Penyusunan Rencana Kuliah dengan Topological Sort (Penerapan Decrease and Conquer)

Diajukan sebagai salah satu tugas kecil mata kuliah Strategi Algoritma pada Semester II Tahun Akademik 2020-2021

oleh

Azmi Muhammad Syazwana 13519151

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

BANDUNG

2021

DAFTAR ISI

BAB 1

DESKRIPSI MASALAH

1.1 Spesifikasi Tugas

Deskripsi tugas :

Pada tugas kali ini, mahasiswa diminta **membuat aplikasi sederhana** yang dapat menyusun rencana pengambilan kuliah, dengan memanfaatkan algoritma **Decrease and Conquer**. Penyusunan Rencana Kuliah diimplementasikan dengan menggunakan pendekatan *Topological Sorting*. Berikut akan dijelaskan tugas yang dikerjakan secara detail.

1. Aplikasi akan menerima daftar mata kuliah beserta prasyarat yang harus diambil seorang mahasiswa sebelum mengambil mata kuliah tersebut. Daftar mata kuliah tersebut dituliskan dalam suatu file teks dengan format:

```
<kode_kuliah_1>,<kode kuliah prasyarat - 1>, <kode kuliah prasyarat - 2>, <kode kuliah prasyarat - 3>.
<kode_kuliah_2>,<kode kuliah prasyarat - 1>, <kode kuliah prasyarat - 2>.
<kode_kuliah_3>,<kode kuliah prasyarat - 1>, <kode kuliah prasyarat - 2>, <kode kuliah prasyarat - 3>, <kode kuliah prasyarat - 4>.
<kode_kuliah_4>.
.
.
.
```

Gambar 1. Format File Teks untuk Masukan Daftar Kuliah

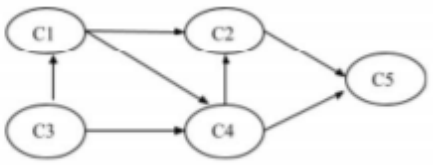
Sebuah kode_kuliah mungkin memiliki nol atau lebih prasyarat kuliah. Kode_kuliah bisa diambil pada suatu semester jika semua prasyaratnya sudah pernah diambil di semester sebelumnya (tidak harus 1 semester sebelumnya). Asumsi semua kuliah bisa diambil di sembarang semester, baik semester ganjil maupun semester genap.

Sebagai contoh, terdapat 5 kuliah yang harus diambil seorang mahasiswa dengan daftar prerequisite dalam file teks sebagai berikut. Dari Gambar 2 terlihat bahwa kuliah C3 tidak memiliki prerequisite.

```
C1, C3.
C2, C1, C4.
C3.
C4, C1, C3.
C5, C2, C4.
```

Gambar 2. Contoh sebuah berkas masukan Daftar Kuliah

Asumsi untuk persoalan ini, kuliah dan prerequisite nya pasti berupa Directed Acyclic Graph (DAG), dan untuk contoh pada Gambar 2, dapat dilihat representasi DAG pada gambar 3.



Gambar 3. DAG dari daftar kuliah pada Gambar 2

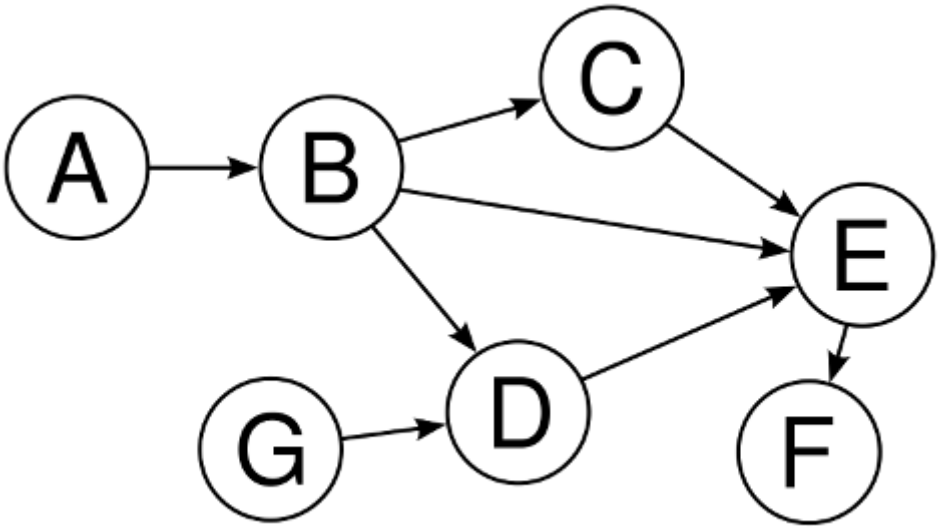
2. Dari file teks yang telah diterima, ditentukan kuliah apa saja yang bisa diambil di semester 1, semester 2, dan seterusnya. Sebuah kuliah tidak mungkin diambil pada semester yang sama dengan prerequisitenya. Untuk menyederhanakan persoalan, tidak ada Batasan banyaknya kuliah yang bisa diambil pada satu semester.

BAB II
TEORI SINGKAT

2.1 Topological Sort

Dalam dunia competitive programming, topological sort sudah menjadi problem yang cukup populer. Kebanyakan orang sering menyingkat topological sort menjadi toposort. Karena males nulis panjang-panjang selanjutnya disini juga akan disingkat jadi toposort. Problem toposort ini termasuk dalam problem graph. Lebih khususnya lagi, problem ini membahas pengurutan node pada directed acyclic graph (DAG). Untuk lebih mengenal toposort, kita harus mengenal DAG terlebih dahulu.

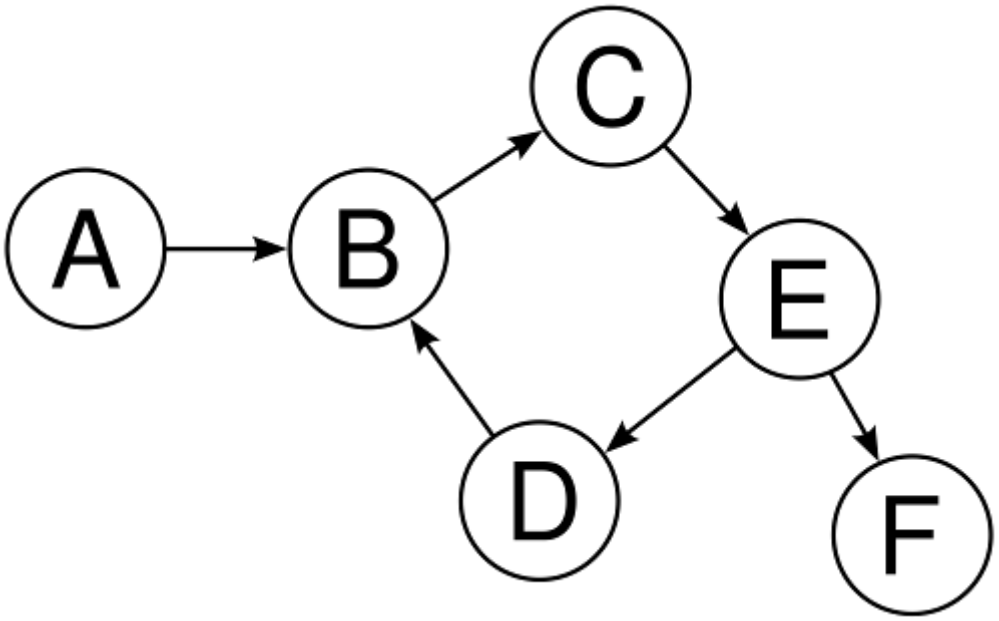
Direct itu artinya berarah, acyclic berarti tidak memiliki loop. Intinya DAG adalah graph berarah yang tidak memiliki loop. Untuk lebih jelasnya tentang DAG dapat dibaca di sumber lain, cukup banyak artikel yang sudah membahas DAG.



Gambar di atas merupakah salah satu contoh DAG. Jika kita lihat pada graph tersebut, kita tidak menemui adanya loop. Artinya jika kita menelusuri graph tersebut dari vertex manapun dengan memilih edge manapun, maka tidak akan pernah kembali lagi ke vertex yang sama untuk kedua kalinya. Dan pasti selalu ada vertex dimana kita tidak bisa kemana-mana lagi.

Karena itu lah, topological sorting menjadi mungkin pada graph DAG. Topological sorting pada suatu graph DAG adalah pengurutan vertex pada graph sehingga setiap ada edge yang menghubungkan vertex A dengan B, A selalu datang sebelum B pada hasil pengurutan. Karena tidak adanya loop pada graph DAG, toposort menjadi mungkin dilakukan. Tinjau kasus untuk graph yang

bukan DAG berikut :



Jika kita mengurutkan vertex pada graph di atas maka akan terjadi keambiguan. Karena adanya loop pada vertex B-C-E-D, maka tidak dapat ditentukan urutannya. Apakah yang paling awal vertex B, C, E atau D? Hal tersebut tidak dapat ditentukan karena loop tersebut.

Salah satu algoritma yang cukup populer digunakan dalam menyelesaikan masalah toposort ini adalah Kahn's Algorithm. Algoritma tersebut memiliki kompleksitas waktu kira-kira $O(E + V)$ dimana E adalah jumlah edge dan V adalah jumlah vertex. Karena kompleksitas yang relatif rendah ini, algoritma inilah yang sering digunakan.

Untuk menyelesaikan toposort kita tahu satu hal : vertex yang tidak memiliki edge masuk atau dengan kata lain indegree-nya adalah nol pasti bisa menjadi vertex yang pertama dalam sorting. Pada contoh DAG di atas, vertex A dan G tidak memiliki edge yang masuk ke dalamnya, oleh karena itu A dan G dapat menjadi vertex pertama dalam pengurutan. Misalkan kita ambil vertex A sebagai vertex pertama. Jika hal itu kita lakukan, maka B dapat menjadi vertex berikutnya karena syarat B diambil adalah sudah diambilnya A. Pada dasarnya ketika kita mengambil suatu vertex X, kita dapat menghilangkan syarat vertex lain. Contohnya pada DAG diatas syarat untuk diambilnya vertex E adalah : sudah diambilnya vertex B, sudah diambilnya vertex C, dan sudah diambilnya vertex D. Jika kita mengambil vertex B, artinya syarat diambilnya vertex E berkurang satu, begitu juga ketika kita mengambil vertex C dan D.

Dengan melakukan observasi seperti diatas, kita dapat definisikan “syarat” untuk setiap vertex sebagai banyaknya edge yang masuk ke dalam vertex tersebut (indegree). Dalam contoh diatas artinya :

- syarat A = 0
- syarat B = 1
- syarat C = 1
- syarat D = 2
- syarat E = 3
- syarat F = 1
- syarat G = 0

Setiap vertex yang memiliki “syarat” nol dapat diambil. Setiap vertex X diambil, jika X berhubungan dengan vertex Y, maka syarat Y dapat dikurangi satu.

2.2 Algoritma *Decrease and Conquer*

Decrease and conquer adalah metode perancangan algoritma dengan mereduksi persoalan menjadi dua upa-persoalan (*sub-problem*) yang lebih kecil, tetapi selanjutnya hanya memproses satu sub-persoalan saja. Berbeda dengan *divide and conquer* yang memproses semua upa-persoala dan menggabung semua solusi setiap sub-persoalan. Di dalam literatur lama, semua algoritma yang membagi persoalan menjadi dua upa-persoalan yang lebih kecil dimasukkan ke kategori *divide and conquer*. Meskipun demikian, tidak kedua upa-persoalan hasil pembagian diselesaikan. Jika hanya satu upa-persoalan yang diselesaikan, maka tidak tepat dimasukkan sebagai algoritma *divide and conquer*. Mereka dikategorikan sebagai *decrease and conquer*.

Algoritma *decrease and conquer* terdiri dari dua tahapan:

1. *Decrease*: mereduksi persoalan menjadi beberapa persoalan yang lebih kecil (biasanya dua upa-persoalana).
2. *Conquer*: memproses satu upa-persoalan secara rekursif

Tidak ada tahap *combine* dalam *decrease and conquer,* karena hanya satu upa-persoalan yang diselesaikan.

Tiga varian *decrease and conquer*:

1. *Decrease by a constan:* ukuran instans persoalan direduksi sebesar konstanta yang sama setiap iterasi algoritma. Biasayan konstanta = 1.
2. *Decrease by a constant factor*: ukuran instans persoalan direduksi sebesar faktor konstanta yang

sama setiap iterasi algoritma. Biasanya faktor konstanta = 2.

3. *Decrease by a variable size*: ukuran instans persoalan direduksi bervariasi pada setiap iterasi algoritma.

BAB III

PENYELESAIAN MASALAH DENGAN ALGORITMA TOPOLOGICAL SORT

Jadi algoritma yang dipakai yaitu decrease and conquer by a constant untuk topological sort ini.

Langkah – langkah penyelesaian masalah adalah sebagai berikut:

1. Program membaca semua isi file dari input.txt
2. Memisahkan kode matkul dan prereq kemudian memasukkan ke dalam list courses
3. Mencari courses yang tidak mempunyai preq sebagai node awal
4. Membaca semua kemungkinan node
5. Membaca node yang merupakan cabang dari preq awal
6. Menampilkan node berdasarkan semesternya

Untuk cara kerja program lebih jelasnya terdapat pada komentar di source code di bawah ini

Source Code:

1. ProgramTopologicalSorting_13519151.java

```
package programtopologicalsorting;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Hashtable;
import java.util.List;

public class ProgramTopologicalSorting_13519151 {

    // nama file yang akan diproses
    String FILE_NAME = "test/input.txt";

    // Buat String pesan semester. Pesan yang akan disampaikan di akhir program na
nti.
    String PesanSemester = "";

    // Inisialisasi map kuliah
    Hashtable<String, Course_13519151> courses = new Hashtable<>();
    ArrayList<String> listMatkul = new ArrayList<>();

    /*
    Integer           = semester
```

```
    ArrayList<String> = list matkul
     */
    Hashtable<Integer, ArrayList<String>> sollution = new Hashtable<>();

    // kedalaman node
    int depth = 0;

    private ArrayList<String> getNode(int init) {
        ArrayList<String> hasil = new ArrayList<>();

        int countSollution = sollution.size();  // mencari banyak solusi / banyak
semester yang sudah ada
        for (int i = init - 1; i < countSollution; i++) {   // mengulang sebanyak
banyak semester

            int countCourses = courses.size();          // mencari banyak course
            for (int k = 0; k < countCourses; k++) {        // mengulang sebanyak
banyak course

                int countMatkul = sollution.get(i).size();// mencari banyak matkul
 didalam solusi
                for (int j = 0; j < countMatkul; j++) {     // mengulang sebanyak
matkul yang ada didalam solusi

                    List<String> values = courses.get(listMatkul.get(k)).prerequis
ites;
                    if (!values.isEmpty()) {  // jika data matkulnya tidak kosong,
 maka proses

                        int countValues = values.size();   // mencari banyak preq
didalam courses
                        int countSameValues = 0;  // variabel untuk mencari nilai
yang sama
                        for (int l = 0; l < countValues; l++) {
                            if (values.get(l).equals(sollution.get(i).get(j))) {
                                countSameValues++;
                            }
                        }

                        if (countSameValues == countValues) {// jika "banyak preq"
 dan "banyak nilai yang sama" nilainya sama
                            hasil.add(courses.get(listMatkul.get(k)).name); //maka
 masukkan kedalam solusi
                        }
                    }
                }
            }
```

```
            System.out.println("\n");

        }
        return hasil;
    }

    /* fungsi untuk sorting*/
    private void sort() {
        int countCourses = courses.size();
// ambil banyak course
        ArrayList<String> initSollution = new ArrayList<>();
// array list untuk inisialisasi solusi

        /* fungsi untuk mencari kedalaman node */
        for (int i = 0; i < countCourses; i++) {
            List<String> preq = courses.get(listMatkul.get(i)).prerequisites;
            int countPreq = preq.size();

            /* mengeset kedalam node */
            if (countPreq > depth) {        // jika banyak preq lebih besar dari d
epth
                depth = countPreq;          // ganti nilai depth dengan banhyaknya
 preq
            } else if (countPreq == 0) {
                initSollution.add(courses.get(listMatkul.get(i)).name);
            }
        }
        depth++;
        /* akhir fungsi untuk mencari kedalaman node */

        if (initSollution.isEmpty()) {                      // jika tidak ada ini
t sollution, maka tidak ada solusi
            System.out.println("Solusi tidak ditemukan!");
        } else {
            System.out.println("\n\nSolusi bisa dicari");
            sollution.put(0, initSollution);
            for (int i = 1; i < depth; i++) {
                ArrayList<String> xxx = getNode(i);
                sollution.put(i, xxx);
            }

            for (int i = 0; i < depth; i++) {
                System.out.print("Semester " + (i + 1) + " :");
                int countMatkul = sollution.get(i).size();
                for (int j = 0; j < countMatkul; j++) {
                    System.out.print(" " + sollution.get(i).get(j));
                }
```

```
                System.out.println("");
            }
        }

    }

    /* fungsi untuk menampilkan course yang berhasil tersimpan */
    private void showCourses() {
        int countCourses = courses.size();
        for (int i = 0; i < countCourses; i++) {
            List<String> preq = courses.get(listMatkul.get(i)).prerequisites;
            preq.forEach((temp) -> {
                System.out.print(temp + " ");
            });
            System.out.println();
        }
    }

    private void mainFuction() {
        try {
            BufferedReader br = new BufferedReader(new FileReader(new File(FILE_NA
ME)));
            String kodeKuliah;

            // proses pembacaan dari file input.txt
            while ((kodeKuliah = br.readLine()) != null) {
                kodeKuliah = kodeKuliah.replace(".", "");// mengganti titik dengan
 spasi kosong
                String[] listKode = kodeKuliah.split(",");      // memotong baris
berdasarkan koma

                /* proses memasukkan pre-requisite kedalam list */
                Course_13519151 temp = new Course_13519151(listKode[0]);
                int countPreq = listKode.length;
                for (int i = 1; i < countPreq; i++) {
                    temp.AddPrerequisite(listKode[i]);
                }
                /* akhir proses memasukkan pre-requisite kedalam list */

                courses.put(listKode[0], temp);         // menambah course

                listMatkul.add(listKode[0]);     // tambahkan kedalam listMatkul
            }

            br.close();

//            showCourses();
```

```
            sort();             // panggil fungsi sort

        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public static void main(String[] args) {
        ProgramTopologicalSorting_13519151 p = new ProgramTopologicalSorting_13519
151();
        p.mainFuction();
    }

}
```

2. Course_13519151.java

```
package programtopologicalsorting;

import java.util.ArrayList;
import java.util.List;

public class Course_13519151 {

    public String name;
    public List<String> prerequisites;
    public int semester;
    public int time_begin;
    public int time_finish;

    public Course_13519151(String _name) {
        name = _name;
        prerequisites = new ArrayList<>();
        semester = 0;
    }

    public void AddPrerequisite(String preq) {
        prerequisites.add(preq);
    }

    public boolean contains(String preq) {
        return prerequisites.contains(preq);
    }
}
```

Kode program dapat diakses di sini:
https://github.com/azmisyazwana/Tucil-2-Stima.git

BAB V

INPUT DAN OUTPUT

No	Input	Output
1.		Solusi benar
2.		Solusi kurang tepat
3.		Solusi kurang tepat
4.		Solusi kurang tepat
5.		Solusi kurang tepat

6.		Solusi kurang tepat
7.		Solusi kurang tepat
8.		Solusi kurang tepat

LAMPIRAN

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

DAFTAR PUSTAKA

- https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Tugas-Kecil-2-(2021).pdf
- https://cp-itb.github.io/blog//graph/sorting/2017/05/04/toposort.html