

Laporan Tugas Kecil 3 IF2211 Strategi Algoritma
Implementasi Algoritma UCS dan A* untuk Menentukan Lintasan Terpendek



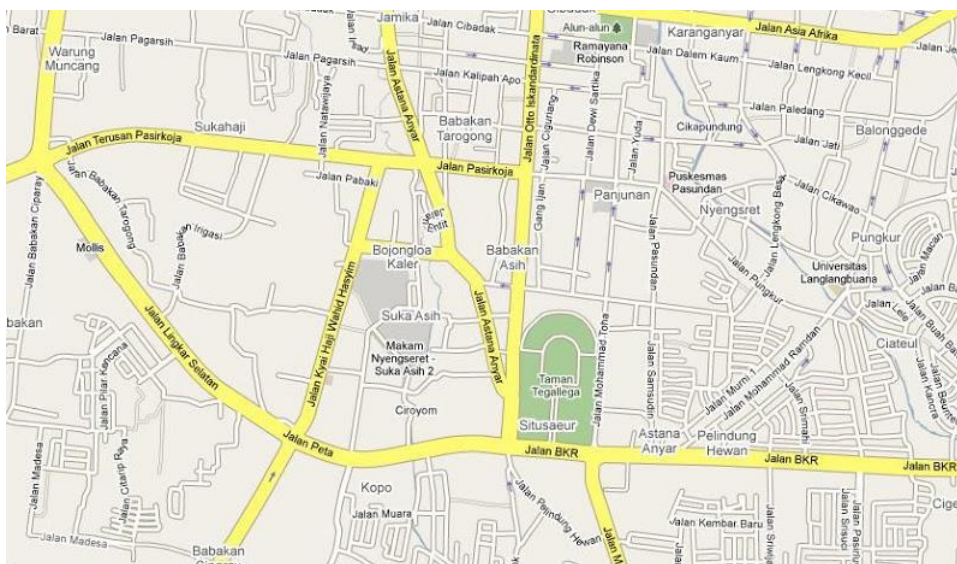
Disusun oleh:
Ahmad Ghulam Ilham (13521118)
Muhammad Naufal Nalendra (13521152)

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
Tahun 2022/2023

BAB I

Deskripsi Tugas

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 antar 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.

Spesifikasi program:

1. Program menerima input file graf (direpresentasikan sebagai matriks ketetanggaan berbobot), jumlah simpul minimal 8 buah.
2. Program dapat menampilkan peta/graf
3. Program menerima input simpul asal dan simpul tujuan.
4. Program dapat menampilkan lintasan terpendek beserta jaraknya antara simpul asal dan simpul tujuan.
5. Antarmuka program bebas, apakah pakai GUI atau command line saja.

BAB II

Landasan Teori

1) Uniform Cost Search (UCS)

Uniform-Cost Search merupakan algoritma pencarian tanpa informasi (uninformed search) yang menggunakan biaya kumulatif terendah untuk menemukan jalur dari node sumber ke node tujuan.

Algoritma ini beroperasi di sekitar ruang pencarian berbobot terarah untuk berpindah dari node awal ke salah satu node akhir dengan biaya akumulasi minimum.

Algoritma Uniform-Cost Search masuk dalam algoritma pencarian uninformed search atau blind search karena bekerja dengan cara brute force, yaitu tidak mempertimbangkan keadaan node atau ruang pencarian.

Cara Kerja:

1. Masukkan node root ke dalam priority queue
2. Ulangi langkah berikut saat antrian (queue) tidak kosong:
 - Hapus elemen dengan prioritas tertinggi
 - Jika node yang dihapus adalah node tujuan, cetak total biaya (cost) dan hentikan algoritma
 - Jika tidak, enqueue semua child dari node saat ini ke priority queue, dengan biaya kumulatifnya dari root sebagai prioritas

Disini node root adalah node awal untuk jalur pencarian, dan priority queue tetap untuk mempertahankan jalur dengan biaya paling rendah untuk dipilih pada traversal berikutnya. Jika 2 jalur memiliki biaya traversal yang sama, node diurutkan berdasarkan abjad.

Time complexity pada algoritma uniform cost search dapat dirumuskan:

$$O(b(1 + C / \epsilon))$$

Dimana:

- b - branching factor
- C - biaya optimal
- ϵ - biaya setiap langkah

2) A* (A star)

Pengertian:

Idea: hindari memperluas jalur yang sudah mahal

Fungsi evaluasi $f(n) = g(n) + h(n)$

- $g(n)$ = biaya sampai saat ini untuk mencapai n
- $h(n)$ = perkiraan biaya dari n ke tujuan
- $f(n)$ = perkiraan total biaya jalur melalui n ke tujuan

Jika $f(n) = g(n) \rightarrow$ Uniform Cost Search (UCS)

Jika $f(n) = h(n) \rightarrow$ Greedy Best First Search

Jika $f(n) = g(n) + h(n) \rightarrow A^*$

A* Special

Tujuan: mencari jalur dengan biaya minimum

Notasi:

- $c(n, n')$ adalah biaya busur (n, n')
- $g(n)$ = biaya jalur saat ini dari awal ke simpul n dalam pohon pencarian.
- $h(n)$ = perkiraan biaya termurah dari jalur dari n ke tujuan.
- Fungsi evaluasi khusus: $f = g + h$
- $f(n)$ memperkirakan jalur solusi dengan biaya termurah yang melewati n .
- $h^*(n)$ adalah biaya termurah sebenarnya dari n ke tujuan.
- $g^*(n)$ adalah jalur terpendek sebenarnya dari awal s , ke n .
- Jika fungsi heuristik, h selalu memperkirakan biaya yang lebih rendah dari biaya sebenarnya ($h(n)$ lebih kecil dari $h^*(n)$), maka A* dijamin akan menemukan solusi optimal \rightarrow dapat diterima; dan juga harus konsisten.

Sifat-sifat A*

- Lengkap? Ya, kecuali jika ada tak terhingga banyaknya simpul dengan $f \leq f(G)$
- Waktu? Eksponensial: $O(b^m)$
- Ruang? Simpan semua simpul dalam memori: $O(b^m)$
- Optimal? Ya

BAB III

Aplikasi Algoritma UCS dan A*

Algoritma UCS dan A* yang digunakan pada implementasi program tucil ini adalah sebagai berikut:

Input:

1. Mendefinisikan matriks ketetanggaan
2. Mendefinisikan simpul awal dan simpul tujuan

Algoritma UCS dan A* (bagian utama):

3. Khusus algoritma A*, terlebih dahulu dilakukan perhitungan fungsi heuristic berdasarkan matriks ketetanggaan dan simpul tujuan. Fungsi heuristic dibagi menjadi 2, yaitu jika koordinat titik diketahui dan tidak diketahui. Jika koordinat diketahui, fungsi heuristic berisi Euclidean distance menggunakan pendekatan haversine (karena koordinat berupa latitude dan longitude) dari semua simpul pada matriks ketetanggaan ke simpul tujuan. Jika koordinat tidak diketahui, fungsi heuristic berisi jarak terpendek yang diperoleh dari hasil algoritma UCS dari semua simpul pada matriks ketetanggaan ke simpul tujuan.
4. Inisialisasi banyak iterasi dengan 0, sebuah variable bernama visited bertipe data set (himpunan) kosong, dan variable bernama queue bertipe data heapq (priority queue) berisi PrioritizedItem(0, start, []), yang merepresentasikan cost masih 0, node adalah simpul awal, dan path kosong
5. Selama node belum mencapai simpul tujuan, banyak iterasi akan bertambah 1, akan dibuat variable item yang di assign dengan nilai PrioritizedItem hasil heappop dari queue. Kemudian, didefinisikan current cost, current node, dan current path berdasarkan item yang baru saja dipop dari queue
6. Jika current node belum berada di himpunan visited, node akan 'ditandai' dan dimasukkan ke dalam visited. Bagian ini mengubah simpul hidup menjadi simpul ekspan
7. Next path akan dibuat dengan menambahkan current node pada current path. Jika current node sudah mencapai simpul tujuan, algoritma akan mengembalikan banyak iterasi, jarak terdekat, dan rute terpendek
8. Jika current node belum mencapai simpul tujuan, algoritma akan mencari simpul-simpul tetangga dari current node yang belum terdapat pada himpunan visited (belum diekspan)
9. Pada algoritma UCS, untuk setiap simpul tetangga akan dihitung next cost berdasarkan current cost dan nilai pada matriks ketetanggaan. Bagian ini mengimplementasikan rumus UCS sebagai $f(n) = g(n)$
10. Pada algoritma A*, untuk setiap simpul tetangga akan dihitung next cost berdasarkan current cost, nilai pada matriks ketetanggaan, dan nilai pada fungsi heuristic. Bagian ini mengimplementasikan rumus A* sebagai $f(n) = g(n) + h(n)$

11. Kemudian, tuple next cost, next node (simpul tetangga), dan next path akan ditambahkan pada queue dengan menggunakan heappush sehingga queue akan tetap teratur menaik berdasarkan nilai cost (jarak tempuh)
12. Selama queue tidak kosong, langkah 4 s.d. 10 akan terus dilakukan. Jika rute ditemukan, maka algoritma UCS dan A* akan melakukan return hasil dan berhenti. Jika tidak ditemukan rute yang menghubungkan simpul awal ke simpul tujuan, algoritma akan mengembalikan banyak iterasi, jarak tak hingga, dan rute kosong

Output:

13. Menampilkan informasi penting terkait hasil algoritma
14. Menampilkan visualisasi graf atau peta beserta rute terpendek

Untuk memenuhi algoritma UCS tersebut, berikut adalah tipe data tambahan, fungsi, dan prosedur yang digunakan dalam tucil ini.

Tipe data tambahan:

Nama	Atribut	Deskripsi
PrioritizedItem	Cost: float Node: int Path: str[]	Tipe data PrioritizedItem digunakan sebagai masukan heapq (priority queue). Tipe data ini berfungsi untuk mempertahankan urutan pop heapq jika dua buah item memiliki cost yang sama.

Tabel 3.1 Daftar tipe data dalam program

Fungsi:

Nama	Parameter Input	Parameter Output	Deskripsi
heuristic	Int[][] Int Str[]	Float[]	Mencari nilai heuristik dari setiap simpul dengan UCS (koordinat simpul tidak diketahui)
haversine	Int[][] Graph Int	Float[]	Mencari nilai heuristik dari setiap simpul dengan haversine (koordinat tiap simpul diketahui)
astar	Int[][] Int Int Str[] Float[]	Int Float Str[]	Mencari lintasan terpendek menggunakan A*
ucs	Int[][] Int Int	Int Float Str[]	Mencari lintasan terpendek menggunakan UCS

	Str[]		
inputMethod	-	Boolean	Mendapatkan metode input pilihan user
inputFile	-	Str[] Int[][]	Mendapatkan matriks ketetanggaan berdasarkan file masukan pengguna
inputMap	-	(Float, Float) Int Graph Str[] Int[][] Float[]	Mendapatkan matriks ketetanggaan dan graf berdasarkan koordinat peta masukan pengguna
inputNode	Str[]	Int Int	Mendapatkan input pilihan simpul awal dan simpul tujuan
inputPoint	Graph Float[]	Int Int	Mendapatkan koordinat titik awal dan titik tujuan

Tabel 3.2 Daftar fungsi dalam program

Prosedur:

Nama	Parameter Input	Deskripsi
plot	Graph Str[] Str[]	Membuat visualisasi graf dan menunjukkan lintasan terpendek berdasarkan matriks ketetanggaan dan hasil algoritma UCS atau A*
map	Graph (Float, Float) Str[] Str	Memvisualisasi graf pada sebuah peta dan menunjukkan lintasan terpendek berdasarkan matriks ketetanggaan dan hasil algoritma UCS atau A*
result	Str Str[] Int Int Int Float Str[] Float	Menampilkan informasi penting terkait hasil algoritma UCS dan A*. Informasi tersebut adalah: Rute yang ditempuh dari simpul awal menuju simpul tujuan Jarak terpendek yang diperoleh algoritma Banyak iterasi (langkah) yang dilakukan algoritma Lama waktu yang diperlukan algoritma

Tabel 3.3 Daftar prosedur dalam program

BAB IV

Implementasi dan Pengujian

Implementasi

- input.py

```
You, 42 minutes ago | 2 authors (You and others)
1 import os
2 import os.path as path
3 import networkx as nx
4 import numbers
5
6 # function to get the input method from the user
7 def inputMethod():
8     # get the input method
9     print("Valid Input Method:")
10    print("1. File Input")
11    print("2. Map Coordinates Input")
12    while True:
13        choice = input("Choose the input method: ")
14        if choice == "1":
15            isFile = True
16            break
17        elif choice == "2":
18            isFile = False
19            break
20        else:
21            print("Enter a valid input method")
22    # if the input is valid, return the input method
23    return isFile
24
25 # function to get input file from the user
26 def inputFile():
27    print("-----")
28    while True:
29        # opening input file
30        filename = input("Enter the name of the input file :\n")
31        filepath = os.path.join('..\Tucil3_13521118_13521152\test', filename)
32
33        try:
34            with open(filepath, 'r') as f:
35                # reading the name of nodes
36                line = f.readline()
37                name = [str(x) for x in line.strip().split(', ')]
38                # get the number of nodes
39                size = len(name)
40                # check if the number of nodes is at least 8
41                if size < 8:
42                    raise ValueError("Input file must contain at least 8 nodes.")
43
44                # reading the adjacency matrix
45                matrix = []
46                for i in range(size):
47                    line = f.readline()
48                    row = line.strip().split()
49                    # check if row length is valid
50                    if len(row) != size:
51                        raise ValueError("The adjacency matrix is not rectangular.")
52                    # check if all elements are non-negative integers
53                    try:
54                        row = [int(element) for element in row]
55                        if any(element < 0 for element in row):
56                            raise ValueError("Elements of the adjacency matrix must be non-negative integers.")
57                    except ValueError:
58                        raise ValueError("File contains non-integer elements.")
59                    matrix.append(row)
60                # if the matrix is valid, return the name and matrix
61                return name, matrix
62    except (FileNotFoundError, ValueError) as e:
63        print(f"Error: {e}")
64        continue
65
66 # function to get input map from the user
67 def inputMap():
68    print("-----")
69    valid = False
```



```

69 valid = False
70 while not valid:
71     # get the center point of the area
72     print("Valid latitude range: -90 to 90 (in degrees)")
73     while True:
74         try:
75             lat = float(input('Enter the latitude of the center point: '))
76             if lat < -90 or lat > 90:
77                 print("Enter a valid latitude")
78                 continue
79             break
80         except ValueError:
81             print("Enter a number (in degrees) for latitude")
82     print("Valid longitude range: -180 to 180 (in degrees)")
83     while True:
84         try:
85             lon = float(input('Enter the longitude of the center point: '))
86             if lon < -180 or lon > 180:
87                 print("Enter a valid longitude")
88                 continue
89             break
90         except ValueError:
91             print("Enter a number (in degrees) for longitude")
92     center = (lat, lon)
93
94     # get the radius of the area
95     print("Valid radius range: 500 to 5000 (in meters)")
96     while True:
97         try:
98             radius = int(input('Enter the radius of the area: '))
99             if radius < 500 or radius > 5000:
100                 print("Enter a valid radius")
101                 continue
102             break
103         except ValueError:
104             print("Enter a number for radius")
105
106     try:
107         # make a graph from the center point and radius
108         graph = ox.graph_from_point(center, dist=radius, network_type='drive')
109         graph = nx.relabel.convert_node_labels_to_integers(graph)
110
111         # get the name of the nodes
112         name = []
113         for node in graph.nodes():
114             name.append(node)
115
116         # convert graph to adjacency matrix
117         adj_matrix = nx.to_numpy_array(graph, weight='length')
118         # get the bounding box of the graph
119         bbox = ox.utils_geo.bbox_from_point(center, dist=radius, project_utm=False)
120
121         # If all inputs are valid, return the center point, radius, graph, name, adjacency matrix, and bounding box
122         valid = True
123         return center, radius, graph, name, adj_matrix, bbox
124     # if there are no nodes in the graph, print the error and ask for new center point and radius
125     except ValueError as e:
126         print(f"Error: {e}")
127         print("Enter a new center point and radius")
128
129 # function to get input node from the user
130 def inputNode(name: list):
131     print("-----")
132     print("List of valid nodes:")
133     for i in range(len(name)):
134         print(f"{i+1}. {name[i]}")
135     # get the starting node
136     while True:
137         try:
138             inputNode = int(input("Choose the starting node : "))
139             inputNode = int(input("Choose the starting node : "))
140             if inputNode < 1 or inputNode > len(name) - 1:
141                 print("Enter a valid starting node")
142                 continue
143             startNode = inputNode - 1
144             break
145         except ValueError:
146             print("Enter the number (integer) of the node")
147     print("-----")
148     print("List of valid nodes:")
149     for i in range(len(name)):
150         if i < startNode:
151             print(f"{i+1}. {name[i]}")
152         elif i > startNode:
153             print(f"{i}. {name[i]}")
154     # get the destination node
155     while True:
156         try:
157             inputNode = int(input("Choose the destination node : "))
158             if inputNode < 1 or inputNode > len(name) - 1:
159                 print("Enter a valid destination node")
160                 continue
161             if inputNode - 1 < startNode:
162                 endNode = inputNode - 1
163             else:
164                 endNode = inputNode
165             break
166         except ValueError:
167             print("Enter the number (integer) of the node")
168     # if the input is valid, return the starting and destination node
169     return startNode, endNode
170
171 # function to get input point from the user
172 def inputPoint(graph, bbox):
173     print("-----")
174     # get the starting point

```

```

172 print("-----")
173 # get the starting point
174 print("Valid latitude range:", bbox[1], "to", bbox[0], "(in degrees)")
175 while True:
176     try:
177         startlat = float(input('Enter the latitude of the start point: '))
178         if not isinstance(startlat, numbers.Number):
179             raise ValueError()
180         if startlat < bbox[1] or startlat > bbox[0]:
181             print("Enter a valid start point latitude")
182         else:
183             break
184     except ValueError:
185         print("Enter a number (in degrees) for latitude")
186 print("Valid longitude range:", bbox[3], "to", bbox[2], "(in degrees)")
187 while True:
188     try:
189         startlon = float(input('Enter the longitude of the start point: '))
190         if not isinstance(startlon, numbers.Number):
191             raise ValueError()
192         if startlon < bbox[3] or startlon > bbox[2]:
193             print("Enter a valid start point longitude")
194         else:
195             break
196     except ValueError:
197         print("Enter a number (in degrees) for longitude")
198
199 print("-----")
200 # get the destination point
201 print("Valid latitude range:", bbox[1], "to", bbox[0], "(in degrees)")
202 while True:
203     try:
204         endlat = float(input('Enter the latitude of the destination point: '))
205         if not isinstance(endlat, numbers.Number):
206             raise ValueError()
207         if endlat < bbox[1] or endlat > bbox[0]:
208             print("Enter a valid destination point latitude")
209         else:
210             break
211     except ValueError:
212         print("Enter a number (in degrees) for latitude")
213 print("Valid longitude range:", bbox[3], "to", bbox[2], "(in degrees)")
214 while True:
215     try:
216         endlon = float(input('Enter the longitude of the destination point: '))
217         if not isinstance(endlon, numbers.Number):
218             raise ValueError()
219         if endlon < bbox[3] or endlon > bbox[2]:
220             print("Enter a valid destination point longitude")
221         else:
222             break
223     except ValueError:
224         print("Enter a number (in degrees) for longitude")
225
226 # define the starting and destination node
227 startNode = ox.nearest_nodes(graph, startlon, startlat)
228 endNode = ox.nearest_nodes(graph, endlon, endlat)
229
230 # if the input is valid, return the starting and destination node
231 return startNode, endNode
232

```

- prioitem.py

```

src > lib > prioitem.py > ...
You, 2 seconds ago | 1 author (You)
1 from dataclasses import dataclass, field
2 from typing import Any
3
You, 2 seconds ago | 1 author (You)
4 @dataclass(order=True)
5 class PrioritizedItem:
6     cost: int
7     node: Any = field(compare=False)
8     path: Any = field(compare=False)
9

```

- ucs.py

```

You, 16 hours ago | 2 authors (Naufal-Nalendra and others)
1 from heapq import heappush, heappop
2 from lib.prioitem import PrioritizedItem
3
4 # Calculate the shortest path using UCS
5 def ucs(adj_matrix, start, dest, name):
6     iteration = 0
7     visited = set()
8     queue = [PrioritizedItem(0, start, [])]
9     while queue:
10         iteration += 1
11         item = heappop(queue)
12         cost, node, path = item.cost, item.node, item.path
13         if node not in visited:
14             visited.add(node)
15             path = path + [name[node]]
16             if node == dest:
17                 # if shortest path is found, return the iteration count, cost, and path
18                 return (iteration, cost, path)
19             for neighbor in range(len(adj_matrix[node])):
20                 if adj_matrix[node][neighbor] != 0 and neighbor not in visited:
21                     # Calculate the actual cost f(n) = g(n)
22                     actual_cost = cost + adj_matrix[node][neighbor]
23                     heappush(queue, PrioritizedItem(actual_cost, neighbor, path))
24 # if no path is found, return the iteration count, infinity, and empty path
25 return (iteration, float("inf"), [])
26

```

- astar.py

```

You, 16 hours ago | 1 author (You)
1 from heapq import heappush, heappop
2 from lib.prioitem import PrioritizedItem
3 from lib.ucs import ucs
4 from math import radians, cos, sqrt
5
6 EARTH_RADIUS = 6371000
7
8 # Calculate the heuristic value for each node using UCS
9 def heuristic(adj_matrix, dest, name):
10     h = [0] * len(adj_matrix)
11     for node in range(len(adj_matrix)):
12         iteration, cost, path = ucs(adj_matrix, node, dest, name)
13         h[node] = cost
14     return h
15
16 # Calculate the heuristic value for each node using Haversine
17 def haversine(adj_matrix, graph, destination_node):
18     h = [0] * len(adj_matrix)
19     dest_lat = radians(graph.nodes[destination_node]['y'])
20     dest_lon = radians(graph.nodes[destination_node]['x'])
21     for u in graph.nodes():
22         lat1 = radians(graph.nodes[u]['y'])
23         lon1 = radians(graph.nodes[u]['x'])
24         x = (lon1 - dest_lon) * cos(0.5 * (lat1 + dest_lat))
25         y = lat1 - dest_lat
26         euclidean_distance = EARTH_RADIUS * sqrt(x * x + y * y)
27         h[u] = euclidean_distance
28     return h
29
30 # Calculate the shortest path using A*
31 def astar(adj_matrix, start, dest, name, heuristic):
32     iteration = 0
33     visited = set()
34     queue = [PrioritizedItem(0, start, [])]
35     while queue:
36

```

```

35 while queue:
36     iteration += 1
37     item = heappop(queue)
38     cost, node, path = item.cost, item.node, item.path
39     if node not in visited:
40         visited.add(node)
41         path = path + [name[node]]
42         if node == dest:
43             # if shortest path is found, return the iteration count, cost, and path
44             return (iteration, cost, path)
45         if (node != start):
46             cost -= heuristic[node]
47         for neighbor in range(len(adj_matrix[node])):
48             if adj_matrix[node][neighbor] != 0 and neighbor not in visited:
49                 # Calculate the actual cost f(n) = g(n) + h(n)
50                 actual_cost = cost + adj_matrix[node][neighbor] + heuristic[neighbor]
51                 heappush(queue, PrioritizedItem(actual_cost, neighbor, path))
52 # if no path is found, return the iteration count, infinity, and empty path
53 return (iteration, float("inf"), [])
54

```

- output.py

```

You, 1 hour ago | 1 author (You)
1 import networkx as nx      You, 5 days ago • feat: add plot weighted graph ...
2 import folium
3 import osmnx as ox
4 import os
5 import matplotlib.pyplot as plt
6
7 # Plot the graph and highlight the shortest path
8 def plot(graph, name, path):
9     # Define the vertex labels
10    labels = {k: v for k, v in enumerate(name)}
11
12    # Convert the adjacency matrix to a weighted graph
13    G = nx.Graph()
14    for i in range(len(graph)):
15        for j in range(i+1, len(graph[i])):
16            if graph[i][j] != 0:
17                G.add_edge(labels[i], labels[j], weight=graph[i][j])
18
19    # Plot the weighted graph with the shortest path highlighted
20    pos = nx.spring_layout(G)
21    nx.draw(G, pos, with_labels=True, font_weight='bold')
22    edge_labels = nx.get_edge_attributes(G, 'weight')
23    nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels, font_weight='bold')
24    edge_colors = ['b' if (path[i], path[i+1]) in nx.edges(G) else 'k' for i in range(len(path)-1)]
25    nx.draw_networkx_edges(G, pos, edgelist=[(path[i], path[i+1]) for i in range(len(path)-1)], edge_color='r', width=4)
26    nx.draw_networkx_edges(G, pos, edge_color=edge_colors)
27    plt.show()
28
29 # Plot the graph on a map and highlight the shortest path
30 def map(graph, center, route, algorithm):
31     filename = f'route-{algorithm}.html'
32     filepath = os.path.join('..\\Tucil3_13521118_13521152\\test', filename)
33     m = folium.Map(Location=[center[0], center[1]], zoom_start=12)
34     ox.plot_route_folium(graph, route, route_map=m, line_color='blue', line_weight=5)
35     # Save the map to an HTML file
36     m.save(filepath)
37
38 # Print the result of the algorithm
39 def result(function, name, start, end, iteration, cost, path, time):
40     print(f"-----")
41     print(f"Algorithm: {function}")
42     print(f"Start node: {name[start]}")
43     print(f"End node: {name[end]}")
44     print(f"Shortest path: {path}")
45     print(f"Shortest distance: {cost}")
46     print(f"Iteration count: {iteration}")
47     print(f"Elapsed time: {time} milliseconds")
48

```

- program.py

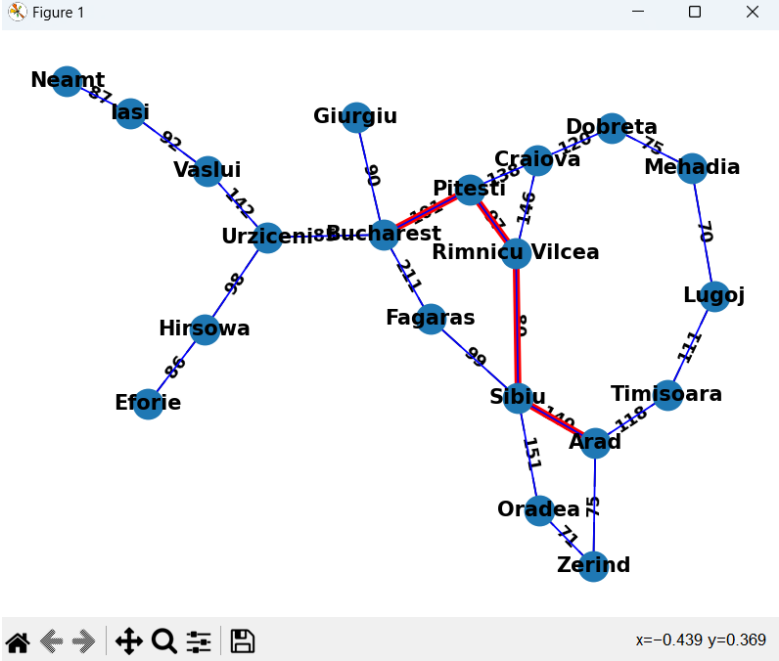
```

1  You, 1 hour ago • You, 8 hours ago • chore: update some error me
2  from lib. (module) output tMethod, inputFile, inputMap, inputNode, inputPoint
3  from lib.output import plot, map, result
4  from lib.ucs import ucs
5  from lib.astar import heuristic, haversine, astar
6  from timeit import timeit
7
8  # Get the input method
9  isFile = inputMethod()
10
11 if (isFile):
12     # Read the input file
13     name, adj_matrix = inputFile()
14     # Plot the graph
15     path = []
16     plot(adj_matrix, name, path)
17     # Get the start and end node
18     start, end = inputNode(name)
19 else:
20     # Read the map coordinates
21     center, radius, graph, name, adj_matrix, bbox = inputMap()
22     # Get the start and end point
23     start, end = inputPoint(graph, bbox)
24
25 # Calculate the shortest path using UCS
26 ucs_iteration, ucs_cost, ucs_path = ucs(adj_matrix, start, end, name)
27 ucs_time = timeit(Lambda: ucs(adj_matrix, start, end, name), number=1) * 1000
28 result("UCS", name, start, end, ucs_iteration, ucs_cost, ucs_path, ucs_time)
29 if (len(ucs_path) == 0):
30     print("No path found")
31 else:
32     if (isFile):
33         plot(adj_matrix, name, ucs_path)
34     else:
35         map(graph, center, ucs_path, "UCS")
36
37 # Calculate the shortest path using A*
38 if (isFile):
39     hfunc = heuristic(adj_matrix, end, name)
40 else:
41     hfunc = haversine(adj_matrix, graph, end)
42 astar_iteration, astar_cost, astar_path = astar(adj_matrix, start, end, name, hfunc)
43 astar_time = timeit(Lambda: astar(adj_matrix, start, end, name, hfunc), number=1) * 1000
44 result("A*", name, start, end, astar_iteration, astar_cost, astar_path, astar_time)
45 if (len(astar_path) == 0):
46     print("No path found")
47 else:
48     if (isFile):
49         plot(adj_matrix, name, astar_path)
50     else:
51         map(graph, center, astar_path, "A-star")

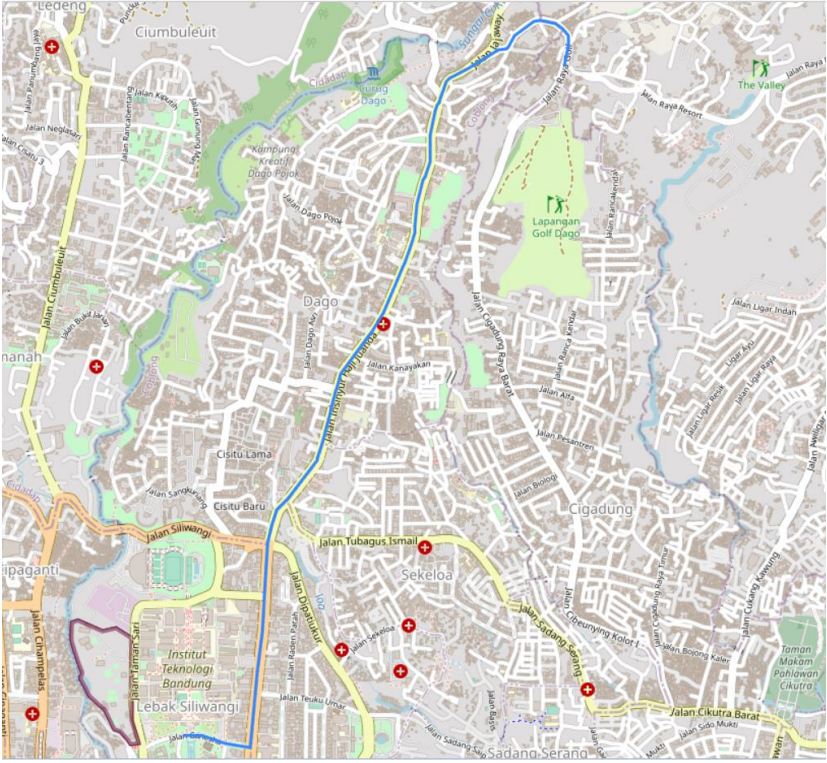
```

Hasil Uji


- Test Case 1 – Input file graph2.txt pada folder test

Keterangan	Screenshoot
Input (masukan)	<pre> 1 Arad, Bucharest, Craiova, Dobreta, Eforie, Fagaras, Giurgiu, Hirsowa, Iasi, Lugoj, 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 140 118 0 0 75 3 0 0 0 0 0 211 90 0 0 0 0 0 0 101 0 0 0 85 0 0 4 0 0 0 120 0 0 0 0 0 0 0 0 0 138 146 0 0 0 0 0 5 0 0 120 0 0 0 0 0 0 75 0 0 0 0 0 0 0 0 0 0 6 0 0 0 0 0 0 86 0 0 0 0 0 0 0 0 0 0 0 0 0 7 0 211 0 0 0 0 0 0 0 0 0 0 0 0 0 99 0 0 0 0 8 0 90 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 9 0 0 0 86 0 0 0 0 0 0 0 0 0 0 0 0 0 98 0 0 10 0 0 0 0 0 0 0 0 0 0 87 0 0 0 0 0 0 0 92 0 11 0 0 0 0 0 0 0 0 0 70 0 0 0 0 0 111 0 0 0 0 12 0 0 0 75 0 0 0 0 70 0 0 0 0 0 0 0 0 0 0 0 13 0 0 0 0 0 0 0 87 0 0 0 0 0 0 0 0 0 0 0 0 14 0 0 0 0 0 0 0 0 0 0 0 0 0 151 0 0 0 0 71 15 0 101 138 0 0 0 0 0 0 0 97 0 0 0 0 0 0 0 0 16 0 0 146 0 0 0 0 0 0 0 97 0 80 0 0 0 0 0 0 17 140 0 0 0 99 0 0 0 0 0 151 80 0 0 0 0 0 0 18 118 0 0 0 0 0 111 0 0 0 0 0 0 0 0 0 0 0 19 0 85 0 0 0 0 98 0 0 0 0 0 0 0 0 0 142 0 20 0 0 0 0 0 0 92 0 0 0 0 0 0 0 0 142 0 0 21 75 0 0 0 0 0 0 0 0 71 0 0 0 0 0 0 0 0 22 </pre>
Output (hasil)	<pre> ----- Algorithm: UCS Start node: Arad End node: Bucharest Shortest path: ['Arad', 'Sibiu', 'Rimnicu Vilcea', 'Pitesti', 'Bucharest'] Shortest distance: 418 Iteration count: 14 Elapsed time: 0.031000003218650818 milliseconds ----- Algorithm: A* Start node: Arad End node: Bucharest Shortest path: ['Arad', 'Sibiu', 'Rimnicu Vilcea', 'Pitesti', 'Bucharest'] Shortest distance: 418 Iteration count: 5 Elapsed time: 0.017300000763498247 milliseconds </pre> 
<p>Penjelasan: File input berisi 20 simpul. Algoritma UCS dan A* menghasilkan jarak dan rute yang sama, tetapi memiliki jumlah iterasi yang berbeda</p>	

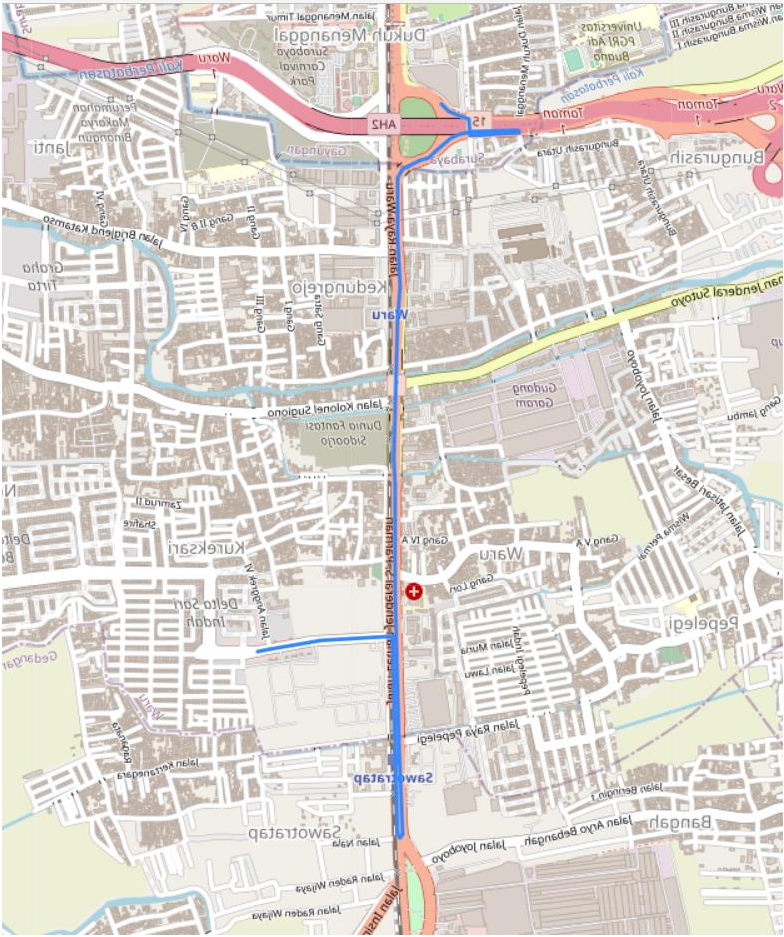
- Test Case 2 – Input koordinat peta jalan sekitar kampus ITB/Dago/Bandung Utara

Keterangan	Screenshoot
Input (masukan)	<pre> Valid latitude range: -90 to 90 (in degrees) Enter the latitude of the center point: -6.89044875 Valid longitude range: -180 to 180 (in degrees) Enter the longitude of the center point: 107.61031355794628 Valid radius range: 500 to 5000 (in meters) Enter the radius of the area: 5000 Valid latitude range: -6.935414766774644 to -6.845482733225356 (in degrees) Enter the latitude of the start point: -6.8933348 Valid longitude range: 107.5650204048073 to 107.65560671108527 (in degrees) Enter the longitude of the start point: 107.6104652 Valid latitude range: -6.935414766774644 to -6.845482733225356 (in degrees) Enter the latitude of the destination point: -6.84148685 Enter a valid destination point latitude Enter the latitude of the destination point: -6.865147 Valid longitude range: 107.5650204048073 to 107.65560671108527 (in degrees) Enter the longitude of the destination point: 107.6262249 </pre>
Output (hasil)	<pre> Algorithm: UCS Start node: 547 End node: 10785 Shortest path: [547, 543, 4396, 1223, 11812, 1226, 14850, 14853, 1202, 5208, 1205, 5273, 1208, 2618, 2877, 8674, 4531, 14372, 1200, 1211, 1221, 5269, 1201, 2570, 1204, 1216, 5285, 5284, 1209, 1220, 1207, 1222, 4385, 14302, 1203, 4383, 12363, 621, 1906, 620, 619, 8863, 8861, 618, 1808, 8859, 10785] Shortest distance: 4457.239000000001 Iteration count: 5156 Elapsed time: 15947.04510000156 milliseconds Algorithm: A* Start node: 547 End node: 10785 Shortest path: [547, 543, 4396, 1223, 11812, 1226, 14850, 14853, 1202, 5208, 1205, 5273, 1208, 2618, 2877, 8674, 4531, 14372, 1200, 1211, 1221, 5269, 1201, 2570, 1204, 1216, 5285, 5284, 1209, 1220, 1207, 1222, 4385, 14302, 1203, 4383, 12363, 621, 1906, 620, 619, 8863, 8861, 618, 1808, 8859, 10785] Shortest distance: 4457.239 Iteration count: 511 Elapsed time: 1798.5517999986769 milliseconds </pre> <p>13521152/test/route-A-star.html</p> 
<p>Penjelasan: Koordinat titik pusat peta (-6.89044875, 107.61031355794628) merupakan Institut Teknologi Bandung. Koordinat titik awal (-6.8933348, 107.6104652) adalah Monumen Kubus. Koordinat titik tujuan (-6.865147, 107.6262249) adalah Dago Pakar. Kedua algoritma menghasilkan jarak dan rute yang sama, tetapi algoritma A* hanya memerlukan 511 langkah dan 1798.5517999986769 miliseconds dibandingkan algoritma UCS yang memerlukan 5156 langkah dan 15947.04510000156 miliseconds</p>	

- Test Case 3 – Input koordinat peta jalan sekitar kampus ITS (Surabaya)

Keterangan	Screenshoot
Input (masukan)	<pre> Valid latitude range: -90 to 90 (in degrees) Enter the latitude of the center point: -7.2826954 Valid longitude range: -180 to 180 (in degrees) Enter the longitude of the center point: 112.79567824449973 Valid radius range: 500 to 5000 (in meters) Enter the radius of the area: 5000 Valid latitude range: -7.327661416774644 to -7.237729383225355 (in degrees) Enter the latitude of the start point: -7.2826954 Valid longitude range: 112.75034652649688 to 112.84100996250257 (in degrees) Enter the longitude of the start point: 112.79567824449973 Valid latitude range: -7.327661416774644 to -7.237729383225355 (in degrees) Enter the latitude of the destination point: -7.26625925 Valid longitude range: 112.75034652649688 to 112.84100996250257 (in degrees) Enter the longitude of the destination point: 112.7839482168426 </pre>
Output (hasil)	<pre> Algorithm: UCS Start node: 4448 End node: 438 Shortest path: [4448, 602, 605, 604, 8633, 7987, 7988, 601, 4597, 603, 4615, 597, 680, 7214, 9348, 346, 402, 4626, 9823, 5404, 5693, 5534, 9617, 8630, 405, 33, 4, 385, 4533, 8479, 399, 5585, 1281, 1274, 228, 227, 234, 4535, 235, 236, 4573, 6427, 237, 238, 9264, 186, 177, 908, 259, 185, 833, 7392, 438] Shortest distance: 4771.616 Iteration count: 3736 Elapsed time: 7495.670899998408 milliseconds Algorithm: A* Start node: 4448 End node: 438 Shortest path: [4448, 602, 605, 604, 8633, 7987, 7988, 601, 4597, 603, 4615, 597, 680, 7214, 9348, 346, 402, 4626, 9823, 5404, 5693, 5534, 9617, 8630, 405, 33, 4, 385, 4533, 8479, 399, 5585, 1281, 1274, 228, 227, 234, 4535, 235, 236, 4573, 6427, 237, 238, 9264, 186, 177, 908, 259, 185, 833, 7392, 438] Shortest distance: 4771.616000000003 Iteration count: 456 Elapsed time: 1261.4784000041593 milliseconds </pre> <p>./test/route-A-star.html</p> 
<p>Penjelasan: Koordinat titik pusat peta dan titik simpul awal merupakan Institut Teknologi Sepuluh Nopember (ITS Surabaya). Koordinat titik tujuan adalah Universitas Airlangga (UNAIR). Kedua algoritma menghasilkan jarak dan rute yang sama.</p>	

- Test Case 4 – Input koordinat peta jalan di sekitar Sidoarjo

Keterangan	Screenshoot
Input (masukan)	<pre> Valid latitude range: -90 to 90 (in degrees) Enter the latitude of the center point: -7.34670285 Valid longitude range: -180 to 180 (in degrees) Enter the longitude of the center point: 112.73917433880675 Valid radius range: 500 to 5000 (in meters) Enter the radius of the area: 5000 Valid latitude range: -7.391668866774644 to -7.301736833225355 (in degrees) Enter the latitude of the start point: -7.3648783 Valid longitude range: 112.69383611975316 to 112.78451255786034 (in degrees) Enter the longitude of the start point: 112.7338581 Valid latitude range: -7.391668866774644 to -7.301736833225355 (in degrees) Enter the latitude of the destination point: -7.345515300000001 Valid longitude range: 112.69383611975316 to 112.78451255786034 (in degrees) Enter the longitude of the destination point: 112.727543922244 </pre>
Output (hasil)	<pre> Algorithm: UCS Start node: 2784 End node: 8340 Shortest path: [2784, 474, 473, 5783, 6228, 120, 121, 11067, 13201, 4436, 4373, 8274, 6130, 2590, 6134, 85, 4266, 126, 125, 46, 7918, 9579, 8347, 7919, 7500, 7501, 5772, 12004, 8340] Shortest distance: 4676.493000000001 Iteration count: 5708 Elapsed time: 16976.3127000002 milliseconds Algorithm: A* Start node: 2784 End node: 8340 Shortest path: [2784, 474, 473, 5783, 6228, 120, 121, 11067, 13201, 4436, 4373, 8274, 6130, 2590, 6134, 85, 4266, 126, 125, 46, 7918, 9579, 8347, 7919, 7500, 7501, 5772, 12004, 8340] Shortest distance: 4676.4929999999995 Iteration count: 1400 Elapsed time: 4228.569300001254 milliseconds </pre> 
<p>Penjelasan: Koordinat titik pusat peta adalah Perumahan Makarya Binangun, koodinat titik awal adalah Indomaret Delta Sari, koordinat titik tujuan adalah City of Tomorrow Mall. Kedua algoritma menghasilkan jarak dan rute yang sama.</p>	

BAB V

Kesimpulan

Komentar

Cukup seru, awalnya gak kebayang cara ngerjain bonus. Ternyata banyak library yang sangat membantu pengerjaan bonus. Jadi bisa explore.

Kesimpulan

Berdasarkan hasil pembuatan tugas dapat disimpulkan bahwa struktur data graph dan matriks ketetanggaan dapat digunakan untuk menggambarkan titik-titik pada peta. Hasil dari matriks dan graph yang didapat selanjutnya dapat dicari lintasan terpendek antar dua titik menggunakan algoritma UCS dan A*

Metode pencarian dengan menggunakan UCS dan A* dapat diterapkan pada pencarian lintasan terpendek antara dua simpul graph. Kedua algoritma dapat menemukan solusi dengan tepat dan membutuhkan waktu yang kurang lebih sama. Perbedaan dari kedua algoritma terlihat pada penelusuran simpul, dimana algoritma A* dapat mencari lintasan terpendek dengan langkah yang lebih sedikit.

Saran

Terdapat beberapa saran yang dapat kami berikan pada pembaca agar pengerjaan tugas dapat lebih baik :

1. Melakukan eksplorasi terhadap berbagai API yang dapat digunakan dalam visualisasi peta
2. Mencari gambaran dari bahasa pemrograman dan library yang ada agar dapat memilih variasi yang dirasa paling optimal

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	✓	

Tabel 5.1 Pemenuhan Spesifikasi

DAFTAR PUSTAKA

Referensi

- <https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2022-2023/Route-Planning-Bagian1-2021.pdf>
- <https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2022-2023/Route-Planning-Bagian2-2021.pdf>
- <https://www.trivusi.web.id/2022/10/apa-itu-algoritma-uniform-cost-search.html>

Repository GitHub

https://github.com/Agilham/Tucil3_13521118_13521152