PENYUSUNAN RENCANA KULIAH DENGAN *TOPOLOGICAL SORT* (PENERAPAN *DECREASE AND CONQUER*)

Laporan Tugas Kecil 2 IF 2211 Strategi Algoritma

Semester II Tahun 2020/2021

Oleh:

Dionisius Darryl Hermansyah

13519058 / Kelas 02

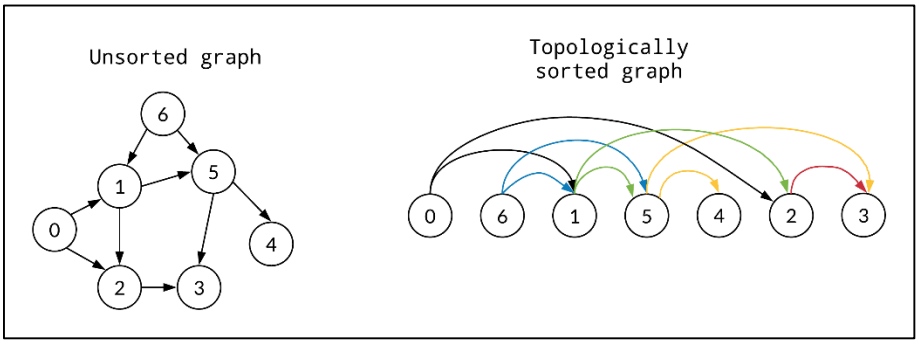
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2021

I. ALGORITMA *TOPOLOGICAL SORT* DAN *DECREASE AND CONQUER*

Topological sort merupakan sebuah algoritma yang umumnya digunakan untuk melakukan *topological ordering*. *Topological ordering* adalah kegiatan mengurutkan objek-objek (biasanya direpresentasikan dalam bentuk graf berarah), dimana, untuk setiap sisi (*edge*) dari simpul (*vertex*) A ke simpul B, simpul A harus berada pada urutan di depan simpul B. Contoh dari permasalahan *topological ordering* yang dapat diselesaikan menggunakan *topological sort* adalah pengambilan mata kuliah yang memilih mata kuliah prasyarat. Pada algoritma *topological sort*, ada sebuah simpul yang harus menjadi *source vertex*, dimana, simpul tersebut memiliki derajat masuk 0. Hal ini juga dijelaskan secara visual dengan adanya *directed acyclic graphs* (DAG) yang menyatakan bahwa *topological sort* hanya dapat diimplementasikan pada graf yang valid yaitu graf yang tidak memiliki sirkuit.



Gambar 1. Contoh penerapan algoritma *topological sort* pada sebuah *directed acyclic graph* (DAG)

Algoritma *topological sort* merupakan salah satu penerapan dari algoritma *decrease and conquer*. Algoritma *decrease and* conquer memanfaatkan metode perancangan algoritma dengan umumnya mereduksi persoalan menjadi dua upa-persoalan yang lebih kecil, tetapi selanjutnya melakukan seleksi dan hanya memproses satu sub-persoalan saja. Berikut merupakan penjelasan lebih lanjut mengenai langkah-langkah algoritma *topological sort* secara umum, serta kaitannya dengan algoritma *decrease and conquer*:

1. Asumsikan graf telah dibaca dari file .txt dan graf merupakan DAG. Pada kasus ini diambil contoh graf pada gambar 1.

2. Hitung seluruh derajat masuk (*in-degree*) dari masing-masing simpul (*vertex*) yang ada pada graf. Derajat masuk merupakan jumlah busur (*edge*) yang masuk menuju ke simpul terkait. Pada gambar 1, derajat masing-masing simpul adalah:

   Simpul 0: 0
   Simpul 1: 2
   Simpul 2: 1
   Simpul 3: 2
   Simpul 4: 1
   Simpul 5: 2
   Simpul 6: 0

3. Pilih simpul dengan derajat masuk paling rendah (Umumnya pada iterasi pertama adalah 0). Dari contoh, didapatkan simpul 0 dan 6 (*Divide*).
4. Hapus simpul beserta semua busur yang keluar dari simpul tersebut, kemudian kurangi derajat simpul yang berhubungan dengan simpul terkait (*Conquer*). Setelah itu, graf akan menjadi:

   Simpul 1: 0
   Simpul 2: 1
   Simpul 3: 2
   Simpul 4: 1
   Simpul 5: 1

5. Ulangi langkah 3 dan 4 hingga semua simpul pada graf telah dipilih dan terhapus (kondisi graf kosong, didefinisikan sebagai graf yang tidak memiliki simpul dan busur).
6. Keluarkan hasil akhir pemilihan mata kuliah berdasarkan *topological sort*, dalam hal ini:

   Semester I : 0, 6
   Semester II : 1
   Semester III : 2, 5
   Semester IV : 3, 4

Algoritma *topological sort* termasuk ke dalam algoritma *divide and conquer*. Hal ini karena algoritma ini memanfaatkan strategi yang serupa dengan algoritma *divide and conquer*, yaitu membagi persoalan ke upa-persoalan yang lebih kecil, menyeleksi upa-persoalan dan memprosesnya, serta meninggalkan upa-persoalan lainnya sehingga tidak diproses. Pada contoh pemilihan simpul graf di atas, mula-mulanya algoritma *topological sort* akan membagi (*divide*) himpunan simpul menjadi 2 upa-himpunan yaitu simpul dengan derajat masuk terkecil (misalnya A) dan simpul lainnya (misalnya B). Upa-himpunan simpul dengan derajat masuk terkecil akan dihilangkan dari graf dan diabaikan (*conquer*) dengan menghapus setiap simpul yang ada serta busur yang terkait dengan simpul tersebut. Setelah itu, hanya upa-himpunan B yang akan diproses

dengan strategi yang sama dengan pemrosesan sebelummnya. *Topological sort* akan berakhir saat graf sudah kosong, yaitu setelah graf tidak memiliki simpul dan bidang apapun lagi. Jadi, berdasarkan penjelasan tersebut, maka *topological sort* merupakan sebuah aplikasi nyata dari algoritma *divide and conquer* dalam kehidupan sehari-hari, contohnya dalam penyusunan jadwal perkuliahan.

II. *SOURCE CODE* PROGRAM

Dalam pembuatan program, diambil beberapa asumsi sebagai berikut:

a. Input data uji yang dimasukkan sudah pasti merupakan DAG jika dikonversi ke dalam graf.
b. Input data mata kuliah sudah pasti dapat diselesaikan dalam 8 semester atau kurang.
c. Masa kuliah (semester) mahasiswa hanya ada 8 semester.
d. Dalam satu semester, mahasiswa dapat mengambil jumlah mata kuliah tanpa batas.

Program ini dibuat menggunakan bahasa Python secara modular. Ada 2 modul utama yaitu main.py dan graph.py. Program utama diletakkan pada main.py, sedangkan graph.py berisi definisi dan deskripsi kelas graph beserta segala fungsi yang terkait dengannya. *Source code* yang dilampirkan telah disesuaikan kode, komentar, dan indentasinya agar lebih rapi dan singkat. Berikut merupakan *source code* dari program:

main.py

```
"""
Main Program
"""

from graph import *

# Variabel Global
semester_romawi = {1: 'I', 2: 'II', 3: 'III', 4: 'IV',
                   5: 'V', 6: 'VI', 7: 'VII', 8: 'VIII'
                   } # Konversi semester angka ke romawi
exit = False        # Status exit program
show_steps = False   # Apakah ingin menampilkan langkah penyelesaian

print("================================================")
print("    ___                                  ___              ")
print("  / __\___  _   _ _ __ ___  ___  /___\_ __ __ _ _ ")
print(" / /  / _ \| | | | '__/ __|/ _ \//  // '__/ _` |")
print("/ /__| (_) | |_| | |  \__ \  __/ \_//| | | (_| |")
print("\____/\___/ \__,_|_|  |___/\___\___/ |_|  \__, |")
print("                                            |___/ ")
print("================================================")
print("[ Dionisius Darryl Hermansyah / 13519058 / K02 ]")
print("       Selamat datang di Course Organizer       ")
print("================================================")
```

```
show_steps_input = str(input("Apakah anda ingin menampilkan langkah-langkah
penyelesaian secara detail? (Y/N): "))

if (show_steps_input == "Y" or show_steps_input == "y"):
    show_steps = True

while (not exit):
    result = []     # Hasil pengambilan mata kuliah
    semester = 1    # Semester tracker

    # Meminta input nama file
    # Input dalam bentuk nama file saja, tanpa path dan ekstensi
    file_name = str(input("\nMasukkan nama file (contoh: 3): "))

    # Membuat graph dari file teks terkait
    g2 = makeGraphFromTxt(file_name)

    print("\nGraf yang anda masukkan: \n")
    g2.printGraph()

    # Lakukan iterasi pencarian mata kuliah selama graph belum kosong
    while (not g2.isGraphEmpty()):
        lowest_degree_v = getLowestDegree(g2)   # Mengambil vertex lowest in degree

        if (show_steps):
            print(f"\nSemester {semester_romawi[semester]} mengambil: {',
'.join(lowest_degree_v)}")  # Output matakuliah yang dapat diambil

        # Manipulasi graph dengan mendelete vertex terkait
        curr = []
        for v in lowest_degree_v:
            curr.append(v)
            removeAllEdgeFrom(g2, v)
        result.append(curr)

        if (show_steps):
            # Mencetak graph
            print("\nGraph:")
            g2.printGraph()

            # Mencetak sisa jumlah mata kuliah
            print(f"Jumlah mata kuliah tersisa: {g2.V}")

        semester += 1

    # Output hasil akhir
    print("\nSelesai merencanakan pengambilan mata kuliah.\n")

    for i in range(len(result)):
        print(f"Semester {semester_romawi[i+1]}: ", end="")
        print(result[i][0], end="")
        for j in range(1, len(result[i])):
            print(f", {result[i][j]}", end="")
        print()

    # Pilihan exit
    exit_choice = str(input("\nApakah anda ingin memproses file lain? (Y/N): "))
    if (exit_choice == "N" or exit_choice == "n"):
        exit = True

print("\nTerima kasih telah menggunakan program ini!")
```

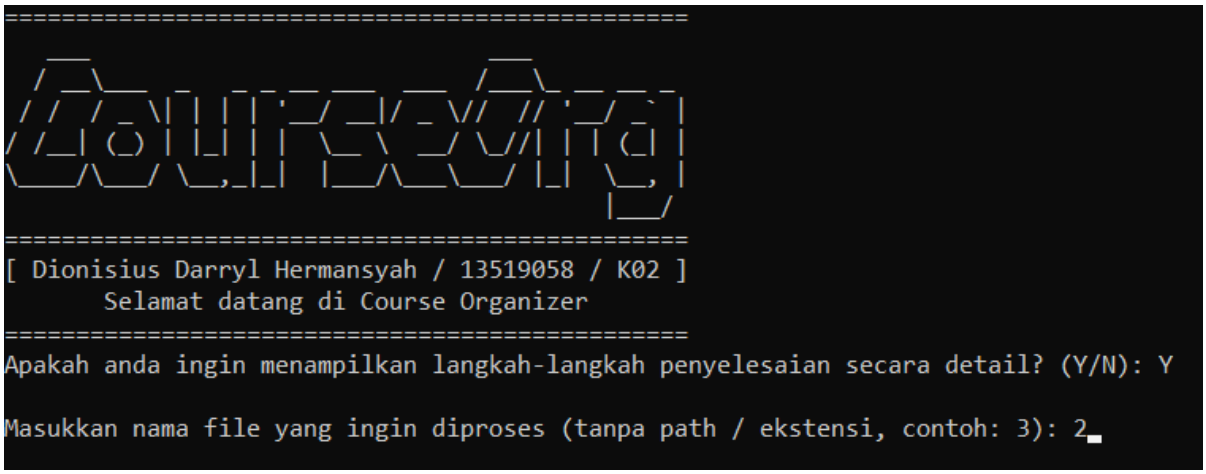
graph.py

```
"""
Definisi kelas Graph dan method yang berhubungan dengan pemrosesan graph
"""

class Graph:
    def __init__ (self, vertices):
        self.E = dict()     # Edges / Kumpulan edge
        self.V = vertices   # Vertices / Kumpulan vertex

    # Menambahkan sebuah edge ke Graph
    def addEdge(self, e, v):
        if v == None: # Vertex tanpa derajat masuk
            self.E[e] = []
        else:
            if e not in self.E.keys():
                self.E[e] = [v]
            else:
                self.E[e].append(v)

    # Mengoutput struktur graph dengan format contoh:
    # A -> B -> C
    # D
    # E -> F
    def printGraph(self):
        for k, v in self.E.items():
            print(str(k) + " ", end="")
            for v_item in v:
                print("->", end="")
                print(f" {v_item} ", end="")
            print()

    # Cek apakah sebuah graph kosong
    # Graph kosong jika vertices dan edge kosong (0)
    def isGraphEmpty(self):
        return self.V == 0

# Membaca data graph dari file .txt dan mereturn sebuah Graph
# Path default : "./test"
# file_name tanpa dituliskan ekstensi (.txt)
def makeGraphFromTxt(file_name):
    # Variabel
    res = []
    res_cleaned = []
    all_vertex = list()

    # Open dan read file
    f = open(f"../test/{file_name}.txt", "r")

    # Append setiap line ke array result
    for line in f:
        res.append(line.split(","))

    # Cleaning tanda ' ', '.' dan '/n'
    for el in res:
        res_cleaned.append([])
        for i in range(len(el)):
            el_cleaned = el[i].replace(" ", "").replace(".", "").replace("\n", "")
            res_cleaned[len(res_cleaned)-1].append(el_cleaned)

    # Menyimpan semua vertex yang ada
    for el in res_cleaned:
```

```
        for v in el:
            all_vertex.append(v)

    all_vertex = set(all_vertex)

    # Membuat sebuah graph dengan vertex sebanyak panjang himpunan all_vertex
    graph = Graph(len(all_vertex))

    # Adding setiap edge yang ada
    for el in res_cleaned:
        if (len(el) == 1): # Vertex tanpa derajat masuk
            graph.addEdge(el[0], None)
        else:
            for i in range(1, len(el)):
                graph.addEdge(el[0], el[i])

    return graph

# Mereturn vertex dengan derajat masuk terendah dari sebuah graph
def getLowestDegree(graph):
    d = {}

    # Mengambil value (derajat masuk) setiap vertex
    for k, v in graph.E.items():
        d[k] = len(v)

    # Mencari value (derajat masuk) terendah
    min_v = min(d.values())
    res = [k for k, v in d.items() if v==min_v]

    return res

# Menghapus semua hubungan edge pada graph p dari semua vertex ke vertex x
def removeAllEdgeFrom(graph, x):
    # Jika vertex k terhubung dengan x, hapus edge antara k dan x
    for k, v in graph.E.items():
        if x in v:
            v.remove(x)

    # Jika vertex tidak punya derajat masuk, maka hapus vertex
    if len(graph.E[x]) == 0:
        graph.E.pop(x)
        graph.V -= 1
```

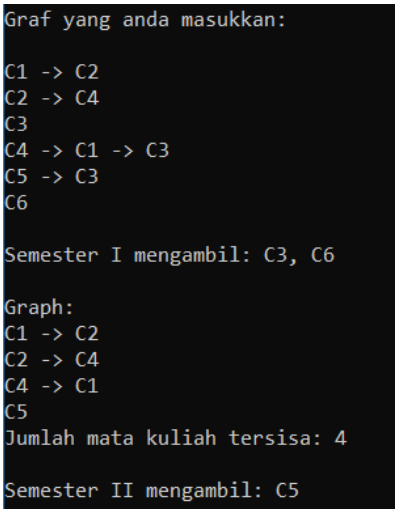
III. INPUT DAN OUTPUT

Program tugas kecil 2 ini diberi nama *Course Organizer* atau CourseOrg. Pada menu awal program, pengguna dapat memilih apakah ingin menampilkan langkah-langkah penyelesaian menggunakan algoritma *topological sort* secara detail atau hanya ingin menampilkan hasil akhirnya. Berikut ini merupakan contoh tampilan awal program:

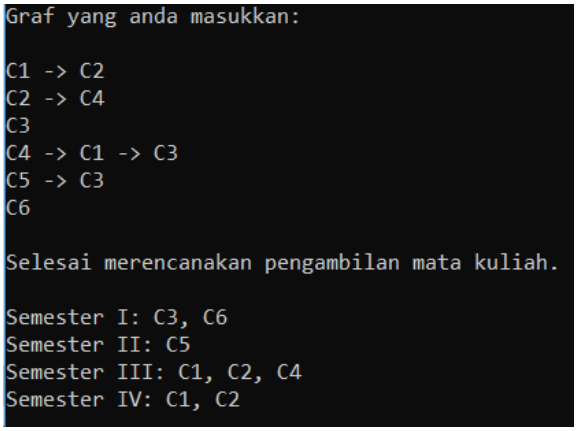


Gambar 2. Tampilan awal program

Gambar 3 (a) menunjukkan contoh output program jika pengguna memilih untuk menampilkan langkah-langkah penyelesaian. Gambar 3 (b) menunjukkan contoh output program jika hanya hasil akhir yang dipilih untuk ditampilkan.



(a)

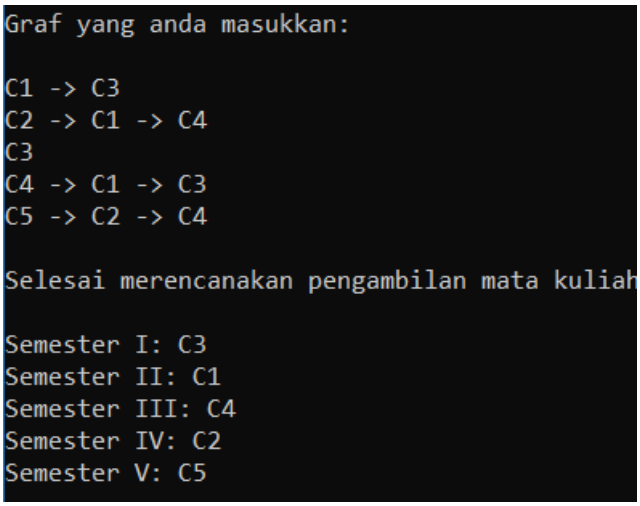
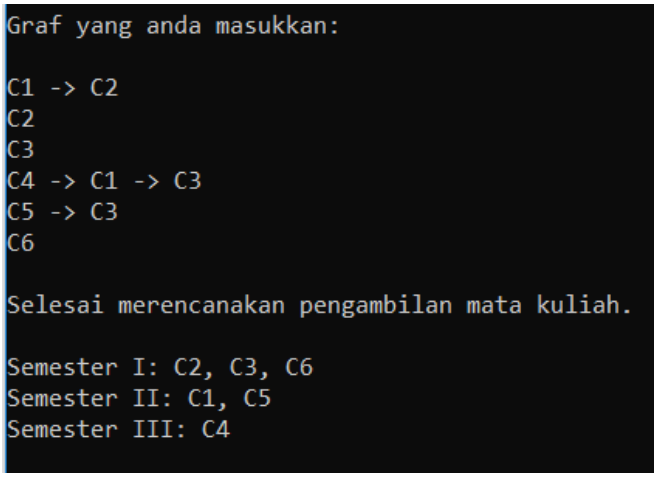


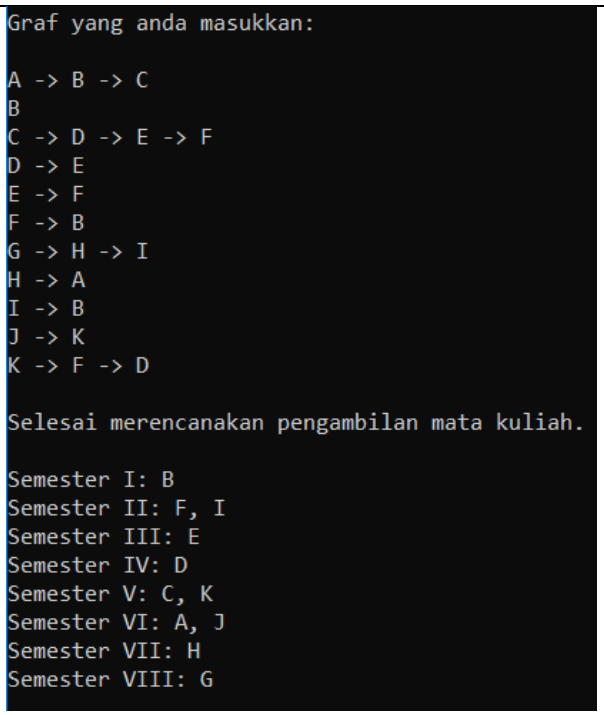
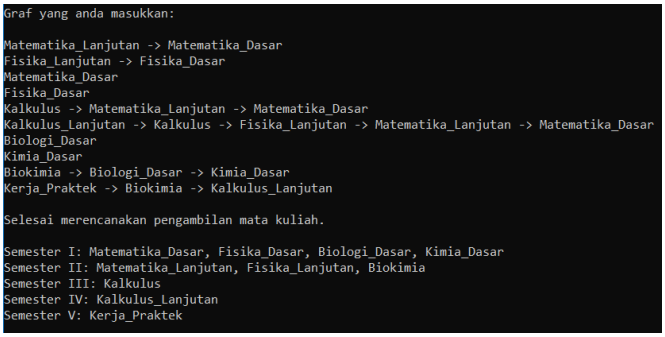
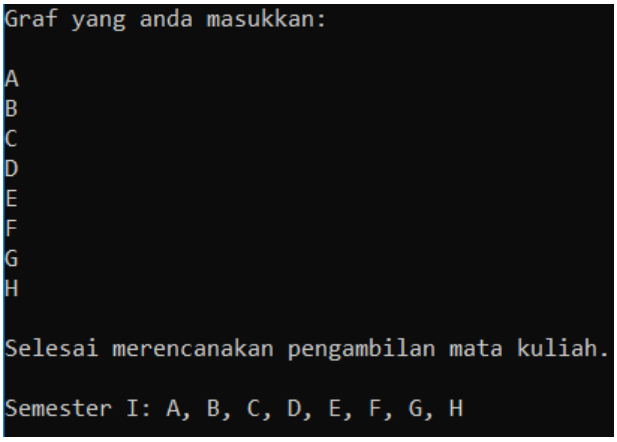
(b)

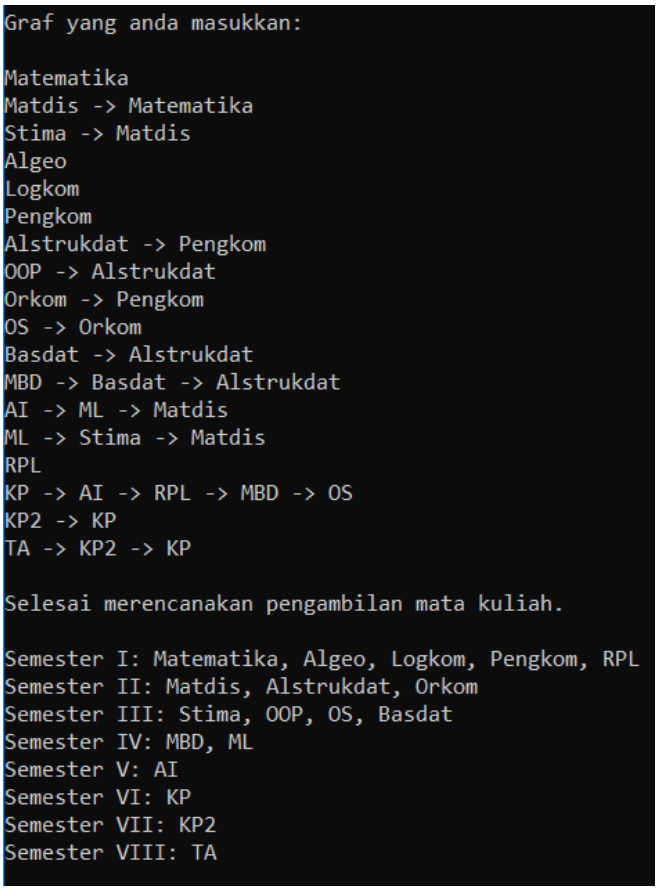
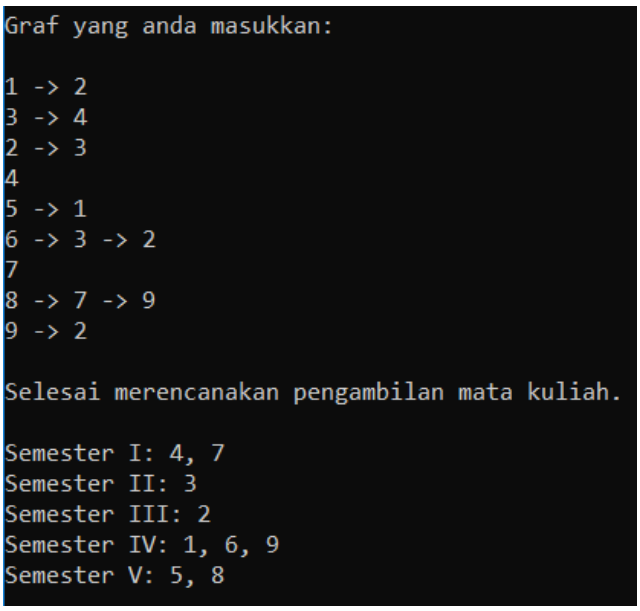
Gambar 2. (a) Tampilan program dengan langkah-langkah penyelesaian dan (b) opsi tanpa langkah-langkah penyelesaian

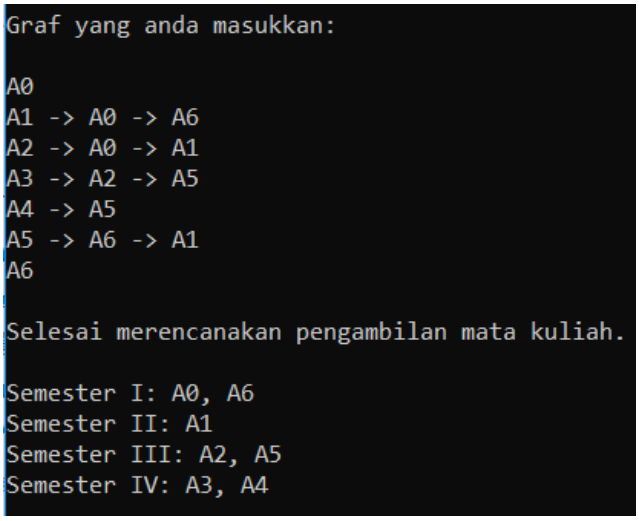
Tabel 1. menunjukkan hasil *test case* yang telah dipastikan merupakan DAG.

Tabel 1. Input dan output program

No	Input	Output
1	C1, C3. C2, C1, C4. C3. C4, C1, C3. C5, C2, C4.	
2	C1, C2. C2. C3. C4, C1, C3. C5, C3. C6.	
3	A, B, C. B. C, D, E, F. D, E.	

	E, F. F, B. G, H, I. H, A. I, B. J, K. K, F, D.	
4	Matematika_Lanjutan, Matematika_Dasar. Fisika_Lanjutan, Fisika_Dasar. Matematika_Dasar. Fisika_Dasar. Kalkulus, Matematika_Lanjutan, Matematika_Dasar. Kalkulus_Lanjutan, Kalkulus, Fisika_Lanjutan, Matematika_Lanjutan, Matematika_Dasar. Biologi_Dasar. Kimia_Dasar. Biokimia, Biologi_Dasar, Kimia_Dasar. Kerja_Praktek, Biokimia, Kalkulus_Lanjutan.	
5	A. B. C. D. E. F. G. H.	

6	Matematika. Matdis, Matematika. Stima, Matdis. Algeo. Logkom. Pengkom. Alstrukdat, Pengkom. OOP, Alstrukdat. Orkom, Pengkom. OS, Orkom. Basdat, Alstrukdat. MBD, Basdat, Alstrukdat. AI, ML, Matdis. ML, Stima, Matdis. RPL. KP, AI, RPL, MBD, OS. KP2, KP. TA, KP2, KP.	
7	1, 2. 3, 4. 2, 3. 4. 5, 1. 6, 3, 2. 7. 8, 7, 9. 9, 2.	

8	A0. A1, A0, A6. A2, A0, A1. A3, A2, A5. A4, A5. A5, A6, A1. A6.	

Evaluasi program secara umum ditunjukkan oleh tabel 2, dimana, program telah dapat memenuhi seluruh poin persyaratan yang ada.

Tabel 2. Evaluasi program

Poin	Ya	Tidak
1. Program berhasil dikompilasi	V	
2. Program berhasil *running*	V	
3. Program dapat menerima berkas input dan menuliskan output	V	
4. Luaran sudah benar untuk semua kasus input	V	

IV. ALAMAT GITHUB

Berikut merupakan alamat *repository* GitHub dari *source code* yang digunakan, laporan, program dalam bentuk file Python (.py), beserta file test. Untuk menguji program, harus dipastikan bahwa file test berada dalam direktori ./test/.

https://github.com/dionisiusdh/courses-organizer

V. DAFTAR PUSTAKA

[1] Levitin, A. 2012. *Introduction to the Design & Analysis of Algorithms, 3rd Edition*. London: Pearson.

[2] Munir, R. 2021. *Algoritma Decrease and Conquer*. Bandung: Institut Teknologi Bandung.

[3] Svirin, A. 2021. *Topological Sorting*. Dilansir dari www.math24.net pada 24 Februari 2021.