1. Algoritma *topological sort*

Pertama-tama program membuka file text yang akan berisi *test case* pada folder test. PERHATIKAN BAHWA DALAM FILE TEXT YANG DIBERIKAN TIDAK BOLEH ADA BARIS YANG KOSONG DARI AWAL HINGGA AKHIR FILE. Lalu program akan membaca tiap baris yang ada dan menaruhnya dalam suatu *vector* string. Dari *vector* string tersebut, akan didata *node* apa saja yang muncul, sekaligus membuat *adjacency list* dari tiap-tiap *node*-nya. Karena program tidak menggunakan data struktur khusus, mapping menggunakan indeks *vector* list *node* menjadi andalan ketika menyelesaikan permasalahan *topological sort* ini. Sehingga *adjacency list* yang terbentuk mempunyai tipe *integer*. Hal ini untuk memudahkan implementasi DFS yang saya pilih. Setelah itu lakukan DFS untuk mendapatkan *topological sort*-nya dengan langkah-langkah sebagai berikut :

1. Jika sebuah *node* belum dikunjungi, tandai *node* tersebut
2. Lalu untuk setiap tetangga dari *node* tersebut yang belum dikunjungi, lakukan DFS
3. Setelah melakukan DFS ke tetangga-tetangganya (atau jika tetangganya tidak ada), masukkan *node* tersebut ke *vector* order yang akan berisi urutan *node* secara terbalik
4. Lakukan proses ini hingga tiap *node* berhasil dikunjungi
5. *Reverse vector* order sehingga urutan yang diinginkan tercapai.

Lalu, buatlah sebuah *array of vector* bertipe *integer* yang akan berisi indeks *node* mata kuliah mana saja yang perlu diambil di tiap semester. Iterasi tiap *node* yang ada dalam *vector* order, lalu jika *node* yang diambil ternyata bertetangga dengan salah satu *node* yang berada dalam elemen *vector* ke i dari *array* semester, maka iterasi i nya (semesternya bertambah). Sehingga dipastikan output yang dihasilkan memenuhi syarat mata kuliah yang diambil tidak boleh bersamaan dengan mata kuliah *prerequisite*-nya.

Algoritma DFS ini bisa dikategorikan sebagai algoritma *Decrease and Conquer* karena setiap melakukan proses DFS, yang diproses hanya *node* yang belum dikunjungi. Sehingga DFS yang dilakukan dianggap memperkecil permasalahan graf secara *decrease by constant*. Selain DFS, ada juga algoritma *source removal* yang bisa digunakan untuk menyelesaikan permasalahan *topological sorting* ini. Namun, penulis memilih menggunakan DFS karena lebih menyukai algoritma penelusuran *graph* :D. Kompleksitas algoritma keduanya sama, yaitu O(V+E) dengan V = banyak *vertices* / *nodes* dan E = banyak *edges*. Hal ini disebabkan karena pada kedua algoritma, proses dijalankan untuk mengecek seluruh *node* (O(V)) lalu mengecek tetangga-tetangga tiap *node* (O(EV)). Jika ditotal, maka kompleksitasnya O(V + ) = O (V+E).

2. Source code program

Isi dari 13519105-main.cpp :

#include <bits/stdc++.h>

#include "13519105-modul.hpp"

using namespace std;

int main(){

string namafile;

getline(cin,namafile);

ifstream fin("../test/" +namafile);

vector<string> graph;

int dummy = openFile(graph,fin); //buka isifile kedalam vector graph, satu barisnya sebagai 1 elemen

if (dummy) return 1;

vector<string> node;

listNode(graph,node); //list node-node yang ada dalam graph

vector<int> adjlist [node.size()];

makeAdjList(graph,node,adjlist); //buat adjacency list dari tiap-tiap node

vector<int> order;

vector<bool> visited; //set semua node belum dikunjungi

for(int i = 0; i < node.size(); ++i){

visited.push\_back(false);

}

topoSort(visited,order,adjlist); //lakukan topological sort

vector<int> Semester[node.size()];

showSMT(Semester,order,adjlist,node); //tampilkan hasilnya

cout<< endl;

fin.close();

return 0;

}

Isi dari 13519105-modul.hpp:

#ifndef \_\_13519105MODUL\_HPP

#define \_\_13519105MODUL\_HPP

#include <bits/stdc++.h>

using namespace std;

static string trim(const string& str);

static bool isMember(const vector<string>& v, const string& str);

static int indexKe(const vector<string>& v, const string& str);

static string stringKe(const vector<string>& v, const int& i);

int openFile(vector<string> & graph, ifstream & fin);

void listNode(const vector<string> & graph, vector<string>& node);

void makeAdjList(const vector<string> & graph, const vector<string> & node, vector<int> adjlist[]);

void topoSort(vector<bool> &visited, vector<int>& order, vector<int> adjlist[]);

void DFSTopo(int node, vector<bool> & visited, vector<int> & order, vector<int>adjlist[]);

void showSMT(vector<int>Semester[],const vector<int>& order, vector<int>adjlist[], const vector<string> & node);

#endif

Isi dari 13519105-modul.cpp:

#include "13519105-modul.hpp"

#include <bits/stdc++.h>

using namespace std;

string trim(const string& str)

{

size\_t first = str.find\_first\_not\_of(" \n\t\r\v");

size\_t last = str.find\_last\_not\_of(" \n\t\r\v");

return str.substr(first, (last-first+1));

}

bool isMember(const vector<string>& v, const string& str){

for (auto string : v){

if (string == str) return true;

}

return false;

}

int indexKe(const vector<string>& v, const string& str){

if (isMember(v,str)){

int i;

for (i = 0; i < v.size(); ++i){

if (v[i] == str) return i;

}

}else return -1;

}

string stringKe(const vector<string>& v, const int& i){

if (i < v.size()) return v[i];

return "Null";

}

int openFile(vector<string> & graph,ifstream & fin){

string line;

if (!fin){

cout << "Problem opening" << endl;

return 1;

}else{

while(getline(fin,line, '.')){ //ambil setiap line yang diakhiri titik

graph.emplace\_back(trim(line)); //trim agar tidak ada typo di dalamnya

}

cout << line;

}

return 0;

}

void listNode(const vector<string> & graph, vector<string>& node){

for (auto sentence : graph){

string n00de;

size\_t prekoma = 0;

size\_t koma = sentence.find\_first\_of(","); //cari posisi koma sebagai pemisah

if (koma == string::npos){ //jika ternyata tidak ada koma, masukkan sentence ke dalam node jika belum ada

if (!isMember(node, sentence)){

node.emplace\_back(sentence);

}

}

while (koma != string::npos){ //selama koma masih dapat dicari

n00de = sentence.substr(prekoma,(koma-prekoma)); //ambil substring dari indeks prekoma sampai koma

prekoma = koma+1; //cari prekoma baru

while(sentence[prekoma] == ' ' or sentence[prekoma] == '\t'){

prekoma++;

}

koma = sentence.find\_first\_of(",", koma+1); //cari koma baru

if (!isMember(node,n00de)){ //jika sentence belum ada di dalam node, masukkan ke dalam node

node.emplace\_back(n00de);

}

}

n00de = sentence.substr(prekoma,sentence.length()); //ambil kata terakhir dan cek jika dimasukkan atau belum

if (!isMember(node,n00de)){

node.emplace\_back(n00de);

}

}

}

void makeAdjList(const vector<string> & graph,const vector<string> & node, vector<int> adjlist[]){

for (auto sentence :graph){

string head;

size\_t prekoma = 0;

size\_t koma = sentence.find\_first\_of(","); //cek posisi koma sebagai pemisah

if (koma == string::npos){

//koma tidak ada, satu line hanya berisi satu node atau string, skip ke sentence selanjutnya

}else{

head = sentence.substr(prekoma,(koma-prekoma)); //ambil substring node pertama

int idxh = indexKe(node,head);//cari indeks node head

prekoma = koma+1; //cari prekoma baru

while(sentence[prekoma] == ' ' or sentence[prekoma] == '\t'){

prekoma++;

}

koma = sentence.find\_first\_of(",", koma+1); //cari koma baru

string body;

while (koma != string::npos){ //selama koma masih dapat dicari

body = sentence.substr(prekoma,(koma-prekoma));

prekoma = koma+1; //cari prekoma baru

while(sentence[prekoma] == ' ' or sentence[prekoma] == '\t'){

prekoma++;

}

koma = sentence.find\_first\_of(",",koma+1); //cari koma baru

int idxb = indexKe(node,body); //cari indeks node body

adjlist[idxb].push\_back(idxh); //masukkan node indeks head ke dalam tetangga indeks body

}

body = sentence.substr(prekoma,sentence.length()); //ambil kata terakhir

int idxb = indexKe(node,body); //cari indeksnya

adjlist[idxb].push\_back(idxh);//masukan node indeks head ke dalam tetangganya

}

}

}

void topoSort(vector<bool> & visited, vector<int>& order, vector<int> adjlist[]){

for (int i = 0; i < visited.size(); ++i){

if (visited[i] == false){

DFSTopo(i,visited,order,adjlist); //jika belum dikunjungi, lakukan dfs

}

}

reverse(order.begin(),order.end()); //reverse order hasil dfs sehingga topological sort terurut

}

void DFSTopo(int node,vector<bool> &visited, vector<int>& order, vector<int> adjlist[]){

visited[node] = true;

for (int i = 0; i < adjlist[node].size() ; i++){

int neighbour = adjlist[node][i];

if (!visited[neighbour]){

DFSTopo(neighbour, visited, order, adjlist);

}

}

order.push\_back(node);

}

void showSMT(vector<int>Semester[],const vector<int>& order, vector<int>adjlist[], const vector<string>& node){

int i = 0;

int j = 1;

Semester[0].push\_back(order[0]);

while(j < order.size()){

bool found = false;

for (int k = 0; k < Semester[i].size(); k++){

for (int l = 0; l < adjlist[Semester[i][k]].size(); l++){

if (adjlist[Semester[i][k]][l] == order[j]){

//jika elemen order selanjutnya merupakan tetangga dari salah satu node dalam semester tertentu

found = true;

}

}

}

if (found){

i++; //maka semester dianggap cukup anggotanya dan pindah ke semester selanjutnya

}else{

Semester[i].push\_back(order[j]); // jika tidak tambahkan node ke anggota semester itu

j++; //cek node selanjutnya

}

}

for (int k =0; k <= i; k++){ //print semua node dalam tiap2 elemen semester yang sudah terisi

cout << "Semester " << k+1 << " : ";

for (int j = 0; j < Semester[k].size();j++){

cout << stringKe(node,Semester[k][j]);

if (j == Semester[k].size()-1){

cout <<endl;

}else{

cout << ", ";

}

}

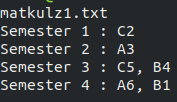
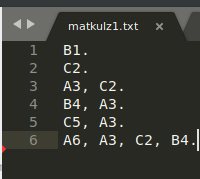
}

}

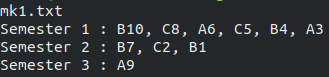
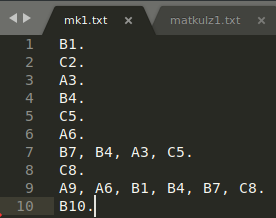
Sepertinya jika di*copy paste* ke file dokumen, formatnya menjadi hancur. Oleh sebab itu saya menaruhnya di <https://github.com/widyaput/Tucil2STIMA> (yang akan dibuka setelah *deadline* pengumpulan tugas selesai)

3. Contoh input/output :

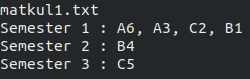
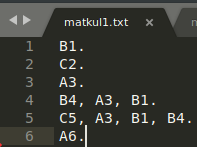
Input/output 1 :



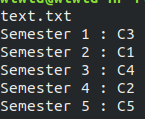
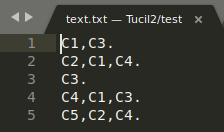
Input/output 2 :



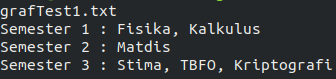
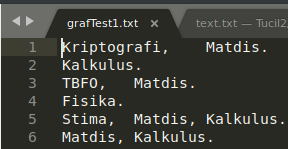
Input/output 3:



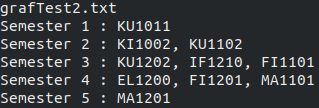
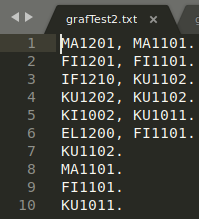
Input/output 4:



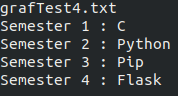
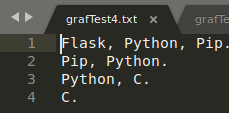
Input/output 5 :



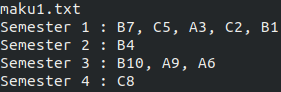
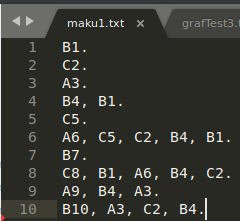
Input/output 6 :



Input/output 7 :



Input/output 8 :



Perhatikan bahwa solusi *topological sort* bukanlah solusi yang unik, sehingga jika keluaran yang dikeluarkan program berbeda-beda ketika diberi masukan yang sama maka hal itu menjadi hal yang lumrah asalkan jawabannya dapat dipastikan kebenarannya masing-masing.

4. Alamat repository program

*Project* ini dapat dilihat di <https://github.com/widyaput/Tucil2STIMA> yang akan di*public* ketika *deadline* selesai.

|  |  |  |
| --- | --- | --- |
| Poin | Ya | Tidak |
| 1. Program berhasil dikompilasi | ✔ |  |
| 1. Program berhasil *running* | ✔ |  |
| 1. Program dapat menerima berkas *input* dan menuliskan *output* | ✔ |  |
| 1. Luaran sudah benar untuk semua kasus | ✔ |  |