T.C

MANİSA CELÂL BAYAR

UNIVERSITY

ENGINERRING FACULTY

COMPUTER ENGINEERING DEPARTMENT

ARTIFICIAL INTELLIGENCE

AI – PA2 & PA3 Report

Prepared by

Berkay Şahin 170316012
Ayşenur Büyükbal 170316007
Mustafa Deniz Demir 180316043

Instructor

Dr. Zeynep ÇİPİLOĞLU YILDIZ

FALL 2021-2022

TABLE OF CONTENT

Development Environment

Implementation machine's operating system is Windows 10 Home Pro. We used Microsoft Visual Studio Code environment to implement Simple AI Library's searching problem class to our N-Queens Problem and Python version 3.8.12 is used.

Problem Formulation

This problem is to find an arrangement of N-Queens on a chess board, such that no queen can attack any other queens on the board. The chess queens can attack in any direction as horizontal, vertical, horizontal, and diagonal way. But since we assume that each column has only one queen, so we do not need to check our vertical violation cases.

We keep our current arrangement state in a string which has length N and consists only integers between 1,…, N.

In this assignment, we are expected to formulate the N-Queens problem as a search problem and solve it using uninformed and informed search algorithms:

- **PA2 (Uninformed)**
  - Breadth-First Search (BFS)
  - Uniform Cost Search (UCS)
  - Depth First Search(DFS)
  - Depth Limited Search (DLS)
  - Iterative Deepening Search (IDS).
- **PA3 (Informed)**
  - A*
  - Greedy

We can move a single queen in its row at each step and each move is of cost 1. In other words, each action will be in form of "***Move Queen*** i ***to row*** j", where $\mathbf{1 \leq i, j \leq N}$.

To define a search algorithm, we must describe and figure out some terms. A problem can be defined formally by five components:

1) The **initial state** that the agent starts in.

2) A description of the possible **actions** available to the agent.

3) **Transition model**: A description of what each action does.

4) The **goal test**, which determines whether a given state is a goal state.

5) A **path cost** function that assigns a numeric cost to each path.

In our problem we can describe these five terms like:

<u>initial state</u> : given by user in string type such as "3214" for N=4 case.

<u>actions</u> : when the algorithm take the state string, looks for the possible steps. We have formulated a solution algorithm to generate possible steps. For example, solution works for state "3242" in this way:

- Take the n number of current problem and create a list like [1,2,3,4] to check which rows we can move that queen to.
- Take string and iterate over the string's each integer and iterate over the list we created above.
- Check if the current integer of state string is equal to current value of list item. If they are equal it means that you can not move that row our queen. Let's say we are iterating over our state string "3214" and our current queen is the first queen, and its value is 3. Algorithm checks this 3 and each item of [1,2,3,4].

  3 and 1 – they are not equal, this 3 may move to 1
  3 and 2 – they are not equal, this 3 may move to 2
  3 and 3 – they are equal, this 3 can't move to 3
  3 and 4 – they are not equal, this 3 may move to 4

- Do it for all other integers in the state string and after each checking add all possibilities to another list in tuples each element such as (queen number, current value, new value). As an actions list, return it.

transition model : Transition model describes that what will be done after any action. After completing the action, we have a result. In our case, as a result, algorithm takes one of the possible actions (priority depends on the search algorithm) and does what the action says. returns the new state again to algorithm back.

goal test : We already have count attacking pair function from AI – PA1 assignment. In this component, we use this function to test our current state if it is our goal state or not. If the count attacking pair amount is zero, this means that this state is one of the goal states. When the algorithm encounters a value like zero, then it stops running and quits searching.

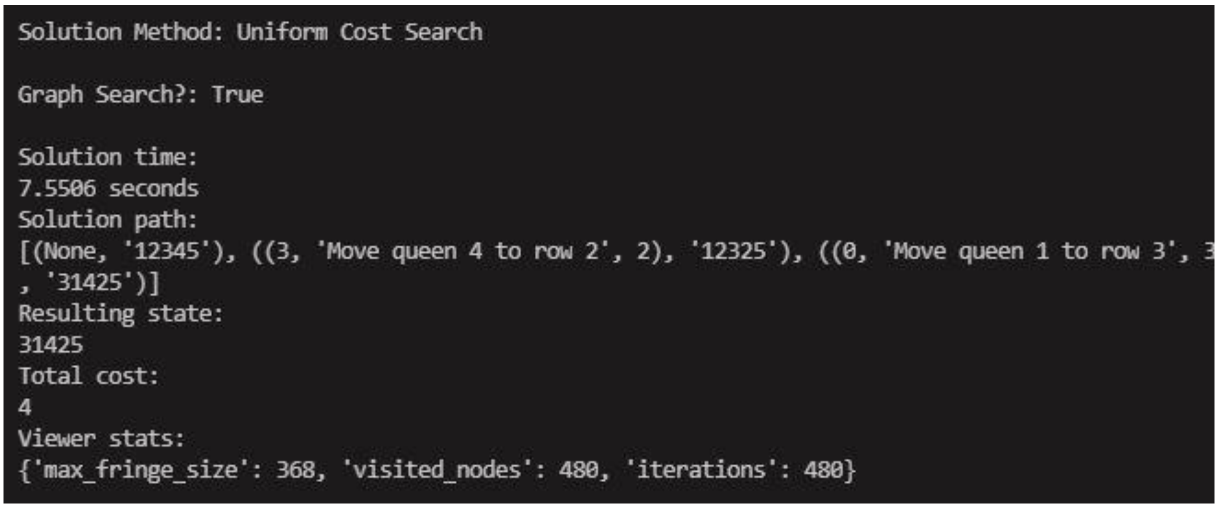
path cost : This value may change in different cases but in our case we have a default value as cost which is 1 per action.

AI – PA2

Results for PA2

To compare results, we have chosen state "12345" for all algorithms.

UCS:

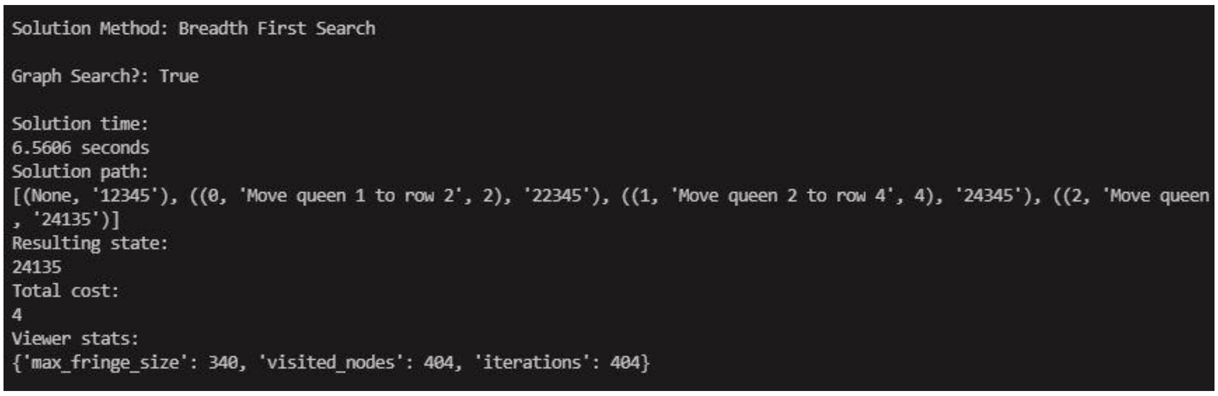


```
Solution Method: Uniform Cost Search

Graph Search?: True

Solution time:
7.5506 seconds
Solution path:
[(None, '12345'), ((3, 'Move queen 4 to row 2', 2), '12325'), ((0, 'Move queen 1 to row 3', 3
, '31425')]
Resulting state:
31425
Total cost:
4
Viewer stats:
{'max_fringe_size': 368, 'visited_nodes': 480, 'iterations': 480}
```

BFS:

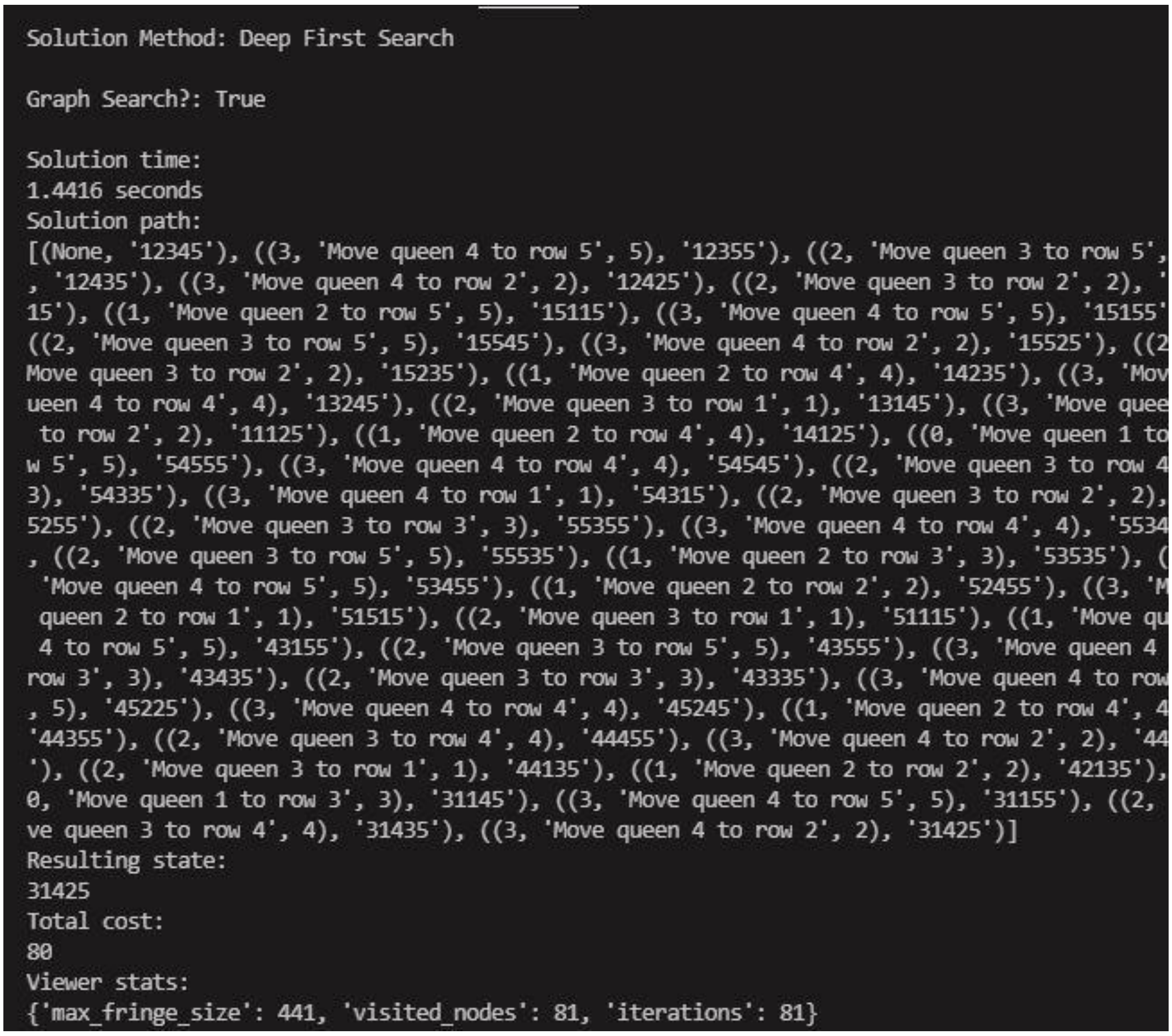


```
Solution Method: Breadth First Search

Graph Search?: True

Solution time:
6.5606 seconds
Solution path:
[(None, '12345'), ((0, 'Move queen 1 to row 2', 2), '22345'), ((1, 'Move queen 2 to row 4', 4), '24345'), ((2, 'Move queen
, '24135')]
Resulting state:
24135
Total cost:
4
Viewer stats:
{'max_fringe_size': 340, 'visited_nodes': 404, 'iterations': 404}
```

DFS:



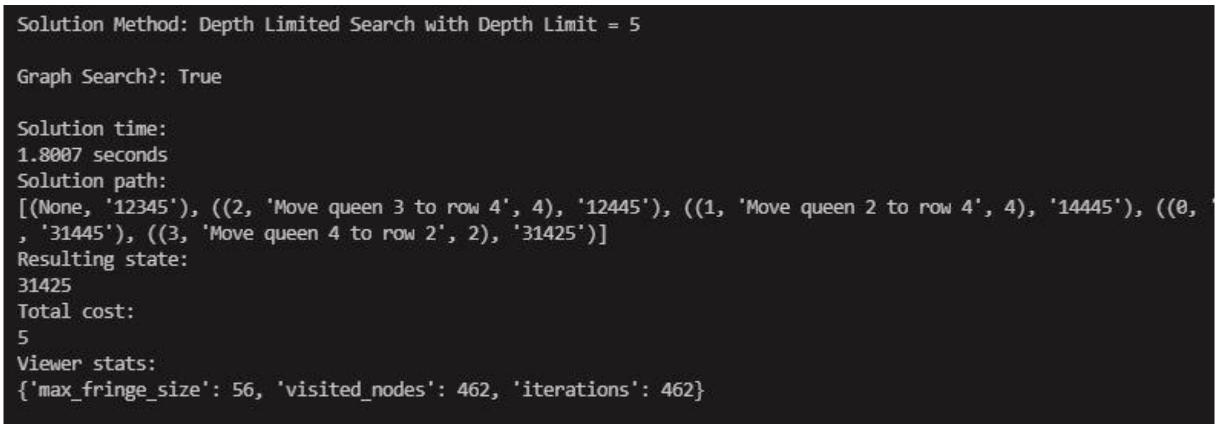
```
Solution Method: Deep First Search

Graph Search?: True

Solution time:
1.4416 seconds
Solution path:
[(None, '12345'), ((3, 'Move queen 4 to row 5', 5), '12355'), ((2, 'Move queen 3 to row 5',
, '12435'), ((3, 'Move queen 4 to row 2', 2), '12425'), ((2, 'Move queen 3 to row 2', 2), '
15'), ((1, 'Move queen 2 to row 5', 5), '15115'), ((3, 'Move queen 4 to row 5', 5), '15155'
((2, 'Move queen 3 to row 5', 5), '15545'), ((3, 'Move queen 4 to row 2', 2), '15525'), ((2
Move queen 3 to row 2', 2), '15235'), ((1, 'Move queen 2 to row 4', 4), '14235'), ((3, 'Mov
ueen 4 to row 4', 4), '13245'), ((2, 'Move queen 3 to row 1', 1), '13145'), ((3, 'Move quee
 to row 2', 2), '11125'), ((1, 'Move queen 2 to row 4', 4), '14125'), ((0, 'Move queen 1 to
w 5', 5), '54555'), ((3, 'Move queen 4 to row 4', 4), '54545'), ((2, 'Move queen 3 to row 4
3), '54335'), ((3, 'Move queen 4 to row 1', 1), '54315'), ((2, 'Move queen 3 to row 2', 2),
5255'), ((2, 'Move queen 3 to row 3', 3), '55355'), ((3, 'Move queen 4 to row 4', 4), '5534
, ((2, 'Move queen 3 to row 5', 5), '55535'), ((1, 'Move queen 2 to row 3', 3), '53535'), (
 'Move queen 4 to row 5', 5), '53455'), ((1, 'Move queen 2 to row 2', 2), '52455'), ((3, 'M
 queen 2 to row 1', 1), '51515'), ((2, 'Move queen 3 to row 1', 1), '51115'), ((1, 'Move qu
 4 to row 5', 5), '43155'), ((2, 'Move queen 3 to row 5', 5), '43555'), ((3, 'Move queen 4
row 3', 3), '43435'), ((2, 'Move queen 3 to row 3', 3), '43335'), ((3, 'Move queen 4 to row
, 5), '45225'), ((3, 'Move queen 4 to row 4', 4), '45245'), ((1, 'Move queen 2 to row 4', 4
'44355'), ((2, 'Move queen 3 to row 4', 4), '44455'), ((3, 'Move queen 4 to row 2', 2), '44
'), ((2, 'Move queen 3 to row 1', 1), '44135'), ((1, 'Move queen 2 to row 2', 2), '42135'),
0, 'Move queen 1 to row 3', 3), '31145'), ((3, 'Move queen 4 to row 5', 5), '31155'), ((2,
ve queen 3 to row 4', 4), '31435'), ((3, 'Move queen 4 to row 2', 2), '31425')]
Resulting state:
31425
Total cost:
80
Viewer stats:
{'max_fringe_size': 441, 'visited_nodes': 81, 'iterations': 81}
```

We tried DLS for 5 different depth limits.

DLS Limit = 5:

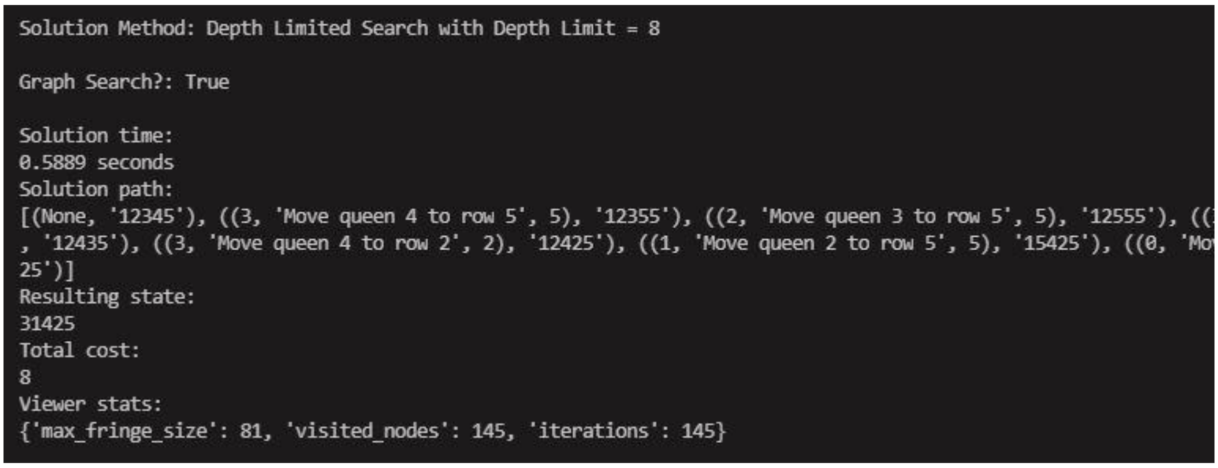


```
Solution Method: Depth Limited Search with Depth Limit = 5

Graph Search?: True

Solution time:
1.8007 seconds
Solution path:
[(None, '12345'), ((2, 'Move queen 3 to row 4', 4), '12445'), ((1, 'Move queen 2 to row 4', 4), '14445'), ((0, '
, '31445'), ((3, 'Move queen 4 to row 2', 2), '31425')]
Resulting state:
31425
Total cost:
5
Viewer stats:
{'max_fringe_size': 56, 'visited_nodes': 462, 'iterations': 462}
```

DLS Limit = 8:

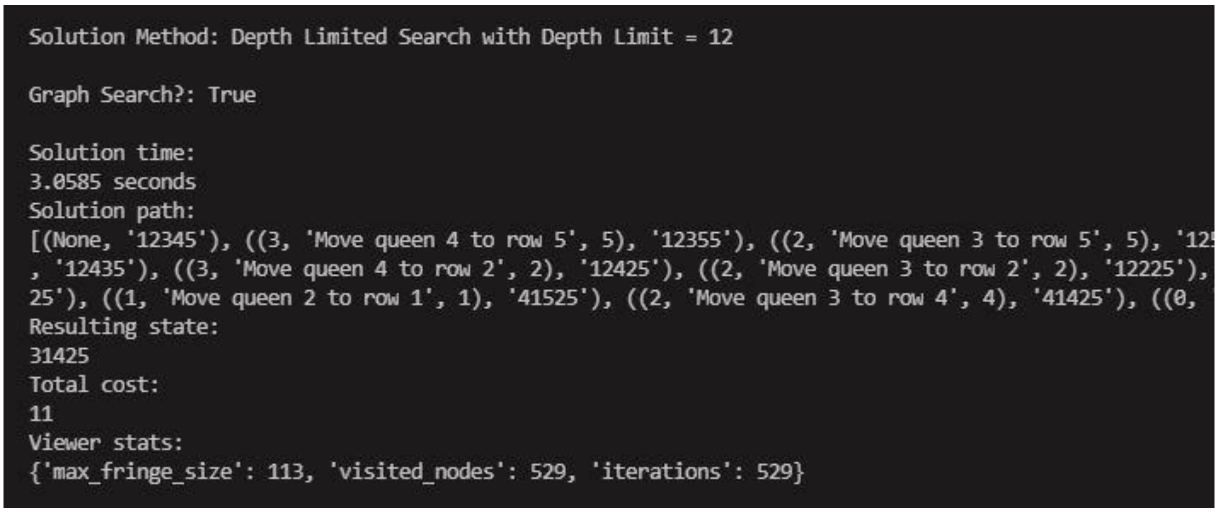


```
Solution Method: Depth Limited Search with Depth Limit = 8

Graph Search?: True

Solution time:
0.5889 seconds
Solution path:
[(None, '12345'), ((3, 'Move queen 4 to row 5', 5), '12355'), ((2, 'Move queen 3 to row 5', 5), '12555'), ((
, '12435'), ((3, 'Move queen 4 to row 2', 2), '12425'), ((1, 'Move queen 2 to row 5', 5), '15425'), ((0, 'Mo
25')]
Resulting state:
31425
Total cost:
8
Viewer stats:
{'max_fringe_size': 81, 'visited_nodes': 145, 'iterations': 145}
```

DLS Limit = 12:

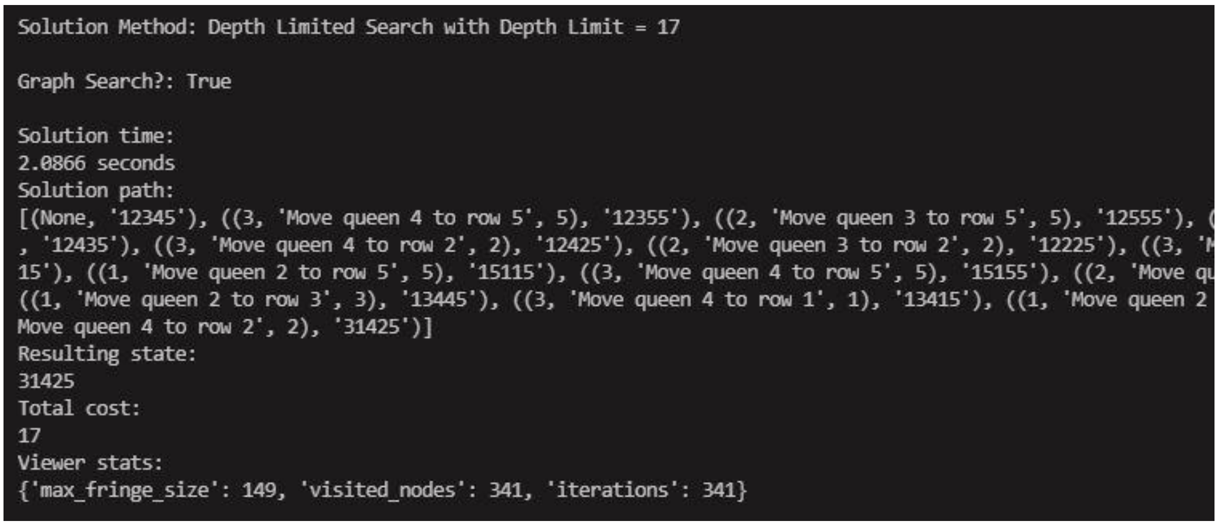


```
Solution Method: Depth Limited Search with Depth Limit = 12

Graph Search?: True

Solution time:
3.0585 seconds
Solution path:
[(None, '12345'), ((3, 'Move queen 4 to row 5', 5), '12355'), ((2, 'Move queen 3 to row 5', 5), '125
, '12435'), ((3, 'Move queen 4 to row 2', 2), '12425'), ((2, 'Move queen 3 to row 2', 2), '12225'),
25'), ((1, 'Move queen 2 to row 1', 1), '41525'), ((2, 'Move queen 3 to row 4', 4), '41425'), ((0,
Resulting state:
31425
Total cost:
11
Viewer stats:
{'max_fringe_size': 113, 'visited_nodes': 529, 'iterations': 529}
```

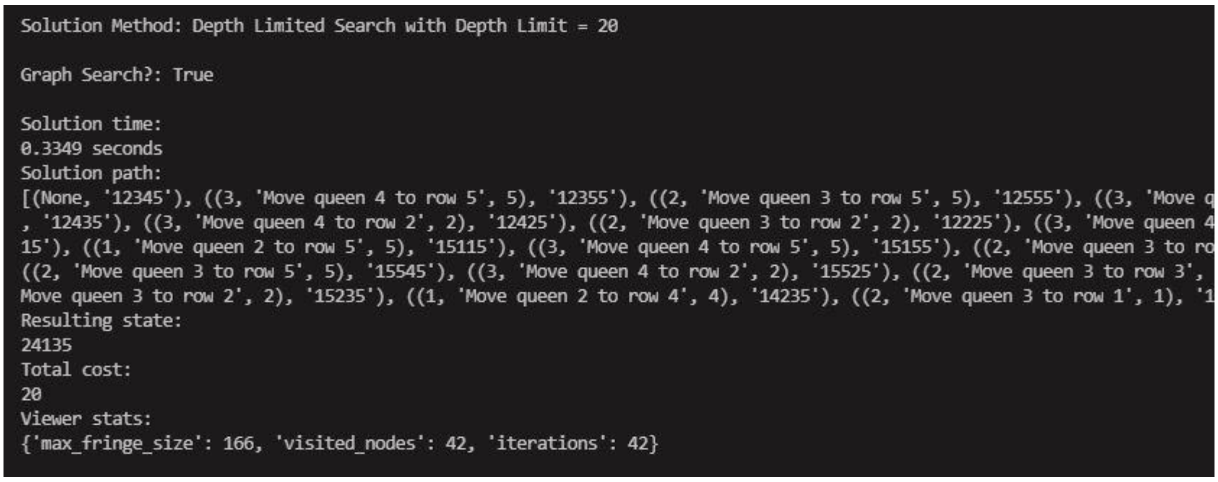
DLS Limit = 17:


```
Solution Method: Depth Limited Search with Depth Limit = 17

Graph Search?: True

Solution time:
2.0866 seconds
Solution path:
[(None, '12345'), ((3, 'Move queen 4 to row 5', 5), '12355'), ((2, 'Move queen 3 to row 5', 5), '12555'), (
, '12435'), ((3, 'Move queen 4 to row 2', 2), '12425'), ((2, 'Move queen 3 to row 2', 2), '12225'), ((3, 'M
15'), ((1, 'Move queen 2 to row 5', 5), '15115'), ((3, 'Move queen 4 to row 5', 5), '15155'), ((2, 'Move qu
((1, 'Move queen 2 to row 3', 3), '13445'), ((3, 'Move queen 4 to row 1', 1), '13415'), ((1, 'Move queen 2
Move queen 4 to row 2', 2), '31425')]
Resulting state:
31425
Total cost:
17
Viewer stats:
{'max_fringe_size': 149, 'visited_nodes': 341, 'iterations': 341}
```

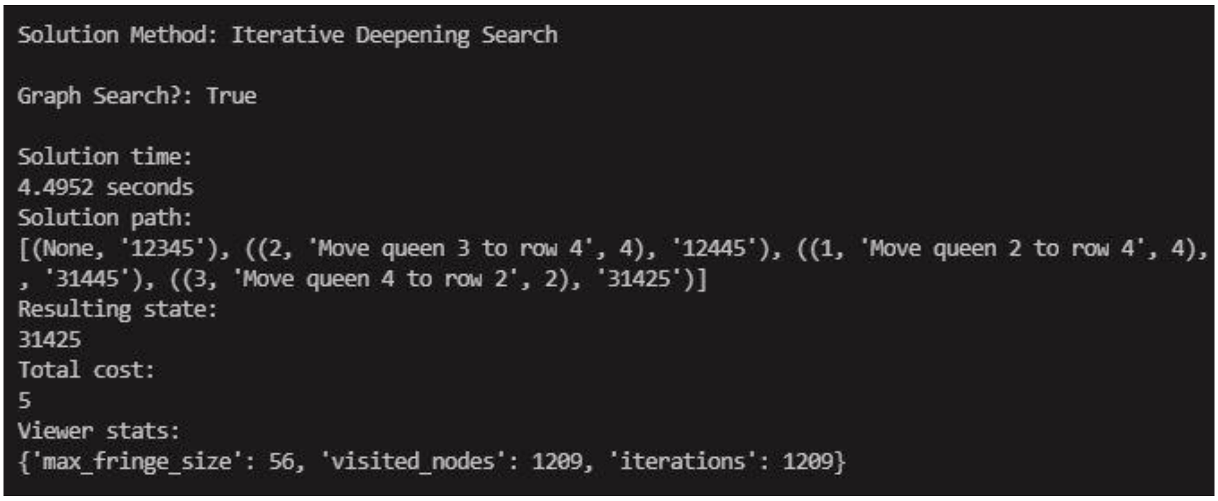
DLS Limit = 20:


```
Solution Method: Depth Limited Search with Depth Limit = 20

Graph Search?: True

Solution time:
0.3349 seconds
Solution path:
[(None, '12345'), ((3, 'Move queen 4 to row 5', 5), '12355'), ((2, 'Move queen 3 to row 5', 5), '12555'), ((3, 'Move q
, '12435'), ((3, 'Move queen 4 to row 2', 2), '12425'), ((2, 'Move queen 3 to row 2', 2), '12225'), ((3, 'Move queen 4
15'), ((1, 'Move queen 2 to row 5', 5), '15115'), ((3, 'Move queen 4 to row 5', 5), '15155'), ((2, 'Move queen 3 to ro
((2, 'Move queen 3 to row 5', 5), '15545'), ((3, 'Move queen 4 to row 2', 2), '15525'), ((2, 'Move queen 3 to row 3',
Move queen 3 to row 2', 2), '15235'), ((1, 'Move queen 2 to row 4', 4), '14235'), ((2, 'Move queen 3 to row 1', 1), '1
Resulting state:
24135
Total cost:
20
Viewer stats:
{'max_fringe_size': 166, 'visited_nodes': 42, 'iterations': 42}
```

IDS:

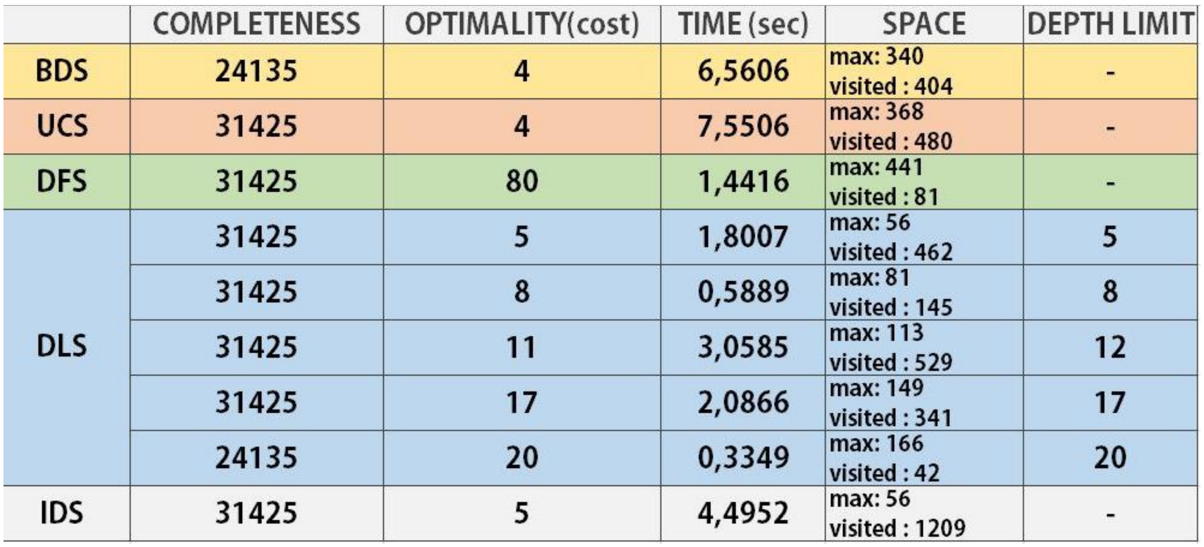

```
Solution Method: Iterative Deepening Search

Graph Search?: True

Solution time:
4.4952 seconds
Solution path:
[(None, '12345'), ((2, 'Move queen 3 to row 4', 4), '12445'), ((1, 'Move queen 2 to row 4', 4),
, '31445'), ((3, 'Move queen 4 to row 2', 2), '31425')]
Resulting state:
31425
Total cost:
5
Viewer stats:
{'max_fringe_size': 56, 'visited_nodes': 1209, 'iterations': 1209}
```

Discussion for PA2

Here is all solutions' statistics. So, we can compare them:

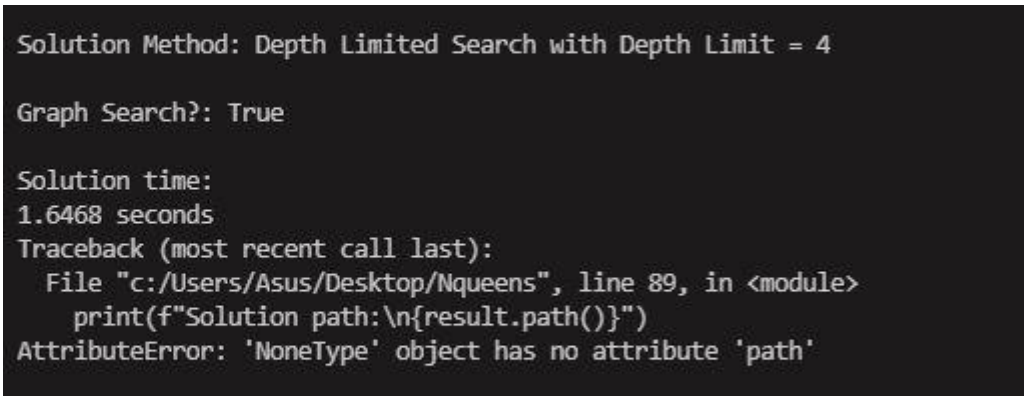


	COMPLETENESS	OPTIMALITY(cost)	TIME (sec)	SPACE	DEPTH LIMIT
BDS	24135	4	6,5606	max: 340 visited : 404	-
UCS	31425	4	7,5506	max: 368 visited : 480	-
DFS	31425	80	1,4416	max: 441 visited : 81	-
DLS	31425	5	1,8007	max: 56 visited : 462	5
	31425	8	0,5889	max: 81 visited : 145	8
	31425	11	3,0585	max: 113 visited : 529	12
	31425	17	2,0866	max: 149 visited : 341	17
	24135	20	0,3349	max: 166 visited : 42	20
IDS	31425	5	4,4952	max: 56 visited : 1209	-

When we look at the table, we see that all the methods return a solution. Let's evaluate all these solutions in terms of the parameters in the columns we see:

Our first column is "completeness". In other words, it tells whether the method gives us the solution if the problem has a solution. As we can see, a problem can have more than one solution, and all our algorithms have returned a solution, albeit different. But here, some of our algorithms cannot provide solutions for some values. For example, when the DLS algorithm has limit = 4, although our problem has a solution, it cannot give us the solution. When we set our limit to 5, the algorithm gives a solution. This means that our solution depth at the closest distance to the root should be at least 5. In this case, we can see how much the limit value given for the DLS algorithm affects the results.

DLS Limit = 4:



```
Solution Method: Depth Limited Search with Depth Limit = 4

Graph Search?: True

Solution time:
1.6468 seconds
Traceback (most recent call last):
  File "c:/Users/Asus/Desktop/Nqueens", line 89, in <module>
    print(f"Solution path:\n{result.path()}")
AttributeError: 'NoneType' object has no attribute 'path'
```

The second column is "optimality". In other words, it is a parameter that shows whether the solution given by the algorithm is the optimal solution. The smallest value we see at N=5 for this state is 4. We can say that the optimal cost is 4. According to this value, we see that the BDS and UCS algorithms give the optimal solution. In addition, we can say that the IDS algorithm gives the first solution at the closest distance to the root in the closest way to the optimal cost. We can state that the DFS algorithm is the costliest algorithm. Last but not least, we can definitely say that the cost value of the DLS algorithm has a very tight relationship with the given depth limit.

Let's consider the third and fourth columns together: "time", that is the parameter that indicates the algorithm's problem-solving speed, and "space", that is, the parameter that indicates the space used by the algorithm while solving the problem.

When we look at the table, we see that the UCS algorithm is the slowest but the algorithm that gives the optimal solution. While the UCS algorithm was running, 404 nodes were visited, and the maximum fringe reached up to 340. This means that the UCS algorithm is rather weak in terms of time and space.

The situation of the BFS algorithm is not different from UCS, but we can say that the BFS algorithm is slightly good than UCS in binary comparison.

Although the DFS algorithm reached 441 fringe size, it only visited 81 nodes and gave the solution in a relatively short time of approximately 1.5 seconds. However, the DFS algorithm had to keep 441 possible nodes in memory at the same time. This is a worse result than UCS and BFS for such a far from optimal solution.

When we look at the DLS algorithm, we see that the time-space relation is not that simple. In connection with the given depth limit, if our limit is an accurate limit, for example limit 20 case, although not optimal, a solution can be found in a short time using less space. Here we can say with certainty that the given depth limit is directly proportional to the max size, but the same is not true for "time".

The IDS algorithm is almost in the middle of the top-bottom values in this whole table for each parameter. In other words, IDS approaches the optimal solution, uses minimum space, but visits many nodes, so it is struggling in the time parameter. We can say that IDS performed well above the average excluding time.

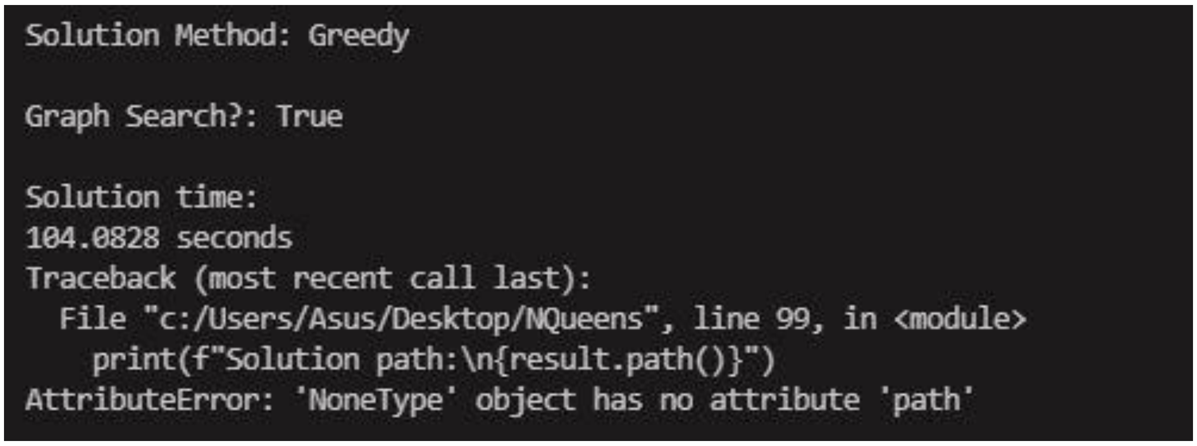
AI - PA3

We added attacking pair number as a heuristic function.

Results for PA3

When we run our code for N = 7 or N = 8, these algorithms didn’t give any solution. Code works, there is no problem with working. Because we tried the code in already arranged or nearly arranged states and code has gave the solution properly. So, we had to show our code’s result in N = 6 (also in this case code works sometimes but sometimes not) :

Greedy N = 7 :

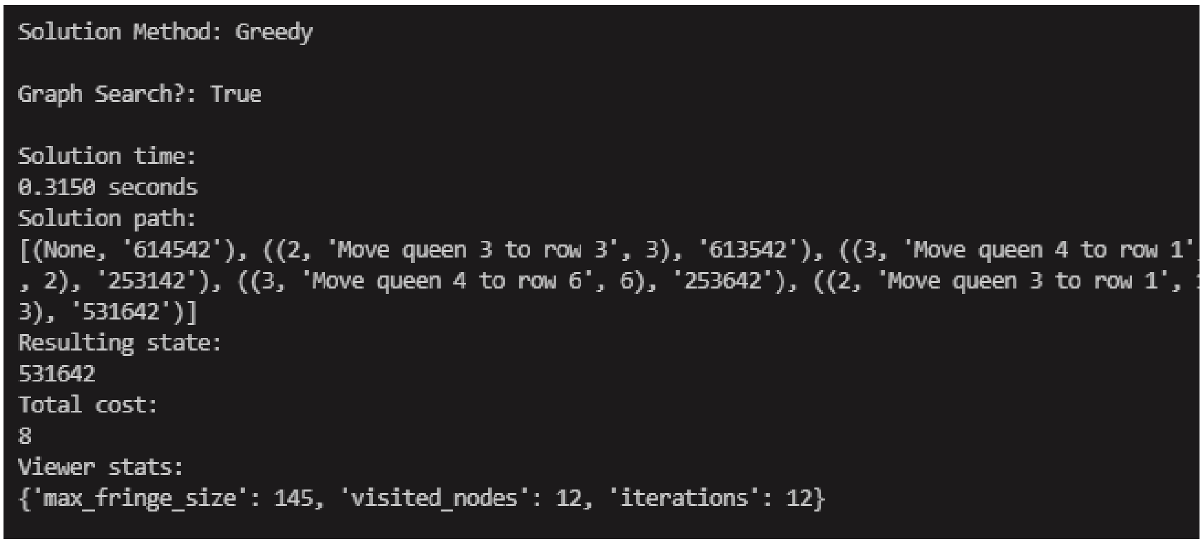


```
Solution Method: Greedy

Graph Search?: True

Solution time:
104.0828 seconds
Traceback (most recent call last):
  File "c:/Users/Asus/Desktop/NQueens", line 99, in <module>
    print(f"Solution path:\n{result.path()}")
AttributeError: 'NoneType' object has no attribute 'path'
```

Greedy N = 6 :

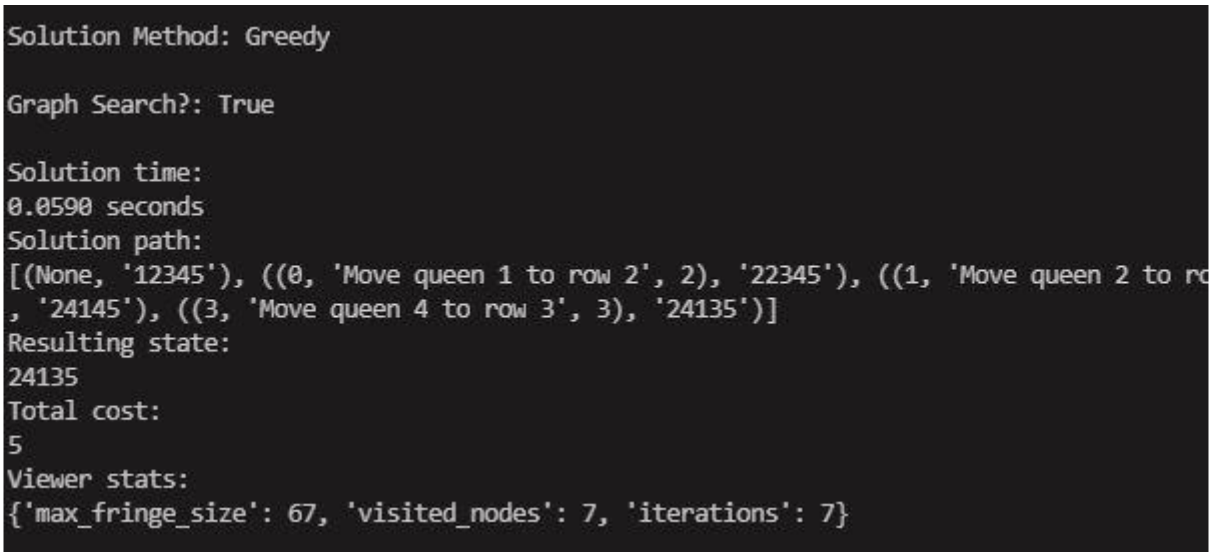


```
Solution Method: Greedy

Graph Search?: True

Solution time:
0.3150 seconds
Solution path:
[(None, '614542'), ((2, 'Move queen 3 to row 3', 3), '613542'), ((3, 'Move queen 4 to row 1'
, 2), '253142'), ((3, 'Move queen 4 to row 6', 6), '253642'), ((2, 'Move queen 3 to row 1', 
3), '531642')]
Resulting state:
531642
Total cost:
8
Viewer stats:
{'max_fringe_size': 145, 'visited_nodes': 12, 'iterations': 12}
```

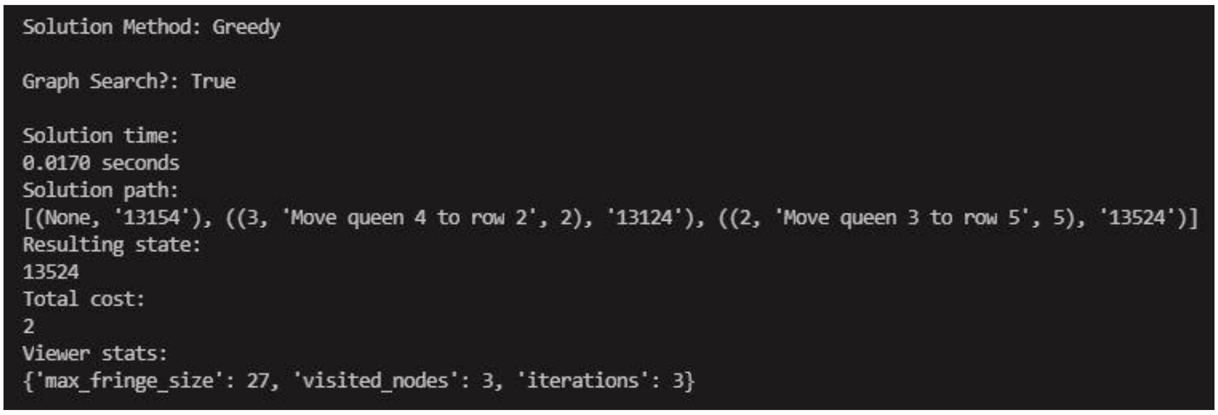
Greedy N = 5



```
Solution Method: Greedy

Graph Search?: True

Solution time:
0.0590 seconds
Solution path:
[(None, '12345'), ((0, 'Move queen 1 to row 2', 2), '22345'), ((1, 'Move queen 2 to ro
, '24145'), ((3, 'Move queen 4 to row 3', 3), '24135')]
Resulting state:
24135
Total cost:
5
Viewer stats:
{'max_fringe_size': 67, 'visited_nodes': 7, 'iterations': 7}
```

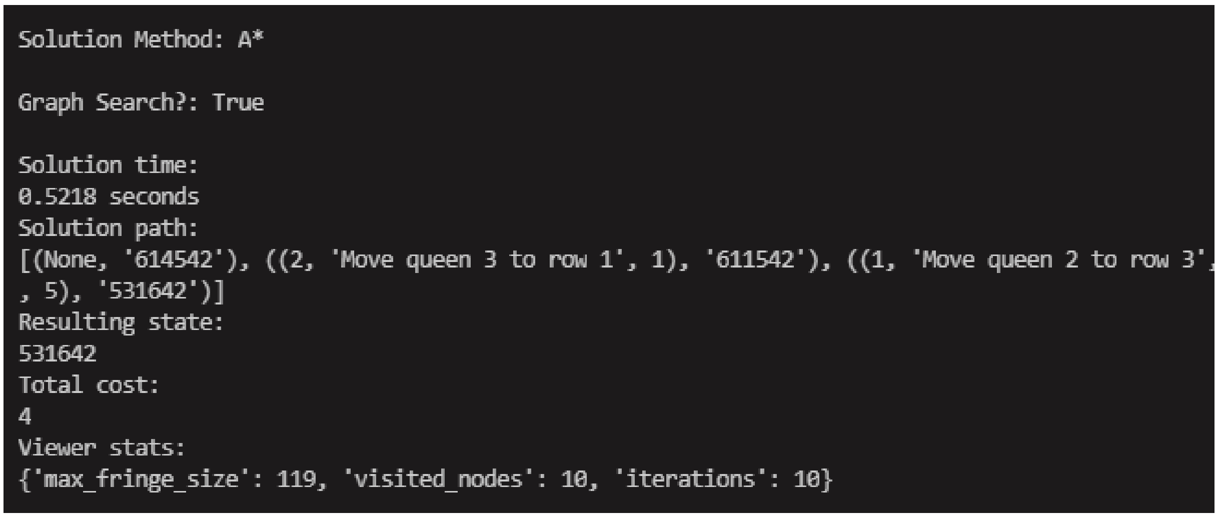


```
Solution Method: Greedy

Graph Search?: True

Solution time:
0.0170 seconds
Solution path:
[(None, '13154'), ((3, 'Move queen 4 to row 2', 2), '13124'), ((2, 'Move queen 3 to row 5', 5), '13524')]
Resulting state:
13524
Total cost:
2
Viewer stats:
{'max_fringe_size': 27, 'visited_nodes': 3, 'iterations': 3}
```

A* N = 6:

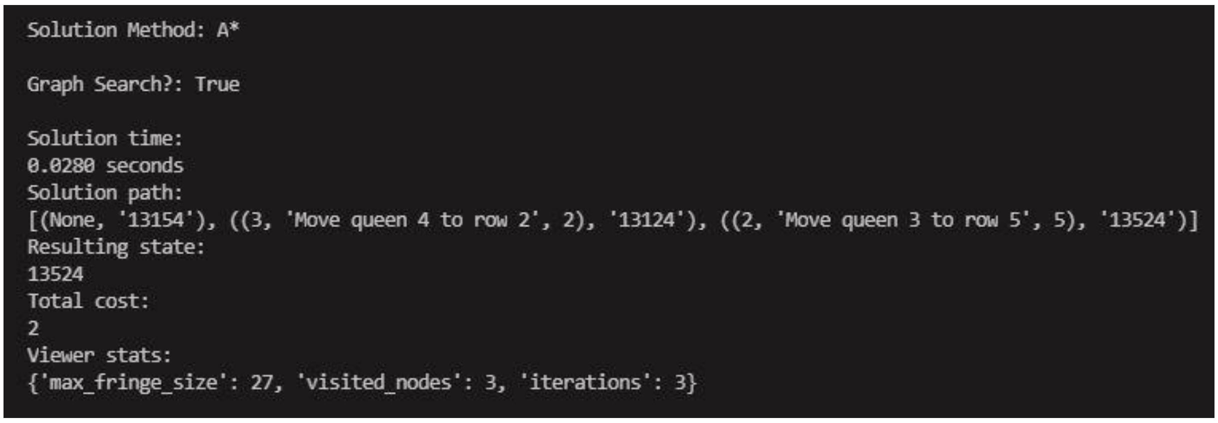


```
Solution Method: A*

Graph Search?: True

Solution time:
0.5218 seconds
Solution path:
[(None, '614542'), ((2, 'Move queen 3 to row 1', 1), '611542'), ((1, 'Move queen 2 to row 3',
, 5), '531642')]
Resulting state:
531642
Total cost:
4
Viewer stats:
{'max_fringe_size': 119, 'visited_nodes': 10, 'iterations': 10}
```

A* N = 5:



```
Solution Method: A*

Graph Search?: True

Solution time:
0.0280 seconds
Solution path:
[(None, '13154'), ((3, 'Move queen 4 to row 2', 2), '13124'), ((2, 'Move queen 3 to row 5', 5), '13524')]
Resulting state:
13524
Total cost:
2
Viewer stats:
{'max_fringe_size': 27, 'visited_nodes': 3, 'iterations': 3}
```

Results for PA3

All results for N = 6 and state "614542"

	Completeness	**Optimality**	**Time (s)**	**Space**
Greedy Search	531642	8	0.3150	Max: 145 Visited: 12
A* Search	531642	4	0.5218	Max: 119 Visited: 10

Looking at this table, unfortunately, we can say that our inputs are not suitable enough to properly test the algorithms. But still, in the light of these data; We can say that both methods give a solution, but the Greedy Search Algorithm gives a more costly path that is far from the optimal solution. Although the solution times are much shorter than the Uninformed Search Algorithms, we can say that the Greedy Algorithm stands out in terms of speed in binary comparison. On the other hand, we can say that the A* algorithm used space more efficiently compared to Greedy and returned an optimal solution.