Laporan Tugas Kecil 1 IF2112 Strategi Algoritma

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

Rafi Raihansyah Munandar
13519154
Program Studi Teknik Informatika

Algoritma *Topological Sort*

Pada permasalahan penyusunan rencana perkuliahan ini dilakukan dengan pendekatan struktur data Graf. Dengan bentuk graf, maka algoritma *topological sort* dapat digunakan antara lain dengan cara berikut

- Terdapat file dalam bentuk txt yang menyimpan daftar mata kuliah
- Pada setiap barisnya, mata kuliah paling pertama merupakan mata kuliah yang memiliki *prerequisite* berupa mata kuliah lainnya yang masih sebaris. Jika pada baris tersebut hanya ada satu mata kuliah, maka mata kuliah tersebut tidak memiliki *prerequisite*.
- Setiap mata kuliah yang menjadi *prerequisite* akan dibentuk dalam struktur graf berarah, yang arahnya menuju mata kuliah utama (mata kuliah yang pertama dalam baris tersebut).
- Graf yang terbentuk tidak boleh siklik, minimal ada satu mata kuliah yang tidak memiliki *prerequisite*
- Akan dipilih graf dengan jumlah *predecessor* sama dengan nol, kemudian ditambahkan ke *list* daftar mata kuliah yang dapat diambil pada semester itu. Setelah itu, *node* yang merepresentasikan mata kuliah tersebut akan dihapus, beserta *edge* yang mengarah ke mata kuliah yang dituju. Jumlah *predecessor* dari mata kuliah yang dituju oleh *node* yang baru dihapus akan dikurangi satu.
- Ulangi langkah sebelumnya hingga sudah tidak ada lagi *node* yang belum terhapus.
- Jika sudah sesuai, maka program akan menampilkan hasil berupa urutan semester dan mata kuliah mana saja yang bisa diambil pada semester itu, kemudian program berhenti.

Pendekatan *Decrease and Conquer* terdapat pada cara algoritma *Topological Sort* membagi sub-persoalannya. Pada kasus ini, algoritma *Topological Sort* hanya membagi persoalan menjadi *node* yang memiliki nol *predecessor* dan yang memiliki lebih dari 0. Kemudian, algoritma ini hanya akan mengerjakan sub-persoalan untuk yang memiliki nol *predecessor* tersebut dahulu. Hal ini dilakukan secara terus menerus hingga seluruh node telah dicek.

Source Program

DirectedGraph.hpp

```
#ifndef DirGRAPH_H
#define DirGRAPH_H

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

typedef struct taddrNode *addrNode;
typedef struct taddrSucc *addrSucc;
typedef struct taddrNode {
    string course;
    int jmlMasuk;
    addrSucc trail;
    addrNode next;
} Node;

typedef struct taddrSucc {
    addrNode succ;
    addrSucc nextTrail;
} SuccNode;

typedef struct {
    addrNode first;
} Graph;

#define Course(N) (N)->course
#define JmlMasuk(N) (N)->jmlMasuk
#define Trail(N) (N)->trail
#define Next(N) (N)->next
#define Succ(T) (T)->succ
#define NextTrail(T) (T)->nextTrail
#define First(G) (G).first

void CreateGraph(Graph *G, string c);
/* I.S. Graf sembarang
F.S. First dari Graf menuju pada node dengan nilai course adalah c */

addrNode AlokNode(string c);
/* Mengembalikan address node yang telah dialokasi
```

```
dengan nama node berupa nama dari course tersebut.
Jika alokasi tidak berhasil maka mengembalikan NULL */

void DealokNode(addrNode P);
/* I.S. P terdefinisi
F.S. P telah didealokasi */

addrSucc AlokSucc(addrNode P);
/* Mengembalikan address successor node yang telah
dialokasi menuju P sebagai successor node-nya.
Jika alokasi tidak berhasil maka mengembalikan NULL */

void DealokSucc(addrSucc Ps);
/* I.S. Ps terdefinisi
F.S. Ps telah didealokasi */

addrNode SearchNode(Graph G, string c);
/* Mengembalikan address dari node yang merepresentasikan
course c. JIka tidak ada maka akan mengembalikan NULL */

addrSucc SearchEdge(Graph G, string prec, string succ);
/* Mengembalikan address dari successor node yang
menghubungkan node dengan course prec dan succ.
JIka tidak ada maka akan mengembalikan NULL */

void InsertNode(Graph *G, string c, addrNode *P);
/* Menambahkan Node dengan course c pada list utama graf */

void DeleteNode(Graph *G, string c);
/* Menghapus node dengan course c pada graf */

void InsertEdge(Graph *G, string prec, string succ);
/* Menambahkan edge yang menghubungkan node dengan course prec
dan node dengan course succ. Successor node ditambahkan pada trail
prec menuju node succ */

void DeleteAllEdges(Graph *G, addrNode P);
/* Menghapus seluruh edges yang dimiliki oleh node P */
```

```
void PrintGraph(Graph G);
/* Menampilkan informasi graf berupa setiap node menunjuk ke node
mana serta jumlah predecessor yang dimiliki oleh masing-masing node */

#endif
```

DirectedGraph.cpp

```
#include "DirectedGraph.hpp"

void CreateGraph(Graph *G, string c) {
    addrNode P;
    P = AlokNode(c);
    First(*G) = P;
}

addrNode AlokNode(string c) {
    addrNode P;
    P = (Node*) malloc(sizeof(Node));
    if (P != NULL) {
        Course(P) = c;
        JmlMasuk(P) = 0;
        Trail(P) = NULL;
        Next(P) = NULL;
    }
    return P;
}

void DealokNode(addrNode P) {
    free(P);
}

addrSucc AlokSucc(addrNode P) {
    addrSucc Ps;
    Ps = (SuccNode*) malloc (sizeof(SuccNode));
    if (Ps != NULL) {
        Succ(Ps) = P;
```

```
        NextTrail(Ps) = NULL;
    }
    return Ps;
}

void DealokSucc(addrSucc Ps) {
    free(Ps);
}

addrNode SearchNode(Graph G, string c) {
    addrNode P;
    P = First(G);
    while (P != NULL && Course(P) != c) {
        P = Next(P);
    }
    return P;
}

addrSucc SearchEdge(Graph G, string prec, string succ) {
    addrNode P;
    P = SearchNode(G, prec);

    if (P != NULL) {
        addrSucc Ps;
        Ps = Trail(P);
        while (Ps != NULL && Course(Succ(Ps)) != succ) {
            Ps = NextTrail(Ps);
        }
        return Ps;
    }

    return NULL;
}

void InsertNode(Graph *G, string c, addrNode *P) {
    (*P) = AlokNode(c);
    if ((*P) != NULL) {
        addrNode Pn;
        Pn = First(*G);
```

```
        while (Next(Pn) != NULL) {
            Pn = Next(Pn);
        }
        Next(Pn) = (*P);
    }
}

void DeleteNode(Graph *G, string c) {
    addrNode Pdel, P;
    Pdel = SearchNode((*G), c);
    P = First((*G));

    if (P != NULL) {
        if (Pdel == First((*G))) {
            First(*G) = Next(P);
        } else {
            while (Next(P) != Pdel) {
                P = Next(P);
            }
            Next(P) = Next(Pdel);
        }
        DeleteAllEdges(G, Pdel);
    }
}

void InsertEdge(Graph *G, string prec, string succ) {
    addrSucc Q;
    Q = SearchEdge((*G), prec, succ);
    if (Q == NULL) {
        addrNode Pr, Ps;
        Pr = SearchNode((*G), prec);
        Ps = SearchNode((*G), succ);

        // if (Pr == NULL) {
        //     InsertNode(G, prec, &Pr);
        // }
        // if (Ps == NULL) {
        //     InsertNode(G, succ, &Ps);
        // }
```

```
        addrSucc Tr;
        Tr = Trail(Pr);
        if (Tr != NULL) {
            while (NextTrail(Tr) != NULL) {
                Tr = NextTrail(Tr);
            }
            NextTrail(Tr) = AlokSucc(Ps);
        } else {
            Trail(Pr) = AlokSucc(Ps);
        }

        JmlMasuk(Ps)++;
    }
}

void DeleteAllEdges(Graph *G, addrNode P) {
    addrSucc Ps;
    Ps = Trail(P);
    while (Ps != NULL) {
        JmlMasuk(Succ(Ps)) -= 1;
        Ps = NextTrail(Ps);
    }
    Trail(P) = NULL;
}

void PrintGraph(Graph G) { // FOR DEBUGGING PURPOSES
    // Print each node and their successor node(s)
    addrNode P = First(G);
    while (P != NULL) {
        cout << "Course " << Course(P) << " is directed to ";

        addrSucc Ps;
        Ps = Trail(P);
        if (Ps == NULL) {
            cout << "none." << endl;
        }

        while (Ps != NULL) {
```

```
            cout << Course(Succ(Ps));
            if (NextTrail(Ps) != NULL) {
                cout << ", ";
            } else {
                cout << "." << endl;
            }
            Ps = NextTrail(Ps);
        }

        P = Next(P);
    }

    // Print each node and their number of predecessor(s)
    P = First(G);
    while (P != NULL) {
        cout << Course(P) << ": " << JmlMasuk(P) << endl;
        P = Next(P);
    }
}
```

main.cpp

```
#include <fstream>
#include "DirectedGraph.hpp"

void FileToGraph (Graph *G, string filename, bool *valid) {
    ifstream file(filename);
    if (file.fail()) {
        (*valid) = false;
    } else {
        string lines;
        bool firstNode;
        bool isHead = true;

        // Process each line from textfile
        while (getline(file, lines, '\n')) {
            string delimiter = " ,.";
```

```
            size_t pos = 0;
            string course;

            addrNode Pmain;
            firstNode = true;
            string mainCourse;

            // Parse word with delimiter as the separator
            while ((pos = lines.find_first_of(delimiter)) !=
string::npos) {
                course = lines.substr(0, pos);

                /***** GRAPH MAKING PROCESS *****/

                if (firstNode) {                            // Check for
the first node on every line
                    if (isHead)                             // First
ever node will be head of graph
                    {
                        CreateGraph(G, course);             // Create
graph with the first course as the head
                        isHead = false;
                    }
                    else
                    {
                        if (SearchNode(*G, course) == NULL)
                        {
                            InsertNode(G, course, &Pmain);  // Insert
node if it has not yet existed
                        }
                    }
                    firstNode = false;                      // Stop
checking for first node
                    mainCourse = course;                    // Setting
the first code from the line as the mainCourse
                }

                // Process the prerequisite for the current mainCourse
```

```
                else {
                    addrNode dummy;
                    if (SearchNode(*G, course) == NULL)
                    {
                        InsertNode(G, course, &dummy);      // Insert
node if it has not yet existed
                    }
                    InsertEdge(G, course, mainCourse);      // Connect
the edge from the current course to the current mainCourse
                }


                /***** END OF PROCESSING LINE *****/

                lines.erase(0, pos + 2);
            }
        }
    }
}

int main() {
    Graph G;
    string input;
    cout << "Please input a file name (example = test.txt) :" << endl;
    cin >> input;
    string testfile = "../test/";
    testfile += input;

    bool valid = true;
    FileToGraph(&G, testfile, &valid);

    if (valid) {
        int smt = 1;
        while (First(G) != NULL) {
            cout << "Semester " << smt << ":" << endl;

            // Set P to always be the head of graph
            addrNode P = First(G);
```

```
            // Create array of addrNode that will be deleted
            addrNode* delArr = new addrNode[50];
            int idx = 0;

            // Getting and showing the courses that has 0 predecessor
            while (P != NULL) {
                if (JmlMasuk(P) == 0) {
                    cout << "\t- " << Course(P) << endl;
                    delArr[idx] = P;
                    idx++;
                }
                P = Next(P);
            }

            // Deleting all the nodes that has 0 predecessor
            for (int i = 0; i < idx; i++)
            {
                DeleteNode(&G, Course(delArr[i]));
            }

            smt++; // Increment semester
        }
    } else {
        cout << "File not found!" << endl;
    }

    return 0;
}
```

Hasil Eksekusi Program

1. Test 1

```
Silakan masukkan nama file!
Contoh : test123.txt
Nama input file : test1.txt
Semester 1:
        - C1
        - C3
Semester 2:
        - C4
Semester 3:
        - C2
Semester 4:
        - C5
```

2. Test 2

```
Silakan masukkan nama file!
Contoh : test123.txt
Nama input file : test2.txt
Semester 1:
        - C3
Semester 2:
        - C1
Semester 3:
        - C4
Semester 4:
        - C2
Semester 5:
        - C5
        - C6
```

3. Test 3

```
Silakan masukkan nama file!
Contoh : test123.txt
Nama input file : test3.txt
Semester 1:
        - C5
Semester 2:
        - C4
Semester 3:
        - C3
Semester 4:
        - C2
Semester 5:
        - C1
```

4. Test 4

```
Silakan masukkan nama file!
Contoh : test123.txt
Nama input file : test4.txt
Semester 1:
        - C1
Semester 2:
        - C2
        - C3
Semester 3:
        - C4
```

5. Test 5

```
Silakan masukkan nama file!
Contoh : test123.txt
Nama input file : test5.txt
Semester 1:
        - IF03
        - IF05
        - IF04
Semester 2:
        - IF01
        - IF02
        - IF06
        - IF07
        - IF08
Semester 3:
        - IF09
        - IF11
Semester 4:
        - IF10
Semester 5:
        - IF12
        - IF13
```

6. Test 6

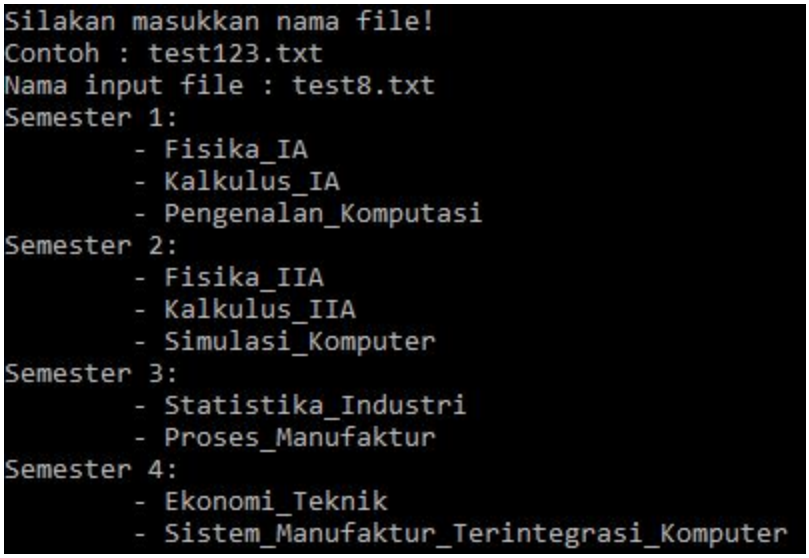
```
Silakan masukkan nama file!
Contoh : test123.txt
Nama input file : test6.txt
Semester 1:
        - MA5173
        - MA7242
        - MA9311
        - MA3449
        - MA7587
        - MA9656
        - MA1725
        - MA3794
        - MA5863
        - MA3104
        - MA1380
        - MA7932
        - MA5518
Semester 2:
        - MA0001
        - MA2070
        - MA4139
        - MA6208
        - MA8277
        - MA0346
        - MA2415
        - MA4484
        - MA8622
        - MA0691
        - MA6898
        - MA8967
        - MA3105
        - MA5174
        - MA9312
        - MA1381
        - MA3450
        - MA4829
        - MA1036
        - MA7243
        - MA2760
        - MA6553
Semester 3:
        - MA5519
        - MA7588
        - MA3795
        - MA1726
        - MA9657
```

```
Semester 4:
        - MA0002
        - MA0347
        - MA0692
        - MA2071
        - MA2416
        - MA4140
        - MA4485
        - MA5864
        - MA6209
        - MA6554
        - MA7933
        - MA8278
        - MA8623
```

7. Test 7

```
Silakan masukkan nama file!
Contoh : test123.txt
Nama input file : test7.txt
Semester 1:
        - IF002-3
        - IF008-4
        - IF010-4
        - IF011-3
Semester 2:
        - IF006-2
Semester 3:
        - IF001-4
Semester 4:
        - IF007-2
        - IF005-1
Semester 5:
        - IF003-4
Semester 6:
        - IF004-3
        - IF009-3
```

8. Test 8



Alamat Kode Program

Repository GitHub: https://github.com/rafirm29/tucil2-stima

Tabel 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	✓	