Tugas Kecil 3 IF2211 Strategi Algoritma
Semester II Tahun 2022/2023

Implementasi Algoritma UCS dan A* untuk Menentukan Lintasan Terpendek

Disusun oleh:

Christian Albert Hasiholan 13521078
Ammar Rasyad Chaeroel 13521136

PROGRAM STUDI
TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO
DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG 2022

1. DESKRIPSI PERSOALAN

Algoritma UCS (*Uniform Cost Search*) dan A* adalah algoritma pencarian untuk menetukan lintasan terpendek dari satu titik ke titik lain. Implementasi UCS dan A* dalam kasus ini adalah menerima graf dalam bentuk matriks ketetanggaan, start node, dan goal node. Kemudian algoritma akan mencari lintasan paling optimal dan menampilkannya dalam bentuk graf, lintasan yang dilalui, dan jarak optimal yang diperoleh dari algoritma.

Algoritma pencarian dalam bentuk naif (*brute force*) merupakan algoritma yang secara komputasional sangat mahal, sehingga perlu *approach* yang dapat membuat algoritma yang dipastikan dapat mendapatkan lintasan terdekat dalam waktu yang minimal agar bisa digunakan secara real-time.

2. SOURCE CODE (C++)

a_star.py

```
from math import sqrt

class Node:
    def __init__(self, label, weight=0, x=0, y=0):
        self.label = str(label)
        self.weight = weight
        self.x = x
        self.y = y

    def __eq__(self, other):
        return self.label == other.label

    def __repr__(self):
        return f'{self.label}'

    def __hash__(self):
        return hash(self.label)


def build(matrix):
    return {Node(i): [Node(j, matrix[i][j], i, j) for j in
range(len(matrix[i])) if matrix[i][j] > 0] for i in
range(len(matrix))}

def retrace_path(came_from, current):
    path = [current]
    while current in came_from:
        current = came_from[current]
        path.append(current)
    return path[::-1]

class Graph:
    def __init__(self, matrix, start, goal):
        self.nodes = build(matrix)
        self.start = Node(start)  # Start Node
        self.goal = Node(goal)  # Goal Node

        # Input validation
        if self.start not in self.nodes:
            raise Exception(f'Start node {self.start} not in
graph')
```

```
        if self.goal not in self.nodes:
            raise Exception(f'Goal node {self.goal} not in
graph')

    def euclidean_distance(self, node):
        return sqrt((node.x - self.goal.x) ** 2 + (node.y -
self.goal.y) ** 2)

    def calculate(self):
        open_set = {self.start}
        came_from = {}
        g_score = {self.start: 0}
        f_score = {self.start:
self.euclidean_distance(self.start)}

        while len(open_set) > 0:
            current = min(open_set, key=lambda x: f_score[x])
            if current == self.goal:
                return retrace_path(came_from, current),
g_score[self.goal]
            open_set.remove(current)
            for neighbor in self.nodes[current]:
                tentative_g_score = g_score[current] +
neighbor.weight
                if neighbor not in g_score or tentative_g_score
< g_score[neighbor]:
                    came_from[neighbor] = current
                    g_score[neighbor] = tentative_g_score
                    f_score[neighbor] = tentative_g_score +
self.euclidean_distance(neighbor)
                    if neighbor not in open_set:
                        open_set.add(neighbor)
        return None
```

ucs.py

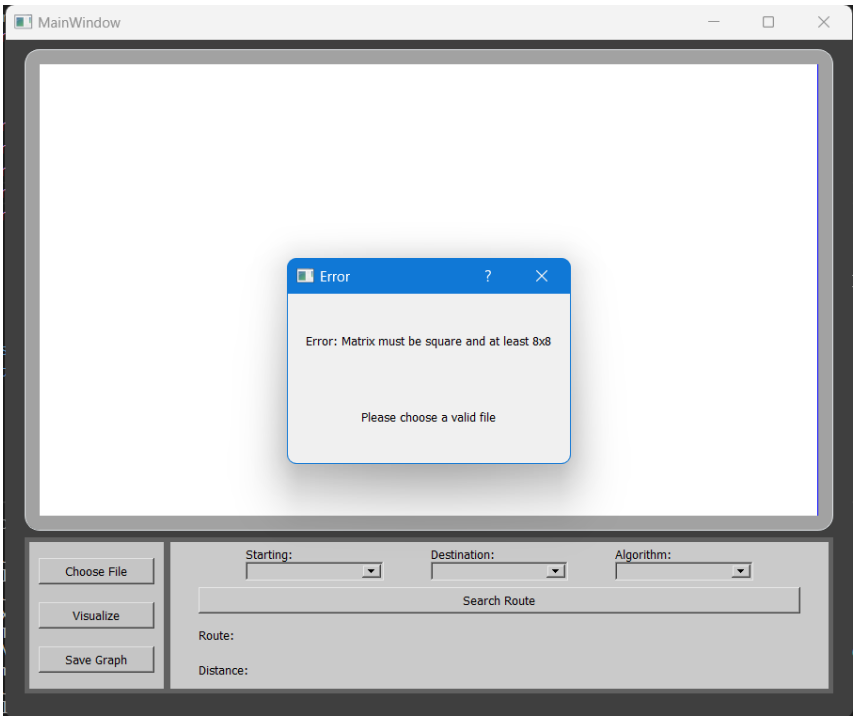
```
def check_node_in_route(route, node):
    return node in route


class UCS:
    def __init__(self, matrix):
        self.matrix = matrix
        self.queue = []

    def get_weight(self, node1, node2):
        return self.matrix[node1-1][node2-1]

    def total_weight(self, route):
        return sum([self.get_weight(route[i], route[i + 1]) for
i in range(len(route)-1)])

    def push_route(self, route):
        i = 0
        if len(self.queue) > 0:
            while len(self.queue) > i and
self.total_weight(self.queue[i]) < self.total_weight(route):
```

```
                i += 1

        self.queue.insert(i, route)
        return

    def get_next_node(self, current_route):
        last_node = current_route[-1]
        next_node = []
        edges = self.matrix[last_node-1]

        for i in range(len(edges)):
            if edges[i] != 0 and not
check_node_in_route(current_route, i + 1):
                next_node.append(i+1)

        return next_node

    def search(self, source, destination):
        self.queue.append([source])
        solution = []
        while len(self.queue) > 0:
            route_now = self.queue.pop(0)

            if route_now[0] == source and route_now[-1] ==
destination:
                solution = route_now
                break

            next_node = self.get_next_node(route_now)
            for node in next_node:
                temp = route_now.copy()
                temp.append(node)
                self.push_route(temp)


        return solution, self.total_weight(solution)


if __name__ == '__main__':
    matrix = []
    while not matrix:
        try:
            filename = input("Masukkan nama file: ")
            with open(filename, 'r') as f:
                for line in f:
                    temp = [int(num) for num in line.split(' ')]
                    matrix.append(temp)
        except IOError | ValueError | FileNotFoundError:
            print("Masukkan file lain")

    ucs = UCS(matrix)

    print(ucs.search(1, 8))
```

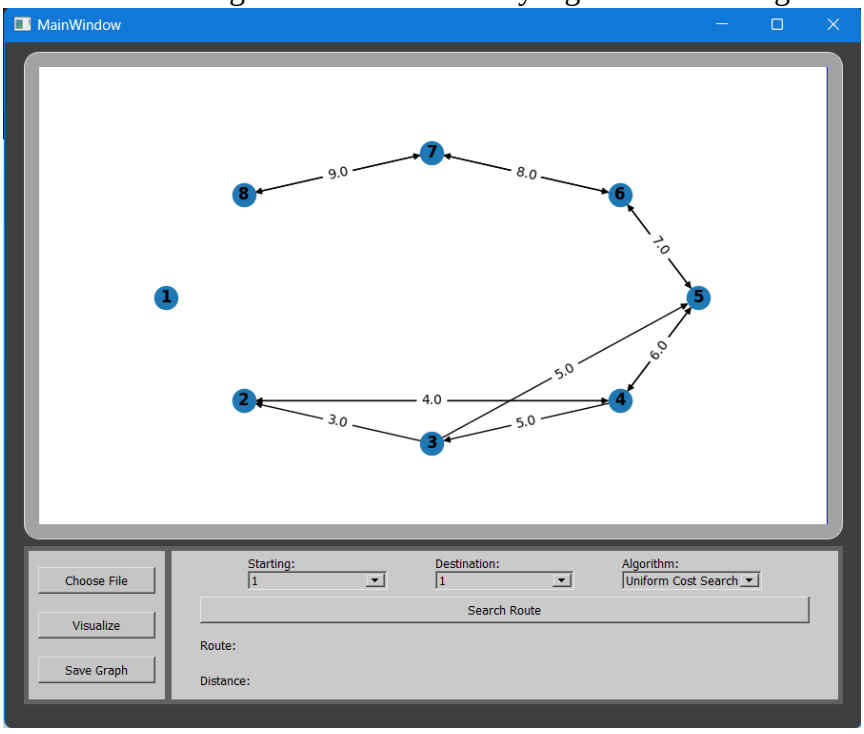
Kode GUI tidak dimasukkan karena bagian yang penting hanya algoritma di atas.

3. CONTOH MASUKAN DAN LUARAN

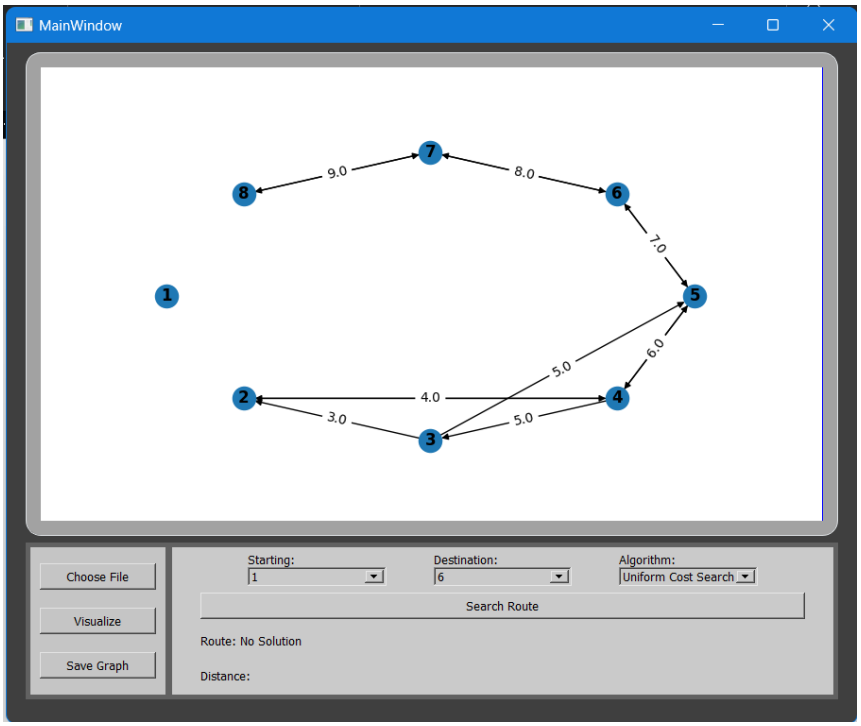
1. Masukan file tidak valid



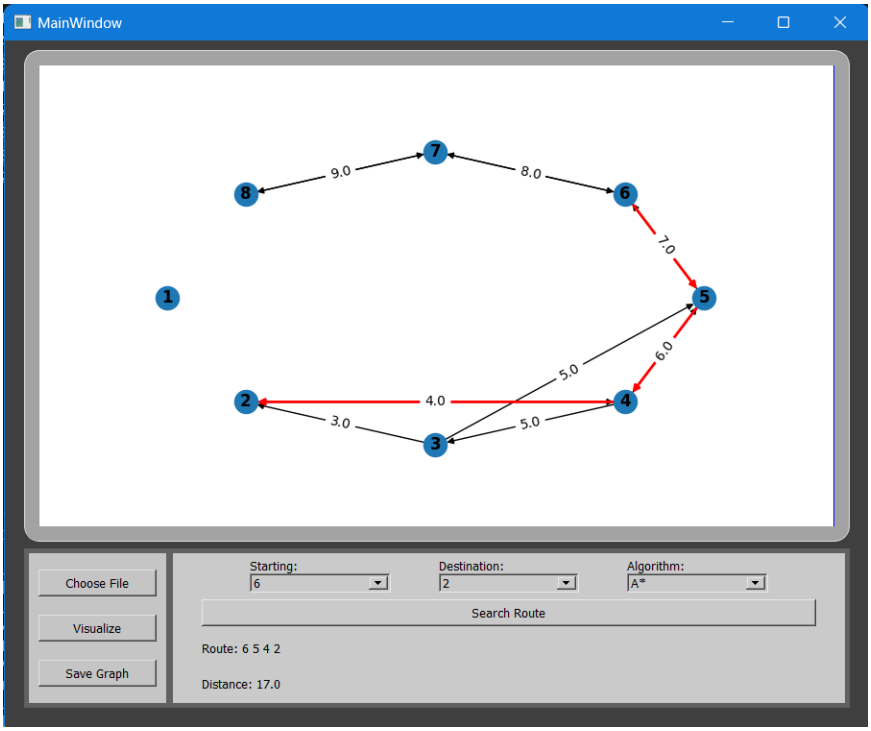
2. File masukan dengan kemunculan node yang tidak terhubung



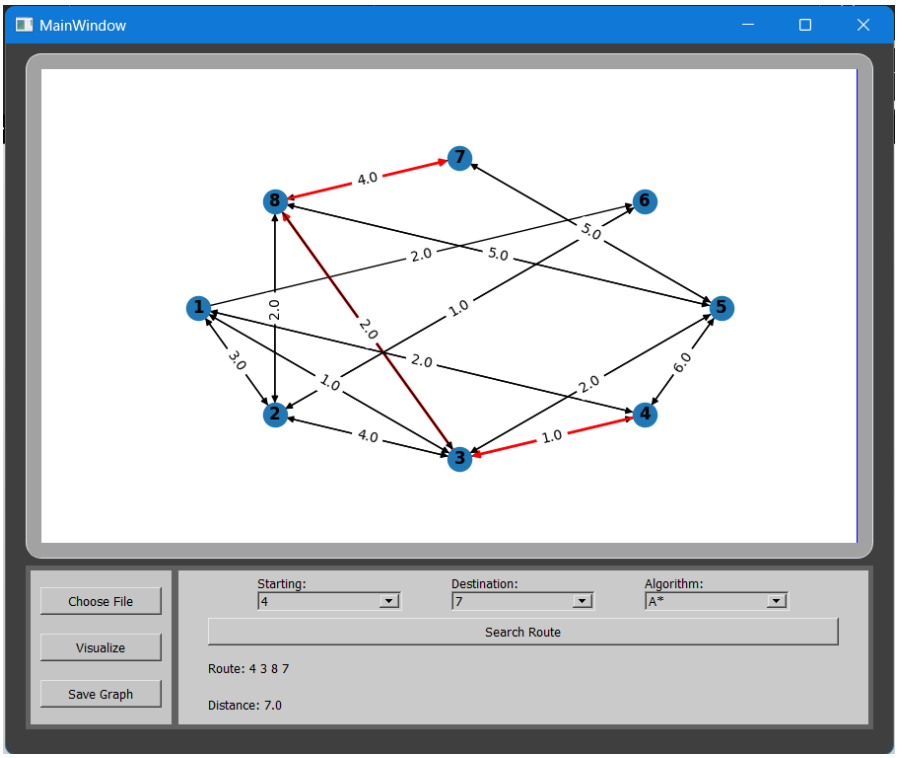
3. Rute solusi tidak ada



4. Pencarian solusi normal



5. Masukan file normal



4. LINK GITHUB

Link Repo GitHub:
https://github.com/ammarasyad/Tucil3_13521078_13521136

1. Program dapat menerima input graf	✓
2. Program dapat menghitung lintasan terpendek dengan UCS	✓
3. Program dapat menghitung lintasan terpendek dengan A*	✓
4. Program dapat menampilkan lintasan terpendek serta jaraknya	✓
5. Bonus: Program dapat menerima input peta dengan Google Map API dan menampilkan peta serta lintasan terpendek pada peta	

5. KESIMPULAN

Kesimpulan yang dapat ditarik adalah algoritma UCS dan A* sama-sama memiliki kelebihan dan kekurangan masing-masing, namun kelebihan A* yang menggunakan fungsi heuristik menjadikan algoritma A* lebih praktis digunakan pada aplikasi navigasi. Selain itu, UCS disebut sebagai *uninformed search* karena tidak menggunakan fungsi heuristik,

sementara A* adalah *informed search*, sehingga algoritma A* dapat mengetahui apakah pencarian sedang mendekat ke tujuan atau tidak.