

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: $O(E * \log V)$, 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 outcome after Dijkstra algorithm :

Randomly show two iteration of outcome

after dikstra algorithm :

3th iteration outcome	Vertex	Distance from Source	37	10000000.0	70	12
0	0		38	12	71	16
1	13		39	28	72	11
2	12		40	22	73	14
3	16		41	18	74	21
4	9		42	9	75	6
5	3		43	14	76	12
6	11		44	12	77	1
7	10000000.0		45	22	78	12
8	13		46	12	79	16
9	20		47	15	80	13
10	16		48	19	81	13
11	13		49	20	82	14
12	14		50	21	83	8
13	12		51	5	84	18
14	18		52	15	85	10
15	20		53	10	86	11
16	8		54	14	87	15
17	23		55	18	88	21
18	15		56	18	89	8
19	12		57	18	90	16
20	10		58	4	91	16
21	3		59	14	92	11
22	24		60	18	93	5
23	10		61	17	94	7
24	14		62	25	95	15
25	15		63	19	96	12
26	2		64	6	97	18
27	18		65	11	98	15
28	7		66	19	99	12
29	15		67	17		
30	14		68	20		
31	13		69	14		
32	4		70	12		
33	11		71	16		
34	9		72	11		
35	10		73	14		
36	12		74	21		
37	10000000.0		75	6		
38	12		76	12		
39	16		77	1		
40	22		78	12		
			79	16		

10th iteration outcome	Vertex	Distance from Source	10th iteration outcome	Vertex	Distance from Source	68	20
0	0		0	0		69	14
1	13		1	13		70	12
2	12		2	12		71	16
3	18		3	18		72	11
4	9		4	9		73	14
5	3		5	3		74	21
6	11		6	11		75	6
7	10000000.0		7	10000000.0		76	12
8	13		8	13		77	1
9	20		9	20		78	12
10	16		10	16		79	16
11	13		11	13		80	13
12	14		12	14		81	13
13	12		13	12		82	14
14	18		14	18		83	8
15	20		15	20		84	18
16	8		16	8		85	10
17	23		17	23		86	11
18	15		18	15		87	15
19	12		19	12		88	21
20	10		20	10		89	8
21	3		21	3		90	16
22	24		22	24		91	16
23	10		23	10		92	11
24	14		24	14		93	5
25	15		25	15		94	7
26	2		26	2		95	15
27	18		27	18		96	12
28	7		28	7		97	18
29	15		29	15		98	15
30	14		30	14			
31	13		31	13			
32	4		32	4			
33	11		33	11			
34	9		34	9			
35	10		35	10			
36	12		36	12			
37	10000000.0		37	10000000.0			
38	12		38	12			
39	28		39	28			
40	22		40	22			

average iteration of 10 times run is : 99.3

This is outcome after Bellman-ford algorithm :

Randomly show two iteration of outcome

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

average iteration of 10 times run is : 19800.0

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

$\text{successor.g} = q.g + \text{the calculated distance between the successor and the } q.$

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.

$\text{successor.f} = \text{successor.g} \text{ plus } \text{successor.h}$

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 Dijkstra is more accurate , and most of distance can have a sharp reduction . But in Bell-man graph , we can see there are still bunch of numbers remaining the same

After 10 iterations , we can see that a* algorithm always can successfully got a shortest path from any random nodes . It is highly and common and efficient way to use in machine learning .

Appendix

https://raw.githubusercontent.com/MaChengYuan/task6/main/task6_dijkstra.py

https://raw.githubusercontent.com/MaChengYuan/task6/main/task6_bellman.py

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

