

TUGAS KECIL 2

PENYUSUNAN RENCANA KULIAH DENGAN TOPOLOGICAL SORT (PENERAPAN *DECREASE AND CONQUER*)

IF2211 STRATEGI ALGORITMA



Oleh

RIZKY ANGGITA S SIREGAR

13519132

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2021

ALGORITMA DECREASE AND CONQUER

Setelah menginput nama file .txt yang berisi Directed Acyclic Graph, selanjutnya proses topological sort terbagi menjadi beberapa tahap:

1. Persiapan

Teks yang terdapat pada file.txt yang kita input akan diolah oleh kelas Graph, dimana graf direpresentasikan menjadi adjacency list. Kelas Graph dirancang memiliki dua atribut, yaitu list of node (simpul) dan adjacency list yang diimplementasikan menggunakan *dictionary* (mencatat node apa saja yang bertetangga).

Method yang digunakan adalah method `add_node` dan method `add_edge_in`. Method `add_node` akan membaca kata pertama pada tiap baris pada file txt yang diinput. Kemudian setelah semua node pada graf tercatat, selanjutnya method `add_edge_in` akan mencatat node apa saja yang bertetangga dengan suatu node, dalam hal ini jika node B memiliki sisi(edge) yang menuju node A, maka B dicatat dalam adjacency list node A. Sebaliknya jika node A memiliki sisi (edge) yang menuju node C, maka node C tidak termasuk adjacency list node A (karena arahnya dari node A ke node C) melainkan node A akan dimasukkan ke adjacency list node C.

Setelah objek graf tersebut terbentuk, maka langkah selanjutnya adalah pemanggilan method topological sort yang akan dibahas pada poin setelah ini.

2. Proses Decrease and Conquer

Selanjutnya ialah melakukan topological sort yang menggunakan algoritma Decrease and Conquer. Method topological sort menggunakan method lain yaitu `count_degree_in` dan `del_node_in`.

Proses decrease and conquer pada topological sort ialah mencari node dengan In-Degree sebanyak 0. Jika suatu node memiliki In-Degree sebesar 0, maka node ini akan dihapus dari graf (**decrease**) dan dicatat sebagai node yang sudah tersortir dan dikunjungi kemudian dilanjutkan untuk graf yang sudah berkurang jumlah node-nya (**conquer**).

Perhitungan In-Degree setiap node menggunakan method `count_degree_in` yang melakukan proses perhitungan jumlah node lain yang memiliki edge menuju node tersebut. Node yang memiliki In-Degree 0 akan dicatat dalam sebuah list. Setelah semua node dikenai method `count_degree_in`, maka setiap node yang terdapat dalam *list of node* dengan In-Degree 0 akan dihapus dari graf menggunakan method `del_node_in`. Proses penghapusan node ini akan dimulai dengan menghapus node tersebut dari adjacency list node lain. Kemudian node ini akan dihapus dari list of node.

Selanjutnya proses topological sort akan dilanjutkan dengan graf yang jumlah node nya sudah berkurang akibat proses decrease sebelumnya dan dilakukan proses yang

sama. Proses topological sort akan berhenti ketika list of nodes yang dimiliki sudah kosong yang menandakan semua node sudah dikunjungi dan tersortir.

Source Code Program

```
# Nama      : Rizky Anggita S Siregar
# NIM       : 13519132
# Tanggal   : 28 Februari 2021
# Deskripsi : Topological Sort (Decrease and Conquer Algorithm)
# TUCIL 2 STRATEGI ALGORITMA

import os

class Graph():
    #Directed Acyclic Graph with List Representation
    def __init__(self):
        self.nodes = []
        self.adj_list = {}

    def add_node(self, node):
        #Adding new node
        self.nodes.append(node)
        self.adj_list[node] = []

    def print_adj_in_list(self):
        print("\nIn Node")
        for node in self.nodes:
            print(node, "<", self.adj_list[node])

    def print_adj_out_list(self):
        print("\nOut Node")
        for n in self.nodes:
            temp = []
            for node in self.adj_list:
                if n in self.adj_list[node] and node != n:
                    temp.append(node)
            print(n, ">", temp)

    def add_edge_in(self, u, v):
        self.adj_list[u].append(v)

    def count_degree_in(self, u):
        #Count in-degree for node u
        return len(self.adj_list[u])

    def count_degree_out(self, u):
        #Count out-degree for node u
        count = 0
        for node in self.adj_list:
            if u in self.adj_list[node]:
                count += 1
        return count

    def del_node_in(self, u):
        # Delete node u from graph
        # Process: delete u from every adjacent of u
        # and delete node u from list of nodes
        for node in self.adj_list:
            if u in self.adj_list[node] and node != u:
```

```

        self.adj_list[node].remove(u)

    self.nodes.remove(u)
    self.adj_list.pop(u, None)

def topological_sort(self):
    #DAG Topological Sort
    list_sorted = []
    list_of_degin_0 = []
    i = 0
    n_node = len(self.nodes)

    while(len(list_sorted) != n_node):
        list_of_degin_0 = []
        for node in self.nodes:
            degin = self.count_degree_in(node)
            if(degin == 0):
                list_of_degin_0.append(node)
                list_sorted.append(node)

        #Didapatkan list node dengan degree-in = 0
        print("\nSemester", i+1, ":", end=" ")
        for node in list_of_degin_0:
            print(node, end=" ")
            self.del_node_in(node)
        i += 1
        # print()

    print("\n\nTopological Sort: ", list_sorted)

def add_graph_from_txt(graf, file):
    for line in f:
        a = line.rsplit(", ")
        for i in range (len(a)):
            if (i==0):
                graf.add_node(a[0].rstrip(".\n").split('.')[0])
            else:
                temp = a[i].rstrip(".\n")
                graf.add_edge_in(a[0].rstrip(".\n"), temp)

#----- MAIN PROGRAM -----#

# Input nama file
filename = input("Masukkan nama file: ")
a = os.path.abspath(os.curdir)

if os.name=='nt':
    file_path = os.path.join("../\test", filename)
else:
    file_path = os.path.join(a, "test", filename)

# Membuat graph dari txt
graf = Graph()
f = open(file_path, "r")
add_graph_from_txt(graf, f)

```

```
print("Graph yang diinput: ")
graf.print_adj_in_list()
graf.print_adj_out_list()

# Topological Sort DAG
graf.topological_sort()
```

Screenshot

```
Masukkan nama file: 1.txt
Graph yang diinput:

In Node
C1 <- ['C3']
C2 <- ['C1', 'C4']
C3 <- []
C4 <- ['C1', 'C3']
C5 <- ['C2', 'C4']

Out Node
C1 -> ['C2', 'C4']
C2 -> ['C5']
C3 -> ['C1', 'C4']
C4 -> ['C2', 'C5']
C5 -> []

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

Topological Sort: ['C3', 'C1', 'C4', 'C2', 'C5']
```

```
Masukkan nama file: 2.txt
Graph yang diinput:

In Node
C1 <- ['C2']
C2 <- ['C4']
C3 <- []
C4 <- ['C3']
C5 <- ['C3']
C6 <- []

Out Node
C1 -> []
C2 -> ['C1']
C3 -> ['C4', 'C5']
C4 -> ['C2']
C5 -> []
C6 -> []

Semester 1 : C3 C6
Semester 2 : C4 C5
Semester 3 : C2
Semester 4 : C1

Topological Sort: ['C3', 'C6', 'C4', 'C5', 'C2', 'C1']
```

Masukkan nama file: 3.txt
Graph yang diinput:

In Node

```
C0 <- ['C5', 'C4']  
C1 <- ['C3', 'C4']  
C2 <- ['C5']  
C3 <- ['C2']  
C4 <- []  
C5 <- []
```

Out Node

```
C0 -> []  
C1 -> []  
C2 -> ['C3']  
C3 -> ['C1']  
C4 -> ['C0', 'C1']  
C5 -> ['C0', 'C2']
```

```
Semester 1 : C4 C5  
Semester 2 : C0 C2  
Semester 3 : C3  
Semester 4 : C1
```

Topological Sort: ['C4', 'C5', 'C0', 'C2', 'C3', 'C1']

Masukkan nama file: 4.txt
Graph yang diinput:

In Node

```
A <- []  
B <- ['A']  
C <- ['A', 'B']  
D <- ['B']  
E <- ['C', 'D', 'G']  
F <- ['D', 'G']  
G <- []
```

Out Node

```
A -> ['B', 'C']  
B -> ['C', 'D']  
C -> ['E']  
D -> ['E', 'F']  
E -> []  
F -> []  
G -> ['E', 'F']
```

```
Semester 1 : A G  
Semester 2 : B  
Semester 3 : C D  
Semester 4 : E F
```

Topological Sort: ['A', 'G', 'B', 'C', 'D', 'E', 'F']

Masukkan nama file: 5.txt
Graph yang diinput:

In Node

```
C0 <- []  
C1 <- ['C0', 'C6']  
C2 <- ['C0', 'C1']  
C3 <- ['C2', 'C5']  
C4 <- ['C5']  
C5 <- ['C1', 'C6']  
C6 <- []
```

Out Node

```
C0 -> ['C1', 'C2']  
C1 -> ['C2', 'C5']  
C2 -> ['C3']  
C3 -> []  
C4 -> []  
C5 -> ['C3', 'C4']  
C6 -> ['C1', 'C5']
```

Semester 1 : C0 C6
Semester 2 : C1
Semester 3 : C2 C5
Semester 4 : C3 C4

Topological Sort: ['C0', 'C6', 'C1', 'C2', 'C5', 'C3', 'C4']

Masukkan nama file: 6.txt
Graph yang diinput:

In Node

```
IF2210 <- ['IF2110']  
IF2211 <- []  
IF2220 <- ['IF2120']  
IF2230 <- ['IF2130']  
IF2240 <- []  
IF2250 <- []  
IF2110 <- []  
IF2120 <- []  
IF2130 <- []
```

Out Node

```
IF2210 -> []  
IF2211 -> []  
IF2220 -> []  
IF2230 -> []  
IF2240 -> []  
IF2250 -> []  
IF2110 -> ['IF2210']  
IF2120 -> ['IF2220']  
IF2130 -> ['IF2230']
```

Semester 1 : IF2211 IF2240 IF2250 IF2110 IF2120 IF2130
Semester 2 : IF2210 IF2220 IF2230

Topological Sort: ['IF2211', 'IF2240', 'IF2250', 'IF2110', 'IF2120', 'IF2130', 'IF2210', 'IF2220', 'IF2230']

Masukkan nama file: 7.txt
Graph yang diinput:

In Node

```
A <- []  
B <- []  
C <- ['A']  
D <- ['A', 'B']  
E <- ['B']  
F <- ['C', 'D']  
G <- ['E']  
H <- ['F', 'G']
```

Out Node

```
A -> ['C', 'D']  
B -> ['D', 'E']  
C -> ['F']  
D -> ['F']  
E -> ['G']  
F -> ['H']  
G -> ['H']  
H -> []
```

```
Semester 1 : A B  
Semester 2 : C D E  
Semester 3 : F G  
Semester 4 : H
```

Topological Sort: ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H']

Masukkan nama file: 8.txt
Graph yang diinput:

In Node

```
a <- []  
b <- ['a']  
c <- ['a']  
d <- ['b']  
e <- ['a', 'b']  
f <- ['c']  
g <- ['d', 'e']  
h <- ['c', 'e', 'f']  
i <- ['e', 'g', 'h']
```

Out Node

```
a -> ['b', 'c', 'e']  
b -> ['d', 'e']  
c -> ['f', 'h']  
d -> ['g']  
e -> ['g', 'h', 'i']  
f -> ['h']  
g -> ['i']  
h -> ['i']  
i -> []
```

```
Semester 1 : a  
Semester 2 : b c  
Semester 3 : d e f  
Semester 4 : g h  
Semester 5 : i
```

Topological Sort: ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i']

Masukkan nama file: 9.txt

Graph yang diinput:

In Node

1 <- []

2 <- ['1']

3 <- ['1']

4 <- ['2', '3']

5 <- ['2', '4']

6 <- ['3', '4']

Out Node

1 -> ['2', '3']

2 -> ['4', '5']

3 -> ['4', '6']

4 -> ['5', '6']

5 -> []

6 -> []

Semester 1 : 1

Semester 2 : 2 3

Semester 3 : 4

Semester 4 : 5 6

Topological Sort: ['1', '2', '3', '4', '5', '6']

ALAMAT DRIVE & GITHUB

https://drive.google.com/drive/folders/1D5zCvm_Om5c2BWtRdk4cDoxZVPi5UJwb?usp=sharing

<https://github.com/RizkyAnggita/topological-sort>

Poin	Ya	Tidak
1. Program berhasil dikompilasi.	✓	
2. Program berhasil <i>running</i> .	✓	
3. Program dapat menerima berkas input dan menuliskan output.	✓	
4. Luaran sudah benar untuk semua kasus input.	✓	