TUGAS KECIL 3 IF2211 STRATEGI ALGORITMA

SEMESTER II TAHUN 2022/2023

IMPLEMENTASI ALGORITMA UCS DAN A* UNTUK MENENTUKAN LINTASAN TERPENDEK

Disusun Oleh:

13521054 Wilson Tansil

13521143 Raynard Tanadi

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2023

DAFTAR ISI

BAB I
DESKRIPSI PERSOALAN

1. Deskripsi Persoalan

Algoritma UCS (Uniform cost search) dan A* (atau A star) dapat digunakan untuk menentukan lintasan terpendek dari suatu titik ke titik lain. Pada tugas kecil 3 ini, anda diminta menentukan lintasan terpendek berdasarkan peta Google Map jalan-jalan di kota Bandung. Dari ruas-ruas jalan di peta dibentuk graf. Simpul menyatakan persilangan jalan (simpang 3, 4 atau 5) atau ujung jalan. Asumsikan jalan dapat dilalui dari dua arah. Bobot graf menyatakan jarak (m atau km) antar simpul. Jarak antara dua simpul dapat dihitung dari koordinat kedua simpul menggunakan rumus jarak Euclidean (berdasarkan koordinat) atau dapat menggunakan ruler di Google Map, atau cara lainnya yang disediakan oleh Google Map.

Langkah pertama di dalam program ini adalah membuat graf yang merepresentasikan peta (di area tertentu, misalnya di sekitar Bandung Utara/Dago). Berdasarkan graf yang dibentuk, lalu program menerima input simpul asal dan simpul tujuan, lalu menentukan lintasan terpendek antara keduanya menggunakan algoritma UCS dan A*. Lintasan terpendek dapat ditampilkan pada peta/graf (misalnya jalan-jalan yang menyatakan lintasan terpendek diberi warna merah). Nilai heuristik yang dipakai adalah jarak garis lurus dari suatu titik ke tujuan

BAB II

KODE PROGRAM

2.1 AStar.go

```
package Algorithm

import (
	"container/heap"
	"fmt"
)

type Item struct {
	value    interface{}
	priority float64
	index    int
}

type PriorityQueue []*Item

func (pq PriorityQueue) Len() int { return len(pq) }

func (pq PriorityQueue) Less(i, j int) bool {
	return pq[i].priority < pq[j].priority
}

func (pq PriorityQueue) Swap(i, j int) {
	pq[i], pq[j] = pq[j], pq[i]
	pq[i].index = i
	pq[j].index = j
}

func (pq *PriorityQueue) Push(x any) {
	n := len(*pq)
	item := x.(*Item)
	item.index = n
	*pq = append(*pq, item)
}

func (pq *PriorityQueue) Pop() any {
	old := *pq
	n := len(old)
	item := old[n-1]
	old[n-1] = nil
	item.index = -1
	*pq = old[0 : n-1]
	return item
}

func CopyMatrix(matrix [][]float64) [][]float64 {
	rows := len(matrix)
```

```
	cols := len(matrix[0])
	matrixCopy := make([][]float64, rows)
	for i := range matrixCopy {
		matrixCopy[i] = make([]float64, cols)
		copy(matrixCopy[i], matrix[i])
	}
	return matrixCopy
}

func GetKeyByValue(myMap map[string]int, value int) (string, bool) {
	for key, val := range myMap {
		if val == value {
			return key, true
		}
	}
	return "", false
}

func TurnOffNode(adjMatrix [][]float64, firstNode int, secondNode int) {
	adjMatrix[secondNode][firstNode] = 0
	for i := 0; i < len(adjMatrix[firstNode]); i++ {
		adjMatrix[firstNode][i] = 0
	}
	for i := 0; i < len(adjMatrix); i++ {
		adjMatrix[i][secondNode] = 0
	}
}

func AStar(rangeToGoal map[string]float64, adjMatrix [][]float64, nodeIndex
map[string]int, goal string, start string) ([]string, float64) {
	fmt.Println("You are using A* Algorithm")

	pq := make(PriorityQueue, 1)

	tempAdjMatrix := CopyMatrix(adjMatrix)

	distance := 0.0

	fValue := distance + rangeToGoal[start]

	path := []string{start}

	itemValue := []interface{}{
		path,
		start,
		distance,
	}
	pq[0] = &Item{
		value:    itemValue,
		priority: fValue,
		index:    0,
	}
	heap.Init(&pq)
```

```
	for pq.Len() > 0 {
		currentNode := heap.Pop(&pq).(*Item)
		currentValue := currentNode.value.([]interface{})
		nodeName := currentValue[1].(string)
		currentIdx := nodeIndex[nodeName]
		currentPath := currentValue[0].([]string)
		distanceCost := currentValue[2].(float64)
		fmt.Println("This is the path now", currentPath)
		if nodeName == goal {
			path = currentPath
			distance = distanceCost
			break
		}

		for idx, val := range tempAdjMatrix[currentIdx] {
			if val != 0 {
				nextNode, _ := GetKeyByValue(nodeIndex, idx)
				tempCurrentPath := append([]string{}, currentPath...)
				tempCurrentPath = append(tempCurrentPath, nextNode)
				tempDistanceCost := distanceCost + val
				tempFValue := tempDistanceCost + rangeToGoal[nextNode]
				tempItemValue := []interface{}{
					tempCurrentPath,
					nextNode,
					tempDistanceCost,
				}
				tempItem := &Item{
					value:    tempItemValue,
					priority: tempFValue,
				}
				heap.Push(&pq, tempItem)
			}
		}
	}

	return path, distance
}
```

2.2 UCS.go

```
package Algorithm

import (
	"container/heap"
	"fmt"
)

func UCS(adjMatrix [][]float64, nodeIndex map[string]int, goal string, start string)
([]string, float64) {
```

```
	fmt.Println("You are using UCS Algorithm")
	pq := make(PriorityQueue, 1)

	tempAdjMatrix := CopyMatrix(adjMatrix)

	distance := 0.0

	path := []string{start}

	visited := make(map[string]bool)
	visited[start] = true

	itemValue := []interface{}{
		path,
		start,
		distance,
	}
	pq[0] = &Item{
		value:    itemValue,
		priority: distance,
		index:    0,
	}
	heap.Init(&pq)

	for pq.Len() > 0 {
		currentNode := heap.Pop(&pq).(*Item)
		currentValue := currentNode.value.([]interface{})
		nodeName := currentValue[1].(string)
		visited[nodeName] = true
		currentIdx := nodeIndex[nodeName]
		currentPath := currentValue[0].([]string)
		distanceCost := currentValue[2].(float64)
		fmt.Println("This is the path now", currentPath)
		if nodeName == goal {
			path = currentPath
			distance = distanceCost
			break
		}

		for idx, val := range tempAdjMatrix[currentIdx] {
			if val != 0 {
				nextNode, found := GetKeyByValue(nodeIndex, idx)
				if found && !visited[nextNode] {
					tempCurrentPath := append([]string{},
currentPath...) // create a copy of currentPath
					tempCurrentPath = append(tempCurrentPath, nextNode)
					tempDistanceCost := distanceCost + val
					tempItemValue := []interface{}{
						tempCurrentPath,
						nextNode,
						tempDistanceCost,
					}
					tempItem := &Item{
```

```
                                        value:    tempItemValue,
                                        priority: tempDistanceCost,
                                    }
                                    heap.Push(&pq, tempItem)
                                }
                            }
                        }
                }

                return path, distance
        }
```

2.3 graph.go

```
package Class

import (
        "fmt"
        "log"
        "math"
        "strconv"
)

type Graph struct {
        TotalNodes      int
        AdjacencyMatrix [][]float64
        Nodes           []Node
}

type Node struct {
        Name       string
        Latitude   float64
        Longtitude float64
}

func (graph *Graph) GetTotalNodes() int {
        return graph.TotalNodes
}

func (graph *Graph) GetNodes() []Node {
        return graph.Nodes
}

func (node *Node) GetLatitude() float64 {
        return node.Latitude
}

func (node *Node) GetLongtitude() float64 {
        return node.Longtitude
```

```
}

func (node *Node) GetName() string {
	return node.Name
}

func (graph *Graph) IsTetanggaFloat(from string, destination string) float64 {
	index1 := -1
	index2 := -1
	for i := 0; i < len(graph.Nodes); i++ {
		if graph.Nodes[i].Name == from {
			index1 = i
		}
		if graph.Nodes[i].Name == destination {
			index2 = i
		}
	}
	if index1 == -1 || index2 == -1 {
		log.Fatal("Error : Node tidak ditemukan2")
	}
	return graph.AdjacencyMatrix[index1][index2]
}

func degToRad(deg float64) float64 {
	return deg * (math.Pi / 180)
}

func (graph *Graph) GetDistance(from string, destination string) float64 {
	isTetangga := graph.IsTetanggaFloat(from, destination)
	distance := -1.0
	earthRadius := 6371.0
	if isTetangga == 1 {
		index1 := graph.GetIndex(from)
		index2 := graph.GetIndex(destination)

		lat1Rad := degToRad(graph.Nodes[index1].Latitude)
		lat2Rad := degToRad(graph.Nodes[index2].Latitude)
		lon1Rad := degToRad(graph.Nodes[index1].Longtitude)
		lon2Rad := degToRad(graph.Nodes[index2].Longtitude)

		deltaLat := lat2Rad - lat1Rad
		deltaLon := lon2Rad - lon1Rad

		a := math.Sin(deltaLat/2)*math.Sin(deltaLat/2) +
			math.Cos(lat1Rad)*math.Cos(lat2Rad)*
				math.Sin(deltaLon/2)*math.Sin(deltaLon/2)

		// Calculate the angular distance in radians
		c := 2 * math.Atan2(math.Sqrt(a), math.Sqrt(1-a))

		// Calculate the distance in kilometers
		distance := earthRadius * c
		return distance
```

```
	}
	log.Print("Node tidak bertertangga")
	return distance

}

func (graph *Graph) GetDistanceToGoal(from string, destination string) float64 {
	distance := -1.0
	earthRadius := 6371.0

	index1 := graph.GetIndex(from)
	index2 := graph.GetIndex(destination)
	lat1Rad := degToRad(graph.Nodes[index1].Latitude)
	lat2Rad := degToRad(graph.Nodes[index2].Latitude)
	lon1Rad := degToRad(graph.Nodes[index1].Longtitude)
	lon2Rad := degToRad(graph.Nodes[index2].Longtitude)

	deltaLat := lat2Rad - lat1Rad
	deltaLon := lon2Rad - lon1Rad

	a := math.Sin(deltaLat/2)*math.Sin(deltaLat/2) +
math.Cos(lat1Rad)*math.Cos(lat2Rad)*math.Sin(deltaLon/2)*math.Sin(deltaLon/2)

	c := 2 * math.Atan2(math.Sqrt(a), math.Sqrt(1-a))

	//dalam km
	distance = earthRadius * c
	return distance

}

func (graph *Graph) GetIndex(nodeSearched string) int {
	for i := 0; i < len(graph.Nodes); i++ {
		if graph.Nodes[i].Name == nodeSearched {
			return i
		}
	}
	log.Print("Error : node tidak ditemukan3")
	return 0
}

func PrintAllNodes(nodes []Node) {
	count := 0
	for _, tempNode := range nodes {
		count++
		strCount := strconv.Itoa(count)
		fmt.Printf(strCount + ". " + tempNode.Name)
		fmt.Printf("\n")
	}
}

func PrintAllWeight(graph *Graph) {
	for i := 0; i < graph.TotalNodes; i++ {
```

```
            for j := 0; j < graph.TotalNodes; j++ {
                  if graph.IsTetanggaFloat(graph.Nodes[i].Name,
graph.Nodes[j].Name) == 1 {
                        distance := graph.GetDistance(graph.Nodes[i].Name,
graph.Nodes[j].Name)
                        distanceString := strconv.FormatFloat(distance, 'f', -1,
64)
                        fmt.Println(graph.Nodes[i].Name, " - ",
graph.Nodes[j].Name, " = ", distanceString)
                  }
            }
      }
}

func PrintAdjacencyNodes(adjacencyMatrix [][]float64) {
      for _, tempMatrix := range adjacencyMatrix {
            fmt.Println(tempMatrix)
      }
}

func WeightedAdjacencyMatrix(adjacencyMatrix [][]float64, graph *Graph) [][]float64
{
      rows := len(adjacencyMatrix)
      cols := len(adjacencyMatrix[0])

      result := make([][]float64, rows)
      for i := 0; i < rows; i++ {
            result[i] = make([]float64, cols)
      }

      for i := 0; i < rows; i++ {
            for j := 0; j < cols; j++ {
                  if graph.IsTetanggaFloat(graph.Nodes[i].Name,
graph.Nodes[j].Name) > 0 {
                        distance := graph.GetDistance(graph.Nodes[i].Name,
graph.Nodes[j].Name)
                        result[i][j] = adjacencyMatrix[i][j] * distance
                  } else {
                        result[i][j] = 0
                  }
            }
      }

      return result
}
```

2.4 gui.py

```
from tkinter import *
import tkintermapview
```

```
import socket
from tkinter import filedialog
import os
import subprocess
import customtkinter


class Application(Frame):
    def __init__(self, master=None):
        # Get the width and height of the screen
        screen_width = root.winfo_screenwidth()
        screen_height = root.winfo_screenheight()

        # Set the size and position of the window
        window_width = 1150
        window_height = 648
        x = (screen_width - window_width) // 2
        y = (screen_height - window_height) // 2
        master.geometry(f"{window_width}x{window_height}+{x}+{y}")
        master.configure(bg='#282B34')
        master.resizable(False, False)
        super().__init__(master)
        self.pack()
        self.create_widgets()

    def change_map(self, new_map: str):
        if new_map == "OpenStreetMap":

self.map_widget.set_tile_server("https://a.tile.openstreetmap.org/{z}/{x}/{y}.png")
        elif new_map == "Google normal":

self.map_widget.set_tile_server("https://mt0.google.com/vt/lyrs=m&hl=en&x={x}&y={y}&
z={z}&s=Ga", max_zoom=22)
        elif new_map == "Google satellite":

self.map_widget.set_tile_server("https://mt0.google.com/vt/lyrs=s&hl=en&x={x}&y={y}&
z={z}&s=Ga", max_zoom=22)

    def send_input_to_go_algorithm(self):
        input_str = self.selected_file_name + "@" + self.radio_var.get() + "@" + \
            self.entry_start.get() + "@" + self.entry_goal.get()
        # Connect to Go Algorithm
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        s.connect(('localhost', 8080))
        s.sendall(input_str.encode())

        # Receive result from Go Algorithm
        result = s.recv(1024).decode()
        result_list = result.split(" ")
        n_result_list = len(result_list)
        result_list_path = result_list[:n_result_list-1]
        my_path = []
        for path in result_list_path:
```

```
            my_path.append(self.my_marker_dic[path].position)
        result_path = ' '.join(result_list[:n_result_list-1])
        result_cost = result_list[n_result_list-1]
        self.map_widget.delete_all_path()
        self.label_result_container_path.config(text="Path:" + result_path)
        self.label_result_container_cost.config(text="Cost: " + result_cost + " km")
        self.map_widget.set_path(my_path)

    def choose_file_name(self):
        subprocess.Popen(['go', 'run', 'main.go'])  # Run the Go file
        self.selected_file_name = filedialog.askopenfilename(
            initialdir="./test", title="Select A File", filetypes=(("txt files",
"*.txt"), ("all files", "*.*")))
        self.read_file()
        self.my_marker_dic = {}
        position = []
        count = 0
        self.map_widget.delete_all_marker()
        self.map_widget.delete_all_polygon()
        self.map_widget.delete_all_path()
        for markerKey in self.my_dic:
            latitude, longitude = self.my_dic[markerKey]
            name = markerKey
            self.my_marker_dic[markerKey] = self.map_widget.set_marker(
                float(latitude), float(longitude), name)
        # print(self.adjMatrix)
        for i in range(len(self.adjMatrix)):
            for j in range(len(self.adjMatrix[i])):
                if (self.adjMatrix[i][j] == '1'):
                    # print(self.nodeIdx[i], "to", self.nodeIdx[j])
                    position.append([self.my_marker_dic[self.nodeIdx[i]].position,
                                     self.my_marker_dic[self.nodeIdx[j]].position])
        for coor in position:
            self.map_widget.set_polygon(coor, outline_color="red")

    def read_file(self):
        self.my_dic = {}
        self.adjMatrix = []
        self.nodeIdx = []
        count = 0
        with open(self.selected_file_name, 'r') as file:
            content = file.readlines()
            n = int(content[0])
            for line in content[1:n+1]:
                parts = line.split()
                self.my_dic[parts[0]] = (parts[1], parts[2])
                self.nodeIdx.append(parts[0])
            # Find node with most neighbors and set as center node
            max_neighbors = -1
            for node in self.my_dic:
                neighbors = sum(
                    1 for adj in content[n+1:] if adj.startswith(node))
                if neighbors > max_neighbors:
```

```
                    self.key_center = node
                    max_neighbors = neighbors
            for adj in content[n+1:]:
                splitAdjData = adj.strip().split()
                self.adjMatrix.append(splitAdjData)
        latitude, longitude = self.my_dic[self.key_center]
        self.map_widget.set_position(
            float(latitude), float(longitude), self.key_center)

    # def mark_location(self):

    def create_widgets(self):
        # Create container
        # left container
        self.left_container = Frame(self, bg="black", width=450, height=700)

        # Choose File Part
        self.file_container = Frame(
            self.left_container, width=300, height=200, bg='black')
        self.label_file_container = Label(self.file_container, text="Choose File",
font=(
            "Arial", 20), fg='white', bg='black')
        self.label_file_container.pack(pady=(60, 20))
        self.choose_file_btn = Button(self.file_container)
        self.choose_file_btn["text"] = "Upload File"
        self.choose_file_btn["command"] = self.choose_file_name
        self.choose_file_btn["font"] = ("Courier New", 12)
        self.choose_file_btn["bg"] = 'gray'
        self.choose_file_btn.pack(pady=(10, 30))

        # Choose Input Part
        self.input_container = Frame(
            self.left_container, width=300, height=150, bg='black')
        self.label_input_start = Label(self.input_container, text="Input Starting
Point : ",
                                       anchor="nw", justify="left", fg="white",
bg='black', font=("Courier New", 12))
        self.label_input_start.grid(row=0, column=0, padx=10)
        self.entry_start = Entry(self.input_container)
        self.entry_start.grid(row=0, column=1, padx=10, pady=10)
        self.label_input_goal = Label(self.input_container, text="Input Goal Point
: ",
                                      anchor="nw", justify="left", fg="white",
bg='black', font=("Courier New", 12))
        self.label_input_goal.grid(row=1, column=0, padx=10, pady=(10, 30))
        self.entry_goal = Entry(self.input_container)
        self.entry_goal.grid(row=1, column=1, padx=10, pady=(10, 30))

        # Choosing Algorithm Part
        self.algorithm_container = Frame(
            self.left_container, width=300, height=150, bg='black')

        # Create radio buttons
```

```
        self.radio_var = StringVar()
        self.radio_var.set(None)

        self.radio_button_1 = Radiobutton(self.algorithm_container, text=" A*
Algorithm", variable=self.radio_var,
                                          value="Option 1", fg="white", bg='black',
font=("Courier New", 12), selectcolor="gray")
        self.radio_button_2 = Radiobutton(self.algorithm_container, text="UCS
Algorithm", variable=self.radio_var,
                                          value="Option 2", fg="white", bg='black',
font=("Courier New", 12), selectcolor="gray")

        # Place radio buttons in the frame
        self.radio_button_1.pack()
        self.radio_button_2.pack()

        # Pack child frames of the left container
        self.file_container.pack(side="top", padx=10, pady=10)
        self.input_container.pack(side="top", padx=10, pady=10)
        self.algorithm_container.pack(side="top", padx=10, pady=10)

        # Button container
        self.button_container = Frame(
            self.left_container, width=300, height=150, bg="black")
        self.submit_button = Button(self.button_container)
        self.submit_button["text"] = "Search"
        self.submit_button["bg"] = "gray"
        self.submit_button["font"] = ("Courier New", 11)
        self.submit_button["command"] = self.send_input_to_go_algorithm
        self.submit_button.pack()
        self.button_container.pack(padx=10, pady=(20, 10))

        # Map label container
        self.map_label = customtkinter.CTkLabel(self.left_container, text="Tile
Server:", anchor="w", font= ("Courier New", 12))
        self.map_label.pack(side="left", padx=(20, 20), pady=(0, 0))
        self.map_option_menu = customtkinter.CTkOptionMenu(self.left_container,
values=["OpenStreetMap", "Google normal", "Google satellite"],

command=self.change_map)
        self.map_option_menu.pack(side= "left", padx=(20, 20), pady=(0, 0))

        self.left_container.pack(side="left", fill=BOTH)

        # Right Container
        self.right_container = Frame(self, width=850, height=700)

        # Map Container
        self.map_container = Frame(self.right_container, width=850, height=600)
        self.map_widget = tkintermapview.TkinterMapView(
            self.map_container, width=850, height=600, corner_radius=0)
        self.map_widget.pack()
        self.map_container.pack(side="top")
```

```
        # Result Container
        self.result_container = Frame(
            self.right_container, width=850, height=100)
        self.label_result_container_path = Label(
            self.result_container, text="Path : ", anchor="nw", justify="left",
fg="white", bg='gray', font=("Courier New", 12), width=850)
        self.label_result_container_path.pack()
        self.label_result_container_cost = Label(
            self.result_container, text="Cost : ", anchor="nw", justify="left",
fg="white", bg='gray', font=("Courier New", 12), width=850)
        self.label_result_container_cost.pack()
        self.result_container.pack()
        self.right_container.pack(side="right", fill=BOTH)


root = Tk()
app = Application(master=root)
app.mainloop()
```

2.5 main.go

```
package main

import (
       "TUCIL3_13521054_13521143/src/Algorithm"
       "TUCIL3_13521054_13521143/src/Class"
       "bufio"
       "fmt"
       "log"
       "net"
       "os"
       "strconv"
       "strings"
)

func read(filepath string, algo string, start string, goal string) ([]string,
float64) {
       // Read input from file
       // filepath := "../test/test.txt"
       file, err := os.Open(filepath)
       fmt.Println(filepath)
       if err != nil {
              log.Fatal(err)
       }
       defer file.Close()

       scanner := bufio.NewScanner(file)
```

```
	scanner.Split(bufio.ScanLines)

	// Read totalNodes
	scanner.Scan()
	totalNodes, err := strconv.Atoi(scanner.Text())
	if err != nil {
		log.Fatal(err)
	}

	// Initialize Graph
	graph := Class.Graph{
		TotalNodes:      totalNodes,
		AdjacencyMatrix: make([][]float64, totalNodes),
		Nodes:           make([]Class.Node, totalNodes),
	}

	// Read nodes data
	for i := 0; i < totalNodes; i++ {
		scanner.Scan()
		line := strings.Fields(scanner.Text())
		edge := line[0]
		for j := 1; j < len(line)-2; j++ {
			edge = edge + " " + line[j]
		}
		latitude, _ := strconv.ParseFloat(line[len(line)-2], 64)
		longitude, _ := strconv.ParseFloat(line[len(line)-1], 64)
		graph.Nodes[i] = Class.Node{edge, latitude, longitude}
	}

	// Read adjacency matrix
	for i := 0; i < totalNodes; i++ {
		scanner.Scan()
		adjacencyData := strings.Split(scanner.Text(), " ")

		graph.AdjacencyMatrix[i] = make([]float64, totalNodes)
		for j := 0; j < totalNodes; j++ {
			weight, err := strconv.ParseFloat(adjacencyData[j], 64)
			if err != nil {
				log.Fatal(err)
			}
			graph.AdjacencyMatrix[i][j] = weight
		}
	}
	// Print all nodes
	fmt.Println("Nodes:")
	Class.PrintAllNodes(graph.Nodes)
	fmt.Print("\n")
	fmt.Println("AdjacencyMatrix:")
	Class.PrintAdjacencyNodes(graph.AdjacencyMatrix)
	fmt.Print("\n")
	fmt.Println("Weight:")
	Class.PrintAllWeight(&graph)
	rangeToGoal := make(map[string]float64)
```

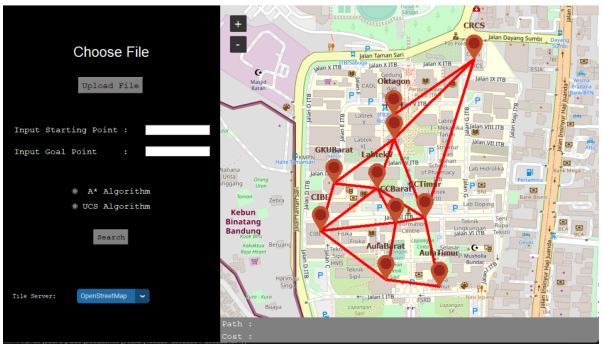
```
	keys := []string{}
	for i := 0; i < graph.TotalNodes; i++ {
		keys = append(keys, graph.Nodes[i].Name)
	}
	for _, key := range keys {
		distance := graph.GetDistanceToGoal(key, goal)
		rangeToGoal[key] = distance
	}
	// fmt.Println(rangeToGoal)
	// include adj matrix
	nodeIdx := make(map[string]int)
	for i := 0; i < graph.TotalNodes; i++ {
		nodeIdx[graph.Nodes[i].Name] = graph.GetIndex(graph.Nodes[i].Name)
	}
	adjMatrix2 := Class.WeightedAdjacencyMatrix(graph.AdjacencyMatrix, &graph)
	if algo == "Option 2" {
		path, distance := Algorithm.UCS(adjMatrix2, nodeIdx, goal, start)
		return path, distance
	} else {
		path, distance := Algorithm.AStar(rangeToGoal, adjMatrix2, nodeIdx,
goal, start)
		return path, distance
	}
}

func main() {
	ln, err := net.Listen("tcp", ":8080")
	if err != nil {
		fmt.Println("Error:", err)
		return
	}
	defer ln.Close()

	fmt.Println("Go Algorithm is ready and listening on :8080")

	for {
		conn, err := ln.Accept()
		if err != nil {
			fmt.Println("Error:", err)
			continue // Continue listening for new connections
		}

		// Handle connection in a separate goroutine
		go handleConnection(conn)
	}
}

func handleConnection(conn net.Conn) {
	defer conn.Close()

	buf := make([]byte, 1024)
	n, err := conn.Read(buf)
	if err != nil {
```

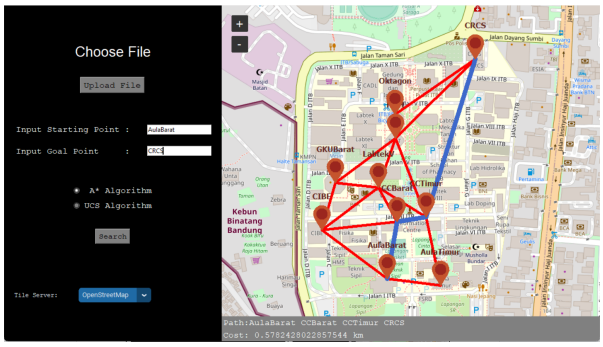
```
		fmt.Println("Error:", err)
		return
	}

	// Process input from GUI
	input := string(buf[:n])
	result := processInput(input)

	// Send result back to GUI
	_, err = conn.Write([]byte(result))
	if err != nil {
		fmt.Println("Error:", err)
		return
	}

	fmt.Println("Go Algorithm finished")
	fmt.Println(result)
}

func processInput(input string) string {
	inputSplit := strings.Split(input, "@")
	wordArray := []string{}
	for _, word := range inputSplit {
		wordArray = append(wordArray, word)
	}
	path, cost := read(wordArray[0], wordArray[1], wordArray[2], wordArray[3])
	result := ""
	for i := 0; i < len(path); i++ {
		result = strings.Join(path, " ")
	}
	result = result + " " + strconv.FormatFloat(cost, 'f', -1, 64)
	return result
}
```

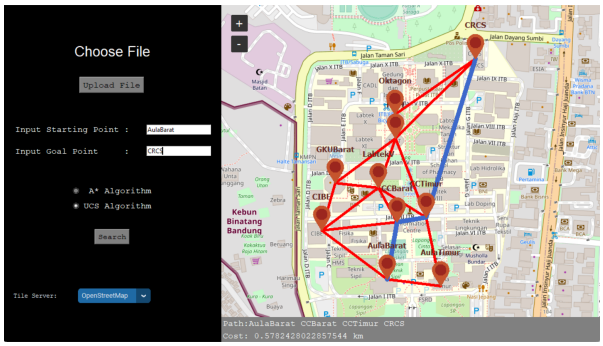
BAB III

PENGUJIAN PROGRAM

3.1 Peta jalan sekitar kampus ITB/Dago/Bandung Utara

<table>
<tr><th>Peta/Graf Input</th><th>Hasil</th></tr>
<tr><td>


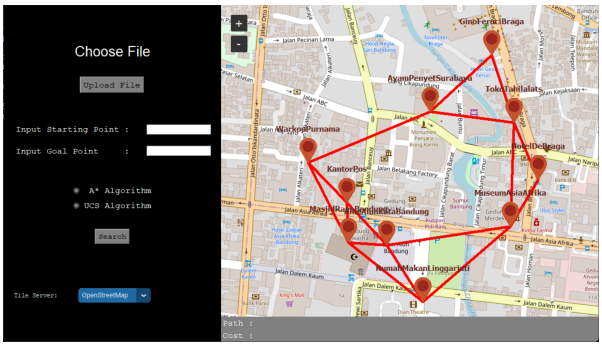
test > tesITB.txt
10
AulaTimur -6.892470442498283 107.6110106483379
AulaBarat -6.8923292596633745 107.60990582398884
CIBE -6.89132271045863 107.60854862632463
GKUBarat -6.8903534388107985 107.6088329404579
CCBarat -6.891152289194305 107.61010430743978
CCTimur -6.891056427219338 107.61071048668443
LabtekV -6.8904546266004845 107.6097180694201
TVST -6.889442747714525 107.61007211097427
Oktagon -6.888947458849591 107.61006675655662
CRCS -6.887813085592755 107.61172972611223
0 1 0 0 0 1 0 0 0 0
1 0 1 0 1 0 0 0 0 0
0 1 0 1 1 0 1 0 0 0
0 0 1 0 0 0 1 1 0 0
0 1 1 0 0 1 1 0 0 0
1 0 0 0 1 0 1 1 0 1
0 0 1 1 1 1 0 1 0 0
0 0 0 1 0 1 1 0 1 1
0 0 0 0 0 0 0 0 0 1
0 0 0 0 0 1 0 1 1 0
</td><td>


Start : AulaBarat
Goal : CRCS
Path : AulaBarat CCBarat CCTimur CRCS
Cost : 0.578248022857544 km
AStar Algorithm


Start : AulaBarat
Goal : CRCS
Path : AulaBarat CCBarat CCTimur CRCS
Cost : 0.578248022857544 km
UCS Algorithm
</td></tr>
</table>

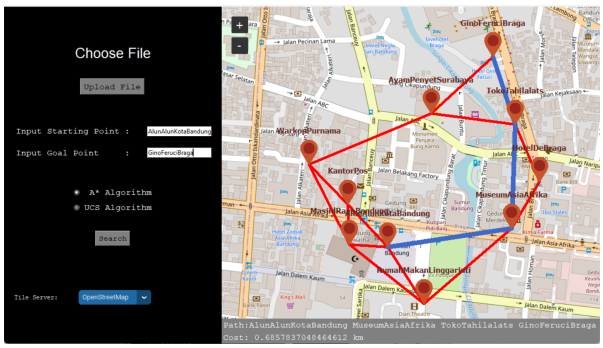
3.2 Peta jalan sekitar Alun-alun Bandung

Peta/Graf Input	Hasil

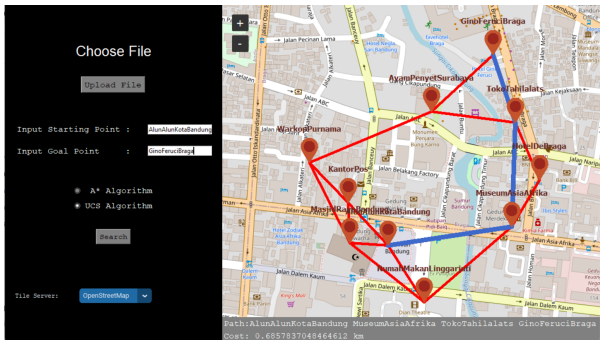


test > tesAlunAlun.txt

```
10
AlunAlunKotaBandung -6.921618216168829 107.60694944454413
KantorPos -6.9207494436636 107.60612993209766
MasjidRayaBandung -6.921564414976234 107.60616442594767
WarkopPurnama -6.9199276224277 107.60532277600689
MuseumAsiaAfrika -6.921234378617717 107.60952191638391
TokoTahilalats -6.919107499818133 107.60959818792549
AyamPenyetSurabaya -6.918887179847317 107.60785632493403
HotelDeBraga -6.920275861516295 107.6100958631183
GinoFeruciBraga -6.917725491064756 107.60913413947199
RumahMakanLinggarjati -6.922779484049875 107.60770164210942
0 1 1 0 1 0 0 0 0 1
1 0 1 1 0 0 0 0 0 0
1 1 0 1 0 0 0 0 0 1
0 1 1 0 0 0 1 0 0 0
1 0 0 0 0 1 0 1 0 1
0 0 0 0 1 0 1 1 1 0
0 0 0 1 0 1 0 0 1 0
0 0 0 0 1 1 0 0 0 0
0 0 0 0 0 1 1 0 0 0
1 0 1 0 1 0 0 0 0 0
```



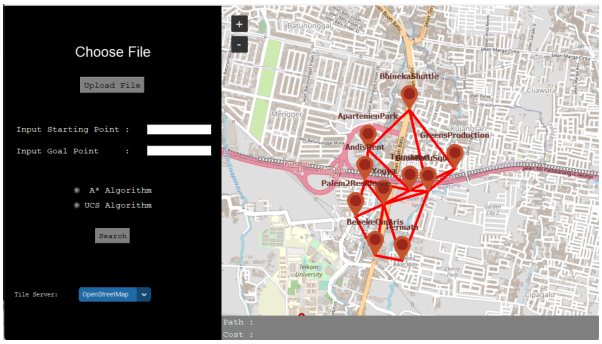
Start : AlunAlunKotaBandung
Goal : GinoFeruciBraga
Path : AlunAlunKotaBandung MuseumAsiaAfrika TokoTahilalats GinoFeruciBraga
Cost : 0.6857837048464612 km
AStar Algorithm



Start : AlunAlunKotaBandung
Goal : GinoFeruciBraga
Path : AlunAlunKotaBandung MuseumAsiaAfrika TokoTahilalats GinoFeruciBraga
Cost : 0.6857837048464612 km
UCS Algorithm

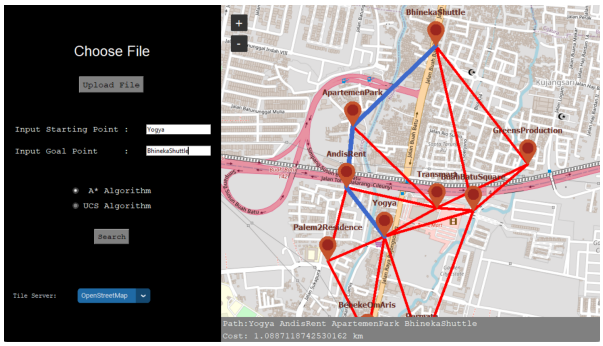
3.3 Peta jalan sekitar Buahbatu atau Bandung Selatan

Peta/Graf Input	Hasil

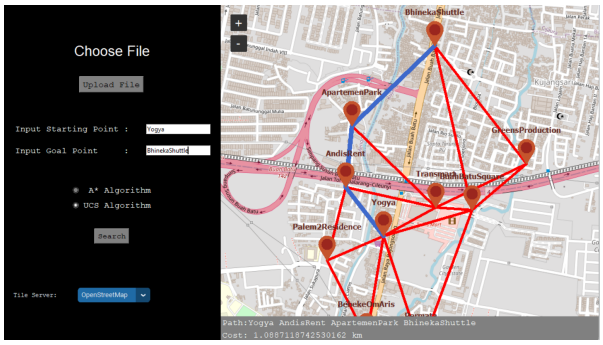


test > tesBuahBatu.txt

```
10
BuahBatuSquare -6.966676402220131 107.6407459644164
Transmart -6.9665699059741 107.63924392747579
ApartemenPark -6.963225911546734 107.63576778484182
BhinekaShuttle -6.959945791944222 107.63920101213463
Permata -6.972384565632324 107.63862165503203
Yogya -6.967741363377569 107.6370670275026
AndisRent -6.965742634484649 107.63551163123819
GreensProduction -6.964769978955752 107.64298567261696
BebekeOmAris -6.971921810714622 107.63635702409339
Palem2Residence -6.968736875114074 107.63474309227892
0 1 0 1 1 1 0 1 0 0
1 0 1 0 0 1 1 1 0 0
0 1 0 1 0 0 1 0 0 0
1 0 1 0 0 0 0 1 0 0
1 0 0 0 0 1 0 0 1 0
1 1 0 0 1 0 1 0 1 1
0 1 1 0 0 1 0 0 0 1
1 1 0 1 0 0 0 0 0 0
0 0 0 0 1 1 0 0 0 1
0 0 0 0 0 1 1 0 1 0
```



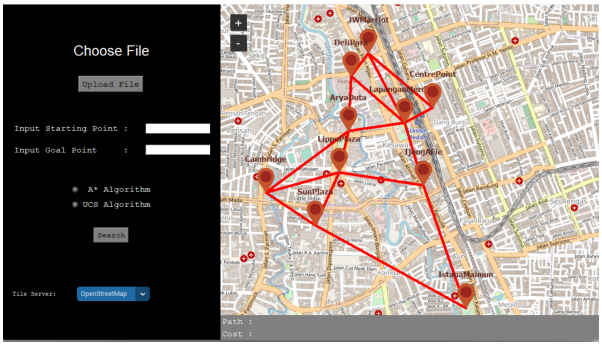
Start : Yogya
Goal : BhinekaShuttle
Path : Yogya AndisRent ApartemenPark BhinekaShuttle
Cost : 1.0887118742530162 km
AStar Algorithm



Start : Yogya
Goal : BhinekaShuttle
Path : Yogya AndisRent ApartemenPark BhinekaShuttle
Cost : 1.0887118742530162 km
UCS Algorithm

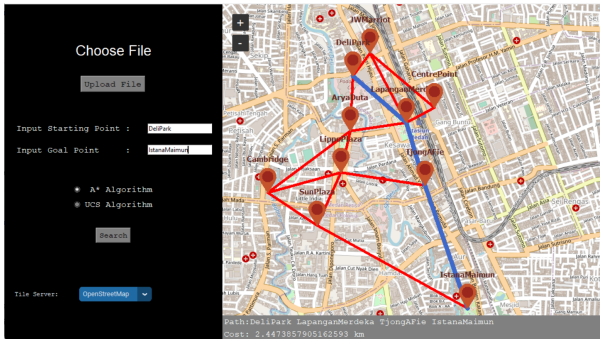
3.4 Peta jalan sebuah kawasan di kota asalmu

Peta/Graf Input	Hasil

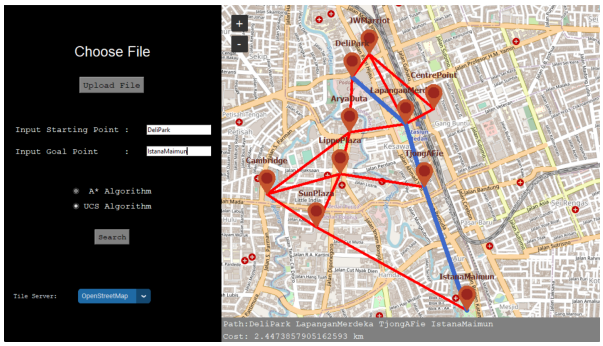


test > Origin.txt

```
1   10
2   LapanganMerdeka 3.590725613100053 98.67878734462185
3   JWMarriot 3.5964797900495675 98.67579814091495
4   DeliPark 3.5945891978352935 98.67437740020448
5   CentrePoint 3.591974265857527 98.68113459560803
6   IstanaMaimun 3.575394092772645 98.68387318003592
7   SunPlaza 3.5822487785088115 98.67143287084477
8   AryaDuta 3.5900592733098318 98.6741526949407
9   TjongAFie 3.5855517397198935 98.68033757012518
10  Cambridge 3.5849646914965847 98.66736085550018
11  LippoPlaza 3.586643938200874 98.67339601317252
12  0 1 1 1 0 0 1 1 0 0
13  1 0 1 1 0 0 0 0 0 0
14  1 1 0 0 0 0 1 0 0 0
15  1 1 0 0 0 0 0 0 0 0
16  0 0 0 0 0 1 0 1 0 0
17  0 0 0 0 1 0 0 0 1 1
18  1 0 1 0 0 0 0 0 1 1
19  1 0 0 0 1 0 0 0 0 1
20  0 0 0 0 0 1 1 0 0 1
21  0 0 0 0 0 1 1 1 1 0
```



Start : DeliPark
Goal : IstanaMaimun
Path : DeliPark LapanganMerdeka TjongAFie IstanaMaimun
Cost : 2.4473857905162593 km
AStar Algorithm



Start : DeliPark
Goal : IstanaMaimun
Path : DeliPark LapanganMerdeka TjongAFie IstanaMaimun
Cost : 2.4473857905162593 km
UCS Algorithm

BAB IV
PENUTUP

4.1 Kesimpulan

Melalui Tugas Kecil 3 IF2211 Strategi Algoritma, kami berhasil membuat sebuah program untuk mencari lintasan terpendek dari suatu titik ke titik lain dengan menggunakan algoritma A* dan algoritma UCS. Untuk keberjalanan program, kami memanfaatkan priority queue di kedua algoritma utama tersebut. Selain itu, kami juga menggunakan rumus jarak euclidean berdasarkan koordinat peta untuk menghitung jarak suatu titik ke titik lainnya.

Dengan membandingkan kedua algoritma yang telah dibuat, dapat dilihat bahwa algoritma A* dan algoritma UCS sama - sama dapat menemukan jarak lintasan terpendek dari suatu titik ke titik lainnya. Akan tetapi, algoritma A* lebih teroptimasi dengan menggunakan heuristik dari suatu titik ke tujuan atau h(n) sehingga algoritma A* cenderung lebih cepat dalam mencari jalur optimal jika dibandingkan algoritma UCS.

DAFTAR REFERENSI

1. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2022-2023/Tucil3-Stima-2023.pdf
2. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Route-Planning-Bagian1-2021.pdf
3. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Route-Planning-Bagian2-2021.pdf

LAMPIRAN

Lampiran 1: Tautan Repositori GitHub

Tautan repositori GitHub Tugas Kecil 3 IF2211 Strategi Algoritma:

https://github.com/Raylouiss/Tucil3_13521054_13521143

Lampiran 2: Tabel Penilaian

Poin	Ya	Tidak
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		