikut:

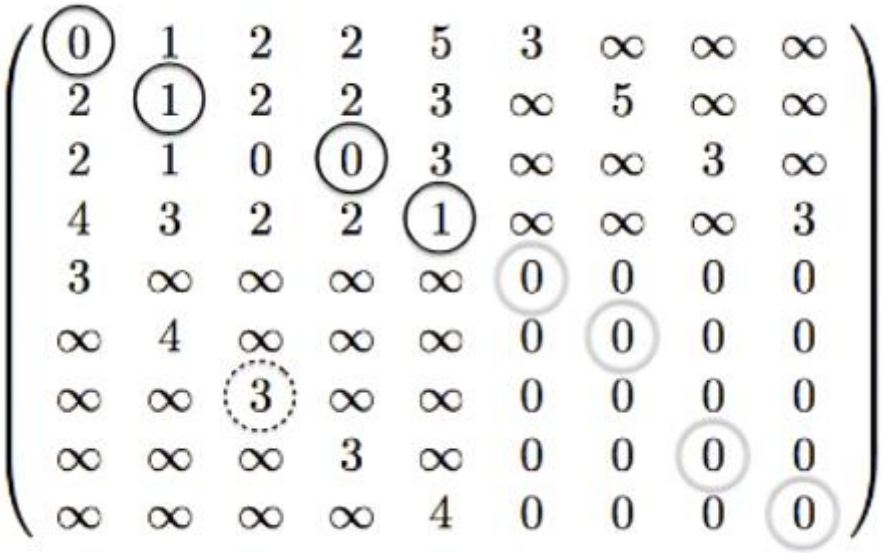
1. Untuk G_1 merupakan *control flow graph* pertama dengan jumlah *node* V_1, dan G_2 merupakan *control flow graph* kedua dengan jumlah *node* V_2, bentuk matriks seukuran $(|V_1| + |V_2|) \times (|V_1| + |V_2|)$.
2. Isi seluruh elemen matriks dengan nilai awal 0.
3. Untuk masing-masing elemen pada submatriks pertama di kiri atas seukuran $|V_1| \times |V_2|$ ganti nilai masing-masing elemen menggunakan rumus (II-1.

4. Untuk masing-masing elemen pada submatriks kedua di kanan atas seukuran $|V_1| \times |V_1|$, ganti nilai masing-masing elemen menggunakan rumus (II-2. Untuk setiap kasus ketika nilai i tidak sama dengan j, ganti nilai elemen dengan ∞
5. Untuk masing-masing elemen pada submatriks ketiga di kiri bawah seukuran $|V_2| \times |V_2|$, ganti nilai masing-masing elemen menggunakan rumus (II-2, namun indeks i pada OE dan IE diganti dengan indeks j. Untuk setiap kasus ketika nilai i tidak sama dengan j, ganti nilai elemen dengan ∞.

Karena elemen pada *cost matrix* di-inisialisasi dengan 0, maka tidak diperlukan perlakuan khusus untuk submatriks keempat di kanan bawah. Contoh visualisasi dari *cost matrix* yang digunakan dapat dilihat pada Gambar II.6 Setelah *cost matrix* selesai terbentuk, maka jumlah *cost* akan dihitung menggunakan algoritma Hungaria (Hungarian Algorithm) sesuai dengan algoritma yang digunakan oleh Hu dkk. (2009). Visualisasi penggunaan algoritma tersebut dapat dilihat pada Gambar II.7 *Cost* yang dipilih akan dihitung menjadi nilai *whitebox* yang melambangkan kemiripan kode hasil submisi peserta didik terhadap kode referensi dari pendidik.



Gambar II.6 Visualisasi *Cost Matrix* yang Digunakan Pada Algoritma Hu dkk. (2009).



Gambar II.7 Visualisasi pemilihan *cost* minimal pada *cost matrix* menggunakan algoritma Hungaria

Sumber: (Chan, P. P. F., & Collberg, C., 2014)

II.7 *Web Service*

Web service adalah aplikasi modular mandiri yang dapat diakses melalui internet (Curbera & Nagy, 2001). *Web service* tersebut merupakan realisasi paling populer dari *service-oriented-architecture*. *Web service* ini biasanya berupa perangkat lunak yang dapat dipanggil melalui *web* melalui format tertentu seperti XML (eXtensible Markup Language) dan JSON (Javascript Object Notation), beberapa operasi beserta format input dan output tersebut dijelaskan menggunakan WSDL (*Web Services Description Language*) (Curbera, 2001).

Web service menyediakan komunikasi yang lebih baik diantara beberapa aplikasi dan *computing platform* (Shahidul Islam, Ramesh Kumar, & Ab Rashid Dar, 2018). Aplikasi kini dapat mengkonsumsi *web service* dengan mengikuti standar *web service* tersebut, tanpa perlu terintegrasi dengan *web service* tersebut secara menyeluruh.

II.7.1 Arsitektur *web service*

Terdapat 3 entitas yang ada pada arsitektur *web service* (T. Gauravra, 2017):

1. ***Service Provider***
   Pembuat layanan web dan yang melakukan publikasi ke dunia luar.
2. ***Service Requester***
   Konsumen yang melakukan permintaan dengan mengikuti standar suatu *web service* yang ditetapkan.
3. ***Registry***
   Pemelihara *registry* di mana penyedia layanan *web* dapat menerbitkan layanan baru atau menemukan yang sudah ada.

II.8 *Microservices Architecture*

Microservices Architecture – varian dari tipe arsitektur *service-oriented architecture (SOA)* – adalah arsitektur yang mengatur aplikasi sebagai kumpulan layanan yang digabungkan secara *loosely coupled*. Dalam *microservices architecture*, layanan bersifat *fine-grained* dan protokolnya *lightweight*. Tujuan dari arsitektur ini agar setiap *services* dapat memberdayakan layanan secara independen dari *service* yang lain.

Pandangan konsensus telah berkembang dari waktu ke waktu di industri. Beberapa karakteristik yang menentukan sebuah *microservices architecture* yang sering dikutip meliputi:

- *Services* dalam *microservices architecture* sering kali merupakan proses yang berkomunikasi melalui jaringan untuk memenuhi tujuan menggunakan protokol agnostik teknologi seperti HTTP (Sam Newman, 2015).
- *Services* diatur berdasarkan kemampuan bisnis (Cesare Pautasso 2017)
- *Services* dapat diimplementasikan menggunakan bahasa pemrograman, basis data, perangkat keras dan lingkungan perangkat lunak yang berbeda, tergantung pada apa yang paling cocok (Lianping Chen 2018)

- *Services* berukuran kecil, mendukung pesan, dibatasi oleh konteks, dikembangkan secara mandiri, dapat digunakan secara independent (Lianping Chen 2018)

II.9 Flask *Web Framework*

Flask adalah *web framework* yang ditulis dengan Python. Diklasifikasikan sebagai *microframework* karena tidak memerlukan alat atau pustaka tertentu. Flask tidak memiliki lapisan abstraksi database, validasi formulir, atau komponen lain di mana perpustakaan pihak ketiga yang sudah ada menyediakan fungsi umum. Namun, Flask mendukung ekstensi yang dapat menambahkan fitur aplikasi seolah-olah diimplementasikan di Flask itu sendiri. Terdapat ekstensi untuk pemetaan relasional objek, validasi formulir, penanganan unggahan, berbagai teknologi autentikasi terbuka, dan beberapa alat terkait kerangka umum. Flask ini mudah digunakan untuk membuat *web service* karena bersifat *lightweight* dan sederhana untuk dipakai.

BAB III
ANALISIS DAN PERANCANGAN SISTEM

III.1 Analisis Masalah

Sesuai dengan uraian pada BAB I, dalam membuat Sistem *autograder* Notasi Algoritmik diperlukan integrasi terhadap dua sistem dari hasil penelitian berikut:

1. Pembangkitan *Abstract Syntax Tree* (AST) dan *Control Flow Graph* (CFG) Notasi Algoritmik oleh Sofyana dkk. (2021)
2. Evaluasi Tugas Pemrograman dengan Pendekatan *Control Flow Graph* (CFG) oleh Sendjaja dkk. (2021)

Kemudian, perlu diketahui juga bahwa sistem *autograder* Notasi Algoritmik tidak melakukan eksekusi atau *compile* pada Notasi Algoritmik karena Notasi Algoritmik tidak dibuat untuk hal tersebut, sehingga sistem *autograder* Notasi Algoritmik yang dibuat akan mendekati cara penilaian otomatis *whitebox*.

Sistem *autograder* yang sudah dilakukan integrasi dari dua sistem tersebut kemudian harus dibuat menjadi *web service*, agar dapat digunakan oleh subsistem Integrasi *Autograder* Abstrak dengan Moodle dan GitLab demi mencapai tujuan *capstone* tugas akhir ini, yakni terbuatnya satu sistem *autograder* yang utuh.

Hasil CFG yang didapat dari hasil sistem Sofyana dkk. (2021) berbeda dengan CFG yang digunakan oleh sistem Sendjaja dkk. (2021), oleh karena itu diperlukan suatu utilitas penengah yang dapat menyelesaikan permasalahan ini. Walaupun kedua CFG yang digunakan tersebut berbeda, *source code* yang ada untuk membuat kedua sistem tersebut sama-sama dibuat dengan bahasa pemrograman Python. Selain itu, kedua CFG yang digunakan sebetulnya sama-sama dapat direpresentasikan menjadi sebuah *Directed Graph*. Oleh karena itu, untuk menyelesaikan permasalahan integrasi ini, dapat dibuat sebuah utilitas penengah berupa *Module Python* yang berisi beberapa *Class Python* yang dapat membantu integrasi antara kedua sistem seperti Kelas *Directed Graph* dan Kelas *Node*.

Sistem *whitebox autograder* untuk tulisan Notasi Algoritmik yang dihasilkan dari integrasi ini kemudian perlu dibuat menjadi *web service*. Seluruh implementasi sistem dibuat dengan bahasa pemrograman Python, oleh karena itu diperlukan penggunaan *library python* seperti *Flask* untuk membuat *web service* tersebut. Kemudian, *web service* yang dibuat ini hanya memerlukan beberapa *endpoint* yang dapat diakses, diantaranya adalah *endpoint health check*, yang digunakan untuk melakukan pengecekan *health web service*, kemudian perlu juga ada *endpoint* untuk menerima *file* tulisan Notasi Algoritmik beberapa referensi pengajar dengan *file* tulisan Notasi Algoritmik pelajar yang kemudian mengembalikan hasil *response* berupa nilai dan informasi lainnya yang dibutuhkan. Hasil nilai ini didapat dari nilai kemiripan CFG beberapa hasil CFG pengajar dengan hasil CFG pengajar.

III.2 Analisis Solusi

Berdasarkan masalah yang telah disebutkan pada subbab sebelumnya, dalam membuat *Autograder* Notasi Algoritmik Berbasis *Control Flow Graph* terdapat dua hal utama yang perlu dilakukan dalam analisis solusi:

1. Analisis Solusi Integrasi Sistem Hasil Penelitian
2. Analisis Solusi Pembuatan *Web Service*

Setelah didapat analisis solusi dari masing-masing point tersebut, dibuat sebuah rancangan sistem untuk menjadi referensi dalam melakukan implementasi.

III.2.1 Analisis Solusi Integrasi Sistem Hasil Penelitian

Dalam melakukan integrasi dari kedua sistem berikut:

1. Pembangkitan *Abstract Syntax Tree* (AST) dan *Control Flow Graph* (CFG) Notasi Algoritmik oleh Sofyana dkk. (2021)
2. Evaluasi Tugas Pemrograman dengan Pendekatan *Control Flow Graph* (CFG) oleh Sendjaja dkk. (2021)

Dapat dibuat sebuah *Module Python* Bernama *Intermediate Module* yang merupakan penengah sekaligus penyatu dari kedua sistem tersebut. Beberapa hal yang perlu ada dalam *Intermediate Module* tersebut adalah:

1. *Class Node*

   Class Node merupakan kelas yang berfungsi untuk merepresentasikan *node* pada suatu *graph*. Pada kelas ini dapat didefinisikan beberapa atribut sebagai berikut:

   a. *Information*

      Atribut *information* pada *Node* berfungsi dalam merepresentasikan potongan tulisan notasi algoritmik yang didapat dari submisi notasi algoritmik.

   b. *Adjacent node list*

      Atribut *adjacent node list* berfungsi untuk mengasosiasikan setiap simpul dalam graf dengan kumpulan simpul atau sisi tetangganya. Dengan adanya *adjacent node list*, bisa dilakukan penelurusan suatu graph dari suatu *node*. Pada kasus implementasi ini, apabila *node* V berada pada *adjacent list node* U, maka terdapat *edge* dari U ke V, namun belum tentu sebaliknya.

   c. *In nodes list*

      Pada dasarnya *in nodes list* ini merupakan kebalikan dari *adjacent node list*, sehingga apabila *node U* berada pada *in nodes list node V* maka terdapat *edge* dari *U* ke *V*. Atribut ini akan berfungsi dalam algoritma *grading* berbasis *control flow graph* yang akan digunakan.

   d. *Label*

      Atribut *label* merupakan suatu *identification number* dari suatu *Node*. Atribut ini berguna untuk membedakan suatu *node* dengan *node* lainnya.

   Pada *Class Node* tersebut dapat dibuat beberapa *method* yang menggunakan atribut *Node* seperti *method traversal* yang berguna dalam melakukan penelurusan graf.

2. *Class Graph*

   Class Graph merupakan sebuah kelas yang berfungsi untuk merepresentasikan suatu *graph*. Pada sistem yang dibuat ini, karena *graph* yang digunakan merupakan sebuah *control flow graph*, maka implementasi

graph yang diperlukan haruslah berupa suatu *Directed Graph*, oleh karena itu dalam pembuatan atribut dan metode harus dilakukan penyesuaian yang memenuhi kriteria suatu *Directed Graph*. Pada kelas ini dapat didefinisikan beberapa atribut sebagai berikut:

a. *Nodes List*

   Atribut *nodes list* menyatakan *nodes* yang dimiliki suatu graf.

b. *Label to Node*

   Atribut *label to node* berfungsi dalam memetakan suatu *label* pada suatu *node* tertentu. Atribut ini berguna dalam memudahkan pengambilan *node* berdasarkan suatu *label*.

Pada *Class Graph* ini dapat dibuat *method* yang dapat mengubah representasi *Graph* sistem yang dibuat Sofyana dkk. (2021) dan Sendjaja dkk. (2021) ke dalam *Class Graph* ini.

3. *Class Constant*

   Class Constant merupakan sebuah kelas yang berisi beberapa nilai-nilai *constant* yang dapat digunakan pada keseluruhan sistem, contoh dari *constant* ini adalah *maximum score* yang bisa didapat saat melakukan *autograding*.

Sebelum membuat *Intermediate Module* tersebut, tentunya masing-masing dari hasil sistem penelitian perlu ditempatkan pada *source code* keseluruhan sistem dan dibuatkan *Module*-nya sendiri, sehingga untuk setiap hasil sistem penelitian dapat dibuat *module* sebagai berikut:

1. *CFG Generator Module*

   CFG Generator Module ini merupakan *module* yang diambil dari Sistem Sofyana dkk. (2021). Pada *module* ini, hal utama yang diperlukan adalah fungsi *CFG Generator* yang menerima parameter berupa tulisan Notasi Algoritmik dan membangkitkannya menjadi sebuah *CFG*. Hasil dari *CFG* ini kemudian dapat dikonversi menjadi *Class Graph* yang berada pada *Intermediate Module*.

2. *Graph Grader Module*

Graph Grader Module ini merupakan *module* yang diambil dari Sistem Sendjaja dkk. (2021). Pada *module* ini, hal utama yang diperlukan adalah fungsi *Graph Collapser* dan fungsi *Graph Comparator*. Fungsi *Graph Collapser* bertujuan dalam menyatukan beberapa *node* yang berada pada satu *flow* menjadi satu *node* gabungan. Kemudian, untuk Fungsi *Graph Comparator* ini menerima beberapa *Class Graph* submisi pendidik dan submisi peserta didik, untuk setiap *Class Graph* submisi pendidik akan dinilai kemiripan *Class Graph* tersebut dengan *Class Graph* peserta didik, dan hasil yang dikeluarkan adalah nilai terbesar yang didapatkan dari setiap *Class Graph* pendidik dengan *Class Graph* peserta didik.

Untuk kedua *module* tersebut perlu dilakukan modifikasi dari sistem hasil penelitian, untuk menyesuaikan dengan kebutuhan keseluruhan sistem yang akan dibuat.

III.2.2 Analisis Solusi Pembuatan *Web Service*

Dalam membuat *web service* untuk sistem yang akan dibuat, dapat digunakan *Flask*, sebuah *library python* dalam membuat *web service*. Pada dasarnya, *web service* ini hanya perlu memberikan beberapa *endpoint* yang dapat digunakan oleh subsistem lain dari keseluruhan sistem Tugas Akhir *Capstone*. Beberapa *endpoint* yang perlu ada pada *web service* ini diantaranya adalah:

1. *Endpoint Health Check*

   Pada *endpoint* ini, dapat dilakukan *GET Request* yang akan mengembalikan *response* berupa sebuah validasi yang menyatakan *web service* sudah berjalan atau tidak.
2. *Endpoint Description*

   Pada *endpoint* ini, dapat dilakukan *GET Request* yang akan mengembalikan *response* berupa informasi dari *web service* untuk dapat digunakan oleh subsistem Integrasi *Autograder* Abstrak dengan Moodle dan GitLab.
3. *Endpoint CFG Generator*

Pada *endpoint* ini, dapat dilakukan *POST Request* berisi tulisan Notasi Algoritmik. Hasil dari *request* tersebut akan mengambilkan *response* berupa sebuah CFG dalam bentuk JSON. *Endpoint* ini berfungsi untuk mengecek apakah suatu tulisan Notasi Algoritmik sudah dapat dibuat menjadi CFG untuk dilakukan *grading*.

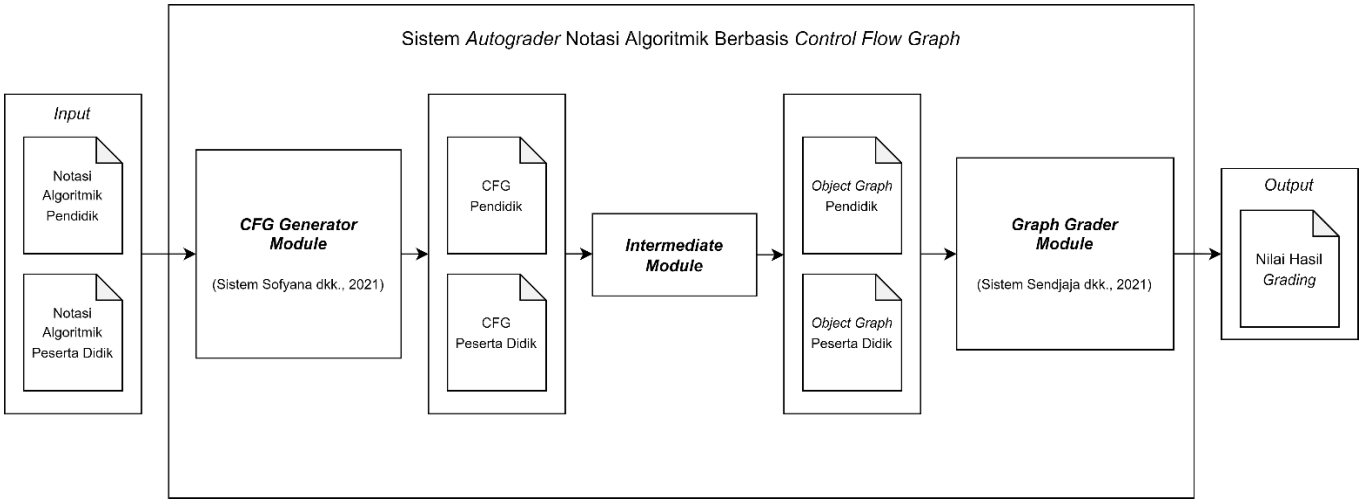
4. *Endpoint Grade*

   Pada *endpoint* ini, dapat dilakukan *POST Request* berisi submisi beberapa notasi algoritmik pendidik dan submisi notasi algoritmik peserta didik. Hasil dari *request* tersebut akan mengambilkan *response* berupa hasil penilaian yang didapat dari *grading*.

Dalam pembuatan *web service* ini, dapat dilakukan *containerization* dengan menggunakan Docker agar sistem lain dapat mudah menjalakan *web service* tersebut tanpa harus melakukan *setup* yang diperlukan. Selain itu, perlu dilakukan penyesuaian format *request* dan *response* untuk menyesuaikan dengan satu sistem *capstone*.

III.3 Rancangan Solusi

Pada bagian ini akan dijelaskan rancangan-rancangan solusi berdasarkan analisis solusi yang telah dibuat. Rancangan solusi ini kemudian menjadi referensi dalam pembuatan implementasi sistem yang akan dibuat. Dalam membuat Sistem *Autograder* Notasi Algoritmik Berbasis *Control Flow Graph*. Secara umum, sistem perlu memanfaatkan beberapa *modules* yang akan dibuat, sehingga sistem yang dibangun ini memiliki diagram umum yang digambarkan pada Gambar III.1.



Gambar III.1 Diagram Blok Sistem *Autograder* Notasi Algoritmik Berbasis *Control Flow Graph*

Berikut merupakan penjelasan sistem secara umum:

1. Sistem menerima *request* berupa *file* tulisan notasi algoritmik pendidik dan *file* tulisan notasi algoritmik peserta didik
2. *Files* tersebut kemudian akan dibangkitkan menjadi CFG oleh *CFG Generator Module* yang merupakan perkembangan dari Sistem Sofyana dkk. (2021)
3. Hasil dari pembangkitan CFG kemudian diubah menjadi *Class Graph* oleh *Intermediate Module*
4. *Class Graph* kemudian akan dilakukan *grading* oleh *Graph Grader* Module yang merupakan perkembangan dari Sistem Sendjaja dkk. (2021).
5. Hasil dari *grading* kemudian dikembalikan menjadi *response* dari sistem.

Untuk setiap bagian pada diagram tersebut, akan diberikan rancangan lebih detil untuk menjelaskan sistem lebih baik.

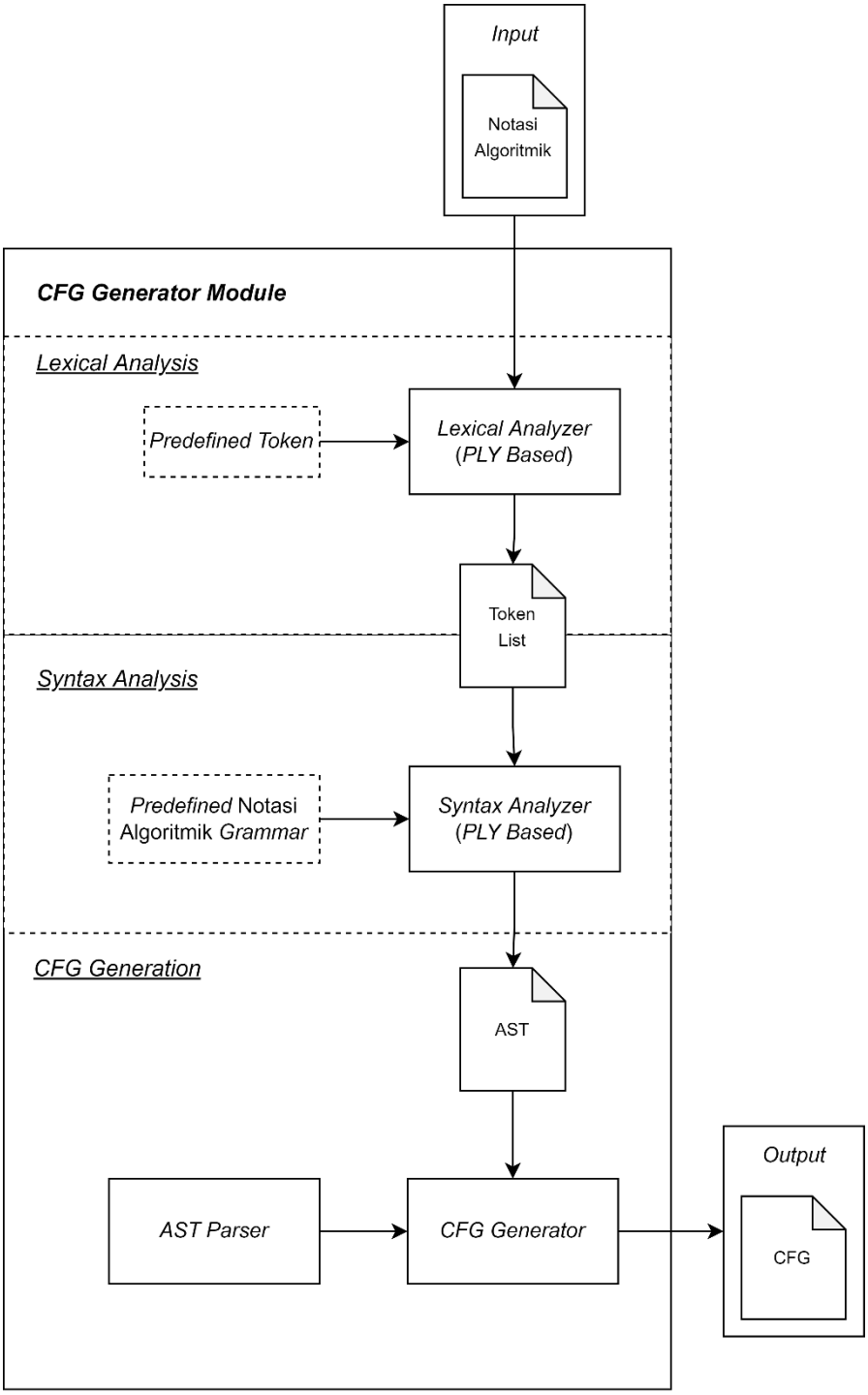
III.3.1 Rancangan *CFG Generator Module*

CFG Generator Module adalah *module* untuk membangkitkan tulisan Notasi Algorimtik menjadi *Control Flow Graph* (CFG). Pada intinya, *module* ini merupakan perkembangan dari Sistem Sofyana dkk. (2021).

Arsitektur dari *module* ini mengikuti arsitektur sistem Sofyana dkk. (2021) pada Gambar II.4. Namun, pada sistem ini, hanya dibutuhkan bagian *backend* saja dari arsitektur tersebut. Sehingga sistem ini terdiri dari beberapa komponen sebagai berikut:

1. *Predefined Token*
2. Komponen *Lexical Analyzer*
3. *Predefined Grammar*
4. Komponen *Syntax Analyzer*
5. Komponen AST *Parser*
6. Komponen CFG *Generator*

Setiap komponen tersebut dapat dilihat penjelasannya pada Subbab II.4, mengenai Rancangan Sistem Sofyana dkk. (2021). Oleh karena itu, rancangan diagram *Module* ini dapat digambarkan oleh Gambar III.2.



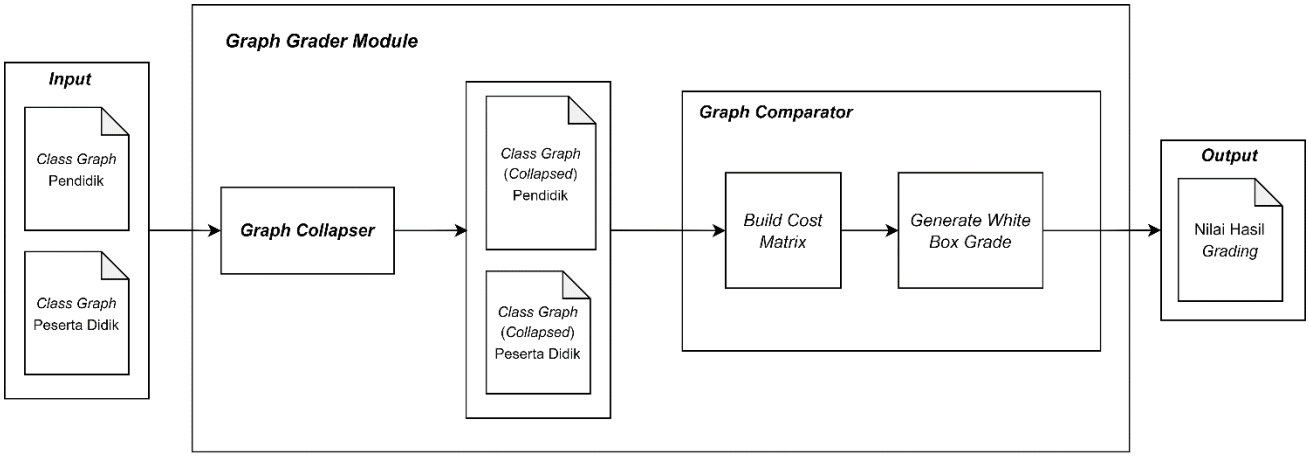
Gambar III.2 Rancangan Diagram *Graph Generator Module*

III.3.2 Rancangan *Graph Grader Module*

Graph Grader Module adalah *module* untuk melakukan penilaian *Class Graph* dengan melakukan komparasi antar *Class Graph*. Pada intinya, *module* ini merupakan perkembangan dari Sistem Sendjaja dkk. (2021).

Arsitektur dari *module* ini mengikuti arsitektur sistem Senjadaja (2021) pada Gambar II.5. Pada hasil Sendjaja dkk. (2021), sistem menerima CFG, namun pada sistem yang akan dibuat, sistem ini akan menerima *Class Graph*, oleh karena itu perlu dilakukan beberapa modifikasi pada implementasi.

Rancangan kelas untuk *module* ini digambarkan oleh Gambar III.3.



Gambar III.3 Rancangan Diagram *Graph Grader Module*

Graph Collapser yang sudah dilakukan implementasi pada sistem Sendjaja dkk. (2021) masih dapat ditingkatkan lebih baik, karena saat *graph* memiliki N simpul, maka dapat terjadi N^2 operasi pada kasus terburuk. Dalam menyelesaikan hal tersebut, dapat dilakukan implementasi *Graph Collapser* menggunakan sebuah data struktur efisien *Disjoint Set Union* (DSU). Dengan menggunakan DSU, operasi dalam menggabungkan dua data hanya membutuhkan kompleksitas waktu $O(1)$ secara rata-rata, sehingga akan terjadi N operasi saja pada kasus terburuk. Beberapa

hal yang ada pada *Disjoint Set Union* untuk menggabungkan *node* adalah sebagai berikut:

1. Atribut *parent* berfungsi untuk mengetahui referensi *node* mana yang sudah digabung.
2. Metode *find parent* berfungsi untuk mengetahui induk dari *node*, yakni yang menjadi perwakilan dari gabungan suatu *node*.
3. Metode *merge* berfungsi untuk menggabungkan dua buah *node*.
4. Metode *check same* berfungsi untuk melakukan pengecekan apakah *node* U sudah bergabung dengan *node* V.

Kemudian, dalam melakukan implementasi *Graph Collapser*, perlu digabungkan beberapa *node* yang berada pada satu *flow*. Misalkan U dan V masing-masing adalah *node* pada G, maka *node* U dan V berada dalam satu *flow* dan dapat digabungkan apabila memenuhi kriteria sebagai berikut.

1. Terdapat tepat 1 *outgoing edge* dari U.
2. Terdapat *edge* dari U menuju V.
3. Terdapat tepat 1 *ingoing edge* dari V.
4. Terdapat maksimal 1 *outgoing edge* dari V.

Dari kriteria tersebut, dapat dibuat sebuah *pseudocode* untuk melakukan implementasi *Graph Collapser* yang ditunjukkan pada Gambar III.4.

```
1: procedure COLLAPSE(G)
2:     nodes ← getNodes(G)
3:     pairNodes ← []                ▷ Used to store pair of nodes to be merged
4:     for node in nodes do
5:         adjNodes ← getAdjacent(nodes)
6:         if len(adjNodes) > 1 then
7:             continue
8:         end if
9:         for adjNode in adjNodes do
10:            if len(getAdjacent(adjNode)) > 1 then
11:                continue
12:            end if
13:            if len(getInNodes(adjNode)) ≤ 1 then
14:                pairNodes.push(node, adjNode)
15:            end if
16:        end for
17:    end for
18:    for (u, v) in pairNodes do    ▷ Node u and v will be in the same flow
19:        merge(u, v)               ▷ Can use DSU to merge this pair of nodes
20:    end for
21: end procedure
```

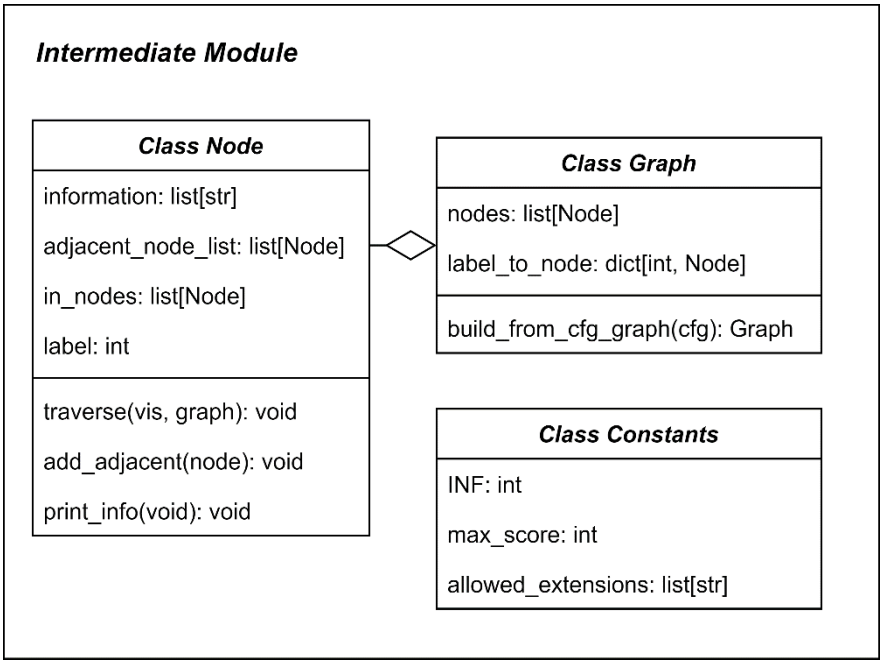
Gambar III.4 *Pseudecode Graph Collapser*

Kemudian, untuk implementasi *Graph Collapser* sendiri akan mengikuti sistem yang sudah, yakni menggunakan Algoritma Hu dkk. (2009)

III.3.3 Rancangan *Intermediate Module*

Intermediate Module merupakan *module* penengah sekaligus penyatu antar sistem Sofyana dkk. (2021), dengan sistem Sendjaja dkk. (2021). Fungsi utama dari *module* ini adalah mengubah *CFG* hasil *CFG Generator Module* menjadi *Class Graph* yang dapat digunakan oleh *Graph Grader Module*.

Module ini hanya memiliki beberapa kelas seperti *Class Node*, *Class Graph*, dan *Class Constants*. Oleh karena itu, rancangan kelas untuk *module* ini dapat digambarkan oleh Gambar III.5.

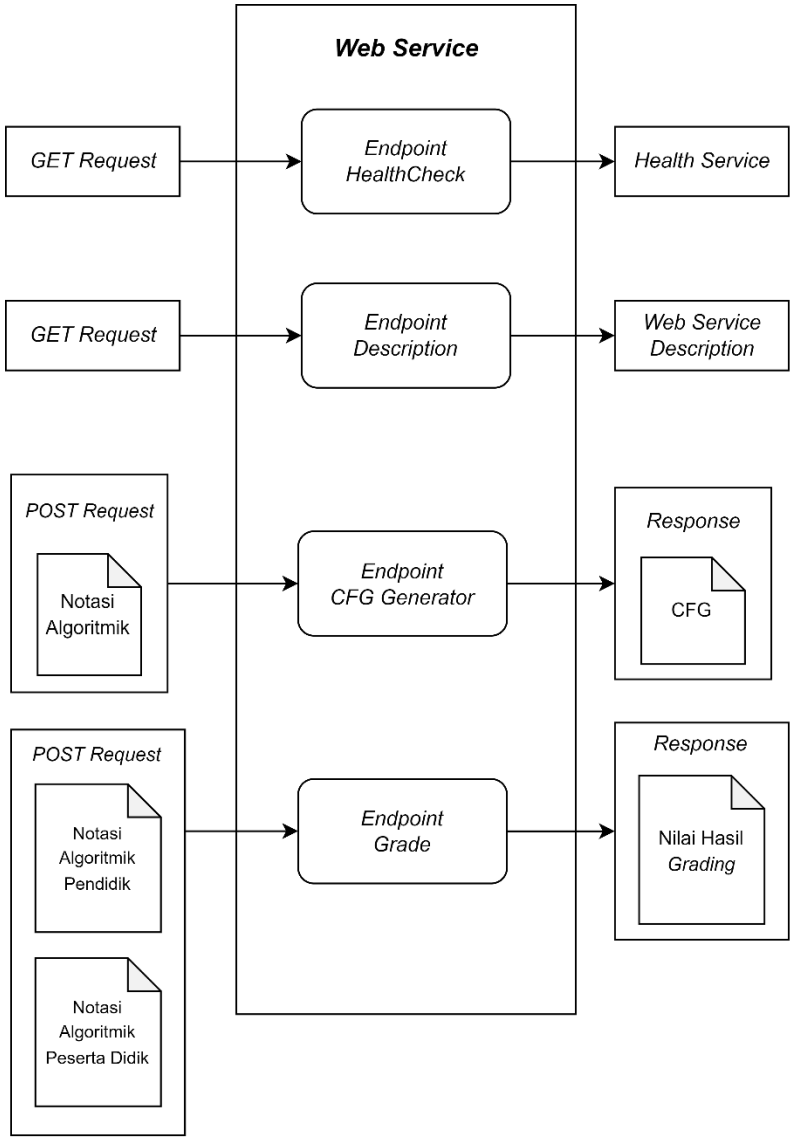


Gambar III.5 Rancangan Diagram Kelas *Intermediate Module*

Masing-masing *classes* sudah dijelaskan pada subbab Analisis Solusi bagian III.2.1.

III.3.4 Rancangan Pembuatan *Web Service*

Pada rancangan pembuatan *web service*, perlu membuat beberapa *endpoint* pada sistem agar dapat diakses oleh sistem lain pada keseluruhan sistem *capstone*. Rancangan *Endpoint Web Service* untuk sistem ini digambarkan pada Gambar III.6.



Gambar III.6 Rancangan *Endpoint Web Service*

Penjelasan setiap *endpoint* sudah dijelaskan pada subbab Analisis Solusi bagian III.2.2.

BAB IV
IMPLEMENTASI

Bagian ini akan menjelaskan secara rinci mengenai hasil implementasi yang dilakukan untuk pengerjaan tugas akhir ini. Penjelasan implementasi akan meliputi lingkungan implementasi, serta implementasi komponen-komponen sebagaimana yang telah dijelaskan pada pada BAB III.

IV.1 Implementasi CFG Generator Module

Dalam membuat implementasi *CFG Generator Module*, perlu dilakukan beberapa modifikasi dari Sistem Sofyana dkk. (2021). Secara umum, sistem yang sudah ada sudah cukup baik dalam melakukan pembangkitan CFG dari Notasi Algoritmik, sehingga fokus utama dalam implementasi *Module* ini adalah menyatukan-nya dengan *Intermediate Module*.

Untuk meningkatkan utilitas sistem Sofyana dkk. (2021) dilakukan terlebih dahulu proses pengujian data ujian Notasi Algoritmik peserta didik *autograder*. Perubahan yang terjadi terletak pada bagSetelah melaksanakan pengujian tersebut, akan dijelaskan secara singkat beberapa perbaikan dari Sistem Sofyana dkk. (2021) yang telah diimplementasikan pada *CFG Generator Module*. Beberapa perubahan ini utamanya terletak pada *predefined token*, *predefined grammar,* dan komponen pembangkitan CFG yang sudah dijelaskan pada subbab III.3.1. Tabel IV.1 menjelaskan masalah beserta perbaikan yang telah dilakukan tersebut.

Tabel IV.1 Kumpulan Masalah dan Perbaikan yang dilakukan pada Sistem Sofyana dkk. (2021)

No	Masalah	Perbaikan
1.	Tidak menerima karakter tab sebagai indentasi ("\t")	Memperbaiki proses *parsing* tulisan Notasi Algoritmik
2.	Fungsi tidak bisa menerima parameter dipisah dengan karakter koma	Memodifikasi *grammar function*
3.	Tidak bisa menggunakan "<--" dan "-->"	Menambahkan "<--" dan "-->" pada definisi *token*
4.	Prosedur tidak bisa menerima parameter dipisah dengan karakter koma	Memodifikasi *grammar procedure*
5.	Wajib mempunyai PROGRAM, KAMUS, dan ALGORITMA (sehingga tidak bisa dipakai untuk menguji fungsi secara langsung)	Menambahkan informasi serta mengubah cara kerja *CFG Generator* pada sistem Sofyana dkk. (2021)
6.	Tidak bisa menggunakan "<>" sebagai Not Equal	Menambahkan "<>" pada definisi token
7.	Blok ALGORITME harusnya bisa digunakan juga (beberapa submisi seperti ini)	Menambahkan "ALGORITME" pada definisi token
8.	KAMUS dan KAMUS LOKAL seharusnya bisa digunakan *interchangeably* (Main hanya bisa KAMUS)	Memodifikasi *grammar* KAMUS
9.	Fungsi tidak bisa *standalone* (wajib mempunyai KAMUS LOKAL dan ALGORITMA)	Memodifikasi *grammar function*
10.	Tidak bisa langsung menguji fungsi / prosedur	Memodifikasi *grammar function / procedure* dan mengubah *source code* bagian penghandalan *function / procedure*
11.	Tidak bisa menggunakan /=	Menambahkan /= pada definisi token
12.	Traversal tidak bisa menerima *expression*	Memodifikasi *grammar traversal*
13.	Tidak bisa menggunakan fungsi / prosedur yang tidak terdefinisi	Merubah *source code* bagian pembangkitan *CFG* dalam penanganan fungsi / prosedur yang tidak terdefinisi
14.	Prosedur tidak bisa dilakukan *assignment*	Memodifikasi *grammar procedure*

IV.2 Implementasi *Graph Grader*

Dalam membuat implementasi *Graph Generator Module*, perlu dilakukan beberapa modifikasi dari Sistem Sendjaja dkk. (2021). Modifikasi utama yang perlu dilakukan adalah melakukan *grading* terhadap *Graph Class* buatan.

Pada *Module* ini dilakukan implementasi ulang dari awal, namun tetap melakukan referensi pada Sistem Sendjaja dkk. (2021). *Module* ini terdiri dari implementasi *Graph Collapser* dan *Graph Comparator*.

IV.2.1 Implementasi *Graph Collapser*

Graph Collapser adalah fungsi yang digunakan untuk menyederhanakan bentuk graf yang ada. pada intinya dua atau lebih *node* yang berada pada satu *flow* akan digabung menjadi satu *node* gabungan. Implementasi *Disjoint Set Union* dilaksanakan berdasarkan penjelasan rancangan solusi pada subbab III.3.2, dan dapat dilihat hasil implementasi tersebut pada Kode IV.1.

```
class DSU:
    def __init__(self, N: int):
        self.par = [i for i in range(N + 1)]

    def find_par(self, u: int):
        if self.par[u] == u:
            return u
        p = self.find_par(self.par[u])
        self.par[u] = p
        return p

    def merge(self, u: int, v: int):
        pu, pv = self.find_par(u), self.find_par(v)
        self.par[pv] = pu

    def check_same(self, u: int, v: int):
        pu, pv = self.find_par(u), self.find_par(v)
        return pu == pv
```

Kode IV.1 Kode Implementasi *Disjoint Set Union*

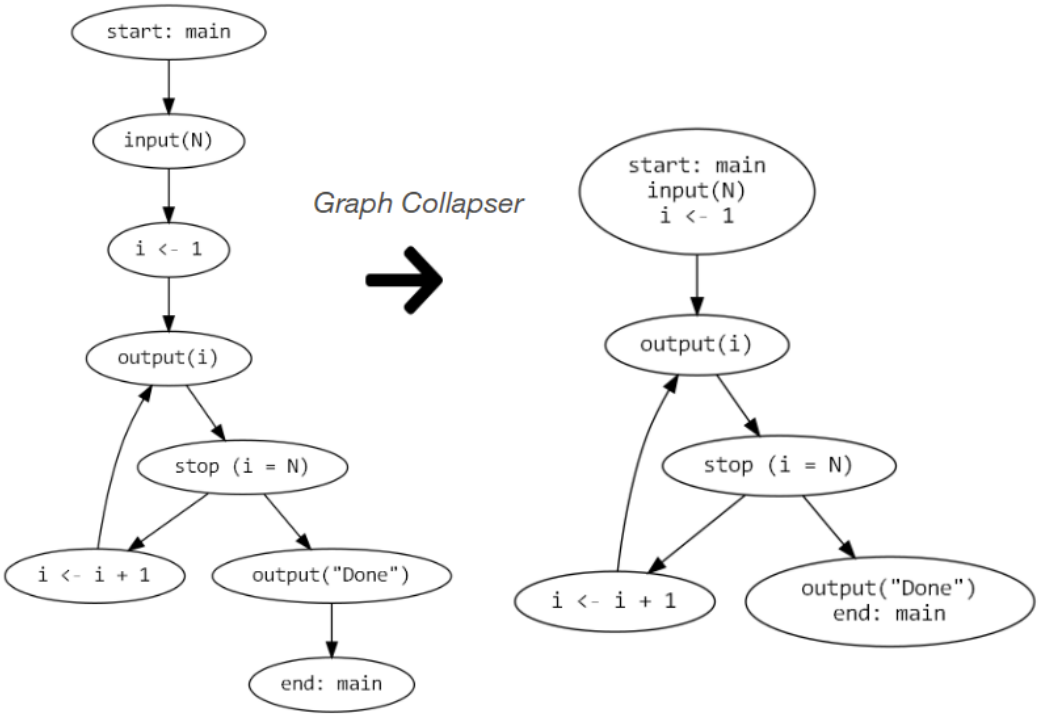
Hasil implementasi dari fungsi *Graph Collapser* mengikuti *pseudocode* rancangan solusi dan dapat dilihat implementasinya pada Kode IV.2.

```
from graph_grader.src.utils.dsu import DSU
from intermediate.src.classes.graph import Graph


def collapse(g: Graph):
    nodes = g.get_nodes()
    dsu = DSU(len(nodes))

    # Merge nodes with the same flow
    for node in nodes:
        adjacent = node.get_adjacent()
        if len(adjacent) > 1:
            continue
        label = node.get_label()
        for adj_node in adjacent:
            if len(adj_node.get_adjacent()) > 1:
                continue
            adj_label = adj_node.get_label()
            if len(adj_node.get_in_nodes()) <= 1:
                dsu.merge(label, adj_label)

    # Store information from merged nodes
    add_infos = {}
    for node in nodes:
        label = node.get_label()
        par_label = dsu.find_par(label)
        if label == par_label:
            continue
        if par_label not in add_infos:
            add_infos[par_label] = []
        for info in node.get_info():
            add_infos[par_label].append(info)

    # Add information to parent node
    for node in nodes:
        label = node.get_label()
        if label in add_infos:
            for info in add_infos[label]:
```

```
                node.add_info(info)

    # Get new adjacency list
    new_adj = {}
    for node in nodes:
        label = node.get_label()
        par_label = dsu.find_par(label)
        for adj_node in node.get_adjacent():
            adj_label = adj_node.get_label()
            par_adj_label = dsu.find_par(adj_label)
            if dsu.check_same(par_label, par_adj_label):
                continue
            if par_label not in new_adj:
                new_adj[par_label] = []
            new_adj[par_label].append(par_adj_label)

    # Reset adjacent and in nodes list for every node
    for node in nodes:
        node.set_adjacent([])
        node.set_in_nodes([])

    # Get new nodes and add new adjacent for every node
    new_nodes = []
    for node in nodes:
        label = node.get_label()
        par_label = dsu.find_par(label)
        if label != par_label:
            continue
        new_nodes.append(node)
        if par_label in new_adj:
            for adj_label in new_adj[par_label]:
                node.add_adjacent(g.get_node(adj_label))

    # Relabel Node
    for idx, node in enumerate(new_nodes):
        node.set_label(idx + 1)

    # Set new node to graph
    g.set_nodes(new_nodes)
```

Kode IV.2 Kode Implementasi *Graph Collapser*

Kemudian, Gambar IV.1 menunjukkan ilustrasi hasil dari *Graph Collapser* pada suatu *control flow graph*.



Gambar IV.1 Ilustrasi Hasil dari *Graph Collapser*

IV.2.2 Implementasi *Graph Comparator*

Graph Comparator merupakan fungsi yang dapat menilai kemiripan antar dua graf. Pada dasarnya, fungsi ini mengikuti implementasi *graph comparator* dari sistem Sendjaja dkk. (2021), yang sudah dijelaskan pada subbab II.6.

Pertama-tama, perlu dibuat terlebih dahulu fungsi untuk membuat *cost matrix*. Hasil implementasi fungsi *cost matrix* ini dapat dilihat pada Kode IV.3.

```
from intermediate.src.classes.constants import Constants
from intermediate.src.classes.graph import Graph


def get_cost_real(a: int, b: int):
```

```
    return a + b - (2 * min(a, b))


def create_cost_matrix(g1: Graph, g2: Graph):
    # Get number of nodes
    nodes1, nodes2 = g1.get_nodes(), g2.get_nodes()
    n, m = len(nodes1), len(nodes2)

    # Initialize cost matrix
    cost_matrix = [[0 for _ in range(n + m)] for _ in range(n + m)]

    # Fill section 1 (real node to real node)
    for i, u in enumerate(nodes1):
        for j, v in enumerate(nodes2):
            inp1 = len(u.get_in_nodes())
            inp2 = len(v.get_in_nodes())
            out1 = len(u.get_adjacent())
            out2 = len(v.get_adjacent())
            cost_matrix[i][j] = get_cost_real(inp1, inp2) +
get_cost_real(out1, out2)

    # Fill section 2 (delete node in graph 1)
    for i, u in enumerate(nodes1):
        for j in range(n):
            if i == j:
                ie = len(u.get_in_nodes())
                oe = len(u.get_adjacent())
                cost_matrix[i][j + m] = 1 + ie + oe
            else:
                cost_matrix[i][j + m] = Constants.INF

    # Fill Section 3 (delete node in graph 2)
    for i, u in enumerate(nodes2):
        for j in range(m):
            if i == j:
                ie = len(u.get_in_nodes())
                oe = len(u.get_adjacent())
                cost_matrix[i + n][j] = 1 + ie + oe
            else:
                cost_matrix[i + n][j] = Constants.INF

    return cost_matrix
```

Kode IV.3 Kode Implementasi Fungsi *Cost Matrix*

Setelah itu, dibuatlah implementasi *graph comparator* yang juga menggunakan fungsi *cost matrix* tersebut. Dalam pengujian yang telah dilakukan, ditemukan bahwa *score* yang didapat bisa bernilai negatif pada kasus membandingkan dua *graph* yang memiliki perbedaan *node* yang cukup besar, oleh karena itu dilakukan modifikasi pada *graph comparator* ini dengan memastikan bahwa *score* minimal yang didapat hanya dapat bernilai sampai 0. Hasil dari implementasi *Graph Comparator* ini dapat dilihat pada Kode IV.4.

```
from munkres import Munkres

from intermediate.src.classes.constants import Constants
from graph_grader.src.grader.cost_matrix import create_cost_matrix
from intermediate.src.classes.graph import Graph


def compare_graph(g1: Graph, g2: Graph):
    munkres = Munkres()
    cost_matrix = create_cost_matrix(g1, g2)
    indexes = munkres.compute(cost_matrix)

    total = 0
    details = []
    for i, j in indexes:
        val = cost_matrix[i][j]
        total += val
        details.append([i, j, val])

    node1 = len(g1.get_nodes())
    node2 = len(g2.get_nodes())
    edge1 = len(g1.get_edges())
    edge2 = len(g2.get_edges())
     score = max(0.0, (1 - (total / (node1 + node2 + edge1 + edge2))) *
Constants.MAX_SCORE)


    return score, total, details
```

Kode IV.4 Kode Implementasi *Graph Comparator*

IV.3 Implementasi Intermediate Module

Dalam membuat implementasi *Intermediate Module*, hanya perlu dibuat beberapa kelas, sebagaimana yang telah dijelaskan pada subbab 0.

IV.3.1 *Class Node*

Class Node yang diimplementasi bisa didapatkan dengan melakukan modifikasi implementasi *Class CFGNode* yang ada pada Sistem Sofyana dkk. (2021). *Class CFGNode* ini merupakan sebuah *Class* yang berada pada komponen *Backend*, dan digunakan dalam berbagai kode program lain seperti pada *Class CFG*. Setelah diselidiki, sebetulnya atribut dan metode yang ada pada *Class CFGNode* merujuk pada *Node* biasa pada graf umum, oleh karena itu dapat dilakukan modifikasi pada kelas ini untuk membuat *Class Node*. Beberapa modifikasi yang dilakukan diantaranya adalah:

1. Menambahkan Atribut *In Nodes* untuk digunakan pada *Module* yang lain seperti *Graph Grader Module*.
2. Menambahkan *type hints* untuk mempermudah proses pengembangan dan pencarian *bug* yang mungkin ada pada kode program.

Setelah dilakukan beberapa modifikasi tersebut, didapat hasil kode *Class Node* yang dapat dilihat pada Kode IV.5

```
class Node:
    def __init__(self, label: int, info=None):
        self.info: list[str] = []
        self.adjacent: list[Node] = []
        self.in_nodes: list[Node] = []
        self.label: int = label
        if info and len(info) > 0:
            self.info = info

    def add_in_nodes(self, node):
        if node not in self.in_nodes:
```

```
            self.in_nodes.append(node)

    def add_info(self, information: str):
        self.info.append(information)

    def add_adjacent(self, node):
        if node not in self.adjacent:
            self.adjacent.append(node)
            node.add_in_nodes(self)

    def print_info(self):
        for info in self.info:
            print(info)

    def get_label(self):
        return self.label

    def get_info(self):
        return self.info

    def get_info_str(self):
        str_info = ''
        for info in self.get_info():
            if str_info == '':
                str_info += info
            else:
                str_info += f'\n{info}'
        return str_info

    def get_in_nodes(self):
        return self.in_nodes

    def get_adjacent(self):
        return self.adjacent

    def set_label(self, label: int):
        self.label = label

    def set_adjacent(self, adjacent: list):
        self.adjacent = adjacent

    def set_in_nodes(self, in_nodes: list):
        self.in_nodes = in_nodes

    def traverse(self, is_visited: dict[int, bool], graph: dict):
```

```
        is_visited[self.label] = True
        if self not in graph:
            graph[self] = self.adjacent
        for adj in self.adjacent:
            if not is_visited[adj.get_label()]:
                adj.traverse(is_visited, graph)
```

Kode IV.5 Kode *Class Node*

IV.3.2 *Class Graph*

Class Graph ini dilakukan implementasi dengan mengikuti penjelasan pada subbab III.2.1. Hasil implementasi dapat dilihat pada Kode IV.6.

```
from intermediate.src.classes.node import Node


class Graph:
    def __init__(self):
        self.nodes: list[Node] = []
        self.label_to_node: dict[int, Node] = {}

    def add_node(self, node: Node):
        self.nodes.append(node)
        self.label_to_node[node.get_label()] = node

    def get_nodes(self):
        return self.nodes

    def get_node(self, label):
        if label not in self.label_to_node:
            raise KeyError("label not in graph")
        return self.label_to_node[label]

    def get_edges(self):
        edges: list[tuple[Node, Node]] = []
        for u in self.nodes:
            for v in u.get_adjacent():
                edges.append((u, v))
```

```
        return edges

    def get_clone(self):
        g = Graph()
        for u in self.nodes:
            new_node = Node(u.get_label())
            for i in u.get_info():
                new_node.add_info(i)
            g.add_node(new_node)

        for i, u in enumerate(self.nodes):
            cur_node = g.get_node(u.get_label())
            for v in u.get_adjacent():
                adj_node = g.get_node(v.get_label())
                cur_node.add_adjacent(adj_node)

        return g

    def set_nodes(self, nodes: list[Node]):
        self.nodes = []
        self.label_to_node = {}
        for node in nodes:
            self.add_node(node)

    def build_from_cfg_graph(self, cfg: dict):
        for node in cfg:
            self.add_node(node)

    def generate_to_cfg_graph(self):
        cfg = {}
        for node in self.nodes:
            cfg[node] = node.get_adjacent()
        return cfg
```

Kode IV.6 Kode *Class Graph*

IV.3.3 *Class Constants*

Class Constants merupakan sebuah kelas yang berisi beberapa *constants* yang diperlukan oleh seluruh sistem. Hasil implementasi *Class Constansts* dapat dilihat pada Kode IV.7.

```
import math


class Constants:
    INF = math.inf
    MAX_SCORE = 100
    ALLOWED_EXTENSIONS = {'txt', 'in', 'docx'}
```

Kode IV.7 Kode *Class Constants*

IV.4 Implementasi Pembuatan *Web Service*

Web Service dari sistem dibuat dengan menggunakan *Flask Framework*. Sebelum membuat beberapa *endpoint web service* yang dibutuhkan, dapat dibuat beberapa utilitas pembantu untuk membantu proses pengembangan serta membuat hasil setiap *endpoint* menjadi seragam. Beberapa utilitas pembantu tersebut adalah sebagai berikut:

1. Fungi *checker* yang berfungsi untuk melakukan validasi format *file* dan *request* yang diterima oleh *web service*.
2. Fungsi *Logger* yang berfungsi untuk menyimpan data-data proses yang dilakukan ketika *web service* sedang digunakan.
3. Fungsi *Wrapper* untuk melakukan standardisasi dari hasil *response* yang akan diberikan.

Endpoint yang akan dibuat mengikuti pemaparan pada subbab III.2.2, yang dapat dirangkum menjadi tabel yang dijelaskan pada Tabel IV.2.

Tabel IV.2 *Endpoint Web Service*

No	Metode	*Endpoint*	Deskripsi
1	*GET*	/health-check	Memeriksa status *server*
2	*GET*	/description	Mengembalikan deskripsi dari *web service*
2	*POST*	/notal-to-cfg	Membangkitkan tulisan Notasi Algoritmik menjadi *control flow graph*
3	*POST*	/grade	Melakukan penilaian *control flow graph* program dengan *autograder*

Kemudian, untuk setiap *endpoint* tersebut akan dibuat *Class Resources* untuk mengatur *request* yang dating. Dari setiap *resources* tersebut, akan ditambahkan ke dalam API *Web Service* agar menjadi *endpoint* yang bisa digunakan.

Setiap *response* yang diberikan oleh *web service* akan mengikuti standar satu tim *capstone* yang dapat dilihat penjelasannya pada Tabel IV.3.

Tabel IV.3 Struktur *Response Endpoint Server*

Atribut	Tipe Data	Wajib / Opsional	Keterangan
error	*Boolean*	Wajib	*Flag* yang menandakan apakah terjadi kegagalan
message	*String*	Wajib	Pesan sukses atau pesan informasi kegagalan yang terjadi pada sistem
data	*Object*	Opsional	Data hasil *request*

IV.4.1 Implementasi *Endpoint Health Check*

Pada *Endpoint Health Check* hanya diperlukan *GET Request* biasa untuk melakukan *hit* dari *endpoint* ini, kemudian apabila *web service* sudah berjalan, maka akan dikembalikan status *health* dari *web service* tersebut.

IV.4.2 Implementasi *Endpoint Description*

Pada *Endpoint Description*, dapat dilakukan *GET Request* untuk mendapatkan *response* berupa deskripsi dari *web service* yang berisi data mengenai *image name*, *display name*, dan *description* yang dapat digunakan oleh Integrasi *Autograder* Abstrak dengan Moodle dan GitLab.

IV.4.3 Implementasi *Endpoint CFG Generator*

Pada *Endpoint CFG Generator*, diperlukan penghandalan *POST request* dari suatu *file* Notasi Algoritmik, kemudian dikembalikan hasil CFG dalam bentuk JSON sebagai *response*. Struktur *request* dari *endpoint* ini dapat dilihat pada Tabel IV.4.

Tabel IV.4 Struktur *Request Endpoint CFG Generator*

Atribut	Tipe Data	Keterangan
solution	*String*	*File* solusi submisi tulisan Notasi Algoritmik peserta didik yang di-*encode* dengan base64
solutionFileName	*String*	Nama *file* dari solusi submisi peserta didik
timeLimit	*Integer*	Batas waktu untuk melakukan pembangkitan *CFG* dari tulisan Notasi Algoritmik

Hasil CFG tersebut didapat dengan memanggil fungsi dari *CFG Generator Module*.

IV.4.4 Implementasi *Endpoint Grade*

Pada *Endpoint Grade* diperlukan penghandalan *POST request* dari beberapa *file* Notasi Algoritmik pendidik dengan *file* Notasi Algoritmik peserta didik, kemudian dikembalikan hasil *grading* tersebut sebagai *response*. Struktur *request* dari *endpoint* ini dapat dilihat pada Tabel IV.5.

Tabel IV.5 Struktur *Request Endpoint Grade*

Atribut	Tipe Data	Keterangan
references	*List of String*	List *file* referensi solusi tulisan Notasi Algoritmik pendidik yang di-*encode* dengan base64
solution	*String*	*File* solusi submisi tulisan Notasi Algoritmik peserta didik yang di-*encode* dengan base64
referencesFileNames	*List of String*	List nama *file* dari solusi referensi
`solutionFileName`	*String*	Nama *file* dari solusi submisi peserta didik
`timeLimit`	*Integer*	Batas waktu untuk melakukan penilaian dalam satuan milisekon

Hasil *grading* tersebut didapat dengan memanggil fungsi dari *Module* yang sudah dibuat.

IV.4.5 Implementasi *Containerization* dengan Docker

Web service yang telah dibuat kemudian dapat dilakukan *containerization* dengan docker agar sistem terisolasi dan dapat mudah dijalankan pada konfigurasi sistem yang berbeda.

Dalam proses melakukan implementasi ini, dikumpulkan terlebih dahulu *requirements python package* yang ada pada *web service* kedalam sebuah file bernama *requirements.txt*.

Kemudian, *web service* dapat dicoba dijalankan melalui *python virtual environment* untuk memastikan *web service* berjalan dengan benar.

Setelah itu, dibuat sebuah *Dockerfile* yang digunakan untuk membangun instruksi pembuatan *image* dari *web service* yang telah dibuat. Hasil dari *image* tersebut kemudian dapat digunakan Docker untuk dapat menjalankan *web service* pada berbagai konfigurasi sistem yang berbeda. *Dockerfile* yang dibuat dapat dilihat implementasinya pada Kode IV.8.

```
FROM python:3.9

WORKDIR /app
COPY . .
ENV PYTHONPATH "."

RUN python3.9 -m pip install -r requirements.txt

EXPOSE 5000

ENTRYPOINT ["python3.9"]
CMD ["web_service/src/main.py"]
```

Kode IV.8 Kode Implementasi *Dockerfile*

BAB V
PENGUJIAN

V.1 Batasan Pengujian

Pengujian dalam *autograding* Notasi Algoritmik dilakukan dengan hanya menguji aspek-aspek yang dinilai penting pada Notasi Algoritmik.

V.2 Tujuan Pengujian

Tujuan dari pengujian ini adalah mengukur akurasi penilaian dari sistem *autograder* Notasi Algoritmik berbasis *Control Flow Graph* terhadap hasil penilaian manual pendidik.

V.3 Proses Pengujian

Proses pengujian yang dilakukan untuk tujuan pertama adalah sebagai berikut:

1. Dikumpulkan terlebih dahulu kumpulan tulisan notasi algoritmik peserta didik yang dapat dibangkitkan menjadi *control flow graph*. Hal ini dilakukan secara manual, karena beberapa dari pengerjaan peserta didik perlu dilakukan perubahan agar dapat diuji, namun perubahan tersebut dilakukan tanpa mengubah aspek-aspek penting yang dinilai.
2. Perangkat lunak penguji akan menjalankan algoritma perbandingan *control flow graph* pada kumpulan kode hasil submisi peserta didik dan kode referensi yang diberikan oleh pendidik. Jika ada lebih dari satu kode referensi yang diberikan pendidik, maka program akan membandingkan kode hasil submisi dengan seluruh kode referensi, dan nilai yang paling tinggi akan disimpan sebagai hasil akhir nilai *grading* peserta didik tersebut.
3. Perangkat lunak penguji kemudian akan membuat file CSV, dengan *header* sebagai berikut:
    a. *Exam Name*: Menyatakan nama ujian yang dilakukan, seperti UTS.

b. *Number*: Menyatakan nomor ujian yang dilakukan *grading* seperti IA pada UTS.
c. NIM: Menyatakan Nomor Induk Mahasiswa dari peserta didik.
d. *Grade*: Menyatakan hasil nilai *auto grading*.

4. Hasil CSV tersebut kemudian dapat dimasukkan ke dalam *google sheets* yang sudah berisi hasil penilaian manual peserta didik yang dilakukan oleh pendidik.
5. Pada *google sheets* tersebut diolah *sheets* baru yang berisi perbandingan dari penilaian *autograder* dengan penilaian manual oleh pendidik.

Pada pengerjaan tugas akhir ini, pemberian nilai manual akan diberikan oleh Bapak Satrio Adi Rukmono, S.T., M.T. selaku dosen dan ahli dalam bidang terkait.

V.4 Data Uji

Data uji yang akan digunakan merupakan Data Jawaban Kuis 2 dan UTS Peserta Mata Kuliah IF2110 Algoritma Struktur Data 2020. Lebih rincinya, data uji ini meliputi 5 persoalan:

1. Kuis 2 IF2110 2020 Nomor IA
2. Kuis 2 IF2110 2020 Nomor IB
3. Kuis 2 IF2110 2020 Nomor IC
4. UTS IF2110 2020 Nomor IA
5. UTS IF2110 2020 Nomor IB

Definisi dari persoalan tersebut akan dijelaskan lebih lanjut pada subbab V.5. Jumlah dari kode hasil submisi untuk setiap data uji dapat berbeda, sebab tidak semua kode hasil submisi dapat dibangkitkan dengan *control flow graph* karena keterbatasan perangkat lunak dan data uji. Kemudian, solusi referensi dari setiap data uji tersebut dapat dilihat pada Lampiran B.

V.5 Definisi Persoalan

Pada setiap data persoalan, peserta didik diminta untuk membuat sebuah tulisan notasi algoritmik yang melakukan realisasi terhadap fungsi / prosedur yang

diberikan. Dalam melakukan penulisan realisasi tersebut, peserta didik diminta untuk tidak membuat fungsi atau prosedur baru

V.5.1 Persoalan Kuis 2 IF2110 2020 Nomor IA

Gambar V.1 menjelaskan deskripsi fungsi yang perlu direalisasikan pada persoalan Kuis 2 IF2110 Nomor IA.

```
function SearchX (L: List, X: infotype) → boolean
{ Mengembalikan true jika X ada pada list L, false jika tidak (termasuk kasus
  list L kosong), dengan L berupa list terurut membesar dan berelemen unik. }
```

Gambar V.1 Persoalan Kuis 2 IF2110 2020 Nomor IA

V.5.2 Persoalan Kuis 2 IF2110 2020 Nomor IB

Gambar V.2 menjelaskan deskripsi prosedur yang perlu direalisasikan pada persoalan Kuis 2 IF2110 Nomor IB.

```
procedure SortedIntersect (input L1, L2: List, output L3: List)
{ I.S. L1, L2 berupa list terurut membesar, berelemen unik, dan mungkin kosong.
  F.S. L3 berupa list dengan elemennya adalah irisan dari elemen L1 dan L2, dan
       terurut membesar (lihat ilustrasi Gambar 1). Jika ada alokasi elemen yang
       gagal, maka L3 berupa list kosong.}
```

Gambar V.2 Persoalan Kuis 2 IF2110 2020 Nomor IB

V.5.3 Persoalan Kuis 2 IF2110 2020 Nomor IC

Gambar V.3 menjelaskan deskripsi prosedur yang perlu direalisasikan pada persoalan Kuis 2 IF2110 Nomor IC.

```
procedure SortedMerge (input/output L1: List, input L2: List, output status: boolean)
{ I.S. L1, L2 terdefinisi dan tidak kosong, dengan L1 elemennya terurut membesar
       dan unik, L2 juga berelemen unik, akan tetapi belum tentu terurut.}
{ F.S. L1 sudah bertambah dengan elemen dari L2, tetap berelemen unik dan
       terurut membesar (lihat ilustrasi Gambar 2(c)). Jika ada alokasi elemen
       yang gagal, maka L1 berupa list tepat sebelum terjadi kegagalan alokasi
       (lihat ilustrasi Gambar 2(d)). status akan bernilai true jika dan hanya
       jika tidak ada alokasi gagal. }
```

Gambar V.3 Persoalan Kuis 2 IF2110 2020 Nomor IC

V.5.4 Persoalan UTS IF2110 2020 Nomor IA

Gambar V.4 menjelaskan deskripsi fungsi yang perlu direalisasikan pada persoalan UTS IF2110 Nomor IA.

```
function IsSegitigaBawah (M: MATRIKS) → boolean
{ Mengirimkan true jika M adalah matriks segitiga bawah,dengan M_ij = 0
  untuk semua i < j.
  Prekondisi: M adalah matriks bujur sangkar (berukuran n × n) }
```

Gambar V.4 Persoalan UTS IF2110 2020 Nomor IA

V.5.5 Persoalan UTS IF2110 2020 Nomor IB

Gambar V.5 menjelaskan deskripsi fungsi yang perlu direalisasikan pada persoalan UTS IF2110 Nomor IB.

```
procedure DetriMatriks (input M: MATRIKS, output Det: integer)
{ Menghitung nilai determinan matriks segitiga M, dimana determinant
  dihitung berdasarkan perkalian semua elemen diagonalnya.
  I.S. Matriks M terdefinisi;
  F.S. Det berisi nilai determinan matriks segitiga bawah M atau 0 jika
       bukan matriks segitiga bawah. }
```

Gambar V.5. Persoalan UTS IF2110 2020 Nomor IB

V.6 Hasil Pengujian dan Evaluasi

Untuk setiap pengujian persoalan yang dilakukan, akan diberikan informasi-informasi sebagai berikut.

1. Jumlah submisi peserta didik yang telah terkumpul.

   Jumlah ini didapatkan dengan mengambil submisi peserta didik yang telah dilakukan penyesuaian dan dapat dibangkitkan menjadi CFG. Penyesuaian tersebut dilakukan tanpa mengubah aspek-aspek penting yang dinilai.
2. Visualisasi *regression linear* perbandingan hasil penilaian *autograder* dengan hasil penilaian manual pendidik.

   Visualisasi ini dapat berfungsi untuk melihat distribusi data serta jarak data terhadap garis regresi yang didapat serta dapat digunakan untuk melakukan prediksi nilai manual pendidik berdasarkan nilai *autograder*.
3. Nilai *Pearson correlation coefficient* antara data penilaian *autograder* dengan data penilaian manual pendidik.

   Nilai ini mengukur kekuatan hubungan antara dua variabel. Pada kasus ini, apabila nilai yang didapatkan positif maka semakin besar nilai *autograder*, semakin besar juga nilai manual pendidik yang didapat, namun apabila nilai tersebut negatif, maka semakin besar nilai *autograder*, semakin kecil nilai manual pendidik yang didapat, hal ini dapat membantu dalam melakukan prediksi nilai. Nilai ini kemudian dapat dilakukan interpretasi berdasarkan interpretasi De Vaus yang dapat dilihat pada Tabel V.1.
4. Nilai *Mean Absolute Error* (MAE) antara data penilaian *autograder* dengan data penilaian manual pendidik.

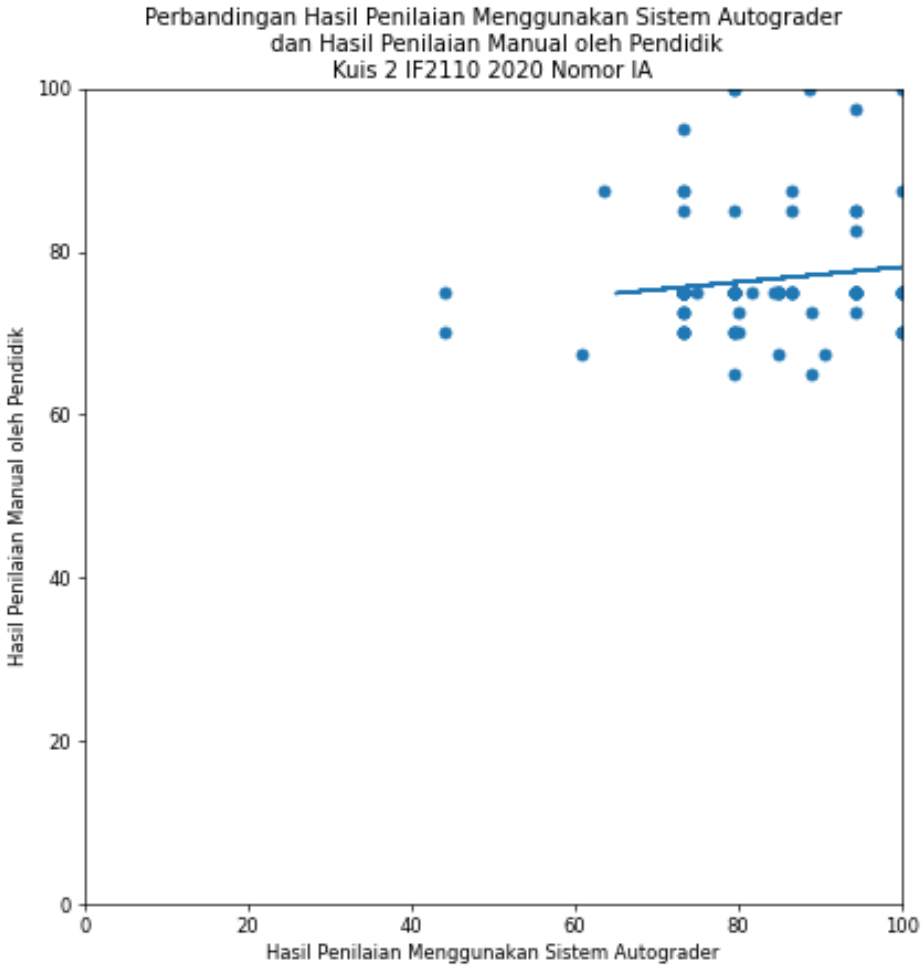
   Nilai ini mengukur besarnya rata-rata kesalahan antara hasil penilaian *autograder* dengan hasil penilaian manual pendidik.

Tabel V.1 Interpretasi *Pearson Coefficient Correlation* De Vaus.

Coefficient Correlation	**Kekuatan Korelasi**
0.00	Tidak ada korelasi
$0.01 - 0.09$	Korelasi tidak signifikan
$0.10 - 0.29$	Korelasi lemah
$0.30 - 0.49$	Korelasi sedang
$0.50 - 0.69$	Korelasi kuat
$0.70 - 0.89$	Korelasi sangat kuat
> 0.90	Korelasi hampir sempurna

V.6.1 Pengujian Persoalan Kuis 2 IF2110 2020 Nomor IA

Pada persoalan ini telah terkumpul sejumlah 117 submisi peserta didik untuk dilakukan pengujian perbandingan terhadap submisi pendidik. Visualisasi *regression linear* perbandingan hasil penilaian *autograder* dengan hasil penilaian manual untuk persoalan ini dapat dilihat pada Gambar V.6.



Gambar V.6 Visualisasi *Regression Linear* Perbandingan Hasil Penilaian *Autograder* dengan Hasil Penilaian Manual untuk Persoalan Kuis 2 IF2110 2020 Nomor IA

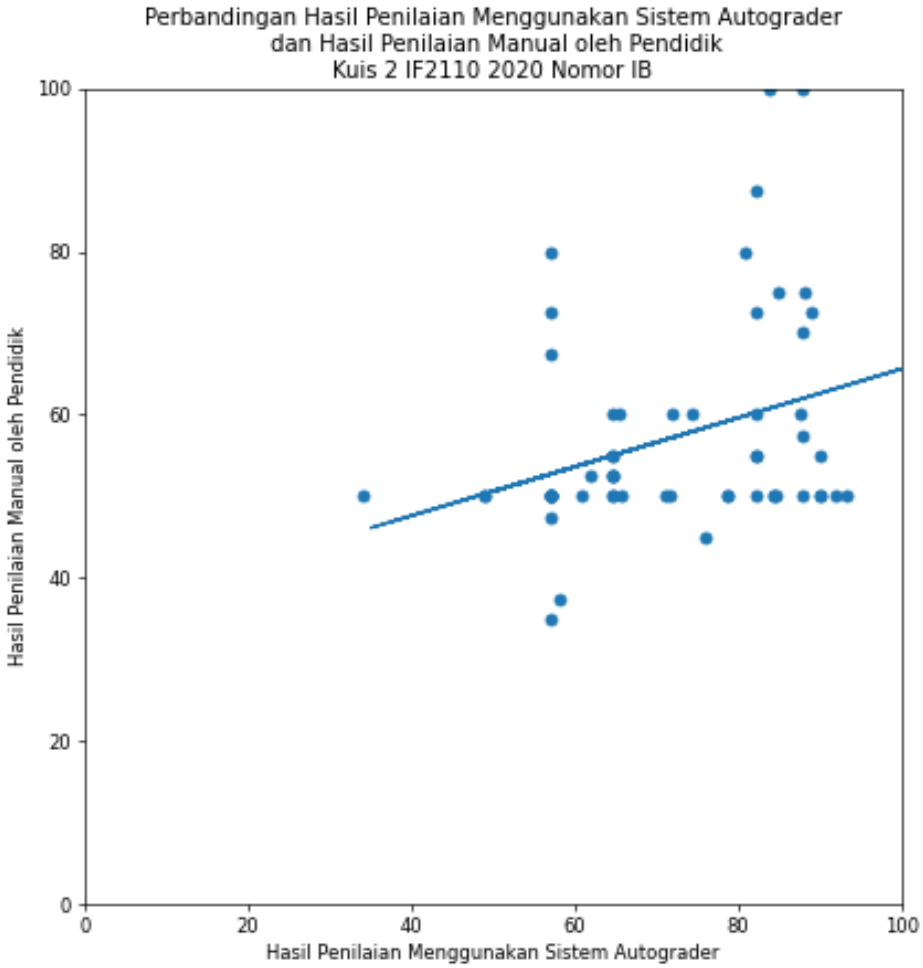
Pada visualisasi *regression linear* tersebut, banyak titik yang masih jauh mengenai garis regresi, sehingga keterkaitan nilai *autograder* dengan nilai manual masih kurang baik.

Pada persoalan ini, terhitung *Pearson correlation coefficient* antar data hasil penilaian *autograder* dengan hasil penilaian manual sebesar 0.144728, yang artinya didapat korelasi positif lemah antara nilai *autograder* dengan nilai manual. Dari seluruh kasus yang diuji dilaksanakan, dengan nilai *autograder* sebagai nilai yang

diamati dan nilai manual sebagai nilai yang diprediksi, terhitung nilai *Mean Absolute Error* (MAE) sebesar 8.5443.

V.6.2 Pengujian Persoalan Kuis 2 IF2110 2020 Nomor IB

Pada persoalan ini telah terkumpul sejumlah 58 submisi peserta didik untuk dilakukan pengujian perbandingan terhadap submisi pendidik. Visualisasi *regression linear* perbandingan hasil penilaian *autograder* dengan hasil penilaian manual untuk persoalan ini dapat dilihat pada Gambar V.7.



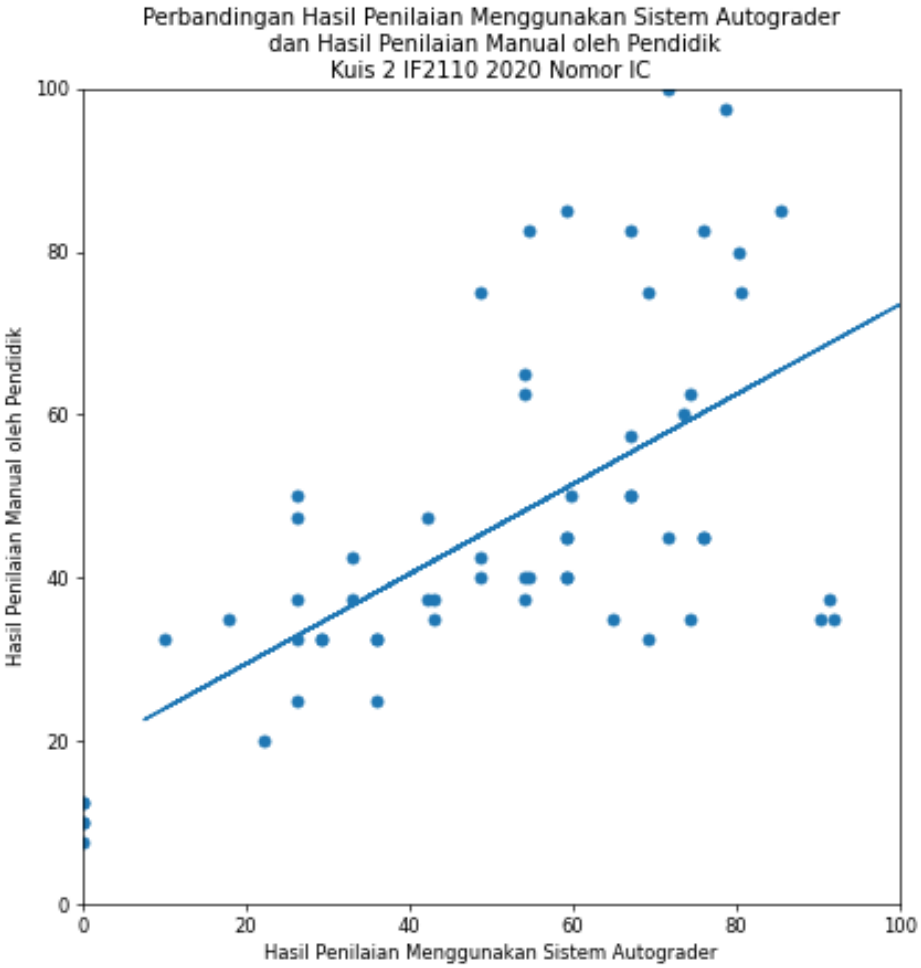
Gambar V.7 Visualisasi *Regression Linear* Perbandingan Hasil Penilaian *Autograder* dengan Hasil Penilaian Manual untuk Persoalan Kuis 2 IF2110 2020 Nomor IB

Pada visualisasi *regression linear* tersebut, banyak titik yang masih jauh mengenai garis regresi, sehingga keterkaitan nilai *autograder* dengan nilai manual masih kurang baik.

Pada persoalan ini, terhitung *Pearson correlation coefficient* antar data hasil penilaian *autograder* dengan hasil penilaian manual sebesar 0.317872, yang artinya didapat korelasi positif sedang antara nilai *autograder* dengan nilai manual. Dari seluruh kasus yang diuji dilaksanakan, dengan nilai *autograder* sebagai nilai yang diamati dan nilai manual sebagai nilai yang diprediksi, terhitung nilai *Mean Absolute Error* (MAE) sebesar 18.296.

V.6.3 Pengujian Persoalan Kuis 2 IF2110 2020 Nomor IC

Pada persoalan ini telah terkumpul sejumlah 61 submisi peserta didik untuk dilakukan pengujian perbandingan terhadap submisi pendidik. Visualisasi *regression linear* perbandingan hasil penilaian *autograder* dengan hasil penilaian manual untuk persoalan ini dapat dilihat pada Gambar V.8.



Gambar V.8 Visualisasi *Regression Linear* Perbandingan Hasil Penilaian *Autograder* dengan Hasil Penilaian Manual untuk Persoalan Kuis 2 IF2110 2020 Nomor IC

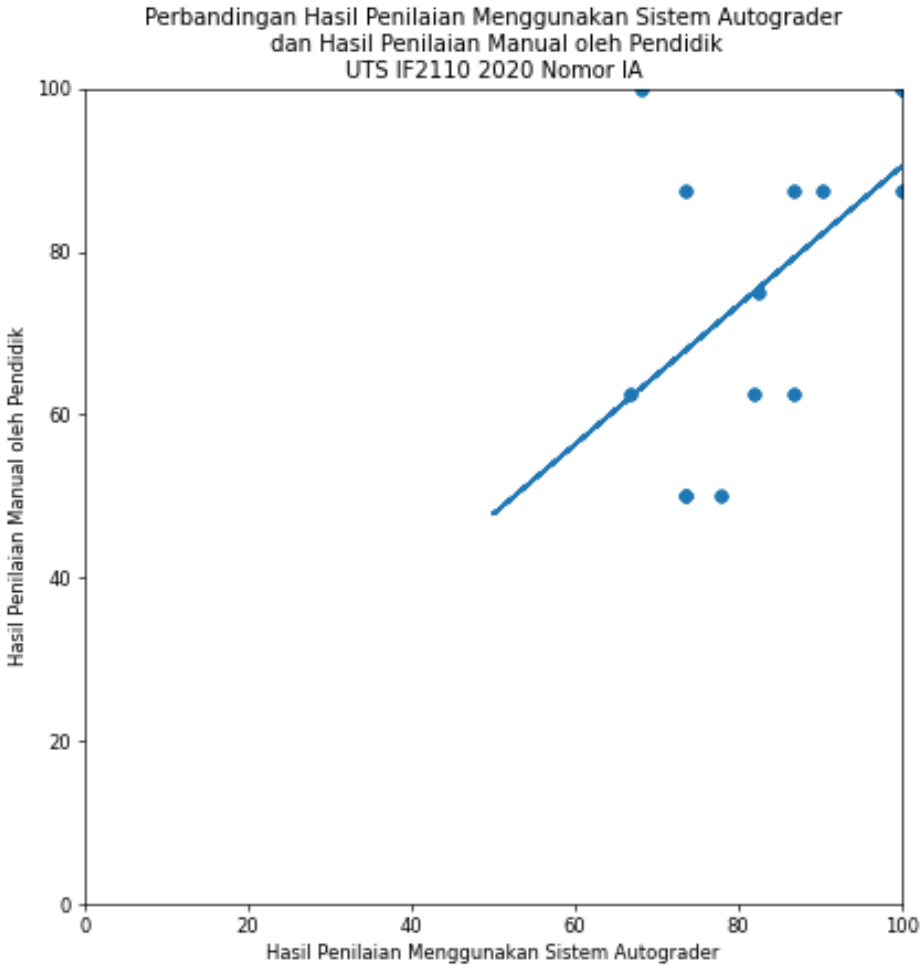
Pada visualisasi *regression linear* tersebut, banyak titik yang masih jauh mengenai garis regresi, sehingga keterkaitan nilai *autograder* dengan nilai manual masih kurang baik.

Pada persoalan ini, terhitung *Pearson correlation coefficient* antar data hasil penilaian *autograder* dengan hasil penilaian manual sebesar 0.634688 yang artinya didapat korelasi positif kuat antara nilai *autograder* dengan nilai manual. Dari seluruh kasus yang diuji dilaksanakan, dengan nilai *autograder* sebagai nilai yang

diamati dan nilai manual sebagai nilai yang diprediksi, terhitung nilai *Mean Absolute Error* (MAE) sebesar 15.8523.

V.6.4 Pengujian Persoalan UTS IF2110 2020 Nomor IA

Pada persoalan ini telah terkumpul sejumlah 14 submisi peserta didik untuk dilakukan pengujian perbandingan terhadap submisi pendidik. Visualisasi *regression linear* perbandingan hasil penilaian *autograder* dengan hasil penilaian manual untuk persoalan ini dapat dilihat pada Gambar V.9.



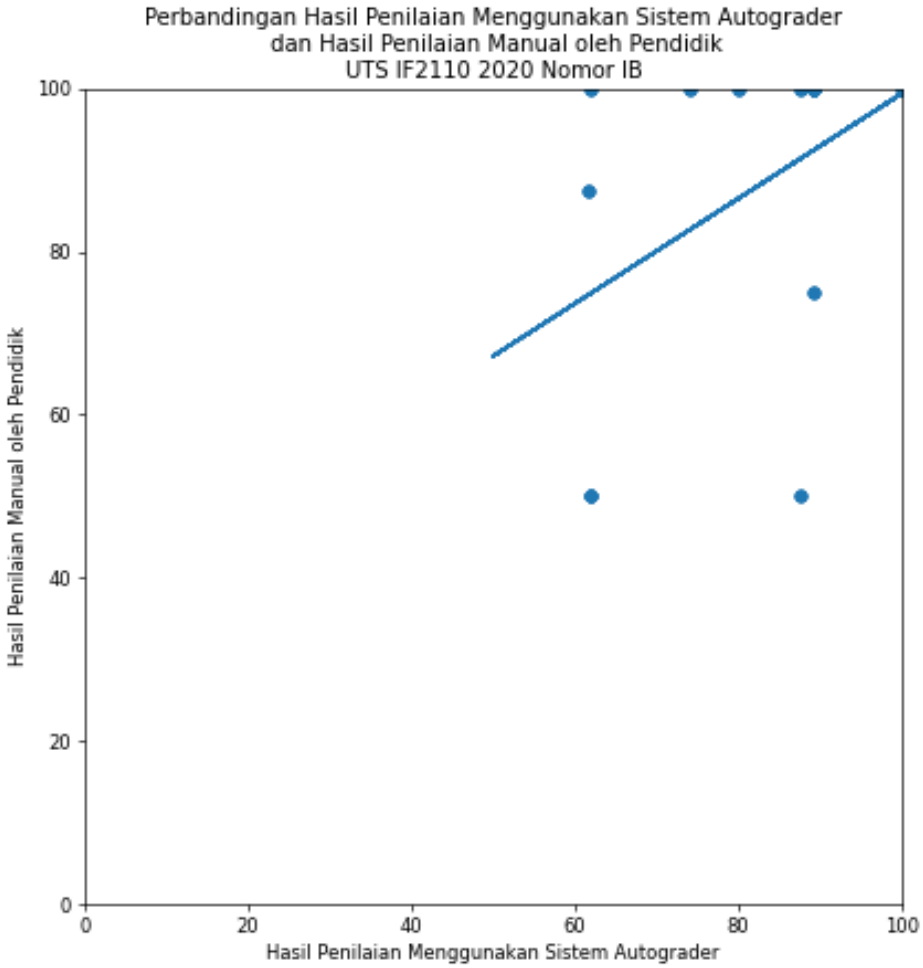
Gambar V.9 Visualisasi *Regression Linear* Perbandingan Hasil Penilaian *Autograder* dengan Hasil Penilaian Manual untuk Persoalan UTS IF2110 2020 Nomor IA

Pada visualisasi *regression linear* tersebut, banyak titik yang masih jauh mengenai garis regresi, sehingga keterkaitan nilai *autograder* dengan nilai manual masih kurang baik.

Pada persoalan ini, terhitung *Pearson correlation coefficient* antar data hasil penilaian *autograder* dengan hasil penilaian manual sebesar 0.510713 yang artinya didapat korelasi positif kuat antara nilai *autograder* dengan nilai manual. Dari seluruh kasus yang diuji dilaksanakan, dengan nilai *autograder* sebagai nilai yang diamati dan nilai manual sebagai nilai yang diprediksi, terhitung nilai *Mean Absolute Error* (MAE) sebesar 13.7335.

V.6.5 Pengujian Persoalan UTS IF2110 2020 Nomor IB

Pada persoalan ini telah terkumpul sejumlah 14 submisi peserta didik untuk dilakukan pengujian perbandingan terhadap submisi pendidik. Visualisasi *regression linear* perbandingan hasil penilaian *autograder* dengan hasil penilaian manual untuk persoalan ini dapat dilihat pada Gambar V.10.



Gambar V.10 Visualisasi *Regression Linear* Perbandingan Hasil Penilaian *Autograder* dengan Hasil Penilaian Manual untuk Persoalan UTS IF2110 2020 Nomor IB

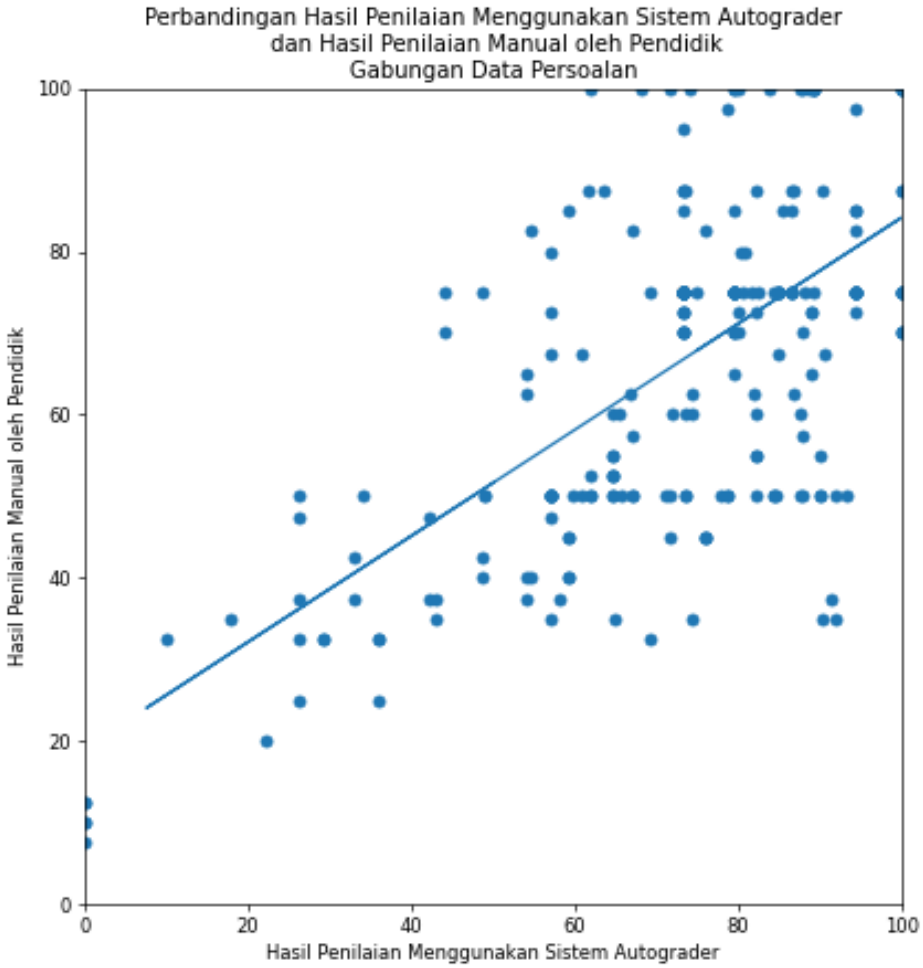
Pada visualisasi *regression linear* tersebut, banyak titik yang masih jauh mengenai garis regresi, sehingga keterkaitan nilai *autograder* dengan nilai manual masih kurang baik.

Pada persoalan ini, terhitung *Pearson correlation coefficient* antar data hasil penilaian *autograder* dengan hasil penilaian manual sebesar 0.404776 yang artinya didapat korelasi positif sedang antara nilai *autograder* dengan nilai manual. Dari seluruh kasus yang diuji dilaksanakan, dengan nilai *autograder* sebagai nilai yang

diamati dan nilai manual sebagai nilai yang diprediksi, terhitung nilai *Mean Absolute Error* (MAE) sebesar 17.18495.

V.6.6 Pengujian Gabungan Data Persoalan

Gabungan data persoalan adalah gabungan data dari hasil pengujian Kuis 2 IF2110 Nomor IA, Kuis 2 IF2110 Nomor IB, Kuis 2 IF2110 Nomor IC, UTS IF2110 Nomor IA, dan UTS IF2110 Nomor IB. Untuk gabungan data persoalan, telah terkumpul total sejumlah 264 submisi peserta didik untuk dilakukan pengujian perbandingan terhadap submisi pendidik. Visualisasi *regression linear* perbandingan hasil penilaian *autograder* dengan hasil penilaian manual untuk persoalan ini dapat dilihat pada Gambar V.11.



Gambar V.11 Visualisasi *Regression Linear* Perbandingan Hasil Penilaian *Autograder* dengan Hasil Penilaian Manual untuk Gabungan Data Persoalan

Pada visualisasi *regression linear* tersebut, banyak titik yang masih jauh mengenai garis regresi, sehingga keterkaitan nilai *autograder* dengan nilai manual masih kurang baik.

Pada persoalan ini, terhitung *Pearson correlation coefficient* antar data hasil penilaian *autograder* dengan hasil penilaian manual sebesar 0.65628 yang artinya didapat korelasi positif kuat antara nilai *autograder* dengan nilai manual. Dari seluruh kasus yang diuji dilaksanakan, dengan nilai *autograder* sebagai nilai yang

diamati dan nilai manual sebagai nilai yang diprediksi, terhitung nilai *Mean Absolute Error* (MAE) sebesar 13.10873.

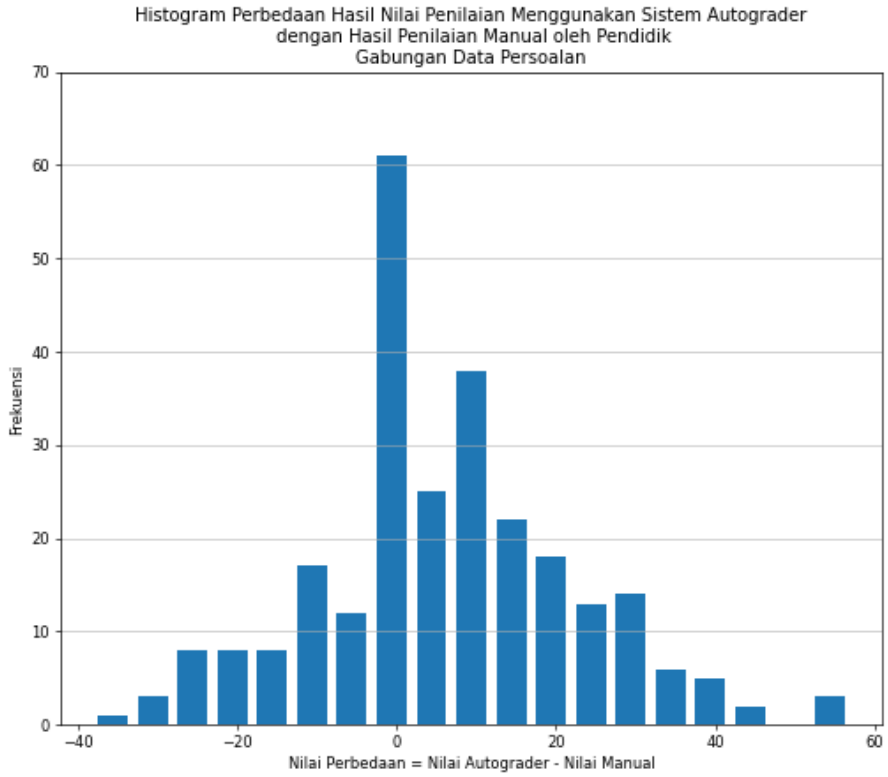
Dari seluruh pengujian yang dilakukan dikumpulkan seluruh nilai *Mean Absolute Error* (MAE) menjadi tabel yang digambarkan pada Tabel V.2.

Tabel V.2 Kumpulan Nilai *Mean Absolute Error* Pengujian Persoalan

Persoalan	**Nilai *Mean Absolute Error***
Kuis 2 IF2110 Nomor IA	8.544
Kuis 2 IF2110 Nomor IB	18.296
Kuis 2 IF2110 Nomor IC	15.852
UTS IF2110 Nomor IA	13.733
UTS IF2110 Nomor IB	17.185
Gabungan	13.109

Dari hasil tersebut dapat dilihat bahwa nilai MAE terkecil didapat oleh persoalan Kuis 2 IF2110 Nomor IA disusul dengan Persoalan UTS IF2110 Nomor IA. Kedua persoalan tersebut dibandingkan dengan ujian yang sama dan nomor berbeda memang memiliki persoalan yang lebih sederhana sehingga hasil solusi dari persoalan tersebut cenderung memberikan hasil akurasi yang lebih baik.

Pada pengujian gabungan data persoalan ini juga dilakukan pembuatan visualisasi histogram dari perbedaan hasil penilaian *autograder* dengan hasil penilaian manual pendidik. Nilai yang dimaksud disini adalah nilai *autograder* dikurang dengan nilai manual pendidik. Visualisasi histogram perbedaan tersebut dapat dilihat pada Gambar V.12.



Gambar V.12 Visualisasi Histogram Perbedaan Hasil Nilai Penilaian *Autograder* dengan Hasil Penilaian Manual untuk Gabungan Data Persoalan

Pada visualisasi tersebut, dapat dilihat bahwa nilai perbedaan cenderung lebih banyak pada nilai ≥ 0, artinya hasil nilai *autograder* ini cenderung lebih meningkat dibandingkan dengan nilai manual pendidik, dengan nilai terjauh mencapai perbedaan 56.8367. Namun, terdapat beberapa kasus juga ketika hasil nilai *autograder* bernilai kurang dari hasil nilai manual pendidik, dengan nilai terjauh mencapai perbedaan -38.095. Pada hasil data perbedaan ini juga, terhitung nilai rata-rata sebesar 5.89 dan nilai standar deviasi sebesar 16.398, sehingga data nilai perbedaan ini cukup tersebar dari rata-rata nilai yang didapatkan.

BAB VI
KESIMPULAN DAN SARAN

VI.1 Kesimpulan

Setelah melakukan implementasi dan pengujian, kesimpulan dari tugas akhir dapat dituliskan sebagai berikut:

1. Integrasi sistem hasil penelitan Sofyana dkk. (2021) dengan sistem hasil penelitian Sendjaja dkk. (2021) dilaksanakan dengan melihat keterkaitan antara dua sistem hasil penelitian tersebut, yang kemudian dibuat sebuah *Intermediate Module* sebagai penengah sekaligus penyatu dari kedua sistem tersebut.
2. Hasil penilaian dari sistem *autograder* Notasi Algoritmik memiliki akurasi yang lebih baik untuk kasus uji persoalan yang lebih sederhana. Hasil penilaian *autograder* juga cenderung memiliki nilai yang lebih besar dibandingkan dengan hasil penilaian manual pendidik. Selain itu, hasil penilaian *autograder* secara umum memiliki korelasi cukup positif dengan nilai manual yang dilakukan pendidik, namun korelasi antara kedua hasil tersebut masih lemah dengan nilai korelasi 0,1 – 0,6.

VI.2 Saran

Terdapat beberapa saran yang dapat digunakan sebagai upaya pengembangan lebih lanjut tugas akhir, seperti:

1. Sistem hasil penelitian Sofyana dkk. (2021) yang sudah dilakukan modifikasi dapat dilakukan modifikasi dan perbaikan lebih lanjut dengan melakukan saran-saran yang terdapat pada Tugas Akhir Sofyana dkk. (2021) dan melakukan pengujian terhadap lebih banyak data ujian Notasi Algoritmik.
2. Algoritma perbandingan *control flow graph* hasil penelitian Sendjaja dkk. (2021) masih terbatas pada perbandingan struktur grafnya saja, sehingga

dapat digunakan algoritma yang lebih baik untuk melakukan perbandingan *control flow graph* tersebut.

3. Dapat dilakukan pengujian data ujian Notasi Algoritmik yang lebih banyak dan lebih bervariasi untuk melakukan analisis lebih baik akan nilai akurat dan korelasi antara hasil penilaian sistem *autograder* dengan hasil penilaian manual pendidik.

DAFTAR PUSTAKA

Chan, P. P. F., & Collberg, C. (2014). *A Method to Evaluate CFG Comparison Algorithms. 2014 14th International Conference on Quality Software. doi: 10.1109/QSIC.2014.28.*

Chen, Lianping (2018). *Microservices: Architecting for Continuous Delivery and DevOps. The IEEE International Conference on Software Architecture* (ICSA 2018). IEEE.

Danutama, K., & Liem, I. (2013). Scalable autograder and LMS Integration. Procedia Technology, 11, 388–395. https://doi.org/10.1016/j.protcy.2013.12.207.

Francisco Curbera, William A. Nagy and Sanjiva Weerawarana (2001). *Web Services: Why and How*.

Ihantola, P., Ahoniemi, T., Karavirta, V., & Seppälä, O. (2010). Review of recent systems for automatic assessment of programming assignments. Proceedings of the 10th Koli Calling International Conference on Computing Education Research - Koli Calling '10. https://doi.org/10.1145/1930464.1930480.

Liem, I. (2007). Draft Diktat Kuliah Dasar Pemrograman (Bagian Pemrograman Prosedural). Bandung: Kelompok Keahlian Rekayasa Perangkat Lunak dan Data STEI ITB.

Naudé, K. A., Greyling, J. H., & Vogts, D. (2010). Marking student programs using graph similarity. Computers & Education, 54(2), 545–561. https://doi.org/10.1016/j.compedu.2009.09.005.

Newman, Sam (2015). *Building Microservices*. O'Reilly Media.

Pautasso, Cesare (2017). "*Microservices in Practice, Part 1: Reality Check and Service Design*". IEEE Software.

Reese T. Prosser (1959). *Applications of Boolean matrices to the analysis of flow diagrams. Papers presented at the December 1–3, 1959, eastern joint IRE-AIEE-ACM computer conference*. pp. 133–138.

Sendjaja, Kevin., Rukmono, S. A., Perdana, R. S. (2021). Evaluasi Tugas Pemrograman Dengan Pendekatan Control Flow Graph.

Shahidul Islam, Dr.S.Britto Ramesh Kumar, Ab Rashid Dar (2018). *A Comprehensive Study on Web Services. International Journal of Advance Research in Science and Engineering* (IJARSE).

Sherman, M., Bassil, S., Lipman, D., Tuck, N., & Martin, F. (2013). Impact of Auto-Grading on an Introductory Computing Course. J. Comput. Sci. Coll. 28, 6, 69–75.

Sofyana, Irfan., Rukmono, S. A., Perdana, R. S. (2021). Pembangkitan Abstract Syntax Tree (AST) dan Control Flow Graph (CFG) Notasi Algoritmik.

Trivenimishra, Gauravraj (2017). *QoS Implementation in Web Services Selection and Ranking using Data Analysis. 7th International Conference on Cloud Computing, Data Science and Engineering-Confluence*. IEEE.

Xin Hu. dkk. 2009. *Large-scale malware indexing using function-call graphs. Proceedings of the* 16th ACM *conference on Computer and communications security* (CCS'09), 611–620. doi:10.1145/1653662.1653736.

Yousefi, Javad (2015). *Masking wrong-successor Control Flow Errors employing data redundancy*. IEEE. pp. 201–205. [Original source: https://studycrumb.com/alphabetizer]

Lampiran A. Dokumen Kelengkapan Proyek Capstone

A.1 Rencana Umum Proyek

Proyek Tugas Akhir *Capstone* yang kami kerjakan berjudul *Advances in Automated Grading for Programming Exercise* ini bertujuan dalam mengembangkan sistem *autograder* yang diharapkan dapat membantu pengajar khususnya di Institut Teknologi Bandung dalam melakukan penilaian hasil pengerjaan latihan pemrograman mahasiswa.

Saat ini sudah ada penelitian mengenai sistem penilaian otomatis atau *autograder*, yang juga merupakan hasil Tugas Akhir beberapa mahasiswa Institut Teknologi Bandung yang dilakukan pada tahun ajaran 2020/2021. Meskipun sudah ada hasil penelitian ini, sistem yang sudah ada masih terpisah-pisah dan belum diintegrasikan.

Maka dari itu, proyek tugas akhir *capstone* ini bertujuan untuk mengembangkan empat buah subsistem dan mengintegrasikan ini menjadi sebuah sistem *autograder* utuh. Tabel A.1 menjelaskan pembagian pengerjaan subsistem tersebut:

Tabel A.1. Pembagian peran subsistem dari sistem *Autograder*

NIM	Nama	Subsistem
13518012	Muhammad Hasan	*Autograder* Notasi Algoritmik berbasis Control Flow Graph
13518113	Muhammad Kamal Shafi	*Web Service White Box Autograder* berbasis *Control Flow Graph*
13518093	Morgen Sudyanto	*Web Service White Box Autograder* Berbasis Semantik
13518069	Dimas Wahyu Langkawi	Pengembangan Sistem Integrasi *Autograder* Abstrak dengan Moodle dan GitLab

A.2 Spesifikasi Kebutuhan Sistem

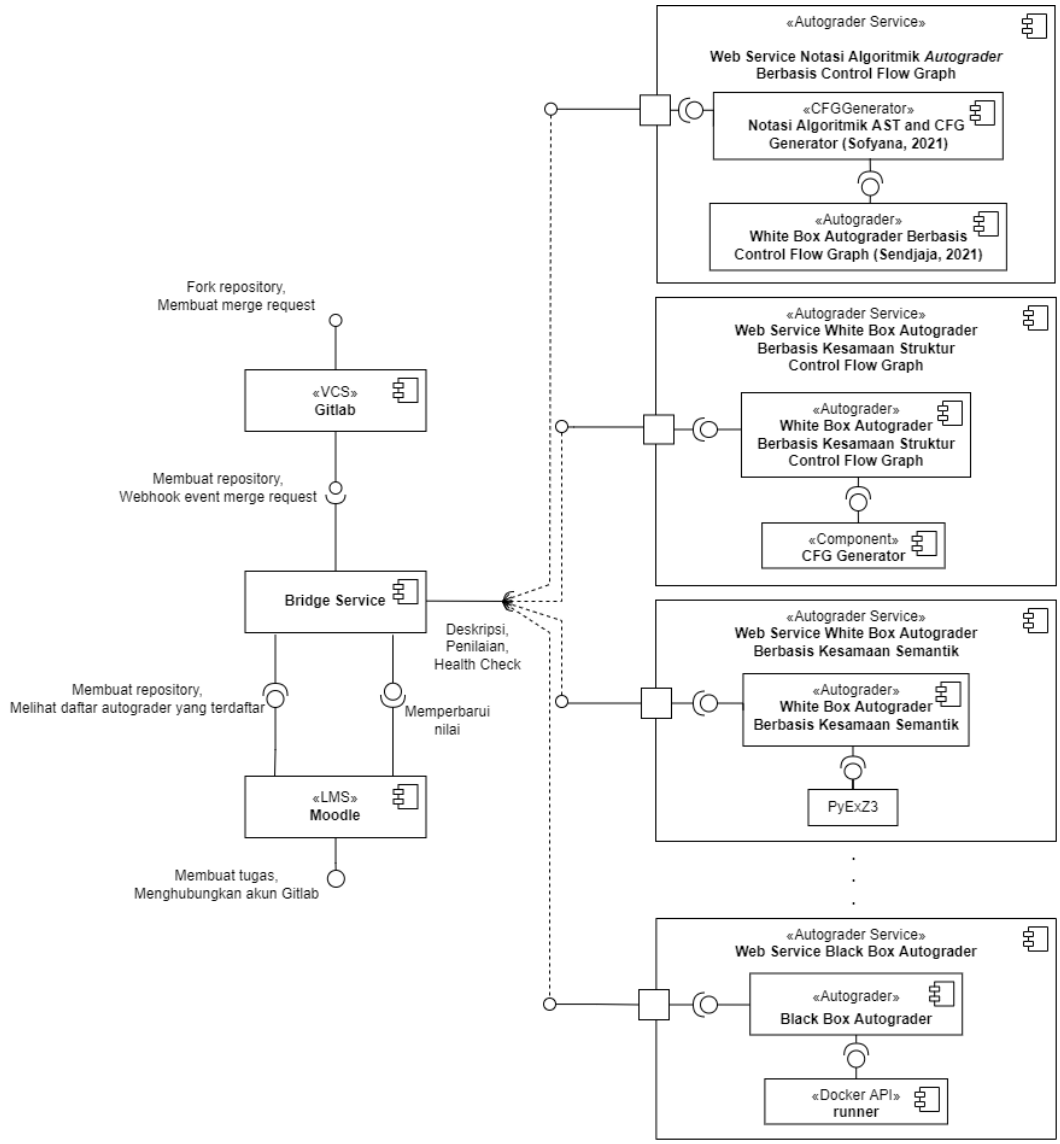
Terdapat beberapa spesifikasi yang harus dipenuhi oleh sistem, hal ini ada pada Tabel A.2 sebagai berikut:

Tabel A.2 Spesifikasi Kebutuhan Sistem

Kode	Kebutuhan
S01	Pengajar dapat membuat sebuah tugas pemrograman.
S02	Pengajar dapat mengunggah kode referensi saat membuat tugas pemrograman.
S03	Pengajar dapat memilih *autograder* yang digunakan untuk soal tersebut.
S04	Pelajar dapat menghubungkan akun Gitlab (*version control*) dengan Moodle (*learning management system*).
S05	Pelajar dapat mengumpulkan pekerjaan melalui Gitlab dengan membuat *merge request*.
S06	Pelajar dapat melihat hasil penilaian melalui Moodle.
S07	Sistem menyediakan tempat untuk menampilkan hasil penilaian *autograder*.
S08	Sistem menyediakan tempat untuk membuat soal pemrograman.
S09	Sistem menyediakan tempat untuk mengunggah *file* program referensi.
S10	Sistem dapat mendaftarkan *autograder* secara otomatis.
S11	Sistem dapat melakukan pemeriksaan ketersediaan *autograder*.

A.3 Rancangan Sistem

Secara umum, rancangan sistem digambarkan pada Gambar A.1:



Gambar A.1 Rancangan sistem *autograder*

Penjelasan dari gambar rancangan sistem tersebut adalah sebagai berikut:

1. LMS (*Learning Management System*) Moodle merupakan *user interface* bagi pengajar dan peserta didik. LMS Moodle ini dapat menerima soal, program

referensi pengajar, beserta program peserta didik. Program yang diterima bisa berupa *file* program python atau *file* program Notasi Algoritmik. Untuk penilaian program python, LMS Moodle dapat memanggil sistem *white box autograder* berbasis kesamaan semantik, kemudian untuk program Notasi Algoritmik, LMS Moodle dapat memanggil sistem *white box autograder* berbasis kesamaan struktur, hasil semua penilaian kemudian dapat LMS Moodle terima kembali dan dapat ditampilkan untuk peserta didik sebagai umpan balik pekerjaan peserta didik. LMS Moodle juga menggunakan Gitlab sebagai media bagi peserta didik untuk mengumpulkan hasil pengerjaan.

2. *Web service autograder* Notasi Algoritmik berbasis *control flow graph* merupakan *autograder* yang dapat menerima *file* tulisan Notasi Algoritmik pengajar dan pendidik, yang kemudian melakukan penilaian berdasarkan kesamaan struktur *control flow graph* tersebut. Subsistem ini menggunakan dua hasil penelitian yang sudah ada, yakni Pembangkitan *Abstract Syntax Tree* (AST) dan *Control Flow Graph* (CFG) Notasi Algoritmik (Sofyana dkk., 2021) dan Evaluasi Tugas Pemrograman dengan Pendekatan *Control Flow Graph* (CFG) (Sendjaja dkk., 2021). Sistem ini berbentuk *web service* yang dapat menerima *request* penilaian program peserta didik dan akan memberi *response* yang berisi nilai kesamaan yang sudah dinormalisasi pada skala tertentu.

3. *Web service white box autograder* berbasis kesamaan struktur *control flow graph* merupakan *autograder* yang dapat menilai kesamaan struktur dari program hasil pengerjaan peserta didik dengan satu atau lebih program referensi yang disediakan oleh pengajar. Penilaian pada *autograder* ini dilakukan terhadap *control flow graph* yang dibangkitkan oleh komponen CFG generator dan kemudian diukur seberapa mirip atau seberapa jauh perbedaan antara keduanya. Sistem ini berbentuk layanan web yang dapat menerima *request* penilaian program peserta didik dan akan memberi *response* yang berisi nilai kesamaan yang sudah dinormalisasi pada skala tertentu.

4. *Web service white box autograder* berbasis kesamaan semantik merupakan *autograder* yang dapat menilai kesamaan semantik antara kode peserta didik

dengan kode referensi. Kesamaan semantik akan diperiksa dengan memanfaatkan *concolic execution* untuk mengeksplorasi *path* eksekusi program. Selain nilai kesamaan semantik, *autograder* juga akan menghasilkan letak perbedaan semantik pada kode peserta didik. Sistem ini berbentuk layanan *web* yang dapat menerima *request* berupa kode peserta didik dan kode referensi dan menghasilkan *response* berupa nilai dan umpan balik berupa letak perbedaan semantik.

5. *Web service black box autograder* merupakan *autograder* yang dapat menilai program peserta didik berdasarkan daftar masukan dan daftar keluaran yang diekspektasi. Penilaian dilakukan dengan mengeksekusi program peserta didik dengan setiap masukan, lalu membandingkan keluaran yang dihasilkan program dengan keluaran yang diekspektasi.

A.4 Pengujian Sistem

Pengujian sistem akan dilakukan dengan melakukan beberapa tahap yang dijelaskan pada Tabel A.3 sebagai berikut:

Tabel A.3 Pengujian Sistem

Kode	Skenario
P01	Menggunakan akun pengajar untuk membuat soal pemrograman dengan Notasi Algoritmik dan mengunggah solusi referensi.
P02	Menggunakan akun peserta didik untuk mengerjakan soal pemrograman tersebut dengan kode solusi dalam Notasi Algoritmik melalui GitLab.
P03	Menggunakan akun peserta didik untuk melihat nilai kode Notasi Algoritmik yang dihasilkan.
P04	Menggunakan *white box autograder* berbasis kesamaan struktur dan *white box autograder* berbasis kesamaan semantik dengan bahasa Python untuk menilai program peserta didik.
P05	Menggunakan akun pengajar untuk membuat soal pemrograman dengan bahasa Python dan mengunggah solusi referensi.
P06	Menggunakan akun peserta didik untuk mengerjakan soal pemrograman tersebut dengan mengunggah kode solusi dalam bahasa Python.
P07	Menggunakan akun peserta didik untuk mengumpulkan hasil pengerjaan kode Python melalui GitLab.
P08	Menggunakan akun peserta didik untuk melihat nilai kode Python yang dihasilkan.
P09	Admin menjalankan program *autograder* pada docker *network* yang sama dan terdaftar secara otomatis.
P10	Admin memberhentikan program *autograder* dan secara otomatis menghapus *autograder* dari daftar *autograder*.

Lampiran B. Solusi Referensi

Tabel B.I berisi solusi referensi dari setiap persoalan yang berada pada data uji.

Tabel B.1 Notasi Algoritmik Solusi Referensi Persoalan Data Uji

Persoalan	Solusi Referensi

Kuis 2
IF2110
2020
Nomor IA
	{{ Answer for Kuis 2 - IA - IF2210 - 2020 }}
	function SearchX (L: List, X : infotype) -> boolean
	KAMUS
	    Pt : address
	    Found : boolean
	ALGORITMA
	    if (IsEmpty(L)) then
	        -> false
	    else
	        Pt <- First(L)
	        while (Next(Pt) ≠ Nil and Info(Pt) < X) do
	            Pt <- Next(Pt)
	        { Next(Pt) = Nil or Info(Pt) = X or Info(Pt) > X }
	        -> Info(Pt) = X

Kuis 2
IF2110
2020
Nomor IB
	procedure SortedIntersect (input L1, L2: List, output L3: List)
	KAMUS
	P, Pt1, Pt2: address
	fail: boolean
	ALGORITMA
	CreateEmpty(L3)
	Pt1 <- First(L1)
	Pt2 <- First(L2)
	fail <- false
	
	while (Pt1 ≠ Nil and Pt2 ≠ Nil and not fail) do
	depend on (Pt1, Pt2)
	Info(Pt1) = Info(Pt2):
	P <- Alokasi(Info(P1))
	if (P ≠ Nil) then
	InsertLast(L3, P)
	Pt1 <- Next(Pt1)
	Pt2 <- Next(Pt2)
	else
	fail <- true
	Info(Pt1) < Info(Pt2):
	Pt1 <- Next(Pt1)
	Info(Pt1) > Info(Pt2):
	Pt2 <- Next(Pt2)
	{Pt1 = nil or pt2 = nil or fail}
	if fail then
	while x = 2 do
	P <- First(L3)
	L3.First <- Next(First(L3))
	Dealokasi(P)

Kuis 2
IF2110
2020
Nomor IC
	procedure SortedMerge (input/output L1: List, input L2: List, output status: boolean)
	KAMUS
	P, Pt1, Pt2, Prec: address
	isStop, status: boolean
	ALGORITMA
	Pt2 ← First(L2)
	isStop ← false
	status ← true
	while (Pt2 ≠ Nil and status) do
	{insert P di L1 sambil menjaga keterurutan dan tetap unik}
	Pt1 ← First(L1)
	Prec ← Nil
	while (Pt1 ≠ Nil and not isStop) do {mencari posisi}
	if Info(Pt1) ≥ Info(Pt2) then
	isStop ← true
	else
	Prec ← Pt1
	Pt1 ← Next(Pt1)
	{Pt1 = Nil or Info(Pt1) ≥ Info(Pt2) or Prec is still Nil}
	if (Pt1 = Nil) then {kasus untuk insert last}
	P ← Alokasi(Info(Pt2))
	if (P ≠ Nil) then
	Next(Prec) ← P {insert last}
	else
	status ← false
	else
	if (Info(Pt1) ≠ Info(P)) then
	P ← Alokasi(Info(Pt2))
	if (P ≠ Nil) then
	if (Prec = Nil) then {kasus insert first}
	Next(P) ← First(L1)
	First(L1) ← P
	else {Info(Pt1) > Info(P), kasus insert after Prec}

	Next(Prec) ← P Next(P) ← Pt1 else status ← false Pt2 ← Next(Pt2)

UTS IF2110 2020 Nomor IA	function IsSegitigaBawah(M: Matriks) -> boolean
	KAMUS LOKAL
	segitigaBawah: boolean
	i, j: indeks
	ALGORITMA
	segitigaBawah <- true
	i <- GetFirstIdxBrs(M)
	while i < GetLastIdxBrs(M) and segitigaBawah do
	j <- i + 1
	while j <= GetLastIdxKol(M) and segitigaBawah do
	if i < j and GetElmt(M, i, j) <> 0 then
	segitigaBawah <- false
	else
	j <- j + 1
	i <- i + 1
	-> segitigaBawah

UTS IF2110 2020 Nomor IB	procedure DetriMatriks(input M: Matriks, output Det: integer)
	KAMUS LOKAL
	i: indeks
	ALGORITMA
	Det <- 1
	if IsSegitigaBawah(M) then
	i traversal [GetFirstIdxBrs(M)..GetLastIdxBrs(M)]
	Det <- Det * GetElmtDiagonal(M, i)
	else
	Det <- 0