LAPORAN TUGAS KECIL 3
IF2211 STRATEGI ALGORITMA
Implementasi Algoritma A* untuk Menentukan Lintasan Terpendek

Oleh :

Mahameru Ds	**13519014**
Ilyasa Salafi Putra Jamal	**13519023**

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA INSTITUT TEKNOLOGI BANDUNG BANDUNG

2021

Kode Program

1. File PriorityQueue.py

```
# Kelas isi/item priority queue
class PriorityQueueItem :
    def __init__(self, item, priority) :
        if(type(priority) != str and type(priority) != int and type(
priority) != float) :
            raise Exception("QueueItem priority must be a string or
int or float")
        self.item = item
        self.priority = priority
    def __str__(self) :
        return "(" + str(self.item) + "," + str(self.priority) + ")"

# Kelas priority queue
class PriorityQueue :
    def __init__(self) :
        self.__values = []
        self.length = 0
    def __str__(self) :
        text = "["
        for i in range(self.length):
            item = self.__values[i]
            text += str(item)
            if(i != self.length-1) :
                text += ","
        text += "]"
        return text

    # Menambahkan item baru ke dalam priority queue
    def enqueue(self, item, priority) :
        newItem = PriorityQueueItem(item, priority)
        self.__values.append(newItem)
        self.length += 1
        self.sort()

    # Menghapus data dari priority queue
    def dequeue(self) :
        item = self.__values.pop(0)
        self.length -= 1
        return item

    # Sort berdasarkan priority
```

```
    def sort(self) :
        for i in range(self.length):
            for j in range(self.length):
                item1 = self.__values[i]
                item2 = self.__values[j]
                if(item1.priority < item2.priority) :
                    temp = item1
                    self.__values[i] = item2
                    self.__values[j] = temp

    # Cek apakah ada item
    def exists(self, item) :
        for val in self.__values :
            if(val.item == item) :
                return True
        return False

    # Cek apakah kosong
    def empty(self) :
        return self.length == 0
```

2. File Node.py

```
class Node :
    def __init__(self, title, x, y) :
        if(type(title) != str ) :
            raise Exception("Node title argument must be a string")
        if(type(x) != int or type(y) != int) :
            raise Exception("X and Y argument must be an integer")
        self.title = title
        self.x = x
        self.y = y
    def __str__(self) :
        return "(" + self.title + "," + str(self.x) + "," + str(self
.y) + ")"
```

3. File Graph.py

```
from Node import Node
from PriorityQueue import PriorityQueue

# Directed Graph
```

```
class Graph :
    def __init__(self) :
        # Node Coordinates
        self.__nodes = {}

        # Edges Weight/Priority
        self.__adjacencyList = {}

    # Getter
    def getNodes(self) :
        return self.__nodes;

    def getAdjacencyList(self) :
        return self.__adjacencyList;

    # Menampilkan semua node
    def printNodes(self) :
        print("[ ", end="")
        for node in self.nodes :
            print(node, end=" ")
        print(" ]")

    # Menambahkan simpul baru
    def addNode(self, nodeTitle, x, y) :
        newNode = Node(nodeTitle, x, y)
        self.__nodes[nodeTitle] = (newNode)
        self.__adjacencyList[newNode.title] = {}

    # Menambahkan sisi baru
    def addEdge(self, nodeStart, nodeEnd, weight) :
        # Error Handling
        if(type(nodeStart) != str or type(nodeEnd) != str ) :
            raise Exception("Node argument must be a string")
        if(type(weight) != int) :
            if(weight != float('inf')) :
                raise Exception("Weight argument must be an integer
or infinity")
        self.__adjacencyList[nodeStart][nodeEnd] = weight

    # Mendapatkan jarak Eucledean
    def getEuclideanDistance(self, x1, x2, y1, y2) :
        return ((x1 - x2)**2 + (y1 - y2)**2)**(0.5)

    # Heuristik
    def getHeuristic(self, nodeStart, nodeEnd) :
```

```
        if(type(nodeStart) != Node or  type(nodeEnd) != Node) :
            raise Exception("Argument must be an instance of Node Ob
ject")
        return self.getEuclideanDistance(nodeStart.x, nodeEnd.x, nod
eStart.y, nodeEnd.y);

    # Mencari Shortest Path antara dua node. Tipe balikan berupa map
 berisi data path dan distance
    # Antara nodeStart dan nodeEnd
    def shortestPath(self, nodeStart, nodeEnd) :
        if(type(nodeStart) != str or type(nodeEnd) != str) :
            raise Exception("Shortest path node argument must be a s
tring")
        # Shortest Path Using A* Algorithm
        nodeSet = PriorityQueue()
        distances = {}
        previous = {}
        visited = []
        # Initial State
        nodeSet.enqueue(nodeStart, 0)
        previous[nodeStart] = None
        # Filling distances with inf
        for node in self.__nodes :
            if(node != nodeStart) :
                distances[node] = float('inf')
            else :
                distances[nodeStart] = 0
        while not nodeSet.empty():
            currentNode = nodeSet.dequeue().item;
            if(currentNode == nodeEnd) :
                path = []
                # distances = 0
                while(previous[currentNode] != None) :
                    path.insert(0,currentNode);
                    currentNode = previous[currentNode];
                path.insert(0, nodeStart)
                return { 'path' : path, 'distance' : distances[nodeE
nd] }

            # Filtering nodes that have inf distance
            adjacencyList = dict(filter(lambda node: node[1] != floa
t('inf'), self.__adjacencyList[currentNode].items()))
            for neighbor in adjacencyList :
                # Continue the for loop if the neighbor is already v
isited
```

```
                if(neighbor in visited) :
                    continue
                weight = self.__adjacencyList[currentNode][neighbor]

                if(weight == float('inf')) :
                    continue
                candidateDistance = distances[currentNode] + int(wei
ght)
                if(candidateDistance < distances[neighbor]) :
                    # Adding neighbor to the previous map and distan
ces map
                    previous[neighbor] = currentNode
                    distances[neighbor] = candidateDistance

                    # Calculating the heuristic value between the st
arting node and the current neighbor.
                    nodeStartInput = self.__nodes[nodeStart]
                    neighborInput = self.__nodes[neighbor]
                    priority = distances[neighbor] + self.getHeurist
ic(nodeStartInput, neighborInput)

                    # Add the current neighbor to the queue.
                    if not nodeSet.exists(neighbor) :
                        nodeSet.enqueue(neighbor, priority)
            visited.append(currentNode)

    # Membuat graf dari file
    # This function only works with specific structure that is speci
fied in the sample.txt file
    def setGraphFromFile(self, filename) :
        # Read the file
        f = open(filename, "r", encoding="utf-8")
        arr = f.readlines()
        temp = []
        for i in arr :
            temp.append(i.rstrip())
        arr = temp
        nodeCount = int(arr[0])
        nodes = arr[1:nodeCount+1]
        adjacencyList = arr[nodeCount+1:]
        # Filling the self.__nodes attribute
        for node in nodes :
            nodeAttributes = node.split();
            title = nodeAttributes[0]
            x = int(nodeAttributes[1])
```

```
            y = int(nodeAttributes[2])
            self.addNode(title, x, y);
        # Filling the self.__adjacencyList attribute;
        for connection in adjacencyList :
            connectionList = connection.split();
            nodeStart = connectionList[0]
            for i in range(len(connectionList[1:])) :
                key = list(self.__nodes.keys())[i]
                nodeEnd = self.__nodes[key].title
                weight = int(connectionList[1:][i])
                if(weight == -1) :
                    weight = float('inf')
                self.addEdge(nodeStart, nodeEnd, weight)
```

4. File main.py

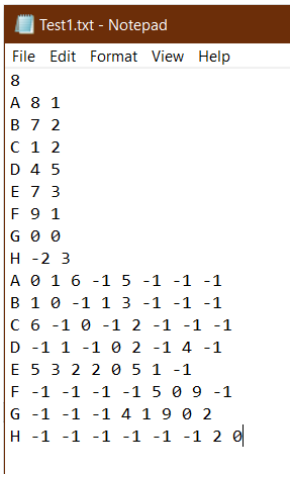
```
import random

# Local Import
from Node import Node
from Graph import Graph

# Dengan mengasumsikan tiap case diberi nama Test1.txt, Test2.txt, d
st
# Automatic test yang akan menjalankan seluruh test case sekaligus d
engan node awal dan akhir random.
def runTestCases(numOfCases) :
    for i in range(numOfCases) :
        filename = f"Test{i+1}.txt"
        graph = Graph()
        graph.setGraphFromFile("../test/" + filename)
        nodes = list(graph.getNodes().keys())
        length = len(nodes)
        n = random.randint(0,length-1)
        start = nodes[n]
        n = random.randint(0,length-1)
        end = nodes[n]
        result = graph.shortestPath(start, end)
        print(f"Test case number {i+1}")
        print(f"Selected nodes are : {start} and {end}")
        print("Path : " + str(result['path']))
        print("Distance : ", result['distance'])
        print()
```

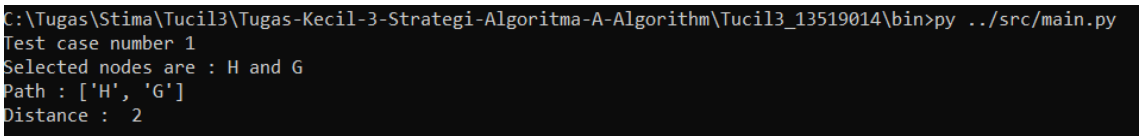
```
def main() :
    # Menjalankan seluruh test cases
    numOfCases = 6
    runTestCases(numOfCases);

    # # Apabila ingin menjalankan test case satu per satu, uncomment
 bagian bawah dan comment runTestCases(numofCases)
    # # Tinggal memasukkan nama file beserta ekstensinya, Contoh: "T
est1.txt"
    # filename = input("Masukkan nama file: \n")
    # graph = Graph()
    # graph.setGraphFromFile("../test/" + filename)

    # # Pastikan nama simpul sesuai dengan pada file input
    # start = input("Masukkan simpul asal: \n")
    # end = input("Masukkan simpul tujuan: \n")
    # result = graph.shortestPath(start, end)
    # print("Path : " + str(result['path']))
    # print("Distance : ", result['distance'])
    # print()


main()
```

Hasil Uji

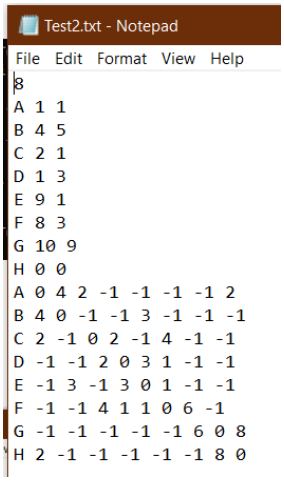
Pengujian dilakukan menggunakan runTestCases() yang langsung menguji semua testcase. Fungsionalitas normal dengan memasukkan secara manual nama file dan kedua node dapat dilakukan dengan cara meng-*comment* baris “runTestCases(numOfCases)” kemudian meng-*uncomment* baris-baris dibawahnya lalu menjalankan file seperti biasa.

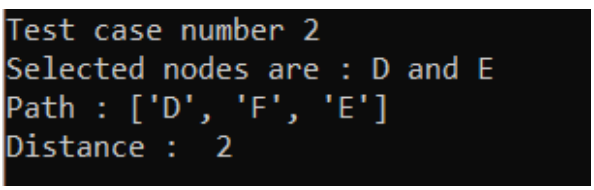
Tes 1:





Tes 2:





Tes 3:

Test3.txt - Notepad

File Edit Format View Help

```
8
kubus -933 104
bonbin -936 84
batan -882 80
sabuga -863 78
dayang -874 115
saraga -856 102
mcdonalds -852 135
borromeus -937 130
kubus 0 27 30 -1 -1 -1 -1 -1
bonbin 27 0 60 -1 -1 -1 -1 -1
batan -1 60 0 62 43 -1 -1 -1
sabuga -1 -1 62 0 54 -1 -1 -1
dayang -1 -1 43 54 0 83 55 -1
saraga -1 -1 -1 -1 83 0 74 -1
mcdonalds -1 -1 -1 -1 55 74 0 98
borromeus 30 -1 -1 -1 -1 -1 98 0
```

```
Test case number 3
Selected nodes are : batan and sabuga
Path : ['batan', 'sabuga']
Distance :  62
```

Tes 4:

Test4.txt - Notepad

File Edit Format View Help

```
8
masjid -218 682
parahyangan -224 629
pendopo -233 701
pln -213 835
museum -211 985
fountain -200 896
sma -239 751
gereja -251 606
masjid 0 81 154 244 -1 -1 407 -1
parahyangan 81 0 152 -1 -1 -1 382 307
pendopo 154 152 0 326 -1 -1 357 -1
pln 407 -1 326 0 214 175 446 -1
museum -1 -1 -1 214 0 277 -1 -1
fountain -1 -1 -1 175 277 0 -1 -1
sma 407 382 357 446 -1 -1 0 267
gereja -1 307 -1 -1 -1 -1 267 0
```

```
Test case number 4
Selected nodes are : gereja and museum
Path : ['gereja', 'parahyangan', 'masjid', 'pln', 'museum']
Distance :  846
```

Tes 5:

Test5.txt - Notepad

File Edit Format View Help

```
8
Borma -554 519
SMPN18 -518 416
Carrefour -466 416
Uninus -645 444
Edelweiss -433 496
Metro -419 588
Dinas -343 625
Pandanwangi -627 515
Borma 0 16 -1 -1 22 -1 -1 13
SMPN18 16 0 6 -1 -1 -1 -1 -1
Carrefour -1 6 0 4 -1 -1 -1 -1
Uninus -1 -1 4 0 7 -1 -1 -1
Edelweiss 22 -1 -1 7 0 12 -1 -1
Metro -1 -1 -1 -1 12 0 15 -1
Dinas -1 -1 -1 -1 -1 15 0 -1
Pandanwangi 13 -1 -1 -1 -1 -1 -1 0
```

```
Test case number 5
Selected nodes are : Dinas and Pandanwangi
Path : ['Dinas', 'Metro', 'Edelweiss', 'Borma', 'Pandanwangi']
Distance :  62
```

Tes 6:

Test6.txt - Notepad

File Edit Format View Help

```
8
Revo -553 699
MM -48 699
Stadium -384 699
Summarecon -269 700
RSUD -422 700
Jakapermai -477 697
GGP -705 697
Kemang -698 699
Revo 0 91 -1 -1 -1 -1 -1 205
MM 91 0 112 -1 214 187 -1 -1
Stadium -1 112 0 134 190 -1 -1 -1
Summarecon -1 -1 134 0 -1 -1 -1 -1
RSUD -1 214 190 -1 0 -1 -1 -1
Jakapermai -1 187 -1 -1 -1 0 36 -1
GGP -1 -1 -1 -1 -1 36 0 25
Kemang 205 -1 -1 -1 -1 -1 25 0
```

```
Test case number 6
Selected nodes are : Stadium and GGP
Path : ['Stadium', 'MM', 'Jakapermai', 'GGP']
Distance :  335


C:\Tugas\Stima\Tucil3\Tugas-Kecil-3-Strategi-Algoritma-A-Algorithm\Tucil3_13519014\bin>PAUSE
Press any key to continue . . .
```

Alamat Source Code

https://github.com/eruds/Tugas-Kecil-3-Strategi-Algoritma-A-Algorithm

(Private)

Folder "../Tucil_13519014/src"

Spesifikasi Program

No.	Poin	Ya	Tidak
1.	Program dapat menerima input graf.	V	
2.	Program dapat menghitung lintasan terpendek.	V	
3.	Program dapat menampilkan lintasan terpendek serta jaraknya.	V	
4.	Bonus: Program dapat menerima input peta dengan Google Map API dan menampilkan peta.		V