ARTIFICIAL INTELLIGENCE

FALL 2022

ASSIGNMENT 1

Baran Gökçekli – 220315006

Yusuf Karaç – 190316065

6 Kasım 2022

A-Development Environment

In this assignment, we used Python version 3.10.2 and the SimpleAI library and we used Microsoft Visual Studio Code as an IDE and Windows as an operating system.

B-Problem Formulation

First, we assigned our initial state as a tuple with all three jugs empty which means (current jug1, current jug2, current jug 3) => (0, 0, 0)

Then, we assigned the capacity values and the target values to all three jugs. If you want, you can specify these values with the inputs you receive from the user (This part is the comment section)

Then, we defined possible actions for our problem. These are fill the jugs (fill 1, fill 2,...), empty jugs (empty 1, empty2,....), pour one to another (j1 to j2,...) and we kept these values as tuples

C-Results

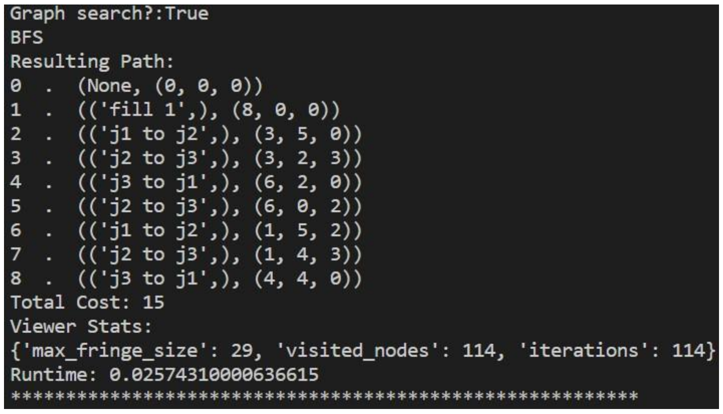
TEST 1

(c1, c2, c3) = (8, 5, 3), (t1, t2, t2) = (4, 4, 0), depth_limit = 15, graph search

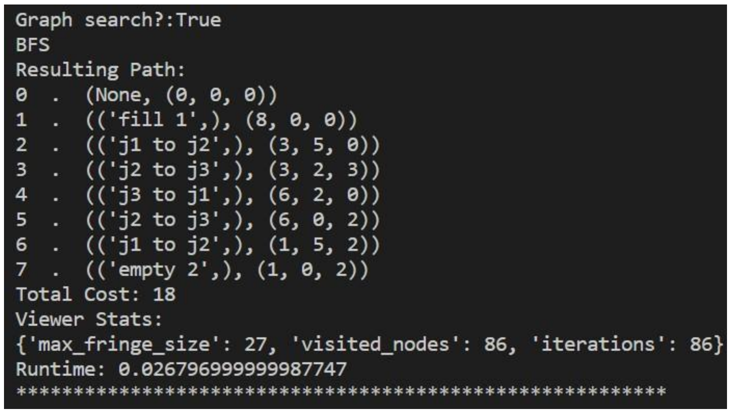
Vs

(c1, c2, c3) = (8, 5, 3), (t1, t2, t2) = (1, 0, 2), depth_limit = 15, graph search :

BFS:

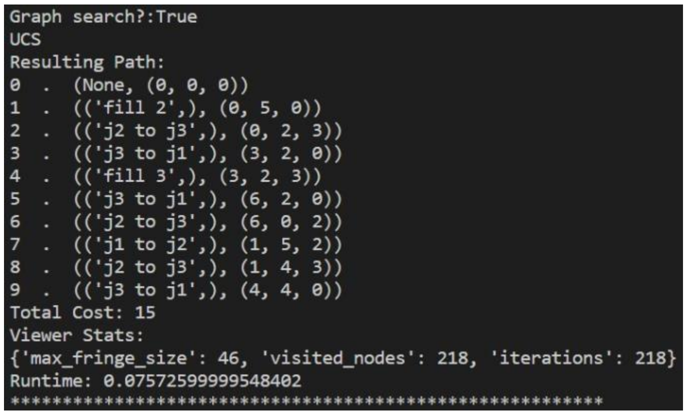


```
Graph search?:True
BFS
Resulting Path:
0  .  (None, (0, 0, 0))
1  .  (('fill 1',), (8, 0, 0))
2  .  (('j1 to j2',), (3, 5, 0))
3  .  (('j2 to j3',), (3, 2, 3))
4  .  (('j3 to j1',), (6, 2, 0))
5  .  (('j2 to j3',), (6, 0, 2))
6  .  (('j1 to j2',), (1, 5, 2))
7  .  (('j2 to j3',), (1, 4, 3))
8  .  (('j3 to j1',), (4, 4, 0))
Total Cost: 15
Viewer Stats:
{'max_fringe_size': 29, 'visited_nodes': 114, 'iterations': 114}
Runtime: 0.02574310000636615
*********************************************************
```

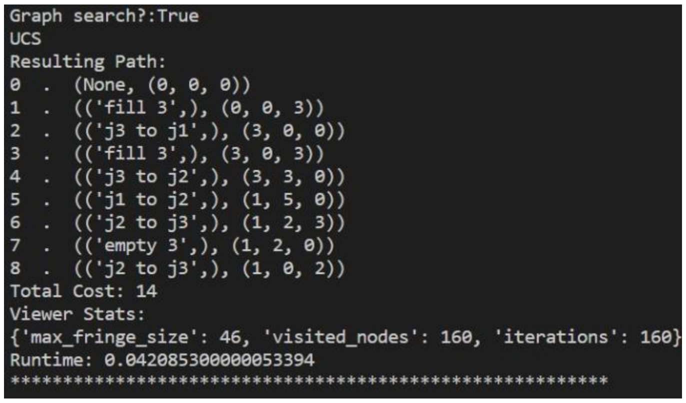


```
Graph search?:True
BFS
Resulting Path:
0  .  (None, (0, 0, 0))
1  .  (('fill 1',), (8, 0, 0))
2  .  (('j1 to j2',), (3, 5, 0))
3  .  (('j2 to j3',), (3, 2, 3))
4  .  (('j3 to j1',), (6, 2, 0))
5  .  (('j2 to j3',), (6, 0, 2))
6  .  (('j1 to j2',), (1, 5, 2))
7  .  (('empty 2',), (1, 0, 2))
Total Cost: 18
Viewer Stats:
{'max_fringe_size': 27, 'visited_nodes': 86, 'iterations': 86}
Runtime: 0.026796999999987747
***********************************************************
```

UCS:

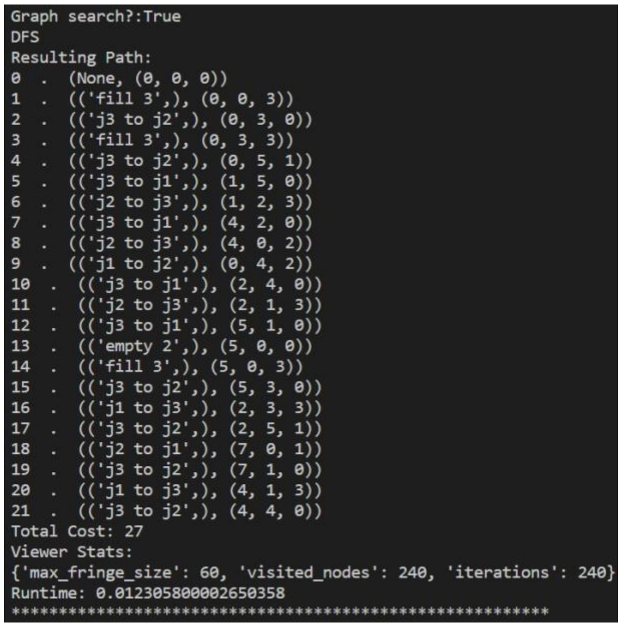


```
Graph search?:True
UCS
Resulting Path:
0  .  (None, (0, 0, 0))
1  .  (('fill 2',), (0, 5, 0))
2  .  (('j2 to j3',), (0, 2, 3))
3  .  (('j3 to j1',), (3, 2, 0))
4  .  (('fill 3',), (3, 2, 3))
5  .  (('j3 to j1',), (6, 2, 0))
6  .  (('j2 to j3',), (6, 0, 2))
7  .  (('j1 to j2',), (1, 5, 2))
8  .  (('j2 to j3',), (1, 4, 3))
9  .  (('j3 to j1',), (4, 4, 0))
Total Cost: 15
Viewer Stats:
{'max_fringe_size': 46, 'visited_nodes': 218, 'iterations': 218}
Runtime: 0.07572599999548402
*********************************************************
```

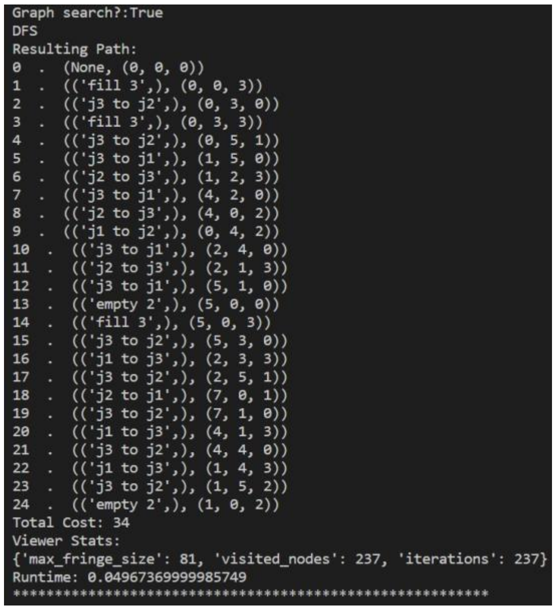


```
Graph search?:True
UCS
Resulting Path:
0  .  (None, (0, 0, 0))
1  .  (('fill 3',), (0, 0, 3))
2  .  (('j3 to j1',), (3, 0, 0))
3  .  (('fill 3',), (3, 0, 3))
4  .  (('j3 to j2',), (3, 3, 0))
5  .  (('j1 to j2',), (1, 5, 0))
6  .  (('j2 to j3',), (1, 2, 3))
7  .  (('empty 3',), (1, 2, 0))
8  .  (('j2 to j3',), (1, 0, 2))
Total Cost: 14
Viewer Stats:
{'max_fringe_size': 46, 'visited_nodes': 160, 'iterations': 160}
Runtime: 0.042085300000053394
*********************************************************
```

DFS:

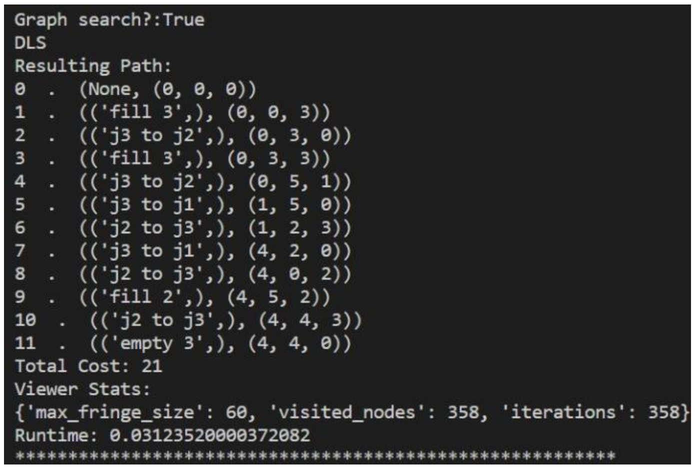


```
Graph search?:True
DFS
Resulting Path:
0  .  (None, (0, 0, 0))
1  .  (('fill 3',), (0, 0, 3))
2  .  (('j3 to j2',), (0, 3, 0))
3  .  (('fill 3',), (0, 3, 3))
4  .  (('j3 to j2',), (0, 5, 1))
5  .  (('j3 to j1',), (1, 5, 0))
6  .  (('j2 to j3',), (1, 2, 3))
7  .  (('j3 to j1',), (4, 2, 0))
8  .  (('j2 to j3',), (4, 0, 2))
9  .  (('j1 to j2',), (0, 4, 2))
10  .  (('j3 to j1',), (2, 4, 0))
11  .  (('j2 to j3',), (2, 1, 3))
12  .  (('j3 to j1',), (5, 1, 0))
13  .  (('empty 2',), (5, 0, 0))
14  .  (('fill 3',), (5, 0, 3))
15  .  (('j3 to j2',), (5, 3, 0))
16  .  (('j1 to j3',), (2, 3, 3))
17  .  (('j3 to j2',), (2, 5, 1))
18  .  (('j2 to j1',), (7, 0, 1))
19  .  (('j3 to j2',), (7, 1, 0))
20  .  (('j1 to j3',), (4, 1, 3))
21  .  (('j3 to j2',), (4, 4, 0))
Total Cost: 27
Viewer Stats:
{'max_fringe_size': 60, 'visited_nodes': 240, 'iterations': 240}
Runtime: 0.012305800002650358
*********************************************************
```

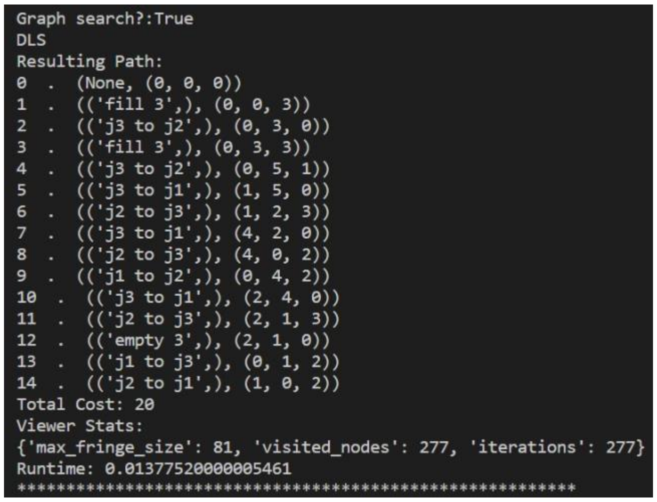


```
Graph search?:True
DFS
Resulting Path:
0  .  (None, (0, 0, 0))
1  .  (('fill 3',), (0, 0, 3))
2  .  (('j3 to j2',), (0, 3, 0))
3  .  (('fill 3',), (0, 3, 3))
4  .  (('j3 to j2',), (0, 5, 1))
5  .  (('j3 to j1',), (1, 5, 0))
6  .  (('j2 to j3',), (1, 2, 3))
7  .  (('j3 to j1',), (4, 2, 0))
8  .  (('j2 to j3',), (4, 0, 2))
9  .  (('j1 to j2',), (0, 4, 2))
10  .  (('j3 to j1',), (2, 4, 0))
11  .  (('j2 to j3',), (2, 1, 3))
12  .  (('j3 to j1',), (5, 1, 0))
13  .  (('empty 2',), (5, 0, 0))
14  .  (('fill 3',), (5, 0, 3))
15  .  (('j3 to j2',), (5, 3, 0))
16  .  (('j1 to j3',), (2, 3, 3))
17  .  (('j3 to j2',), (2, 5, 1))
18  .  (('j2 to j1',), (7, 0, 1))
19  .  (('j3 to j2',), (7, 1, 0))
20  .  (('j1 to j3',), (4, 1, 3))
21  .  (('j3 to j2',), (4, 4, 0))
22  .  (('j1 to j3',), (1, 4, 3))
23  .  (('j3 to j2',), (1, 5, 2))
24  .  (('empty 2',), (1, 0, 2))
Total Cost: 34
Viewer Stats:
{'max_fringe_size': 81, 'visited_nodes': 237, 'iterations': 237}
Runtime: 0.04967369999985749
*********************************************************
```

DLS:

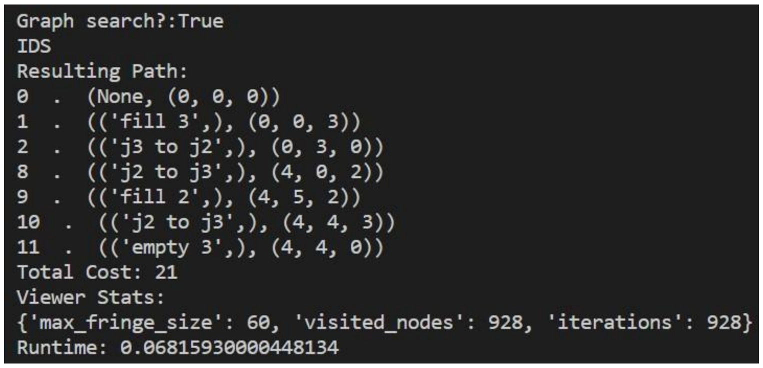


```
Graph search?:True
DLS
Resulting Path:
0  .  (None, (0, 0, 0))
1  .  (('fill 3',), (0, 0, 3))
2  .  (('j3 to j2',), (0, 3, 0))
3  .  (('fill 3',), (0, 3, 3))
4  .  (('j3 to j2',), (0, 5, 1))
5  .  (('j3 to j1',), (1, 5, 0))
6  .  (('j2 to j3',), (1, 2, 3))
7  .  (('j3 to j1',), (4, 2, 0))
8  .  (('j2 to j3',), (4, 0, 2))
9  .  (('fill 2',), (4, 5, 2))
10  .  (('j2 to j3',), (4, 4, 3))
11  .  (('empty 3',), (4, 4, 0))
Total Cost: 21
Viewer Stats:
{'max_fringe_size': 60, 'visited_nodes': 358, 'iterations': 358}
Runtime: 0.03123520000372082
*********************************************************
```

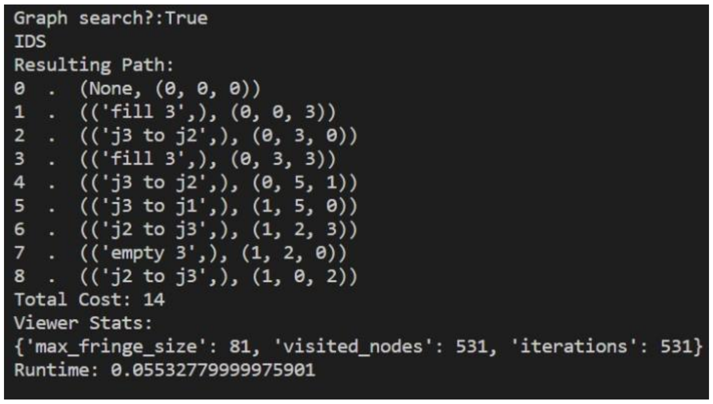


```
Graph search?:True
DLS
Resulting Path:
0  .  (None, (0, 0, 0))
1  .  (('fill 3',), (0, 0, 3))
2  .  (('j3 to j2',), (0, 3, 0))
3  .  (('fill 3',), (0, 3, 3))
4  .  (('j3 to j2',), (0, 5, 1))
5  .  (('j3 to j1',), (1, 5, 0))
6  .  (('j2 to j3',), (1, 2, 3))
7  .  (('j3 to j1',), (4, 2, 0))
8  .  (('j2 to j3',), (4, 0, 2))
9  .  (('j1 to j2',), (0, 4, 2))
10  .  (('j3 to j1',), (2, 4, 0))
11  .  (('j2 to j3',), (2, 1, 3))
12  .  (('empty 3',), (2, 1, 0))
13  .  (('j1 to j3',), (0, 1, 2))
14  .  (('j2 to j1',), (1, 0, 2))
Total Cost: 20
Viewer Stats:
{'max_fringe_size': 81, 'visited_nodes': 277, 'iterations': 277}
Runtime: 0.01377520000005461
*********************************************************
```

IDS:



```
Graph search?:True
IDS
Resulting Path:
0  .  (None, (0, 0, 0))
1  .  (('fill 3',), (0, 0, 3))
2  .  (('j3 to j2',), (0, 3, 0))
8  .  (('j2 to j3',), (4, 0, 2))
9  .  (('fill 2',), (4, 5, 2))
10  .  (('j2 to j3',), (4, 4, 3))
11  .  (('empty 3',), (4, 4, 0))
Total Cost: 21
Viewer Stats:
{'max_fringe_size': 60, 'visited_nodes': 928, 'iterations': 928}
Runtime: 0.06815930000448134
```



```
Graph search?:True
IDS
Resulting Path:
0  .  (None, (0, 0, 0))
1  .  (('fill 3',), (0, 0, 3))
2  .  (('j3 to j2',), (0, 3, 0))
3  .  (('fill 3',), (0, 3, 3))
4  .  (('j3 to j2',), (0, 5, 1))
5  .  (('j3 to j1',), (1, 5, 0))
6  .  (('j2 to j3',), (1, 2, 3))
7  .  (('empty 3',), (1, 2, 0))
8  .  (('j2 to j3',), (1, 0, 2))
Total Cost: 14
Viewer Stats:
{'max_fringe_size': 81, 'visited_nodes': 531, 'iterations': 531}
Runtime: 0.05532779999975901
```

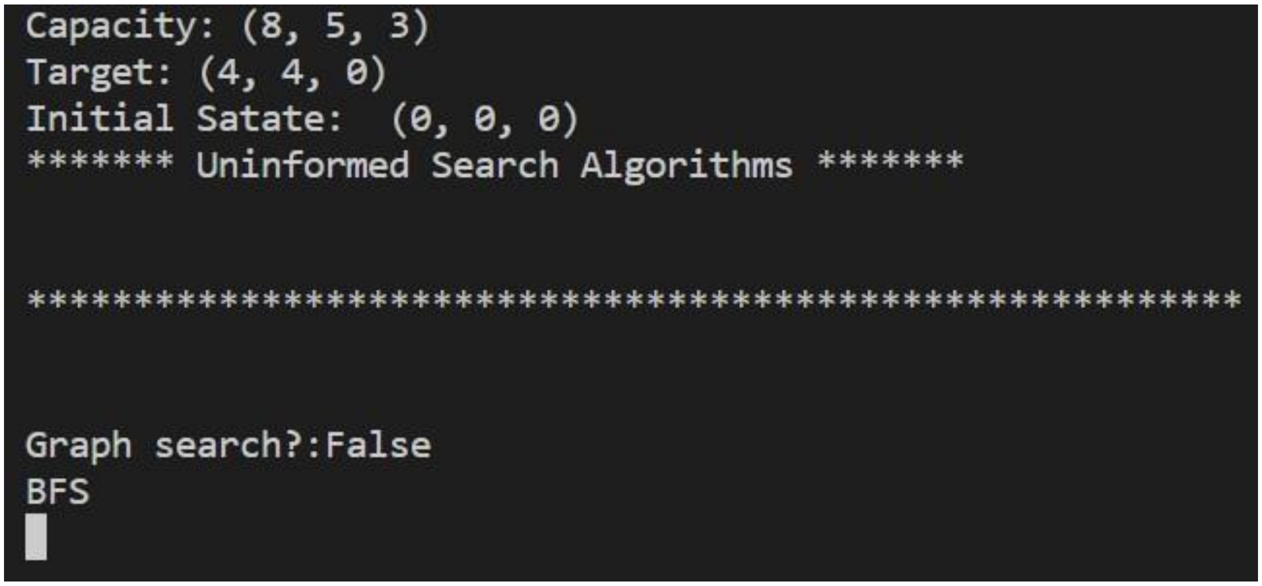
TEST 2

(c1, c2, c3) = (8, 5, 3), (t1, t2, t2) = (4, 4, 0), depth_limit = 15, graph search

Vs

(c1, c2, c3) = (8, 5, 3), (t1, t2, t2) = (4, 4, 0), depth_limit = 15, tree search

We ran into a problem here. When we run the code with the tree search, it didn't give an error, but it didn't return the result either. We think it is because it goes into an infinite loop or it take too long to solve because of high fringe size (we thought that because when we defined target as (6,2,0) it can solve in 29 seconds), but we couldn't figure out how to fix it.

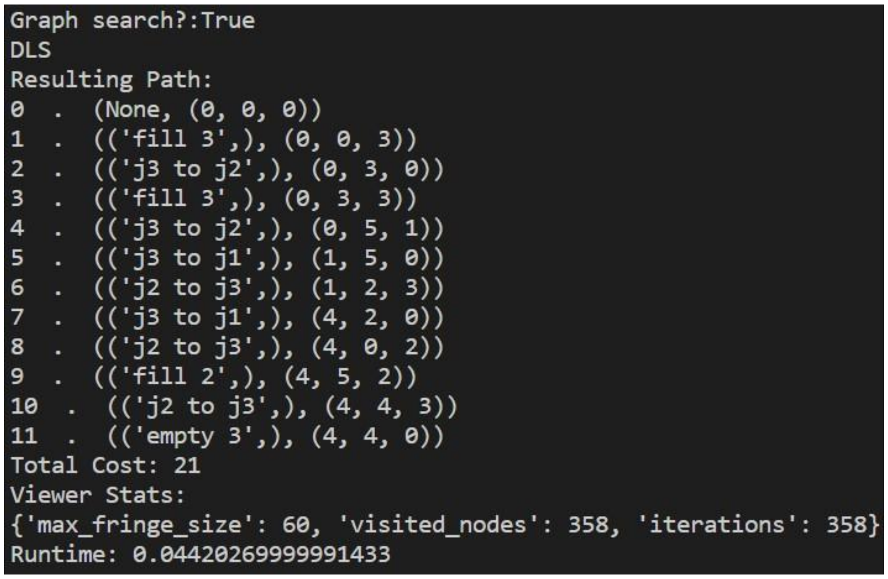


```
Capacity: (8, 5, 3)
Target: (4, 4, 0)
Initial Satate:  (0, 0, 0)
******* Uninformed Search Algorithms *******


**********************************************************


Graph search?:False
BFS
█
```

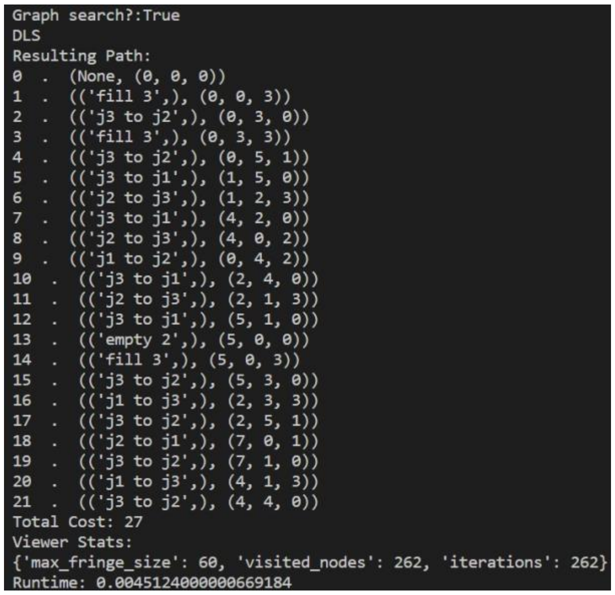
TEST 3

(c1, c2, c3) = (8, 5, 3), (t1, t2, t2) = (4, 4, 0), depth_limit = 15, graph search

(c1, c2, c3) = (8, 5, 3), (t1, t2, t2) = (4, 4, 0), depth_limit = 32, graph search



```
Graph search?:True
DLS
Resulting Path:
0  .  (None, (0, 0, 0))
1  .  (('fill 3',), (0, 0, 3))
2  .  (('j3 to j2',), (0, 3, 0))
3  .  (('fill 3',), (0, 3, 3))
4  .  (('j3 to j2',), (0, 5, 1))
5  .  (('j3 to j1',), (1, 5, 0))
6  .  (('j2 to j3',), (1, 2, 3))
7  .  (('j3 to j1',), (4, 2, 0))
8  .  (('j2 to j3',), (4, 0, 2))
9  .  (('fill 2',), (4, 5, 2))
10  .  (('j2 to j3',), (4, 4, 3))
11  .  (('empty 3',), (4, 4, 0))
Total Cost: 21
Viewer Stats:
{'max_fringe_size': 60, 'visited_nodes': 358, 'iterations': 358}
Runtime: 0.04420269999991433
```



```
Graph search?:True
DLS
Resulting Path:
0  .  (None, (0, 0, 0))
1  .  (('fill 3',), (0, 0, 3))
2  .  (('j3 to j2',), (0, 3, 0))
3  .  (('fill 3',), (0, 3, 3))
4  .  (('j3 to j2',), (0, 5, 1))
5  .  (('j3 to j1',), (1, 5, 0))
6  .  (('j2 to j3',), (1, 2, 3))
7  .  (('j3 to j1',), (4, 2, 0))
8  .  (('j2 to j3',), (4, 0, 2))
9  .  (('j1 to j2',), (0, 4, 2))
10  .  (('j3 to j1',), (2, 4, 0))
11  .  (('j2 to j3',), (2, 1, 3))
12  .  (('j3 to j1',), (5, 1, 0))
13  .  (('empty 2',), (5, 0, 0))
14  .  (('fill 3',), (5, 0, 3))
15  .  (('j3 to j2',), (5, 3, 0))
16  .  (('j1 to j3',), (2, 3, 3))
17  .  (('j3 to j2',), (2, 5, 1))
18  .  (('j2 to j1',), (7, 0, 1))
19  .  (('j3 to j2',), (7, 1, 0))
20  .  (('j1 to j3',), (4, 1, 3))
21  .  (('j3 to j2',), (4, 4, 0))
Total Cost: 27
Viewer Stats:
{'max_fringe_size': 60, 'visited_nodes': 262, 'iterations': 262}
Runtime: 0.004512400000669184
```

D – Discussion

There are five different search methods. As you can see in the results section each search method has pros and cons. For example, BFS is good for completeness and optimality but not good for time complexity and space complexity. It has 29 nodes in its fringe and visited 114 nodes and it took 0.026 seconds to complete the process.

UCS is good for completeness, and optimality but it is not good for time and space complexity. It has 46 nodes in its fringe and visited 218 nodes and it took 0.076 seconds to complete the process.

DFS is not good for completeness, optimality, and time but it is good for space complexity. It has 60 nodes in its fringe and visited 240 nodes and it took 0.012 seconds to complete the process.

DLS is not good for completeness, optimality, and time but it is good for space complexity just like DFS. It has 60 nodes in its fringe and visited 358 nodes and it took 0.031 seconds to complete the process. DLS has another element called depth limit that other search methods don’t have. It is used to prevent depth space from being infinite. As you can see in the outputs, the output with a 15-depth limit is faster than the output with a 32-depth limit.

IDS is good for completeness, optimality, and space complexity but it is not good for time complexity. It has 60 nodes in its fringe and visited 928 nodes and it took 0.068 seconds to complete the process.