**인공지능 Assignment\_1 <N-Queens’s Problem>**

**컴퓨터소프트웨어학부 2016024957 이원석**

1. **함수설명 – 함수별 기능**

텍스트이(가) 표시된 사진

자동 생성된 설명

<전체함수 ScreenShot>

구현한 함수는 총 11개이며, 함수별 기능은 주석에 명시된 것과 같습니다.

1~3은 알고리즘 part 로, 4~11 의 함수와 함께 설명드리겠습니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

1. **heuristics는 현재 Board(Graph)의 휴리스틱값을 계산하는 함수입니다. Column이 아닌 Row를 기준으로 하였기때문에 휴리스틱 값 확인을 왼쪽 아래 대각선, 아래, 오른쪽 아래 대각선 방향에서 attack이 있는지 확인합니다.**

텍스트이(가) 표시된 사진

자동 생성된 설명

**5. placeQueen함수는 Queen을 특정 row에 배치하는 함수로서, 해당 row에 queen을 배치했으면, 다음 row에서도 queen을 배치할 수 있는지 보고, 가능하다면 true를 리턴합니다.**

텍스트이(가) 표시된 사진

자동 생성된 설명

6. fillZero함수는 단순히 size를 전달받아 0으로 가득 찬 (size x size) 배열을 만들어서, 반환해주는 함수입니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

**7. PlaceQueenRandomly는 랜덤으로 각 row별로 queen을 1개씩 배치하는 함수로서 queen을 어떻게 배치했는지에 대한 위치정보 locationInfo 를 반환합니다.**

텍스트이(가) 표시된 사진

자동 생성된 설명

8. CreateBoard함수는 전달받은 queen의 locationInfo를 바탕으로 Board를 생성하여 반환합니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

**9. getOptimalNeighbor 함수는 현재 상태에서 가능한 next 노드 (neigibor board)를 탐색하는데, 그 중 휴리스틱 값이 가장 낮은 Board를 찾아 해당 Board와 그 LocationInfo를 반환합니다.**

텍스트이(가) 표시된 사진

자동 생성된 설명

10. PrintBoard는 2차원 배열형식의 보드를 편하게 콘솔에 출력해주는 함수입니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

**11. chagneRowColumn함수는 제가 Queen을 놓을 때 Column 기준이 아닌 Row를 기준으로 하였고, 첫번째 칸의 index를 1이 아닌 0으로 두었기 때문에 나온 결과값을 output 형식에 맞춰서 변환해주는 함수입니다.**

1. **알고리즘 설명 – 각 알고리즘 동작**
2. **BFS**

텍스트이(가) 표시된 사진

자동 생성된 설명

BFS는 처음 초기화 부분과 휴리스틱 값 확인을 제외하고는 다른 함수의 도움을 받지않고 자체적으로 queue를 이용하여 구현했습니다. 먼저 입력받은 Size를 바탕으로 랜덤으로 Board를 생성하고, 해당 Board의 이웃노드를 탐색하며, 휴리스틱 값이 0인 Board를 발견 할 때까지, BFS를 수행하는 Brute Force 방식입니다. 하지만 해당 보드를 이미 탐색했는지 확인해서 중복 탐색을 방지하는 부분은 Board location 의 Scale이 너무 크기때문에 구현할 수 없었습니다. 따라서 Size가 커지면 속도가 매우 느려집니다.

이론적으로 모든 경우의 수를 찾기때문에, Board Size가 3이하여서 answer이 존재하지 않는 경우를 제외하고는 solution이 있기 때문에, size<4 인 경우를 제외하고는 반복문 내에서 no solution이 되는 경우는 구현하지 않았습니다. 다만 오랜시간동안 탐색을 했는데도 값을 찾지 못하면, 다음 알고리즘을 진행하기 위해서 10만번의 탐색을 넘어서면 종료하게 설정하였습니다.

1. **Hill Climbing**

텍스트이(가) 표시된 사진

자동 생성된 설명

Hill Climb는 먼저 입력받은 크기에 대한 Board를 무작위로 생성합니다. 이후 무한루프 내에서 getOptimalNeighbor() 함수를 통해 가장 휴리스틱 값이 낮은 neighbor Board를 계속해서 찾습니다. 다만 depth가 30을 초과했는데도 휴리스틱 값이 0 (safe state) 이 아니라면, Global 이 아닌 Local Optimal에 Stuck 된 상태이기때문에 다시 랜덤으로 Board를 초기화 해줍니다.

평균적으로 랜덤으로 보드를 초기화 했을때, 30.5번을 초기화하면 99%의 성공률을 보장하므로 randomSettingCount가 31회를 넘어서면 Search를 중단하고 no solution을 출력하도록 했습니다.

1. **CSP**

텍스트이(가) 표시된 사진

자동 생성된 설명

CSP는 다른 두 알고리즘과 다르게 Board를 랜덤으로 세팅하지 않고, Queen을 하나도 두지 않은 상태에서 placeQueen() 함수를 통해 각 row별로 하나씩 queen을 두면서, 해당 row에 queen을 두는게 possible한지 확인하는 알고리즘입니다. 각 상태는 다음 row에서 Safe한 상태가 가능할 때에만 Safe상태로 True를 반환하며 가능한 next stage에서 safe한 상태가 없다면, false를 반환합니다. Next stage로부터 True인지 Fasle인지가 결정되기때문에, BackTracking 형식으로 동작하게 되어있습니다.

1. **실험결과**

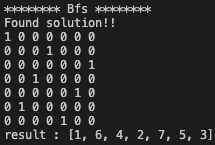
(가시성을 위하여 txt file이 아닌 console 결과를 첨부하였습니다.)

* 1. BFS
     1. Size = 5

텍스트이(가) 표시된 사진

자동 생성된 설명

* + 1. Size = 7



* 1. **Hill Climbing**
     1. Size = 5

텍스트이(가) 표시된 사진

자동 생성된 설명

* + 1. Size = 10

텍스트이(가) 표시된 사진

자동 생성된 설명

* 1. CSP
     1. Size = 5

텍스트이(가) 표시된 사진

자동 생성된 설명

* + 1. Size = 10

텍스트이(가) 표시된 사진

자동 생성된 설명