

LAPORAN TUGAS KECIL 2
IF 2211 STRATEGI ALGORITMA
SEMESTER 2 2022 - 2023

Implementasi Algoritma UCS dan A* untuk Menentukan Lintasan
Terpendek



Disusun Oleh :
Ezra M C M H (13521073)
PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2023

1. Spesifikasi 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 GUI atau command line saja

2. Algoritma UCS dan A*

Algoritma UCS

Algoritma Uniform Cost Search (UCS) adalah algoritma pencarian jalur terpendek (shortest path) di dalam graf berbobot. Algoritma ini mencari jalur terpendek berdasarkan biaya atau jarak dari node awal ke setiap node yang terhubung dengannya

Algoritma UCS memulai pencarian dari node awal dan melangkah ke node tetangga yang memiliki biaya terendah. Jika ada dua atau lebih node dengan biaya yang sama, maka UCS akan memilih salah satu node tersebut secara acak. Algoritma akan terus melangkah ke node tetangga dengan biaya terendah hingga mencapai node tujuan atau mencapai seluruh node yang terhubung

Selama pencarian, algoritma UCS menyimpan daftar node yang sudah dikunjungi dan biaya terpendek dari node awal ke setiap node tersebut. Algoritma UCS juga menggunakan struktur data heap (priority queue) untuk mengurutkan node berdasarkan biaya terkecil.

Algoritma UCS dianggap sebagai salah satu algoritma pencarian jalur terpendek yang paling efisien, karena mengoptimalkan biaya perjalanan dan memastikan jalur terpendek secara optimal. Namun, algoritma ini memiliki kelemahan yaitu tidak efektif untuk graf dengan node dan edge karena dapat memakan waktu dan memori yang besar.

Secara umum, algoritma UCS sangat berguna untuk aplikasi dalam bidang kecerdasan buatan, pemetaan, navigasi, dan perencanaan rute

Berikut langkah - langkah algoritma Uniform Cost Search (UCS) :

1. Inisialisasi node awal dengan biaya 0 dan node-goal dengan biaya tak terhingga.
2. Buat priority queue yang berisi node awal.
3. Lakukan loop sampai priority queue kosong atau node-goal sudah tercapai:
 - a. Ambil node dengan biaya terkecil dari priority queue.
 - b. Jika node tersebut adalah node-goal, maka kembalikan jalur dari node awal ke node-goal
 - c. Jika node tersebut belum dikunjungi, tandai sebagai sudah dikunjungi dan tambahkan ke daftar yang sudah dikunjungi
 - d. Untuk setiap node tetangga yang belum dikunjungi, hitung biaya total dari node awal melalui node ini dan simpan dalam priority queue

- e. Jika biaya total untuk mencapai node tetangga ini lebih rendah dari biaya total saat ini, maka perbarui biaya total dan jalur terpendek yang ditemukan.
4. Jika priority queue kosong dan node-goal belum ditemukan, maka tidak ada jalur yang memungkinkan dari node awal ke node-goal

Algoritma UCS menjamin menemukan jalur terpendek secara optimal, karena mengurutkan node berdasarkan biaya terkecil dan memperbarui jalur terpendek saat menemukan biaya yang lebih rendah. Namun, algoritma ini dapat memakan waktu dan memori yang besar terutama jika graf sangat besar.

Algoritma A*

Algoritma A* adalah algoritma pencarian jalur terpendek pada graf atau peta yang menggunakan fungsi heuristik untuk memperkirakan jarak antara simpul saat ini dan simpul tujuan. Algoritma ini menggabungkan ide dari algoritma Best-First-Search dan Dijkstra dan memiliki performa yang lebih baik daripada kedua algoritma tersebut.

Berikut adalah langkah-langkah algoritma A* secara umum:

1. Tentukan simpul awal dan simpul tujuan pada graf atau peta.
2. Tetapkan nilai $g(s) = 0$ dan $h(s)$ sebagai fungsi heuristik yang mengestimasi jarak antara simpul awal dan simpul tujuan.
3. Buat open list yang berisi simpul awal
4. Pilih simpul dengan nilai $f(n) = g(n) + h(n)$ terkecil dari open list.
5. Jika simpul yang dipilih adalah simpul tujuan, maka pencarian selesai.
Kembalikan jalur dari simpul awal ke simpul tujuan.
6. Generate anak-anak dari simpul saat ini dan hitung nilai $g(n)$ dan $h(n)$ untuk setiap anak.
7. Untuk setiap anak, tambahkan ke open list dan tetapkan nilai $g(n)$ dan $h(n)$ jika anak tidak ada dalam open list atau memiliki $g(n)$ yang lebih kecil.
8. Tetapkan parent anak sebagai simpul saat ini dan lakukan langkah 4-7 hingga simpul tujuan ditemukan atau open list kosong.
9. Jika open list kosong dan simpul tujuan tidak ditemukan, maka tidak ada jalur dari simpul awal ke simpul tujuan.

3. Source Code Program

Main.py

```
1  import math
2
3  def create_coordinates(lines):
4      global list_of_coordinates
5      global list_of_names
6      n = int(lines[0])
7      list_of_coordinates = []
8      list_of_names = []
9      for i in range(1, n+1):
10         coordinates = lines[i].split(" ")
11         coords_list = [float(coordinates[1]), float(coordinates[2])]
12         list_of_coordinates.append(coords_list)
13         list_of_names.append(coordinates[0])
14     return list_of_coordinates, list_of_names
15
16 def create_list_of_lat(c):
17     global list_lat
18     list_lat = []
19     for i in range(len(c)):
20         list_lat.append(c[i][0])
21     return list_lat
22
23 def create_list_of_lon(c):
24     global list_lon
25     list_lon = []
26     for i in range(len(c)):
27         list_lon.append(c[i][1])
28     return list_lon
29
30 def create_matrix(lines):
31     n = int(lines[0])
32     adj_matrix = []
33     for i in range(n+1, n*2+1):
34         line = lines[i].split(" ")
35         adj_matrix.append(line)
36     return adj_matrix
```

```

37
38 def create_adj_list(m):
39     global adj_list
40     global list_of_names
41     adj_list = []
42     for i in range(0, len(m)):
43         neighbor = []
44         for j in range(0, len(m)):
45             if m[i][j] == '1':
46                 name = convert_to_name(j)
47                 neighbor.append(name)
48         adj_list.append(neighbor)
49     return adj_list
50
51 def create_adj_matrix(m):
52     global adj_matrix
53     global list_of_coordinates
54     n = len(m)
55     adj_matrix = [[ 0 for i in range(n)] for j in range(n)]
56     for i in range(0,n):
57         for j in range(0,n):
58             if m[i][j] == '1':
59                 distance = haversineDistance(list_of_coordinates[i],list_of_coordinates[j])
60                 adj_matrix[i][j] = distance
61     return adj_matrix
62
63 def create_heuristic_matrix(m):
64     global heuristic_matrix
65     global list_of_coordinates
66     n = len(m)
67     heuristic_matrix = [[ 0 for i in range(n)] for j in range(n)]
68     for i in range(0,n):
69         for j in range(0,n):
70             if (i!=j):
71                 distance = haversineDistance(list_of_coordinates[i],list_of_coordinates[j])
72                 heuristic_matrix[i][j] = distance
73             else:
74                 heuristic_matrix[i][j] = 0
75     return heuristic_matrix
76

```

```

77
78 def haversineDistance(a,b):
79
80     lat1 = a[0]
81     lon1 = a[1]
82     lat2 = b[0]
83     lon2 = b[1]
84
85     lat1_rad = lat1 * math.pi / 180.0
86     lat2_rad = lat2 * math.pi / 180.0
87
88     delta_lat = (lat2 - lat1) * math.pi / 180.0
89     delta_lon = (lon2 - lon1) * math.pi / 180.0
90
91     a = (pow(math.sin(delta_lat / 2), 2) + pow(math.sin(delta_lon / 2), 2) * math.cos(lat1_rad) * math.cos(lat2_rad))
92
93     r = 6371
94
95     distance = 2 * r * math.asin(math.sqrt(a)) * 1000
96     return distance
97
98
99 def convert_to_idx(node_name):
100     global list_of_names
101     idx = 0
102     for i in range(len(list_of_names)):
103         if (list_of_names[i] == node_name):
104             idx = i
105     return idx
106
107
108 def convert_to_name(idx):
109     global list_of_names
110     name = ''
111     for i in range(len(list_of_names)):
112         if (i == idx):
113             name = list_of_names[i]
114     return name

```



```

115
116
117 def astar(initial, final):
118     global adj_matrix
119     global heuristic_matrix
120     global adj_list
121     global list_of_names
122     global path
123
124
125     idx_initial = convert_to_idx(initial)
126     idx_final = convert_to_idx(final)
127     queue = [[idx_initial, 0, [initial]]]
128     current_node = []
129
130     while(len(queue) != 0):
131
132         current_node = queue.pop(0)
133         current_node_idx = convert_to_idx(current_node[0])
134
135         if (current_node_idx == idx_final):
136             break
137
138         for neighbor in adj_list[current_node_idx]:
139
140             visited_node = []
141             for c in current_node[2]:
142                 visited_node.append(c)
143
144             i = convert_to_idx(neighbor)
145             visited_node.append(neighbor)
146
147             queue.append([neighbor, adj_matrix[current_node_idx][i] + heuristic_matrix[i][idx_final], visited_node])
148
149             queue.sort(key = lambda q : q[1])
150
151
152     path = current_node[2]
153
154     cost = 0
155     path_cost = []
156     for node in path:
157         path_cost.append(convert_to_idx(node))
158     for i in range(len(path)-1):
159         cost += adj_matrix[path_cost[i]][path_cost[i+1]]
160
161     return path, cost

```

```

162
163 def path_coords(path):
164     global list_of_names
165     global list_of_coordinates
166     global list_of_path_coords
167     list_of_path_coords = []
168     for node in path[0]:
169         list_of_path_coords.append(list_of_coordinates[convert_to_idx(node)])
170     return list_of_path_coords
171
172 def print_route(solution):
173     print("Lintasan terpendek: ", end=" ")
174     for i in range(len(solution[0])):
175         if (i == (len(solution[0])-1)):
176             print(solution[0][i])
177         else:
178             print(solution[0][i], end=" -> ")
179     print("Panjang lintasan: ", solution[1], "meter. ")
180
181
182
183 def initialize(file_name):
184     data_folder = "../test/"
185     file_to_open = data_folder + file_name
186     f = open(file_to_open, "r")
187     lines = f.read().splitlines()
188     coordinates = create_coordinates(lines)[0]
189     list_lat = create_list_of_lat(coordinates)
190     list_lon = create_list_of_lon(coordinates)
191     node_names = create_coordinates(lines)[1]
192     matrix = create_matrix(lines)
193     adj_list = create_adj_list(matrix)
194     adj_matrix = create_adj_matrix(matrix)
195     heur_matrix = create_heuristic_matrix(matrix)

```

```
196
197
198     print("Program ini akan menghitung jarak terdekat dengan algoritma UCS atau A*")
199     file_name = input("Masukkan nama file dalam format .txt: ")
200     print("Masukkan pilihan algoritma")
201     print("1. algoritma A*")
202     print("2. algoritma UCS")
203
204     initialize(file_name)
205     pilihan = int(input(""))
206     if(pilihan == 1):
207         start_node = input("Masukkan start node: ")
208         goal_node = input("Masukkan goal node: ")
209         print("Hasil: ")
210         path_solution = astar(start_node, goal_node)
211         list_path = path_coords(path_solution)
212         print_route(path_solution)
213     else:
214         print("Algoritma belum dibuat")
215
```

4. Screenshot

alun2.txt

```
1 16
2 A -6.921141 107.607668
3 B -6.920768 107.604056
4 C -6.918263 107.604216
5 D -6.919038 107.606670
6 E -6.920995 107.606441
7 F -6.922551 107.607619
8 G -6.922771 107.609810
9 H -6.925376 107.610599
10 I -6.926075 107.610524
11 J -6.925835 107.607071
12 K -6.924356 107.607253
13 L -6.924317 107.606160
14 M -6.923950 107.603842
15 N -6.922388 107.606404
16 O -6.923443 107.606298
17 P -6.923100 107.603892
18 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0
19 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 1
20 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0
21 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0
22 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0
23 1 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0
24 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0
25 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0
26 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0
27 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0
28 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0
29 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0
30 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1
31 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0
32 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1
33 0 1 0 0 0 0 0 0 0 0 0 0 1 0 1 0
```

Buahbatu.txt

```
1      8
2    A -6.940351 107.658245
3    B -6.939252 107.663915
4    C -6.943234 107.663564
5    D -6.942138 107.652719
6    E -6.955690 107.654484
7    F -6.956029 107.662112
8    G -6.954222 107.639885
9    H -6.946367 107.641756
10   0 1 0 1 0 0 0 0
11   1 0 1 0 0 0 0 0
12   0 1 0 0 0 1 0 0
13   1 0 0 0 1 0 0 1
14   0 0 0 1 0 1 1 0
15   0 0 1 0 1 0 0 0
16   0 0 0 0 1 0 0 1
17   0 0 0 1 0 0 1 0
```

itb.txt

```
1     12
2    A -6.884893 107.611445
3    B -6.885191 107.613017
4    C -6.885257 107.613733
5    D -6.887256 107.611540
6    E -6.887386 107.613611
7    F -6.887910 107.608289
8    G -6.893882 107.608450
9    H -6.893230 107.610447
10   I -6.893605 107.611944
11   J -6.893780 107.613036
12   K -6.894759 107.611723
13   L -6.894883 107.608839
14   0 1 0 1 0 0 0 0 0 0 0 0
15   1 0 1 1 0 0 0 0 0 0 0 0
16   0 1 0 0 1 0 0 0 0 0 0 0
17   1 1 0 0 1 1 0 0 0 0 0 0
18   0 0 1 1 0 0 0 0 0 1 0 0
19   0 0 0 1 0 0 1 0 0 0 0 0
20   0 0 0 0 0 1 0 1 0 0 0 1
21   0 0 0 0 0 0 1 0 1 0 0 0
22   0 0 0 0 0 0 0 1 0 1 1 0
23   0 0 0 0 1 0 0 0 1 0 0 0
24   0 0 0 0 0 0 0 0 1 0 0 1
25   0 0 0 0 0 0 1 0 0 0 1 0
```

Jakarta.txt

```
1 13
2 A -6.309102 106.858279
3 B -6.309126 106.857437
4 C -6.309160 106.857262
5 D -6.309428 106.857282
6 E -6.309717 106.857293
7 F -6.310017 106.857302
8 G -6.310307 106.857319
9 H -6.310267 106.858366
10 I -6.309965 106.858354
11 J -6.309669 106.858351
12 K -6.309384 106.858332
13 L -6.308814 106.858315
14 M -6.308856 106.857452
15 0 1 0 0 0 0 0 0 0 1 1 0
16 1 0 1 0 0 0 0 0 0 0 0 1
17 0 1 0 1 0 0 0 0 0 0 0 0
18 0 0 1 0 1 0 0 0 0 0 1 0
19 0 0 0 1 0 1 0 0 0 0 0 0
20 0 0 0 0 1 0 1 0 0 0 0 0
21 0 0 0 0 0 1 0 1 0 0 0 0
22 0 0 0 0 0 0 1 0 1 0 0 0
23 0 0 0 0 0 0 0 1 0 1 0 0
24 0 0 0 0 0 0 0 0 1 0 1 0
25 1 0 0 0 0 0 0 0 0 1 0 0
26 1 0 0 0 0 0 0 0 0 0 0 1
27 0 1 0 0 0 0 0 0 0 0 1 0
```

Test menggunakan itb.txt

```
Program ini akan menghitung jarak terdekat dengan algoritma UCS atau A*
Masukkan nama file dalam format .txt: itb.txt
Masukkan pilihan algoritma
1. algoritma A*
2. algoritma UCS
1
Masukkan start node: A
Masukkan goal node: F
Hasil:
Lintasan terpendek: A -> D -> F
Panjang lintasan: 629.142451054661 meter.
```

Test menggunakan jakarta.txt

```
Masukkan nama file dalam format .txt: jakarta.txt
Masukkan pilihan algoritma
1. algoritma A*
2. algoritma UCS
1
Masukkan start node: A
Masukkan goal node: F
Hasil:
Lintasan terpendek: A -> K -> J -> I -> H -> G -> F
Panjang lintasan: 278.2842502839683 meter.
```

Test menggunakan alun2.txt

```
Program ini akan menghitung jarak terdekat dengan algoritma UCS atau A*
Masukkan nama file dalam format .txt: alun2.txt
Masukkan pilihan algoritma
1. algoritma A*
2. algoritma UCS
1
Masukkan start node: A
Masukkan goal node: G
Hasil:
Lintasan terpendek: A -> F -> G
Panjang lintasan: 399.96413122686965 meter.
```

Test menggunakan buahbatu.txt

```
Program ini akan menghitung jarak terdekat dengan algoritma UCS atau A*
Masukkan nama file dalam format .txt: buahbatu.txt
Masukkan pilihan algoritma
1. algoritma A*
2. algoritma UCS
1
Masukkan start node: A
Masukkan goal node: H
Hasil:
Lintasan terpendek: A -> D -> H
Panjang lintasan: 1939.7546674905243 meter.
```

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	

Link Repository : https://github.com/ezramcmh/Tucil3_13521073