FEDERAL STATE AUTONOMOUS EDUCATIONAL INSTITUTION

OF HIGHER EDUCATION

ITMO UNIVERSITY

Report

on the practical task No. 6

"Algorithms on graphs. Path search algorithms on weighted graphs "

Performed by

Cheng-Yuan and Ma

Academic group

J4133c

Petr Chunaev

St. Petersburg 2022

Language : python

Goal : The use of path search algorithms on weighted graphs (Dijkstra's, A* and Bellman- Ford algorithms)

Problem :

I. *Generate a random adjacency matrix for a simple undirected weighted graph of 100 vertices and 500 edges with assigned random positive integer weights (note that the matrix should be symmetric and contain only 0s and weights as elements). Use Dijkstra's and Bellman-Ford algorithms to find shortest paths between a random starting vertex and other vertices. Measure the time required to find the paths for each algorithm. Repeat the experiment 10 times for the same starting vertex and calculate the average time required for the paths search of each algorithm. Analyse the results obtained.*

II. *Generate a 10x20 cell grid with 40 obstacle cells. Choose two random non- obstacle cells and find a shortest path between them using A* algorithm. Repeat the experiment 5 times with different random pair of cells. Analyse the results obtained.*

III. *Describe the data structures and design techniques used within the algorithms.*

Theory:

Bellman Ford's algorithm and **Dijkstra's algorithm** both are single-source shortest path algorithm

Bellman-Ford algorithm is used to find the shortest path from the source vertex to every vertex in a weighted graph

Time Complexity: O(V * E), where V is the number of vertices in the graph and E is the number of edges in the graph

Auxiliary Space: O(E)

Dijkstra's algorithm is used to find the shortest path , which every time, take out the point of the minimum path from the starting point from the point where the shortest path has not been found, and use this point as a bridge to refresh the distance of the point where the shortest path is not found.

Time Complexity: O(E * logV), Where E is the number of edges and V is the number of vertices.

Auxiliary Space: O(V)

The only difference between the Dijkstra algorithm and the bellman ford algorithm is that Dijkstra's algorithm just visits the neighbour vertex in each iteration but the bellman ford algorithm visits each vertex through each edge in every iteration.

A* Search algorithm is one of the best and popular technique used in path-finding and graph traversals.

What A* Search Algorithm does is that at each step it picks the node according to a value-‘f’ which is a parameter equal to the sum of two other parameters – ‘g’ and ‘h’. At each step it picks the node/cell having the lowest ‘f’, and process that node/cell

g = the movement cost to move from the starting point to a given square on the grid, following the path generated to get there.
h = the estimated movement cost to move from that given square on the grid to the final destination

Results:

I.

This is just a adjacency graph which is shown as proof of existence of 500 edges ,


```
defaultdict(<class 'list'>, {0: [10, 34, 42, 54, 80, 98], 1: [12, 20, 39, 40, 55, 72, 80, 83, 94], 2: [24, 28, 31, 37, 52, 66, 75, 81], 3: [12, 14, 21, 22, 24, 31, 32, 47,
94, 97], 4: [6, 9, 18, 27, 29, 42, 43, 56, 59, 60, 74, 77, 78, 97], 5: [14, 25, 27, 30, 40, 41, 43, 48, 49, 53, 58, 63, 85, 88, 96], 6: [4, 16, 25, 31, 62, 85, 88], 7: [9,
16, 22, 29, 35, 54, 57, 66, 69, 73, 91], 8: [12, 27, 30, 43, 50, 58, 63, 67, 70, 73, 77, 83, 93], 9: [4, 7, 11, 17, 23, 28, 44, 47, 69, 71, 72], 10: [0, 13, 20, 35, 41, 77
84, 96, 99], 11: [9, 13, 26, 32, 33, 36, 52, 55, 60, 69, 70, 85, 93, 95], 12: [1, 3, 8, 44, 48, 63, 67, 68, 84, 87, 88, 96], 13: [7, 10, 11, 18, 20, 23, 46, 52, 62, 63, 65
68, 77, 95], 14: [3, 5, 28, 31, 33, 34, 61, 64, 68, 94], 15: [34, 52, 55, 64, 75, 82, 83, 86, 91, 92], 16: [6, 7, 23, 27, 40, 45, 48, 54, 58, 76, 83, 86, 88, 89, 95], 17:
37, 40, 48, 50, 52, 58, 79, 80, 82, 88], 18: [4, 13, 34, 40, 58, 60, 65, 70, 81, 82, 90, 92], 19: [21, 23, 41, 51, 65, 72, 78, 82, 84, 85], 20: [1, 10, 13, 45, 46, 49, 59,
86, 98], 21: [3, 19, 22, 32, 35, 37, 40, 51, 56, 58, 60, 69, 82, 97], 22: [3, 7, 21, 33, 45, 47, 55, 57, 67, 68, 90, 95, 99], 23: [9, 13, 16, 19, 24, 44, 46, 51, 58, 80],
[2, 3, 23, 31, 36, 81], 25: [5, 6, 40, 54, 55, 68, 79], 26: [11, 29, 36, 42, 44, 54, 66, 82], 27: [4, 5, 8, 16, 39, 41, 60, 84, 90, 96, 99], 28: [2, 9, 14, 34, 36, 45, 55,
82], 29: [4, 7, 26, 36, 50, 59, 64, 66], 30: [5, 8, 40, 52, 57, 63, 77], 31: [2, 3, 6, 14, 24, 32, 34, 59, 63, 79, 84, 92], 32: [3, 11, 21, 31, 37, 41, 55, 59, 63, 69, 78,
94], 33: [11, 14, 22, 41, 74, 77, 78, 81, 88], 34: [0, 14, 15, 18, 28, 31, 36, 47, 69, 74, 80, 98], 35: [7, 10, 21, 57, 59, 60, 63, 66, 97], 36: [11, 24, 26, 28, 29, 34,
50, 58, 71, 98], 37: [2, 17, 21, 32, 40, 41, 47, 55, 62, 73, 75, 78, 85, 93], 38: [40, 51, 60, 67, 77, 84, 87, 92], 39: [1, 27, 50, 75, 90], 40: [1, 5, 16, 17, 18, 21, 25,
37, 38, 49, 65, 69, 82, 85], 41: [5, 10, 19, 27, 32, 33, 37, 42, 51, 61, 65, 77, 80, 95, 99], 42: [0, 4, 26, 41, 44, 48, 54, 56, 74, 76, 81, 86, 88, 96], 43: [4, 5, 8, 46,
60, 89], 44: [9, 12, 23, 26, 42, 54, 58, 59, 76], 45: [16, 20, 22, 28, 51, 60, 97], 46: [13, 20, 23, 43, 63], 47: [3, 9, 22, 34, 36, 37, 69, 88, 98], 48: [5, 12, 16, 17,
53, 58, 82, 93, 98], 49: [5, 20, 40, 51, 55, 59, 64, 82, 91], 50: [8, 17, 29, 36, 39, 43, 59, 65, 68, 88], 51: [19, 21, 23, 38, 41, 45, 49, 59, 79, 87, 96], 52: [2, 11, 13
15, 17, 30, 58, 62, 74, 79, 81, 98], 53: [5, 48, 64, 83, 91], 54: [0, 7, 16, 25, 26, 42, 44, 63, 78, 81, 86, 99], 55: [1, 11, 15, 22, 25, 28, 32, 37, 49, 65, 68, 71, 75,
86, 88], 56: [3, 4, 21, 42, 76, 89], 57: [7, 22, 30, 35, 74, 76, 98], 58: [5, 8, 16, 17, 18, 21, 23, 36, 44, 48, 52, 80, 98], 59: [4, 20, 29, 31, 32, 35, 44, 49, 50, 51,
68, 96], 60: [4, 11, 18, 21, 27, 35, 38, 43, 45, 63, 78, 86, 91, 95, 97], 61: [14, 41, 59, 87, 91, 96], 62: [6, 13, 37, 52, 67, 72, 76, 95], 63: [5, 8, 12, 13, 30, 31, 32,
46, 54, 60, 64, 69, 81, 92, 98], 64: [14, 15, 29, 49, 53, 63, 65, 73, 78], 65: [13, 18, 19, 40, 41, 50, 55, 64, 72, 73, 79, 82, 84, 85, 86], 66: [2, 7, 26, 29, 35, 70, 81,
98], 67: [8, 12, 22, 38, 62, 74, 76], 68: [12, 13, 14, 22, 25, 50, 55, 59, 74, 75, 76, 83, 95], 69: [7, 9, 11, 21, 32, 34, 40, 47, 63, 71, 84, 85, 86, 95, 98], 70: [8, 11,
66, 76, 79, 83, 91, 97], 71: [9, 36, 55, 69, 82, 92, 96], 72: [1, 9, 19, 62, 65, 75], 73: [7, 8, 37, 64, 65, 80, 81, 84], 74: [4, 33, 34, 42, 52, 57, 67, 68, 76], 75: [2,
37, 39, 55, 68, 72, 98], 76: [16, 42, 44, 56, 57, 62, 67, 68, 70, 74], 77: [4, 8, 10, 13, 30, 33, 38, 41, 86, 99], 78: [4, 19, 32, 33, 37, 54, 55, 60, 64, 85, 94], 79: [17
25, 31, 51, 52, 65, 70, 90, 97, 99], 80: [0, 1, 17, 20, 23, 34, 41, 58, 73, 87], 81: [2, 18, 24, 33, 42, 52, 54, 63, 66, 73], 82: [15, 17, 18, 19, 21, 26, 28, 40, 48, 49,
71], 83: [1, 8, 15, 16, 53, 68, 70], 84: [10, 12, 19, 27, 31, 38, 65, 69, 73, 90], 85: [5, 6, 11, 19, 37, 40, 65, 69, 78], 86: [15, 16, 20, 42, 54, 55, 60, 65, 69, 77], 87
[12, 38, 51, 61, 80, 90], 88: [5, 6, 12, 16, 17, 33, 42, 47, 50, 55, 95, 96], 89: [16, 43, 56, 99], 90: [18, 22, 27, 39, 79, 84, 87], 91: [7, 15, 49, 53, 60, 61, 70, 97],
[15, 18, 31, 38, 63, 71, 97, 98], 93: [8, 11, 32, 37, 48, 94], 94: [1, 3, 14, 32, 78, 93], 95: [11, 13, 16, 22, 41, 60, 62, 68, 69, 88], 96: [5, 10, 12, 27, 42, 51, 59, 61
71, 88], 97: [3, 4, 21, 35, 45, 60, 70, 79, 91, 92], 98: [0, 20, 34, 36, 47, 48, 52, 57, 58, 63, 66, 69, 75, 92], 99: [10, 22, 27, 41, 54, 77, 79, 89]})
```

This is outcome after Dijkstra algorithm :

```
-------------------------------------------
after dikstra algorithm :

10th iteration outcome
Vertex    Distance from Source
0         0
1         12
2         9
3         7
4         7
5         8
6         10
7         4
8         9
9         9
10        7
11        8
12        10
13        8
14        11
15        6
16        8
17        5
18        6
19        9
20        8
21        10
22        5
23        6
24        7
25        10
26        6
27        8
28        10
29        13
30        11
31        7
32        7
33        10
34        5
35        8
36        7
37        7
38        13
```

```
35        8
36        7
37        7
38        13
39        11
40        8
41        6
42        4
43        11
44        5
45        9
46        7
47        6
48        6
49        9
50        10
51        9
52        10
53        12
54        3
55        6
56        6
57        9
58        5
59        10
60        9
61        7
62        9
63        9
64        10
65        7
66        12
67        9
68        9
69        7
70        10
71        10
72        11
73        5
74        8
75        8
76        7
77        8
```

```
61        7
62        9
63        9
64        10
65        7
66        12
67        9
68        9
69        7
70        10
71        10
72        11
73        5
74        8
75        8
76        7
77        8
78        8
79        8
80        4
81        10
82        8
83        10
84        10
85        8
86        5
87        7
88        7
89        11
90        12
91        12
92        10
93        8
94        8
95        9
96        12
97        9
98        7
99        9
-------------------------------------------
average iteration of 10 times run is : 100.0
```

This is outcome after Bellman-ford algorithm :

```
-------------------------------------------
after Bellman algorithm :

10th iteration outcome
Vertex Distance from Source
0       0
1       inf
2       7
3       10
4       inf
5       inf
6       inf
7       inf
8       43
9       44
10      48
11      52
12      58
13      60
14      61
15      63
16      65
17      inf
18      inf
19      inf
20      inf
21      inf
22      inf
23      inf
24      inf
25      inf
26      inf
27      inf
28      inf
29      inf
30      inf
31      inf
32      inf
33      inf
34      inf
35      inf
36      inf
37      inf
```

```
42      inf
43      inf
44      inf
45      inf
46      inf
47      inf
48      inf
49      128
50      128
51      129
52      129
53      130
54      130
55      131
56      132
57      133
58      133
59      133
60      134
61      135
62      135
63      137
64      138
65      138
66      139
67      139
68      140
69      144
70      145
71      145
72      145
73      146
74      146
75      146
76      inf
77      146
78      146
79      146
80      146
81      inf
82      inf
83      inf
```

```
65      138
66      139
67      139
68      140
69      144
70      145
71      145
72      145
73      146
74      146
75      146
76      inf
77      146
78      146
79      146
80      146
81      inf
82      inf
83      inf
84      146
85      inf
86      inf
87      147
88      147
89      147
90      inf
91      147
92      147
93      147
94      inf
95      inf
96      inf
97      inf
98      inf
99      inf
-------------------------------------------
average iteration of 10 times run is : 49500.0
```

II. These are 3 out of 10 outcomes of a* algorithm to find shortest path

```
-----------------------------------------
2th iteration
[[1 1 1 0 0 0 0 1 0 0 0 1 0 1 0 1 0 0 0 0]
 [0 1 0 0 0 0 0 0 0 0 1 1 0 0 0 1 0 1 0 0]
 [0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0]
 [1 0 1 0 0 0 1 0 0 1 1 0 1 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 0]
 [0 0 0 0 0 0 1 0 0 1 0 0 1 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0]
 [0 0 0 0 0 0 0 0 1 0 1 0 0 1 0 0 1 0 0 0]
 [0 1 0 0 0 0 1 0 1 0 0 0 0 1 0 0 0 0 0 1]
 [0 1 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0]]

start from : [1, 16]
to : [8, 18]

best route is [(1, 16), (2, 17), (3, 18), (4, 18), (5, 18), (6, 18), (7, 18), (8, 18)]

[[1 1 1 0 0 0 0 1 0 0 0 1 0 1 0 1 0 0 0 0]
 [0 1 0 0 0 0 0 0 0 0 1 1 0 0 0 1 7 1 0 0]
 [0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 7 0 0]
 [1 0 1 0 0 0 1 0 0 1 1 0 1 0 0 0 0 0 7 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 7 0]
 [0 0 0 0 0 0 1 0 0 1 0 0 1 0 0 0 0 0 7 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 7 0]
 [0 0 0 0 0 0 0 0 1 0 1 0 0 1 0 0 1 0 7 0]
 [0 1 0 0 0 0 1 0 1 0 0 0 0 1 0 0 0 0 7 1]
 [0 1 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0]]

-----------------------------------------
```

```
-----------------------------------------
10th iteration
[[1 1 1 0 0 0 0 1 0 0 0 1 0 1 0 1 0 0 0 0]
 [0 1 0 0 0 0 0 0 0 0 1 1 0 0 0 1 0 1 0 0]
 [0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0]
 [1 0 1 0 0 0 1 0 0 1 1 0 1 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 0]
 [0 0 0 0 0 0 1 0 0 1 0 0 1 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0]
 [0 0 0 0 0 0 0 0 1 0 1 0 0 1 0 0 1 0 0 0]
 [0 1 0 0 0 0 1 0 1 0 0 0 0 1 0 0 0 0 0 1]
 [0 1 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0]]

start from : [8, 9]
to : [7, 2]

best route is [(8, 9), (9, 8), (8, 7), (7, 6), (7, 5), (7, 4), (7, 3), (7, 2)]

[[1 1 1 0 0 0 0 1 0 0 0 1 0 1 0 1 0 0 0 0]
 [0 1 0 0 0 0 0 0 0 0 1 1 0 0 0 1 0 1 0 0]
 [0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0]
 [1 0 1 0 0 0 1 0 0 1 1 0 1 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 0]
 [0 0 0 0 0 0 1 0 0 1 0 0 1 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0]
 [0 0 7 7 7 7 7 0 1 0 1 0 0 1 0 0 1 0 0 0]
 [0 1 0 0 0 0 1 7 1 7 0 0 0 1 0 0 0 0 0 1]
 [0 1 0 1 0 0 0 0 7 0 0 0 0 0 1 0 0 0 0 0]]
```

```
-----------------------------------------
3th iteration
[[1 1 1 0 0 0 0 1 0 0 0 1 0 1 0 1 0 0 0 0]
 [0 1 0 0 0 0 0 0 0 0 1 1 0 0 0 1 0 1 0 0]
 [0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0]
 [1 0 1 0 0 0 1 0 0 1 1 0 1 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 0]
 [0 0 0 0 0 0 1 0 0 1 0 0 1 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0]
 [0 0 0 0 0 0 0 0 1 0 1 0 0 1 0 0 1 0 0 0]
 [0 1 0 0 0 0 1 0 1 0 0 0 0 1 0 0 0 0 0 1]
 [0 1 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0]]

start from : [7, 16]
to : [6, 13]

best route is [(7, 16), (7, 15), (6, 14), (6, 13)]

[[1 1 1 0 0 0 0 1 0 0 0 1 0 1 0 1 0 0 0 0]
 [0 1 0 0 0 0 0 0 0 0 1 1 0 0 0 1 0 1 0 0]
 [0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0]
 [1 0 1 0 0 0 1 0 0 1 1 0 1 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 0]
 [0 0 0 0 0 0 1 0 0 1 0 0 1 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 1 7 7 1 0 0 0 0]
 [0 0 0 0 0 0 0 0 1 0 1 0 0 1 0 7 7 0 0 0]
 [0 1 0 0 0 0 1 0 1 0 0 0 0 1 0 0 0 0 0 1]
 [0 1 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0]]
```

III.

I used numpy array and list as mainly structure

Firstly I created random matrix using random implementation in python

And converted them into adjacency list to use it with **Dijkstra algorithm** and **Bellman-ford algorithm**

Dijkstra algorithm :

Step-01: Create a list of “distances” equal to the number of nodes and initialize each value to infinity

Step-02: Set the “distance” to the starting node equal to 0

Step-03: Create a list of “visited” nodes set to false for each node (since we haven’t visited any yet)

Step-04: Loop through all the nodes

1. Loop through all the nodes again, and pick the one that is the shortest distance away *and* not yet visited
2. Set that node to visited
3. Set the distance in the distance list to the distance to that node

Step-05: The original “distance” list should now contain the shortest distance to each node or infinity if a node is unreachable from the desired starting node

Bellman-ford algorithm :

Step-01: Find the number of iterations to be performed.

If Total nodes are 6 and the number of iterations will be 1 less than the number of nodes which is 6 – 1 = 5.

Step-02: Initialization Initialize the value of the source node with 0, and the rest of the nodes with infinity as shown below:

Step-03: What to do in each iteration?
For each iteration, visit every edge of the graph and update values accordingly.

---

a* algorithm :

Initial condition - we create two lists - Open List and Closed List.

the following steps are

- The open list must be initialized.
- Put the starting node on the open list (leave its f at zero). Initialize the closed list.
- Follow the steps until the open list is non-empty:

1. Find the node with the least f on the open list and name it “q”.
2. Remove Q from the open list.
3. Produce q's eight descendants and set q as their parent.
4. For every descendant:

i) If finding a successor is the goal, cease looking

ii) Else, calculate g and h for the successor.

successor.g = q.g + the calculated distance between the successor and the q.

successor.h = the calculated distance between the successor and the goal. We will cover three heuristics to do this: the Diagonal, the Euclidean, and the Manhattan heuristics.

successor.f = successor.g plus successor.h

iii) Skip this successor if a node in the OPEN list with the same location as it but a lower f value than the successor is present.

iv) Skip the successor if there is a node in the CLOSED list with the same position as the successor but a lower f value; otherwise, add the node to the open list end (for loop).

- Push Q into the closed list and end the while loop.

We will now discuss how to calculate the Heuristics for the nodes.

Conclusion:

From iterations of bell-man and Dijkstra , we can that Dijkstra is much more efficient , and from outcome , we can see it is more accurate .

After 10 iterations , we can see that a* algorithm always can successfully got a shortest path from any random nodes . It is highly efficient .

Appendix

https://raw.githubusercontent.com/MaChengYuan/task6/main/task6_i.py

https://raw.githubusercontent.com/MaChengYuan/task6/main/task6_a*algorithm.py