Tugas Kecil 2 IF2211 Strategi Algoritma
Semester II tahun 2020/2021

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

Nama : Ahmad Saladin
NIM : 13519187

Program Studi Teknik Informatika

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung

2021

Algoritma Topological Sort

1. Program membaca data mata kuliah dan prerequisitenya dari file .txt lalu mengkonversi data tersebut dalam bentuk graf. Graf di dalam program direpresentasikan dalam bentuk adjacency list dimana setiap node berisi id, nama mata kuliah, derajat masuk, dan juga list yang berisi alamat node yang terhubung (keluar) dari node tersebut.
2. Kemudian untuk program akan menelusuri seluruh graf dan menyimpan id dari semua node yang memiliki derajat 0 dalam sebuah list. Node-node ini berisi mata kuliah yang tidak memiliki prerequisite atau prerequisitenya sudah diambil.
3. Selanjutnya program akan menampilkan output nama-nama dari mata kuliah yang idnya ada di list tersebut.
4. Lalu program akan menghapus semua node dengan id yang terdapat pada list dari graf. Kaitannya dengan pendekatan Decrease and conquer terdapat pada bagian ini. Setiap iterasi akan mengurangi jumlah node yang ada sehingga persoalan dapat dikerjakan lebih mudah dan cepat. Jumlah node yang dikurangi tidak konstan karena tergantung dengan jumlah node yang memiliki derajat masuk bernilai 0.
5. Langkah 2 sampai 3 akan diulangi sampai seluruh node dalam graf habis atau terjadi 8 kali iterasi yang artinya sudah mencapai batas maksimal semester. Jika setelah 8 iterasi graf belum kosong, program akan menampilkan pesan error.

Source code Program

1. node.hpp

```cpp
#ifndef NODE_HPP
#define NODE_HPP

#include <iostream>
#include <string>
#include <list>
using namespace std;

class Node{
    private:
        int id;
        int inDegree;
        string name;
        static int currentId;
        list<Node*> trail;

    public:

        Node(); //Konstruktor
        Node(string name); //konstruktor
        void deleteTrail(); //Menghapus trail milik node
        int getId(); //mengembalikan id node
        int getInDegree(); //mengembalikan derajat masuk node
        int getOutDegree(); //mengembalikan derajat keluar node
        string getName(); //mengembalikan nama node
        void addInDegree(int n); //menambah derajat masuk node
        void minInDegree(int n); //mengurangi derajat masuk node
        void connect(Node& N); //membuat hubungan (Node, N)
        void disconnect(Node& N); //menghapus hubungan (Node, N)
```

```
        bool isConnected(Node& N); //Mengecek apakah Node dan N terhubung
        bool operator==(Node& N); //mengecek apakah id Node dan N sama
        bool operator!=(Node& N); //mengecek apakah id Node dan N berbeda
        void printTrail(); //Menampilkan trail ke layar

};


#endif
```

2. node.cpp

```
#include "node.hpp"

int Node::currentId = 0;

Node::Node(){
    this->id = currentId;
    this->name = "No name";
    this->inDegree = 0;
    currentId++;
}

Node::Node(string name){
    this->id = currentId;
    this->name = name;
    this->inDegree = 0;
    currentId++;
}

void Node::deleteTrail(){
    list<Node*>::iterator p;
    for (p = this->trail.begin();p!=this->trail.end();++p){
```

```
        (*p)->inDegree--;
    }
}

int Node::getId(){
    return this->id;
}

string Node::getName(){
    return this->name;
}

int Node::getInDegree(){
    return this->inDegree;
}

int Node::getOutDegree(){
    int count =0;
    for (Node* N : this->trail){
        count++;
    }
    return count;
}

void Node::addInDegree(int n){
    this->inDegree += n;
}


void Node::minInDegree(int n){
    this->inDegree -= n;
```

```
}

void Node::connect(Node& N){
    this->trail.push_back(&N);
}

void Node::disconnect(Node& N){
    list<Node*>::iterator p;
    p = this->trail.begin();
    while((*p)->getId() != N.getId()){
        advance(p,1);
    }
    this->trail.erase(p);
}

bool Node::isConnected(Node& N){
    list<Node*>::iterator p;
    for (p = this->trail.begin();p!=this->trail.end();++p){
        if ((*p)->getId() == N.getId()){
            return true;
        }
    }
    return false;
}

bool Node::operator==(Node& N){
    return this->getId() == N.getId();
}

bool Node::operator!=(Node& N){
    return this->getId() != N.getId();
```

```
}

void Node::printTrail(){
    list<Node*>::iterator p;
    for(p=this->trail.begin();p!=this->trail.end();++p){
        cout << (*p)->getName() << " ";
    }
}
```

3. Graph.hpp

```
#ifndef GRAPH_HPP
#define GRAPH_HPP
#include "node.hpp"
#include "list"

class Graph{
    private:
        list<Node> Nodes;

    public:
        Graph(); //konstruktor
        ~Graph(); //destruktor
        void addNode(string name); //membuat node dengan nama name
        void deleteNode(Node& N); //menghapus node N dari graph
        void addEdge(Node& N1, Node& N2); //menambah hubungan (N1, N2)
        void deleteEdge(Node& N1, Node& N2); //menghapus hubungan (N1, N2)
        Node& getNode(int id); //mengembalikan node dengan id id
        Node& getNode(string name); //mengembalikan node dengan nama name
        void printGraph(); //menampilkan graph ke layar
        void printInDegree(); //menampilkan derajat masuk semua node
        //mengembalikan list yang berisi semua id node dengan derajat masuk 0
        list<int> getZeroNodes();
```

```
        int countNodes(); //menghitung jumlah semua node
        bool nodeExist(string name); //mengecek apakah terdapat node dengan nama name

};

#endif
```

4. Graph.cpp

```
#include "Graph.hpp"

Graph::Graph(){

}


Graph::~Graph(){

}

void Graph::addNode(string name){
    Node N(name);
    Nodes.push_back(N);
}

void Graph::deleteNode(Node& N){
    //hapus semua hubungan ke N
    list<Node>::iterator p1;
    for(auto p1 = begin(this->Nodes);p1!=end(this->Nodes);++p1){
        if ((*p1).isConnected(N)){
            this->deleteEdge((*p1), N);
        }
    }
```

```
    //hapus node
    list<Node>::iterator p;
    p = Nodes.begin();
    while((*p) != N){
        advance(p,1);
    }
    (*p).deleteTrail();
    Nodes.erase(p);
}

void Graph::addEdge(Node& N1, Node& N2){

    N2.addInDegree(1);
    N1.connect(N2);
}

void Graph::deleteEdge(Node& N1, Node& N2){
    N2.minInDegree(1);
    N1.disconnect(N2);
}

Node& Graph::getNode(int id){
    list<Node>::iterator p;
    p = Nodes.begin();
    while((*p).getId() != id){
        advance(p,1);
    }
    return (*p);
}
```

```
Node& Graph::getNode(string name){
    list<Node>::iterator p;
    p = Nodes.begin();
    while((*p).getName() != name){
        advance(p,1);
    }
    return (*p);
}

void Graph::printGraph(){
    list<Node>::iterator p;

    for (p=this->Nodes.begin();p!=this->Nodes.end();++p){
        cout << (*p).getName() << ": ";
        (*p).printTrail();
        cout << endl;
    }
}

void Graph::printInDegree(){
    list<Node>::iterator p;

    for (p=this->Nodes.begin();p!=this->Nodes.end();++p){
        cout << (*p).getName() << ": " << (*p).getInDegree()<<endl;
    }
}

int Graph::countNodes(){
    int count = 0;
    for (Node N : Nodes){
        count++;
```

```
    }
    return count;
}

list<int> Graph::getZeroNodes(){
    list<int> temp;
    list<Node>::iterator p;
    for (p = this->Nodes.begin();p!=this->Nodes.end();++p){
        if((*p).getInDegree() == 0){
            temp.push_back((*p).getId());
        }
    }
    return temp;
}

bool Graph::nodeExist(string name){
    list<Node>::iterator p;
    for(p=this->Nodes.begin();p!=this->Nodes.end();++p){
        if ((*p).getName() == name){
            return true;
        }
    }
    return false;
}
```

5. Main.cpp

```
include "Graph.hpp"
#include <fstream>
#include <vector>
```

```
void readGraph(string file, Graph& G); //Membaca graph dari file
void topologicalSort(Graph& G); //Melakukan topological sort

int main(){
    Graph G;
    string file;
    cout << "Masukkan nama file: ";
    cin >> file;
    readGraph("../test/"+file, G); //Membaca graph
    topologicalSort(G); //melakukan topological sort
    system("pause");

    return 0;
}

void topologicalSort(Graph& G){
    list<int> temp;
    for (int i =1; i<=8 ; i ++){
        //id semua node dengan derajat masuk 0 disimpan dalam list temp
        temp = G.getZeroNodes();

        if(temp.size() > 0){
            //Menampilkan nama node dengan derajat masuk 0
            cout << "Semester " << i << ": ";
            for (int j : temp){
                cout << G.getNode(j).getName() << " ";
                //menghapus node yang sudah ditampilkan
                G.deleteNode(G.getNode(j));
            }
            cout << endl;
        }
```

```
        //Tidak terdapat node dengan derajat masuk 0
        else if(G.countNodes() > 0){
            cout << "gagal, tidak ada lagi mata kuliah yang bisa diambil"<< endl;
            break;
        }
    }
    //Graph belum kosong setelah 8 semester
    if (G.countNodes() > 0){
        cout << "Masih terdapat mata kuliah yang belum diambil, ";
        cout << "Tidak dapat diselesaikan dalam 8 semester" <<endl;
    }

}

void readGraph(string file, Graph& G){
    ifstream scanner(file);
    string text, temp, head;
    vector<string> nodes;
    int i;

    while(getline(scanner, text)){

        i = 0;

        nodes.clear();
        while(text[i] != '.'){
            temp = "";
            while(text[i] != ',' && text[i] != '.'){
                if (text[i] != ' '){
                    temp = temp + text[i];
                }
```

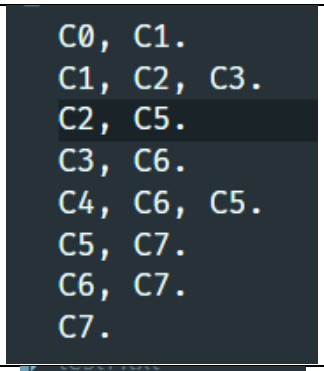
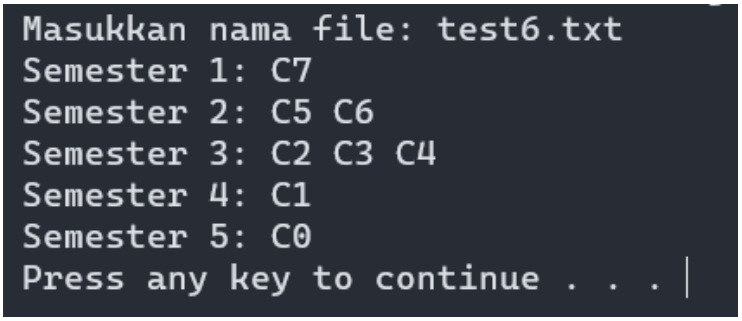
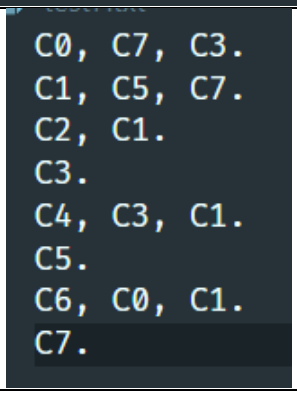
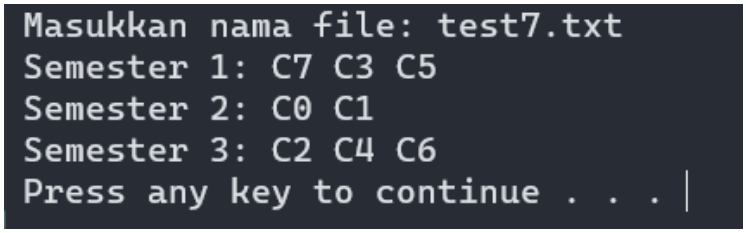
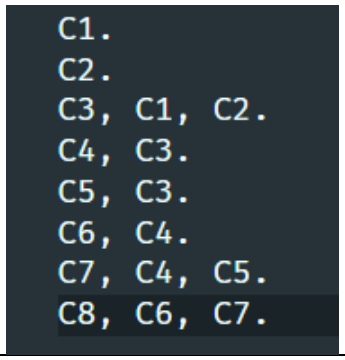
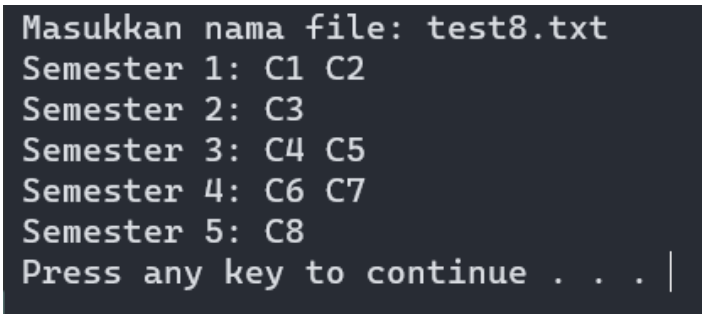
```
                i++;
            }
            nodes.push_back(temp);
            if (text[i] != '.'){
                i++;
            }
        }

        head = nodes[0];
        if (!G.nodeExist(head)){
                G.addNode(head);
            }

        nodes.erase(nodes.begin());
        for (string n : nodes){
            if (!G.nodeExist(n)){
                G.addNode(n);
            }
            G.addEdge(G.getNode(n), G.getNode(head));
        }
    }
}
```

Hasil *input* dan *output*

No.	Input	Output
1	C1, C3. C2, C1, C4. C3. C4, C1, C3. C5, C2, C4.	Masukkan nama file: test1.txt Semester 1: C3 Semester 2: C1 Semester 3: C4 Semester 4: C2 Semester 5: C5 Press any key to continue . . .
2	C5. C11, C5, C7. C2, C11. C7. C8, C7, C3. C9, C8, C11. C3. C10, C3, C11.	Masukkan nama file: test2.txt Semester 1: C5 C7 C3 Semester 2: C11 C8 Semester 3: C2 C9 C10 Press any key to continue . . .
3	C0, C4, C5. C4. C1, C4. C3, C1, C2. C2, C5. C5.	Masukkan nama file: test3.txt Semester 1: C4 C5 Semester 2: C0 C1 C2 Semester 3: C3 Press any key to continue . . .
4	C1. C2, C1. C3, C1. C4, C2, C3. C5, C4, C2. C6, C4, C3.	Masukkan nama file: test4.txt Semester 1: C1 Semester 2: C2 C3 Semester 3: C4 Semester 4: C5 C6 Press any key to continue . . .
5	C1. C2, C1. C3, C1, C2. C4, C3, C5. C5, C1, C2.	Masukkan nama file: test5.txt Semester 1: C1 Semester 2: C2 Semester 3: C3 C5 Semester 4: C4 Press any key to continue . . .

6		
7		
8		

CheckList

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

Alamat Kode Program

https://github.com/Saladin21/STIMA_Tucil2