**<인공지능 assignment1 보고서>**

2019019016 서시언

**1. <2019019016\_assignment\_1.py>**

**- main()**

- input.txt 파일의 내용을 한 줄씩 읽어와서 해당하는 알고리즘을 실행해 그 결과를 output 파일에 작성하는 과정을 반복한다. (input.txt에 있는 N은 1이상의 정수라고 가정한다.)

**2. <bfs.py>**

**[함수설명]**

**- class State**

- 1~N열에 대한 퀸의 행 위치를 저장하는 array, queens와 N값을 가진다.

**- getNextState(self, pos)**

- queens를 deep copy한 newQueens에 매개변수로 받은 row position을 append해준 뒤,

newQueens와 n으로 새로운 State를 만들어 리턴한다.

**- isGoal(self)**

- for문을 돌면서 서로 잡을 수 있는 queen pair가 있다면 False를, 없다면 True를 리턴한다.

**- isLeaf(self)**

- queens의 길이가 n이면 True, 아니면 False를 리턴한다.

**- toString(self)**

- queens를 output file에 쓸 형태로 바꿔서 리턴한다.

**- bfs(n)**

- []와 n으로 State를 만들어 비어있는 array인, queue에 append해준다.

- queue가 빌 때까지, queue의 첫 번째에 있는 state를 pop해서

그 state가 leaf라면, goal인지 테스트해서, goal이면 결과를 리턴한다.

leaf가 아니라면, for문을 돌면서 queue에 새로운 state를 append한다. (expand)

- queue가 비었는데도 답을 찾지 못했다면 “no solution”을 리턴한다.

**[알고리즘 설명]**

**- class State**

- State class는 1~N열에 대한 퀸의 행 위치를 저장하는 array, queens와 n값을 가지는데, queens의 index는 열의 순서를 나타내고, index에 있는 값은 해당 열에서의 행의 위치를 나타낸다.

- 모든 열의 행 위치가 정해지는 leaf node에서의 queens의 length는 n이므로, State의 isLeaf()에서는 queens의 length가 n일 때 True를 리턴한다.

- N-queens 문제에서 goal은 서로 잡을 수 있는 queen pair가 하나도 없는 경우이므로, 모든 열 pair에 대해 같은 행에 위치한 queen이 있거나, 대각선에 위치한 queen이 있는지(열 index 차이와 행 index 차이가 같은지) 확인해서 하나도 없을 때만 True를 리턴한다.

**- bfs(n)**

- n은 n-queens problem의 n이다.

- bfs이므로 FIFO queue를 이용한다.

- 빈 queens를 갖는 State를 queue에 추가해주고, While문을 통해 queue가 빌 때까지 queue의 맨 앞에 있는 State를 pop해서 leaf인지 체크한다.

- 모든 열에서의 행 위치가 정해졌으면(Leaf이면), Goal인지 체크하고, Goal이라면 State의 toString()을 사용해output 파일에 맞는 형태로 결과를 리턴한다.

- 모든 열에서의 행 위치가 정해지지 않았으면(Leaf가 아니면), 아직 expand가 끝나지 않았으므로, append로 가능한 모든 state들을 queue에 추가해준다.

- 이때 첫번째 열부터 순차적으로 expand하므로, State의 getNextClass()를 호출하면 queens에 append를 통해 새로운 행 위치를 추가한 State를 리턴해준다.

- queue에 남은 State가 없는데도 goal을 찾지 못했다면 “no solution”을 리턴한다.

**[실험결과]**



**3. <hc.py>**

**[함수설명]**

**- class State**

- 1~N열에 대한 퀸의 행 위치를 저장하는 array, queens와 n값, heuristic값인 h를 가진다.

**- \_\_init\_\_(self, n)**

- queens에 0~n 사이의 값을 랜덤하게 n개 append한다.

**- getH(self, queens)**

- for문을 돌면서 서로 잡을 수 있는 queen pair의 수를 카운트해서, 리턴한다.

**- moveQueen(self)**

- h값을 minH에 저장하고, for문을 돌면서 queen을 옮길 수 있는 모든 경우에 대해 getH() 결과를 구해서,

그 값이 minH보다 작다면 minH값을 업데이트하고, 그 위치를 coord에 저장한다.

- for문을 빠져나왔을 때 minH값이 h보다 작다면 queens[coord[0]] = coord[1]로 만들고, h에 minH를 저장한다.

- for문을 나왔을 때 minH값이 h보다 작지 않다면 h에 -1을 저장한다.

**- toString(self)**

- queens를 output file에 쓸 형태로 바꿔서 리턴한다.

**- hc(n)**

- n이 2또는 3일때는 “no solution”을 리턴한다.

- while문에서 새로운 State를 생성하고, while문으로 state의 h값이 0보다 크면 moveQueen()을 계속 호출한다.

- 안쪽 while문에서 빠져나왔을 때 state의 h값이 0이면 state의 toString()의 결과를 리턴한다.

**[알고리즘 설명]**

**- class State**

- bfs와 마찬가지로 State class는 1~N열에 대한 퀸의 행 위치를 저장하는 array, queens와 n, h값을 가지는데, queens의 index는 열의 순서를 나타내고, index에 있는 값은 해당 열에서의 행의 위치를 나타내며, h는 heuristic 값으로, 현재 상태에서 서로 잡을 수 있는 queen pair의 수이다.

- heuristic 값이 현재 상태에서 서로 잡을 수 있는 queen pair의 수이므로, getH(self, queens)에서는 for문을 이용해 모든 열 pair에 대해 같은 행에 위치하거나, 대각선에 위치한 (열 index 차이와 행 index 차이가 같은) queen pair의 수를 카운트해서 리턴한다.

- hill climbing은 항상 heuristic값이 더 좋아지는 방향으로 움직이는데, n-queens problem의 경우 heuristic값이 0일 때가 goal이므로 항상 현재 상태보다 더 작은 heuristic값을 갖도록 queen을 움직여야 한다. 따라서 moveQueen()에서는 for문을 이용해 현재 상태에서 queen을 옮길 수 있는 모든 경우에 대해 heuristic값을 구해서 현재 값보다 작으면서 가장 작은 값을 갖는 상태로 queen을 옮긴다.

- 가장 작은 값을 갖는 위치가 여러 개라면, 그 중 먼저 발견한(열 값이 더 작고, 열 값이 같다면 행

값이 더 작은) 위치로 queen을 이동시킨다.

- 현재 값보다 더 작은 값을 갖는 위치가 없다면 peak에 도달했으므로, stuck되었다는 의미로 h에 -1을 저장한다.

**- hc(n)**

- n은 n-queens problem의 n이다.

- 정답이 존재한다면, 정답을 발견할 때까지 반복해서 랜덤한 위치에 queen들을 위치시키고 peak값에 도달할 때까지 heuristic값이 최대한 작아지도록 queen의 위치를 옮기는 과정을 반복하기 때문에, 정답이 없는 경우 (n이 2 또는 3일 때)는 while문이 시작되기 전에 미리 “no solution”을 리턴한다.

- n이 2 또는 3이 아니라면, while문에서 queen들이 랜덤한 위치에 있는 새로운 State를 생성하고 peak값에 도달할 때(state.h<=0이면 peak에 도달한 것)까지 moveQueen()을 반복해서 호출한다.

- peak값에 도달했을 때 state.h가 0이라면 서로 잡을 수 있는 queen pair가 0개 이므로 goal을 찾은 것이다. 따라서 결과값을 리턴해준다.

- state.h가 0이 아니라면, 다시 while문의 처음으로 돌아간다.

**[실험결과]**



**4. <csp.py>**

**[함수설명]**

**- class State**

- 각 열(variable)에 가능한 모든 value값(행 위치)를 저장하고 있는 variables와 n, assign된 variable 수를 카운트하는 varCnt를 가진다.

**- getNextState(self, value)**

- variables를 deep copy한 newVariables의 varCnt번째에 매개변수로 받은 value를 assign해준 뒤, (varCnt, value)의 대각선에 위치하거나 같은 행에 위치한 value들이 제거된 newVariables로 새로운 State를 만들어 리턴한다. (이때 varCnt값을 1 증가시킨다.)

**- isLegalValue(self, value)**

- variables[varCnt].count(value)가 0보다 크면 Ture, 아니면 False를 리턴한다.

**- isEmptyVariable(self)**

- variables[varCnt]의 legth가 0이면 Ture, 아니면 False를 리턴한다.

**- isGoal(self)**

- varCnt == n이면 True, 아니면 False를 리턴한다.

**- toString(self)**

- variables를 output file에 쓸 형태로 바꿔서 리턴한다.

**- csp(n)**

- 0 ~ n-1값을 가지는 array를 n개 갖는 이차원 배열과 n, 0으로 State를 만들어 비어있는 array인, stack에 append한다.

- while문을 이용해 stack이 빌 때까지, stack의 마지막에 있는 state를 pop해서 그 state의 isGoal()이 True이면 state의 toString()을 리턴하고, state의 isEmptyVariable()이 True라면 while문의 처음으로 돌아간다.

나머지 경우에는 for문으로 0~n-1값에 대해 state의 islegalValue()가 True라면 그 값을 파라미터로 getNextState()를호출하고, 리턴된 값을 스택에 append한다.

- stack이 비었는데도 답을 찾지 못했다면 “no solution”을 리턴한다.

**[알고리즘 설명]**

**- class State**

- 각 열(variable)에 가능한 모든 value값(행 위치)를 배열로 저장하고 있는 이차원배열 variables와 n, value가 assign된 variable 수를 카운트하는 varCnt를 가진다. varCnt는 현재 값이 할당될 variable을 가리키는 index로도 사용되며, 값이 할당된 variable은 할당된 값만이 들어있는 배열을 갖는다.

- getNextState(self, value)에서는 value값을 varCnt번째 value에 할당해주고, 아직 할당되지 않은 나머지 variable들에 대해 할당된 값으로 인해 더이상 legal value가 아니게 된 값들을 배열에서 제거해주고, 새로운 State를 만들어 리턴한다.

- isLegalValue()는 넘겨받은 value값이 현재 variable에서 legal value인지 체크하기 위해 count를 이용해 variables[varCnt]에 value가 있는지 확인한다.

- isEmptyVariable()은 현재 값이 할당될 variable에 legal value가 존재하는지 체크하기 위해 variables[varCnt]의 length를 확인한다.

- isGoal()은 현재 State가 goal state인지 확인하기 위해 varCnt가 n인지 확인한다. (variable에 값이 할당되어 있다면 반드시 legal value일 것이므로, 할당된 variable의 수가 n이면 goal state이다.)

**- csp(n)**

- n은 n-queens problem의 n이다.

- variable에 더이상 가능한 값이 없을 때 backtracking을 하기 위해 LIFO stack을 이용한다

- 모든 variable에 가능한 value 목록을 넣어서 만든 이차원 배열, n, varCnt로는 0을 넣어서 초기 State를 만들어 stack에 넣어준다.

- while문을 이용해 stack이 빌 때까지, stack의 맨 마지막에 있는 state를 꺼내 isGoal()을 이용해 goal State인지 확인하고 goal state라면 결과를 리턴하며, isEmptyVariable()을 이용해 현재 값을 할당할 variable에 legal value가 있는지 확인한다. legal value가 없다면 while문의 처음으로 돌아간다. (backtracking). legal value가 있다면 isLegalValue를 이용해 어떤 값이 legal value인지 찾고 그 값을 파라미터로 getNextState()를 호출해 리턴되는 값을 append를 이용해 stack에 추가한다.

- 이때 첫번째 열부터 순차적으로 값을 할당하므로, State의 getNextState()를 호출하면 varCnt번째 variable에 value값을 할당한 결과 state를 리턴해준다.

- stack이 비었는데도 goal을 찾지 못한다면 “no solution”을 리턴한다.

**[실험결과]**

