**Artificial Intelligence Assignment1 Report**

2016025078 강덕영

**-Assignment1**

assignment1에서 N-Queens problem을 search 기반 알고리즘인 BFS, hill-climbing 그리고 CSP를 사용하여 해결합니다.

**-N-Queens problem**

N-Queens problem은 n x n 체스판에 n개의 queen들을 서로 공격하지 못하도록 queen들을 배치하는 문제입니다. queen은 모든 방향(위쪽, 오른쪽 위쪽, 오른쪽, 오른쪽 아래, 아래쪽, 왼쪽 아래, 왼쪽, 왼쪽 위)에 위치한 queen을 공격할 수 있으며, 공격거리에 제한이 없습니다.

<3x3 chess board>

    - \* -         '\*': a queen

     \* - -         '-': an empty sapce

     - - \*

3 x 3 체스판에 위 그림과 같이 queen들이 배치되어 있다면 첫번 째 column에 있는 queen이 두번 째 column에 있는 queen을 공격할 수 있으므로 정답이 아닙니다. 달리 말하면 현재의 state는 goal state가 아닙니다.

<4x4 chess board>

   - \* - -

 - - - \* '\*': a queen

  \* - - - '-': an empty sapce

  - - \* -

4 x 4 체스판에 위 그림과 같이 queen들이 배치되어 있다면, 어떤 queen 쌍도 서로 공격할 수 없으므로 정답이며, 현재 state를 goal state라고 합니다. 이와 같이, 체스판의 queen들의 배치에 따라 state가 달라집니다. BFS, hill-climbing, CSP 알고리즘은 queen들을 적절하게 배치시킴으로써 즉, state를 변경시키며 goal state를 찾는 방식으로 문제를 해결합니다.

체스판의 state를 배열로 나타내면 위 그림은 [2, 1, 3], 아래 그림은 [3, 1, 4, 2]가 됩니다. 세 알고리즘에 거쳐 현재 체스판의 state를 나타내기 위해, 일차원 배열 변수인 'state'를 사용하게 될 것입니다.

**-BFS**

한 state에서 가능한 다음 state들을 확장하고, 확장된 state들을 다시 순서대로 확장해 나가면서 search space를 탐색해 나가는 알고리즘입니다. 확장된 state들을 큐 자료구조에 저장한 후, 큐에서 순서대로 하나씩 state를 빼고 다시 해당 state에서 가능한 state를 확장하여 큐에 저장하는 방식으로 goal state를 탐색해 나갑니다. 알고리즘은 다음 함수로 이루어져 있습니다.

**1. bfs(n):**

bfs 함수는 queen의 개수인 n을 매개변수로 받습니다. 알고리즘은 시작 state인 startState를 큐에 넣음으로써 시작됩니다. 큐에 state가 존재하는 동안, 큐에서 state를 하나씩 꺼냅니다. 해당 state가 goal state인지 아닌지를 isGoalState 함수를 통해 판단 후, goal state가 맞으면 알고리즘은 종료됩니다. goal state가 아니라면 해당 state를 나중에 다시 탐색하지 않기 위해 집합 자료형 변수인 checkedState에 추가하고, 해당 state를 확장한 state들을 큐에 넣습니다.

**2. isGoalState(n, state)**

isGoalState 함수는 queen의 개수인 n과 각 column에서의 queen들의 row 값들을 가지는 state 배열을 매개변수로 받습니다. 만약 state가 [1, 1, 1, 1]이라면 모든 queen들이 체스판에서 첫번 째 row에 위치하는 것입니다.

함수 내부에서 state를 바탕으로 현재 체스판의 layout을 나타내는 2차원 배열인 chessBoard를 생성합니다. 그리고 체스판의 각 queen마다 오른쪽 위, 오른쪽, 오른쪽 아래 방향의 경로를 탐색하면서 해당 경로 상에 queen이 존재하는지 확인합니다. 경로 상에 queen이 하나도 없으면 goal state이므로 True를 반환합니다 경로 상에 queen이 하나라도 있으면 공격이 가능하다는 의미이므로 False를 반환합니다..

**-Hill-climbing**

랜덤한 state에서 시작하여 heuristic funtion의 값이 작아지거나 커지는 방향으로 state를 선택하며 search space를 탐색해 나가는 알고리즘입니다. N-Queens problem에서의 heuristic function은 서로 공격할 수 있는 queen 쌍의 개수를 반환합니다. 때문에, 함수 값이 작아지는 방향으로 state를 탐색하게 됩니다. goal state는 heuristic function의 값이 0인 state이며, 함수 값이 0이 되지 않고 local minimum에 수렴하거나, 특정 값에서 stuck되는 문제점을 방지하기 위해, 여러번 재시작을 합니다. 알고리즘은 다음 함수들로 구성되어 있습니다.

**1. hc(n)**

hc 함수는 queen의 개수인 n을 매개변수로 받습니다.

우선 무작위로 첫 번째 state를 만들고, create\_chessBoard 함수를 호출하여 체스판을 나타내는 2차원 배열인 chessBoard를 생성합니다. 그리고 해당 state, chessBoard 변수를 인자로 heuristic\_func 함수를 호출하여 첫번 째 state의 heuristic function의 값을 얻습니다.

첫번 째 state의 heuristic 값을 얻은 후에는, 현재 state와 chessBoard를 바탕으로 다음 state들의 heuristic function 값들을 구하고 그 중 가장 작은 값을 가지는 state를 탐색합니다. 그리고 또 해당 state의 다음 state들 중 가장 작은 heuristic function 값을 가지는 state를 탐색하는 과정을 반복합니다. 탐색할 다음 state와 그 state의 heuristic function 값을 구하는 과정은 get\_next\_state 함수가 수행합니다.

이러한 과정은 heuristic function 값이 0이 되면 종료되고 그 시점의 state가 goal state가 됩니다. 만약 이전 state보다 더 적은 heuristic function 값을 가지는 state가 없다면 탐색을 멈춥니다. 또한, 함수 값이 5회(MAXSTUCK\_CNT) 동안 그대로라면 local minimum에 stuck되었다고 판단하여 탐색을 멈춥니다. 이렇게 도중에 탐색을 그만두게 되어 goal state를 찾지 못하는 것을 막기 위해서 goal state를 찾을 때 까지 최대 30회(RESTART\_CNT)까지 재시작하도록 하였습니다.

**2. create\_chessBoard(n, state)**

create\_chessBoard 함수는 queen의 개수인 n과 각 column에서의 queen들의 row 값을 원소로 가지는 state 배열을 매개변수로 받습니다. n x n 이차원 배열을 생성 후, queen들의 위치를 -1로 지정하고 배열을 반환합니다.

**3. heuristic\_func(n, state, chessBoard)**

heuristic\_func 함수는 queen의 개수인 n, 각 column에서의 queen들의 row 값을 원소로 가지는 state 배열, 그리고 queen들의 위치를 표시한 2차원 배열인 chessBoard를 매개변수로 받습니다. queen의 위치를 기준으로 오른쪽 위, 오른쪽, 오른쪽 아래 방향으로 chessBoard를 탐색하면서 서로 공격할 수 있는 queen들의 쌍의 개수를 구한 후 반환합니다.

**4. get\_next\_state(n, state, chessBoard)**

get\_next\_state 함수는 heuristic\_func 함수와 같은 매개변수들을 받습니다. 함수 내부에서 현재 state를 확장하여 다음 state들의 heuristic function 값을 딕셔너리 자료형 변수인 states에 저장합니다. 확장된 state에 대한 함수 값들을 states에 저장하는 것이 끝나면, states를 정렬하여 가장 작은 함수 값을 가지는 state와 해당 heuristic function 값을 반환합니다.

현재의 state를 확장하는 과정은 다음과 같습니다. 현재의 state에서 각 column에 위치한 queen들을 순서대로 하나씩 다른 row에 위치시킵니다. 쉽게 말해, 한 column에서 하나의 queen을 위, 아래로 이동시키는 것이고, 이렇게 queen을 이동시켰을 때의 state와 해당 state의 heuristic function 값을 구하여 딕셔너리에 저장합니다. n x n chess board에서 state는 (n - 1) \* n 번 확장되며 그 중 가장 작은 heuristic functinon 값을 가지는 state와 함수 값을 반환합니다. 반환된 heuristic 값(h1)은 바로 직전에 탐색된 state의 heuristic 값(h0)과 비교되는데, h0보다 h1의 값이 작거나 같아야 탐색이 계속될 수 있습니다.

**-CSP**

각 queen들이 가질 수 있는 value들로 구성된 pool에서 가능한 value를 선택해 queen 순서대로 부여하고, 이로 인해 다른 queen들이 가질 수 없는 value들을 pool에서 제거해가는 방식으로 goal state에 도달하는 fowrad checking 기반 알고리즘입니다. 이 방식으로 각 column에 위치한 queen의 row 값을 queen 순서대로 차례차례 지정해 나가다가 모든 queen의 row 위치를 확정하게 되면 그 상태가 goal state가 됩니다. 만약 특정 queen의 row를 지정했더니 다른 queen이 위치할 row 값이 하나도 없게 된다면 즉, 어떤 곳에 위치시켜도 해당 queen이 공격받게 된다면, goal state에 도달할 수 없기 때문에 backtracking을 하게됩니다. 알고리즘은 다음 함수들로 구성되어 있습니다.

**1. csp(n)**

csp 함수는 queen의 개수인 n을 매개변수로 받습니다. 우선 모든 함수에 걸쳐 사용하게 될, queen의 개수를 나타내는 변수 N을 할당합니다. 또한, queen들의 배치 상태를 나타내기 위해 사용되는 state 변수를 초기화합니다. BFS와 hill-climbing과 달리, CSP 알고리즘에서는 state 변수에 queen들의 row 값들이 역순으로 저장됩니다.(따라서 goal state를 원래 순서대로 반환하기 위해 state[::-1]를 반환합니다)그리고 generate\_value\_pool 함수를 호출하여 각 queen들이 가질 수 있는 값들을 포함하고 있는 이차원 배열 pool을 생성합니다. 마지막으로, forward\_check 함수를 호출한 후 리턴 값이 True이면 goal state를 찾았다는 의미이기 때문에 state를 역순으로 반환합니다.

**2. generate\_value\_pool(n)**

generate\_value\_pool 함수는 n개의 queen들이 각각 가질 수 있는 value(row)값들을 포함하는 이차원 배열을 생성하고 반환합니다.

pool = [

    [1, 2, 3, 4], # queen1

    [1, 2, 3, 4], # queen2

    [1, 2, 3, 4], # queen3

    [1, 2, 3, 4]  # queen4

]

4 x 4 체스판이라면, pool은 위처럼 값을 가지게 됩니다. 아직 queen의 위치를 지정하지 않았기 때문에, 모든 queen들이 첫번째 row부터 네번째 row까지 중 어디에도 위치 가능합니다.

<4x4 chess board>

   \*  x  x  x

   -  x  -  -

   -  -  x  -

   -  -  -  x

   q1 q2 q3 q4

pool = [

    [1, 2, 3, 4], # queen1

    [-1, -1, 3, 4], # queen2

    [-1, 2, -1, 4], # queen3

    [-1, 2, 3, -1]  # queen4

]

첫번째 queen이 첫번째 row에 위치하는 상황에서의 pool은 위와 같습니다. 나머지 queen들이 위치하지 못하는 좌표는 pool에서 -1로 표시됩니다. 체스판의 [row][col]위치에 queen을 놓을 수 없음을 나타내기 위해 pool[col][row] 값이 -1으로 지정해줍니다.

**3. forward\_check(N, n, state, pool)**

forward\_check 함수는 가능한 value 값을 queen 순서대로 차례대로 부여하면서 goal state를 찾아가는 재귀함수입니다. 재귀호출을 반복하면서 N 값은 변하지 않지만, n 값은 queen의 row 위치가 지정될 때마다 1씩 줄어듭니다. 즉, 모든 queen들의 row 위치가 알맞게 정해졌다면 n 값은 0이 되고, 이는 forward\_check 함수를 끝내는 조건이 됩니다. 이때 state의 역순은 goal state입니다.

foward\_check 함수는 우선 targetValues 변수에 특정 queen이 가질 수 있는 value 값들을 가지고 있는 일차원 배열을 할당합니다. 그리고 targetValues를 인자로 is\_empty 함수를 호출하여 가능한 value 값이 존재하는지 확인합니다. 만약 없으면 False를 반환함으로써 backtracking을 시작하게 됩니다.

targetValues에 값이 존재한다면, 가능한 value를 하나 골라 state 배열에 반영하고, 변경된 state를 매개변수로 eliminate\_illegal\_value 함수를 호출하여 pool을 업데이트 해줍니다. 다시 말해, queen을 가능한 row에 위치시키고, 변경된 queen의 위치에 따라 다른 queen들이 위치할 수 있는 row 값들을 pool에서 업데이트 하는 것입니다. 그리고 queen의 위치를 옮김으로써 업데이트된 state 배열과 pool 배열을 인자로 foward\_check 함수를 재귀호출 하게됩니다. 이때 호출된 forward\_check 함수의 반환 값이 True라면, goal state를 찾았다는 의미이므로 알고리즘을 끝내기 위해 True를 반환합니다. (n 값이 0이 될때까지 재귀호출을 반복하다 True를 반환했을 것이므로) 그리고 이 모든 과정을 targetValues에 존재하는 나머지 value에 대해 반복함으로써 backtracking이 가능하게 합니다.

**4. is\_empty(values)**

is\_empty 함수는 일차원 배열인 values를 매개변수로 가집니다. 이 함수는 queen에 지정할 수 있는 value가 존재하면 True를, 존재하지 않으면 False를 반환합니다. 만약 두번 째 queen이 위치할 수 있는 row 값들이 존재하는지 확인하고 싶으면 함수의 인자로 pool[1]을 넘겨주면 됩니다.

pool = [

    [1, 2, 3, 4], # queen1

    [-1, -1, 3, 4], # queen2

    [-1, 2, -1, 4], # queen3

    [-1, 2, 3, -1]  # queen4

]

사진에서의 pool[1] 의 값은 [-1, -1, 3, 4] 입니다. 이는 두번 째 queen이 체스판의 3, 4번 째 row에 위치할 수 있음을 의미합니다.

**5. eliminate\_illegal\_value(N, n, state, pool)**

eliminate\_illegal\_value 함수는 state에 따라 pool을 업데이트하고 해당 pool을 반환하는 함수입니다. 이 함수는 forward\_check 함수 내부에서 state 값을 변경한 후, 변경된 state를 기반으로 업데이트된 새로운 pool을 생성하기 위해 호출됩니다. 기존 pool이 아닌 새로운 pool(updatedPool)을 생성하고 반환해야하므로 deepcopy를 사용합니다. 변경된 queen의 위치를 기준으로 오른쪽 위, 오른쪽, 오른쪽 아래의 좌표를 큐에 넣습니다. 그리고 큐에서 좌표를 하나씩 뽑으면서 대응된 updatedPool 배열의 위치에 -1를 지정한 후 updatedPool을 반환합니다. pool의 특정 위치의 값이 -1 이라는 것은 해당 위치에 대응되는 체스판에 queen이 위치할 수 없음을 의미합니다.

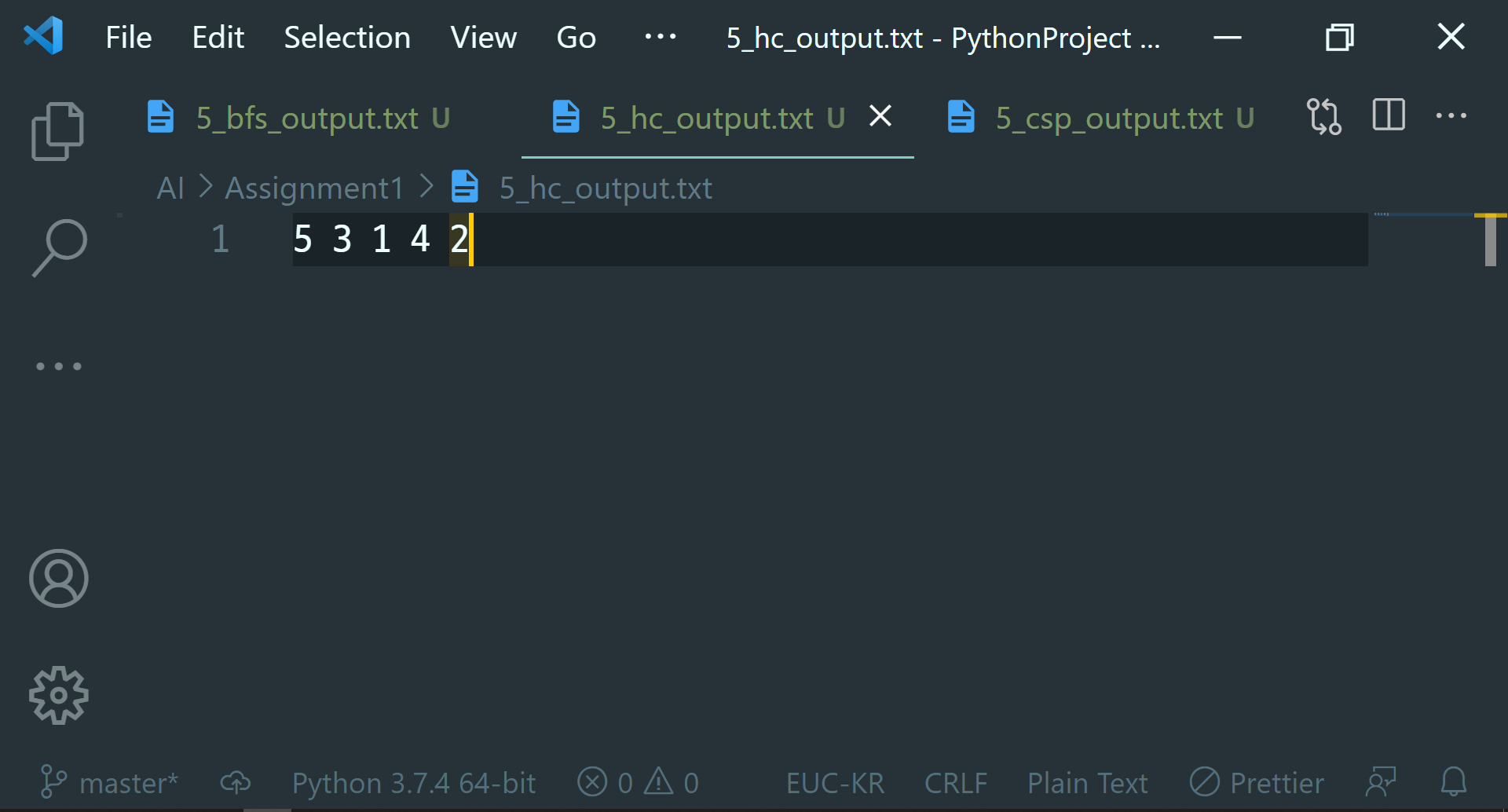
**-5-Queens Problem Solution**

다음은 각 알고리즘 별 5-queens problem에 대한 답입니다.

**1.BFS**



**2.Hill-Climbing**



**3.CSP**

