TUGAS KECIL

IF2211 – Strategi Algoritma

K – 03

Oleh

Hokki Suwanda 13519143

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2021

BAB I

ALGORITMA TOPOLOGICAL SORT

1.1 Decrease and Conquer

Decrease and Conquer adalah algoritma yang mereduksi sebuah persoalan yang besar menjadi dua buah subpersoalan dan memproses salah satu subpersoalan. Beberapa literatur lama mengategorikan algoritma *decrease and conquer* sebagai bagian dari algoritma *divide and conquer*. Terdapat beberapa variasi algoritma *decrease and conquer* berdasarkan ukuran pembagian subpersoalan, yaitu *decrease by a constant, decrease by a constant factor, decrease by a variable size*. Algoritma *decrease and conquer* yang digunakan dalam *topological sort* kali ini adalah *decrease by a variable size*.

1.2 Topological Sort

Pendekatan algoritma *topological sort* yang digunakan adalah pendekatan yang tertera pada spesifikasi Tugas Kecil II IF2211 Strategi Algoritma Semester 2 Tahun 2020/2021. Perlu diperhatikan bahwa semua contoh tertulis yang ada di dokumen ini diambil dari spesifikasi tugas.

Terdapat beberapa tahap dalam pengimplementasian algoritma *topological sort*, tahap pertama adalah tahap *file converting*, yaitu mengubah sebuah file **.txt* menjadi sebuah *adjacency list*. Pada tahap pertama ini, setiap mata kuliah direpresentasikan oleh *tuple* tiga elemen (untuk mempermudah implementasinya di *python*, representasi yang digunakan adalah sebuah *list* tiga elemen). Elemen pertamanya adalah kode mata kuliah, elemen keduanya adalah jumlah *prerequisite*, dan elemen ketiga adalah himpunan *prerequisite* (*adjacency list*). Contohnya adalah (*C2*, 2, {*C1*, *C4*}).

Tahap pertama dilakukan dalam modul *FileConverting* dalam *class Convert_To_Graph* yang memiliki 2 atribut utama, dan 1 atribut transisi. Atribut transisi (*courseByID*) hanya digunakan pada tahap pertama ini, sementara atribut utama (*theGraph,* graf dengan representasi *tuple* dan *numberOfLines*) akan digunakan oleh *main program*. Berikut adalah langkah-langkahnya :

1. Baca *file *.txt* dengan library *csv* dengan *delimiter = ‘,’*. Proses ini akan menghasilkan *list* 2 dimensi. Contoh hasilnya adalah :
```
['C1', ' C3.']
['C2', ' C1', ' C4.']
['C3.']
['C4', ' C1', ' C3.']
['C5', ' C2', ' C4.']
```
2. Tulis hasil pada 1. menjadi sebuah *file outputFile.txt* dengan menghapus titik pada akhir tiap baris. Berikut adalah contoh isi *outputFile.txt*.
```
C1 C3
C2 C1 C4
C3
C4 C1 C3
C5 C2 C4
```

3. Baca *outputFile.txt* dengan library *csv* dengan *delimiter* = ' ' (spasi). Proses ini akan menghasilkan *numberOfLines dan list* 2 dimensi yang bersih dari tanda baca. Contoh hasilnya adalah :
```
['C1', 'C3']
['C2', 'C1', 'C4']
['C3']
['C4', 'C1', 'C3']
['C5', 'C2', 'C4']
```
4. *List* dari hasil pada 3. akan disimpan ke atribut utama *theGraph* terlebih dahulu. Di atribut ini, indeks 0 menunjukkan kode mata kuliah, dan indeks 1 dan seterusnya adalah *prerequisite* mata kuliah tersebut.
5. Pisah *theGraph* menjadi dua buah komponen, yaitu kode mata kuliah (disimpan dalam atribut transisi *courseByID*) dan himpunan *prerequisite*nya (disimpan dalam atribut utama *theGraph*).
6. Untuk setiap $0 \leq i <$ *numberOfLines*, ambil (*courseByID*[*i*], *length* of *theGraph*[*i*], *theGraph*[*i*]) dan masukkan menjadi *theGraph*[*i*] yang baru. Untuk menyederhanakan implementasi di *python*, *tuple* tersebut dijadikan sebuah *list*. *Tuple* ini kemudian akan diteruskan ke *main program* untuk dilanjutkan ke tahap kedua, yaitu *main program topological sort*. *Tuple*nya adalah :
```
['C1', 1, ['C3']]
['C2', 2, ['C1', 'C4']]
['C3', 0, []]
['C4', 2, ['C1', 'C3']]
['C5', 2, ['C2', 'C4']]
```

Pada *main program*, digunakan beberapa variabel seperti yang terdapat pada tabel di bawah. Pendekatan *topological sort* untuk *n* buah mata kuliah yang digunakan adalah selama *numberOfTakenCourse* lebih kecil dari *numberOfLines,* lakukan :

1. Ambil semua mata kuliah tidak memiliki *prerequisite*, sebut saja sebagai C_0, C_1, C_2, ..., C_k dengan k adalah jumlah mata kuliah yang tidak memiliki *prerequisite*. Tiap mata kuliah ini akan dimasukkan ke dalam *takenCourse*[*semester*]. Kompleksitas waktunya $O(n)$.
2. Untuk setiap $c \in \{C_i, 0 \leq i \leq k\}$, hapus c dari himpunan *prerequisite* dan kurangi jumlah *prerequisite* dengan 1. Kompleksitas waktunya $O(n)$.
3. Tambahkan nilai *semester* dengan 1. Kompleksitas waktunya $O(1)$.
4. Lakukan *topological sort* pada $n - k$ mata kuliah lainnya. Kompleksitas waktunya $O(n - k) = O(n)$.

Pendekatan *topological sort* di atas sangatlah sesuai dengan salah satu varian *decrease and conquer*, *decrease by a variable size*. *Variable size* yang dimaksud adalah nilai dari variabel *k* yang akan berubah bergantung dengan banyaknya mata kuliah yang diambil. Karena proses 1 sampai 4 dilakukan selama *numberOfTakenCourse* lebih kecil darip *numberOfLines*, algoritma ini memiliki kompleksitas waktu total $O(n^2)$.

Variabel	Deskripsi
numberOfTakenCourse	Jumlah mata kuliah yang sudah diambil
takenCourse	*List of* mata kuliah yang sudah ambil
myGraph	Atribut utama *theGraph* dari tahap pertama di modul FileConverting
semester	Semester sekarang ini

Bila dituliskan dalam *pseudocode*, adalah :

```
procedure TopSort(input semester : integer, input/output takenCourse : list of list of string,
input/output theGraph : tuple, input numberOfLines : integer, input/output numberOfTakenCourse :
integer)
KAMUS LOKAL
i, numOfPrereq : integer
courseID : string
prereq : list of string

ALGORITMA
if(numberOfTakenCourse < numberOfLines) then
        i ← 0
        while (i < length(theGraph)) do   {length(x) adalah jumlah elemen pada list x}
                (courseID, numOfPrereq, prereq) ← theGraph[i]
                if(numOfPrereq = 0) then
                        {delete elemen ke i dari theGraph}
                        theGraph ← theGraph – theGraph[i]
                        takenCourse[semester] ← takenCourse[semester] ∪ courseID
                        numberOfTakenCourse ← numberOfTakenCourse + 1
                else
                        i ← i + 1
        { sudah tidak ada yang numOfPrereq nya 0 }

        { length(takenCourse[semester]) ekivalen dengan k }
        for i ← 0 to length(takenCourse[semester]) - 1 do
                for j ← 0 to length(theGraph) -1 do
                        (courseID, numOfPrereq, prereq) ← theGraph[j]
                        if(takenCourse[semester][i] ∈prereq)
                                prereq ← prereq – takenCourse[semester][i]
                                numOfPrereq ← numOfPrereq – 1
                                theGraph[j] ← (courseID, numOfPrereq, prereq)

        TopSort(semester + 1, takenCourse, theGraph, numberOfLines, numberOfTakenCourse)
```

Pada awalnya, penulis menambahkan sebuah modul *ListSorting* untuk *merge sort* tetapi setelah selesai dibuat, penulis merasa modul tersebut tidak bermanfaat untuk solusi penulis, maka modul tersebut tidak digunakan.

BAB II

SOURCE CODE

1. Indikator Penilaian

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	

2. Source Code Modul FileConverting

```
import csv

testDirectory = '../test/'
outputFileName = 'outputFile.txt'

class Convert_To_Graph:
    theGraph = [];
    courseByID = [];
    numberOfLines = 0;
    def __init__(self, inputFileName):
        # perlihatkan isi file
        self.printFileContent(inputFileName)
        # tahap pertama langkah 1 dan 2.
        self.formattingFile(testDirectory + inputFileName, outputFileName)
        # tahap pertama langkah 3 dan 4.
        self.theGraph = self.takeFormattedFile(outputFileName)
        # tahap pertana langkah 5.
        self.theGraph = self.intoAdjacencyList()
        # tahap pertama langkah 6.
        self.theGraph = self.intoCourseAndPrereq()

    def printFileContent(self, inputFileName):
        print('Isi ' + inputFileName)
        inputFile = open(testDirectory + str(inputFileName))
        for i in inputFile:
            print(i, end = '')
        inputFile.close()

    def formattingFile(self, inputFileName, outputFileName):
        inputFile = open(str(inputFileName))
        fileContent = csv.reader(inputFile, delimiter = ",")
        list = []

        for row in fileContent:
            content = f'{"".join(row)}'
            contentLength = len(content)
            list.append(content[:contentLength - 1])

        outputFile = open(testDirectory + outputFileName, 'w')
        for row in list:
            outputFile.write(row + '\n')

        outputFile.close()
        inputFile.close()
        return
```

```
    def takeFormattedFile(self, outputFileName):
        outputFile = open(testDirectory + outputFileName, 'r')

        isifile = csv.reader(outputFile, delimiter = ' ')
        complete = []
        for row in isifile:
            complete.append([])
            for kolom in row:
                complete[self.numberOfLines].append(kolom)
            self.numberOfLines += 1

        return complete

    def intoAdjacencyList(self):
        temporaryGraph = []
        for i in range(0, self.numberOfLines, 1):
            # menyimpan kode matkul
            self.courseByID.append(self.theGraph[i][0])
            temporaryGraph.append([])
            # himpunan prereq untuk sementara disimpan ke theGraph dulu
            for j in range(1, len(self.theGraph[i]), 1):
                temporaryGraph[i].append(self.theGraph[i][j])

        return temporaryGraph

    def intoCourseAndPrereq(self):
        temporaryGraph = []
        # tuple (kode matkul, jumlah prereq, himpunan prereq)
        for i in range(0, self.numberOfLines, 1):
            temporaryGraph.append([self.courseByID[i],len(self.theGraph[i]),
self.theGraph[i]])
        return temporaryGraph

    def printGraph(self):
        for node in self.theGraph:
            print(node)
```

3. Source Code Modul ListSorting (tidak digunakan)

```
class Sort_The_List :
    def Sort(self, list, comparedIndex, length):
        return self.MergeSort(list, 0, length - 1, comparedIndex)

    def MergeSort(self, list, firstIndex, lastIndex, comparedIndex):
        if(firstIndex < lastIndex):
            middleIndex = (firstIndex + lastIndex) // 2
            list = self.MergeSort(list, firstIndex, middleIndex, comparedIndex)
            list = self.MergeSort(list, middleIndex + 1, lastIndex, comparedIndex)
            list = self.Merge(list, firstIndex, middleIndex, lastIndex, comparedIndex)

        return list

    def Merge(self, list, firstIndex, middleIndex, lastIndex, comparedIndex):
        result = []
        for isi in list:
            result.append(isi)

        p = firstIndex
        q = middleIndex + 1
        r = firstIndex
        while(p <= middleIndex) and (q <= lastIndex):
            if(list[p][comparedIndex] <= list[q][comparedIndex]):
                result[r] = list[p]
                p += 1
            else:
                result[r] = list[q]
                q += 1

            r += 1
```

```
        while(p <= middleIndex):
            result[r] = list[p]
            p += 1
            r += 1

        while(q <= lastIndex):
            result[r] = list[q]
            q += 1
            r += 1

        return result
```

4. Source Code solusi

```
from importlib import import_module as im

ConverterModule = im('13519143-FileConverting')

inputFileName = input('Masukkan nama file : ')
Converter = ConverterModule.Convert_To_Graph(inputFileName);

takenCourse = [[]]
numberOfTakenCourse = 0
semester = 0
myGraph = Converter.theGraph

while(numberOfTakenCourse < Converter.numberOfLines):
    semester += 1
    takenCourse.append([])

    ''' ambil semua matkul yang prereqnya 0 '''
    i = 0
    while(i < len(myGraph)):
        if(myGraph[i][1] != 0):
            i += 1
        else:
            takenCourse[semester].append(myGraph[i][0])  # kode mata kuliahnya
            myGraph.pop(i)
            numberOfTakenCourse += 1
    ''' ambil semua matkul yang prereqnya 0 '''

    ''' hapus matkul yang sudah diambil dari prereq matkul lainnya '''
    for matkulDiambil in takenCourse[semester]:
        for j in myGraph:
            if(j[2].count(matkulDiambil) > 0):
                j[2].remove(matkulDiambil)
                j[1] -= 1
    ''' hapus matkul yang sudah diambil dari prereq matkul lainnya '''

for i in range(1, semester + 1, 1):
    print(f'Semester {i}:', end='')
    for matkul in takenCourse[i]:
        print(f' {matkul}', end='')
    print()
```

Selain itu, *source code* dapat diakses pada **Main.py**, **ListSorting.py**, **FileConverting.py** dan link github.

Solusi tersebut terdapat pada *folder* ini yang memuat semua *file* masukan, laporan ini, skrinshut, dan *source code* solusi.

BAB III

EKSPERIMEN

```
Masukkan nama file : test1.txt
Isi test1.txt
C1, C3.
C2, C1, C4.
C3.
C4, C1, C3.
C5, C2, C4.

Semester 1: C3
Semester 2: C1
Semester 3: C4
Semester 4: C2
Semester 5: C5
```

```
Masukkan nama file : test2.txt
Isi test2.txt
A, C, E.
B, C.
C.
D, A, E, B.
E.
F, E.
G, D.
H, D.
I, G, H.
J, H, K, F.
K, F.
L, I, J.
M, J.

Semester 1: C E
Semester 2: A B F
Semester 3: D K
Semester 4: G H
Semester 5: I J
Semester 6: L M
```

```
Masukkan nama file : test3.txt
Isi test3.txt
A.
B, A.
C, F, A.
D, I, B.
E, G, C.
F, D.
G.
H, D, E.
I, G.
J, I, B.

Semester 1: A G
Semester 2: B I
Semester 3: D J
Semester 4: F
Semester 5: C
Semester 6: E
Semester 7: H
```

```
Masukkan nama file : test5.txt
Isi test5.txt
0, 5, 4.
1, 4.
2, 5.
3, 1, 2.
4.
5.

Semester 1: 4 5
Semester 2: 0 1 2
Semester 3: 3
```

```
Masukkan nama file : test6.txt
Isi test6.txt
5.
7.
3.
11, 5, 7.
8, 7, 3.
2, 11.
9, 11, 8.
10, 11, 3.

Semester 1: 5 7 3
Semester 2: 11 8
Semester 3: 2 9 10
```

```
Masukkan nama file : test8.txt
Isi test8.txt
CS106A.
CS106BX, CS106A.
CS103, CS106BX.
CS107, CS106BX.
CS109, CS103.
CS221, CS107, CS109.
CS110, CS107.
CS161, CS109.

Semester 1: CS106A
Semester 2: CS106BX
Semester 3: CS103 CS107
Semester 4: CS109 CS110
Semester 5: CS221 CS161
```

```
Masukkan nama file : test4.txt
Isi test4.txt
0, 1, 2, 3.
1.
2.
3.

Semester 1: 1 2 3
Semester 2: 0
```

```
Masukkan nama file : test10.txt
Isi test10.txt
0.
1, 0.
2, 1, 0, 3, 4.
3, 1.
4, 3, 7, 12, 13.
5, 4, 6, 0.
6, 4.
7, 3.
8, 1.
9, 7, 11.
10, 7.
11, 8, 10.
12, 7.
13, 12.
14, 10, 12, 15.
15, 11, 13, 10.
Semester 1: 0
Semester 2: 1
Semester 3: 3 8
Semester 4: 7
Semester 5: 10 12
Semester 6: 11 13
Semester 7: 4 9 15
Semester 8: 2 6 14
Semester 9: 5
```