200316059 İkram Celal KESKİN

200316045 Mert KARDAŞ

200316011 Musa Sina ERTUĞRUL

**NQueens Search Algorithm Report**

**A – Development Environment**

We developed and tested the code in a Python 3.x environment. The operating system used was Windows 10. The code was written and executed in Visual Studio Code, a popular IDE for Python development. The CPU properties include an AMD Ryzen 3 3250U with Radeon Graphics processor.

**B – Problem Formulation**

The problem addressed in the code is the N-Queens problem, where N queens are placed on an N×N chessboard in such a way that no two queens threaten each other. The problem is formulated as a search problem using the SimpleAI library. Here are the key components:

* **State Specification**: The state is represented as a string of length N, where each character corresponds to the row position of the queen in the respective column.
* **Initial State**: The initial state is either manually entered by the user or generated randomly. The validity of the state is ensured through the \_is\_valid method.
* **Possible Actions**: The possible actions represent moving a queen to a different row within its column.
* **Transition Model**: The result method generates a new state by applying an action to the current state.
* **Goal Test**: The goal is reached when there are no attacking pairs of queens on the board.
* **Path Cost**: The cost of each step is constant, set to 1.

**C- Testing**

1. Experiment 1- N = 4, Initial State = "2323"

1.a) A\* Algorithm

Resulting State: “2413”

Resulting Path: [(None, '2323'), ((1, '4'), '2423'), ((2, '1'), '2413')]

Cost of Solution: 1

Viewer Statistics: {'max\_fringe\_size': 12, 'visited\_nodes': 2, 'iterations': 2}

1.b) Breadth-First Algorithm

Resulting State: “2413”

Resulting Path: [(None, '2323'), ((1, '4'), '2423'), ((2, '1'), '2413')]

Cost of Solution: 2

Viewer Statistics: {'max\_fringe\_size': 870, 'visited\_nodes': 80, 'iterations': 80}

1.c) Depth-First Algorithm

Resulting State: “3142”

Resulting Path: [(None, '2421'), ((3, '4'), '2424'), ((2, '4'), '2444'), ((3, '3'), '2443'), ((2, '3'), '2433'), ((3, '2'), '2432'), ((2, '1'), '2412'), ((1, '3'), '2312'), ((3, '4'), '2314'), ((2, '3'), '2334'), ((3, '1'), '2331'), ((2, '4'), '2341'), ((1, '2'), '2241'), ((3, '2'), '2242'), ((2, '2'), '2222'), ((3, '3'), '2223'), ((2, '1'), '2213'), ((1, '1'), '2113'), ((3, '1'), '2111'), ((0, '4'), '4111'), ((3, '4'), '4114'), ((2, '4'), '4144'), ((3, '3'), '4143'), ((2, '3'), '4133'), ((3, '2'), '4132'), ((2, '2'), '4122'), ((1, '4'), '4422'), ((3, '3'), '4423'), ((2, '1'), '4413'), ((1, '3'), '4313'), ((0, '3'), '3313'), ((3, '1'), '3311'), ((2, '2'), '3321'), ((3, '4'), '3324'), ((2, '4'), '3344'), ((3, '2'), '3342'), ((1, '1'), '3142')]

Cost of Solution: 36

Viewer Statistics: {'max\_fringe\_size': 190, 'visited\_nodes': 93, 'iterations': 93}

1. Experiment 2- N = 4, Initial State = "4311"

2.a) A\* Algorithm

Resulting State: “2413”

Resulting Path: [(None, '4311'), ((3, '3'), '4313'), ((1, '4'), '4413'), ((0, '2'), '2413')]

Cost of Solution: 3

Viewer Statistics: {'max\_fringe\_size': 78, 'visited\_nodes': 8, 'iterations': 8}

2.b) Breadth-First Algorithm

Resulting State: “2413”

Resulting Path: [(None, '4311'), ((0, '2'), '2311'), ((1, '4'), '2411'), ((3, '3'), '2413')]

Cost of Solution: 3

Viewer Statistics: {'max\_fringe\_size': 4082, 'visited\_nodes': 372, 'iterations': 372}

2.c) Depth-First Algorithm

Resulting State: “2413”

Resulting Path: [(None, '3333'), ((3, '4'), '3334'), ((2, '4'), '3344'), ((3, '2'), '3342'), ((2, '2'), '3322'), ((3, '1'), '3321'), ((2, '1'), '3311'), ((1, '4'), '3411'), ((3, '4'), '3414'), ((2, '2'), '3424'), ((3, '3'), '3423'), ((2, '4'), '3443'), ((1, '2'), '3243'), ((3, '1'), '3241'), ((2, '3'), '3231'), ((3, '2'), '3232'), ((2, '1'), '3212'), ((1, '1'), '3112'), ((3, '3'), '3113'), ((0, '4'), '4113'), ((3, '4'), '4114'), ((2, '4'), '4144'), ((3, '2'), '4142'), ((2, '3'), '4132'), ((3, '1'), '4131'), ((2, '2'), '4121'), ((1, '4'), '4421'), ((3, '2'), '4422'), ((2, '1'), '4412'), ((1, '3'), '4312'), ((0, '2'), '2312'), ((3, '4'), '2314'), ((2, '2'), '2324'), ((3, '3'), '2323'), ((2, '4'), '2343'), ((3, '1'), '2341'), ((2, '3'), '2331'), ((1, '4'), '2431'), ((3, '4'), '2434'), ((2, '4'), '2444'), ((3, '2'), '2442'), ((1, '2'), '2242'), ((2, '2'), '2222'), ((3, '1'), '2221'), ((2, '1'), '2211'), ((3, '3'), '2213'), ((1, '4'), '2413')]

Cost of Solution: 46

Viewer Statistics: {'max\_fringe\_size': 193, 'visited\_nodes': 74, 'iterations': 74}

1. Experiment 3- N = 4, Initial State = "3442"

3.a) A\* Algorithm

Resulting State: “3142”

Resulting Path: [(None, '3442'), ((1, '1'), '3142')]

Cost of Solution: 1

Viewer Statistics: {'max\_fringe\_size': 12, 'visited\_nodes': 2, 'iterations': 2}

3.b) Breadth-First Algorithm

Resulting State: “3142”

Resulting Path: [(None, '3442'), ((1, '1'), '3142')]

Cost of Solution: 1

Viewer Statistics: {'max\_fringe\_size': 45, 'visited\_nodes': 5, 'iterations': 5}

3.c) Depth-First Algorithm

Resulting State: “2413”

Resulting Path: [(None, '1443'), ((3, '4'), '1444'), ((2, '3'), '1434'), ((3, '2'), '1432'), ((2, '2'), '1422'), ((3, '1'), '1421'), ((2, '1'), '1411'), ((1, '3'), '1311'), ((3, '4'), '1314'), ((2, '2'), '1324'), ((3, '3'), '1323'), ((2, '3'), '1333'), ((1, '2'), '1233'), ((3, '1'), '1231'), ((2, '4'), '1241'), ((3, '2'), '1242'), ((2, '1'), '1212'), ((1, '1'), '1112'), ((3, '3'), '1113'), ((0, '4'), '4113'), ((3, '4'), '4114'), ((2, '4'), '4144'), ((3, '2'), '4142'), ((2, '3'), '4132'), ((3, '1'), '4131'), ((2, '2'), '4121'), ((1, '3'), '4321'), ((3, '2'), '4322'), ((2, '1'), '4312'), ((1, '4'), '4412'), ((0, '3'), '3412'), ((3, '4'), '3414'), ((2, '2'), '3424'), ((3, '3'), '3423'), ((2, '3'), '3433'), ((3, '1'), '3431'), ((2, '4'), '3441'), ((1, '3'), '3341'), ((3, '4'), '3344'), ((2, '3'), '3334'), ((3, '2'), '3332'), ((1, '2'), '3232'), ((2, '2'), '3222'), ((3, '1'), '3221'), ((2, '1'), '3211'), ((3, '3'), '3213'), ((2, '4'), '3243'), ((1, '1'), '3143'), ((0, '2'), '2143'), ((1, '3'), '2343'), ((2, '1'), '2313'), ((1, '4'), '2413')]

Cost of Solution: 51

Viewer Statistics: {'max\_fringe\_size': 192, 'visited\_nodes': 69, 'iterations': 69}

1. Experiment 4- N = 5, Initial State = "12345"

4.a) A\* Algorithm

Resulting State: “25314”

Resulting Path: [(None, '12345'), ((0, '2'), '22345'), ((1, '5'), '25345'), ((3, '1'), '25315'), ((4, '4'), '25314')]

Cost of Solution: 4

Viewer Statistics: {'max\_fringe\_size': 115, 'visited\_nodes': 7, 'iterations': 7}

4.b) Breadth-First Algorithm

Resulting State: “13524”

Resulting Path: [(None, '13154'), ((2, '5'), '13554'), ((3, '2'), '13524')]

Cost of Solution: 2

Viewer Statistics: {'max\_fringe\_size': 4827, 'visited\_nodes': 255, 'iterations': 255}

4.c) Depth-First Algorithm

Resulting State: “14253”

Resulting Path: [(None, '21243'), ((4, '5'), '21245'), ((3, '5'), '21255'), ((4, '4'), '21254'), ((3, '3'), '21234'), ((4, '2'), '21232'), …, ((2, '1'), '24115'), … , ((3, '5'), '54251'), ((2, '1'), '54151'), ((1, '5'), '55151'), …,((4, '3'), '14253')]

Cost of Solution: 441

Viewer Statistics: {'max\_fringe\_size': 2549, 'visited\_nodes': 462, 'iterations': 462}

1. Experiment 5- N = 5, Initial State = "13154"

5.a) A\* Algorithm

Resulting State: “13524”

Resulting Path: [(None, '13154'), ((3, '2'), '13124'), ((2, '5'), '13524')]

Cost of Solution: 2

Viewer Statistics: {'max\_fringe\_size': 39, 'visited\_nodes': 3, 'iterations': 3}

5.b) Breadth-First Algorithm

Resulting State: “13524”

Resulting Path: [(None, '13154'), ((2, '5'), '13554'), ((3, '2'), '13524')]

Cost of Solution: 2

Viewer Statistics: {'max\_fringe\_size': 4827, 'visited\_nodes': 255, 'iterations': 255}

5.c) Depth-First Algorithm

Resulting State: “24135”

Resulting Path: [(None, '22142'), ((4, '5'), '22145'), ((3, '5'), '22155'), ((4, '4'), '22154'), ((3, '3'), '22134'), ((4, '3'), '22133'), ((3, '2'), '22123'), ((4, '1'), '22121'), ((3, '1'), '22111'), ((2, '5'), '22511'), ((4, '5'), '22515'), ((3, '3'), '22535'), ((4, '2'), '22532'), ((3, '5'), '22552'), ((4, '3'), '22553'), ((3, '4'), '22543'), ((4, '4'), '22544'), ((3, '2'), '22524'), ((2, '4'), '22424'), ((4, '5'), '22425'), ((2, '3'), '22325'), ((4, '2'), '22322'), ((3, '1'), '22312'), ((4, '4'), '22314'), ((2, '2'), '22214'), ((4, '3'), '22213'), ((2, '4'), '22413'), ((1, '5'), '25413'), ((4, '5'), '25415'), ((3, '5'), '25455'), ((4, '4'), '25454'), ((3, '4'), '25444'), ((4, '2'), '25442'), ((3, '3'), '25432'), ((4, '1'), '25431'), ((3, '2'), '25421'), ((2, '5'), '25521'), ((4, '5'), '25525'), ((3, '4'), '25545'), ((2, '3'), '25345'), ((4, '3'), '25343'), ((3, '5'), '25353'), ((4, '2'), '25352'), ((2, '2'), '25252'), ((4, '1'), '25251'), ((3, '4'), '25241'), ((2, '1'), '25141'), ((1, '4'), '24141'), ((4, '4'), '24144'), ((3, '2'), '24124'), ((4, '5'), '24125'), ((3, '3'), '24135')]

Cost of Solution: 51

Viewer Statistics: {'max\_fringe\_size': 543, 'visited\_nodes': 52, 'iterations': 52}

1. Experiment 1- N = 6, Initial State = "536142"

6.a) A\* Algorithm

Resulting State: “531642”

Resulting Path: [(None, '536142'), ((2, '1'), '531142'), ((3, '6'), '531642')]

Cost of Solution: 2

Viewer Statistics: {'max\_fringe\_size': 117, 'visited\_nodes': 5, 'iterations': 5}

6.b) Breadth-First Algorithm

Resulting State: “246135”

Resulting Path: [(None, '141331'), ((0, '2'), '241331'), ((2, '6'), '246331'), ((3, '1'), '246131'), ((5, '5'), '246135')]

Cost of Solution: 4

Viewer Statistics: {'max\_fringe\_size': 11355, 'visited\_nodes': 3965, 'iterations': 3965}

6.c) Depth-First Algorithm

Resulting State: “362514”

Resulting Path: [(None, '234436'), ((5, '5'), '234435'), ((4, '6'), '234465'), ((5, '4'), '234464'), ((4, '5'), '234454'), ((5, '3'), '234453'), ((4, '4'), '234443'), ((5, '2'), '234442'), ((4, '2'), '234422'), ((5, '1'), '234421'), ((4, '1'), '234411'), ((3, '6'), '234611'), ((5, '6'), '234616'), ((4, '6'), '234666'), ((5, '3'), '234663'), ((4, '3'), '234633'), ((5, '4'), '234634'), ((4, '4'), '234644'), ((5, '5'), '234645'), ((4, '5'), '234655'), ((5, '2'), '234652'), ((3, '5'), '234552'), ((5, '6'), '234556'), ((4, '4'), '234546'), ((5, '1'), '234541'), ((4, '6'), '234561'), ((3, '3'), '234361'), ((5, '2'), '234362'), ((4, '3'), '234332'), ((3, '2'), '234232'), … ((4, '4'), '362514')]

Cost of Solution: 3240

Viewer Statistics: {'max\_fringe\_size': 40475, 'visited\_nodes': 13072, 'iterations': 13072}

1. Experiment 7- N = 6, Initial State = "532512"

7.a) A\* Algorithm

Resulting State: “362514”

Resulting Path: [(None, '532512'), ((0, '6'), '632512'), ((5, '4'), '632514'), ((1, '6'), '662514'), ((0, '3'), '362514')]

Cost of Solution: 4

Viewer Statistics: {'max\_fringe\_size': 146, 'visited\_nodes': 6, 'iterations': 6}

7.b) Breadth-First Algorithm

Resulting State: “415263”

Resulting Path: [(None, '453162'), ((1, '1'), '413162'), ((2, '5'), '415162'), ((3, '2'), '415262'), ((5, '3'), '415263')]

Cost of Solution: 4

Viewer Statistics: {'max\_fringe\_size': 18930, 'visited\_nodes': 9388, 'iterations': 9388}

7.c) Depth-First Algorithm

IT TAKES TOO LONG

**D – Discussion**

**Completeness and Optimality**

* **A**:\* A\* is complete and optimal, ensuring the optimal solution is found with an admissible heuristic. The heuristic used here is the count of attacking pairs.
* **Breadth-First**: Breadth-First is complete and optimal when all step costs are equal. It explores all nodes at the current depth before moving on to the next depth.
* **Depth-First**: Depth-First is not optimal because it might find a non-optimal solution if it reaches a goal state at a higher depth before finding the optimal solution at a shallower depth. It is also not complete in cases where there are infinite paths.

**Time and Space Complexity**

* **A\***:A\* has time and space complexity that depends on the heuristic's quality. In this case, the heuristic is admissible, ensuring optimality.
* **Breadth-First**: Breadth-First has higher time and space complexity compared to Depth-First. It explores all nodes at each depth level before moving on.
* **Depth-First**: Depth-First has lower memory requirements but may get stuck in deep paths. Its time complexity is heavily influenced by the depth of the solution.

**Conclusion**

In conclusion, the A\* algorithm tends to be the most reliable in finding optimal solutions, while Breadth-First is effective for relatively small problem sizes. Depth-First has its strengths in terms of memory efficiency but lacks optimality and completeness. The depth limit in DLS should be carefully chosen to balance completeness and memory usage.