Programming Assignment 2

Local search : Hill climbing

2012003567

이상화

**1. 문제**

N-Queen 문제를 Hill climbing을 이용해서 해결한다. State를 각 colomn에서의 Queen의 위치로 하는데 초기값을 랜덤값으로 설정해야 한다.

**2. 문제 해결 과정**

Main class Chess.java와 heuristic function 및 각종 문제해결 메소들을 정한 Hillclimbing\_chess.java를 작성하였다. Hillclimbing\_chess 클래스에서 가장 중요한 메소드는 heuristic 값을 구하는 getHeuristic()과 hill climbing을 재귀적으로 수행해서 solution state를 구해내는 hill\_climbing() 메소드이다. heuristic값은 서로 공격을 하는 퀸의 짝의 개수로 정하고 heuristic 값이 0이 되면 그 state를 solution으로 하게 하였다.

**3. 코드 분석**

(1) Hillclimbing\_chess.java

Hillclimbing\_chess 클래스에는 문제해결을 위한 주요 메소드들이 정의되어 있는 클래스이다.

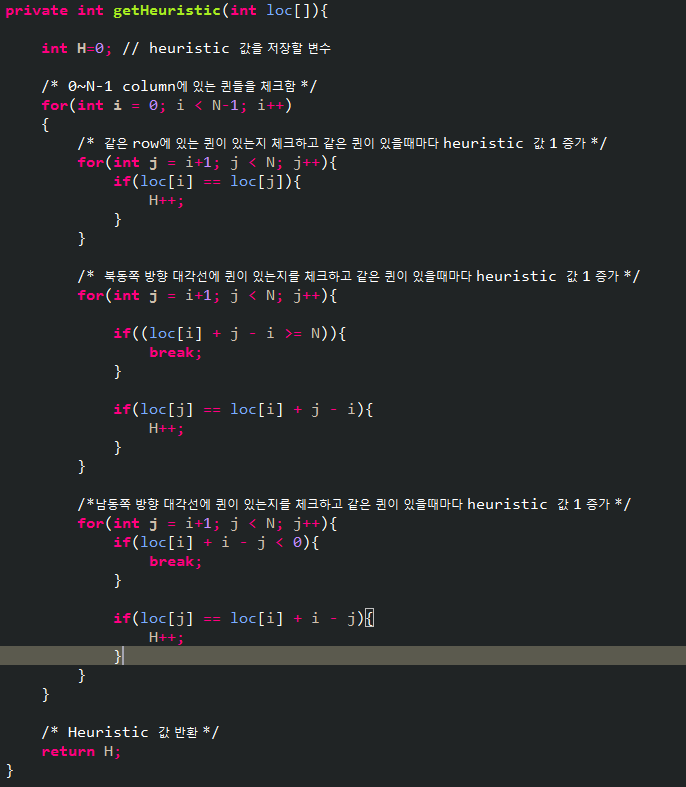
-solve() : 메인 클래스에서 solution을 구할 때 호출하는 메소드. 내부적으로 hillclimbing\_chess()를 호출해서 답을 구한다.

-getLocation() : 결과로 나온 솔루션을 출력하는 메소드이다 int[]형으로 반환하며 Chess.java

에서 파일출력을 위해 사용된다.

-getTime() : 경과시간을 출력하기 위한 메소드이다.

-getHeuristic() : 인자로 받은 state의 heuristic 값을 계산해서 반환해주는 메소드.



BFS,DFS,DFID를 이용해서 NQueen문제를 해결할 때 구현한 goal test 함수를 그대로 사용하였다. 바뀐 점은 H라는 heuristic 값을 저장하는 변수를 선언해서 서로 공격중인 Queen이 있을 때마다 +1을 해주어서 최종적으로 나온 H값을 heuristic 값으로 반환한다.

-hill\_climbing() : successor state들의 heuristic 값 중 가장 작은 값을 이전 state의 heuristic 값과비교해서 재귀적으로 문제를 푸는 메소드



successor\_temp는 heuristic 값을 계산하기 위해 successor state를 담고 있는 역할을 한다. 예를 들어 8X8 체스판에서 다음 state는 총 8X7개가 올 수 있는 8X7개 state를 successor\_temp에 담은 뒤 heuristic을 계산하면서 min\_heuristic 값보다 작은 heuristic 값이 나오면 successor\_best에 방금 검사한 state르르 복사한 뒤에 min 값을 갱신해준다. min\_heuristic의 초기값을 이전 state의 heuristic 값을 해주었기 때문에 8X7개를 검사해야 하지만 8X8개를 전부 계산해도 갱신되는 min 값에는 변화가 없다. 편의를 위해 8X8개를 전부 검사하였다.

If else statement가 하는 역할을 다음과 같다.

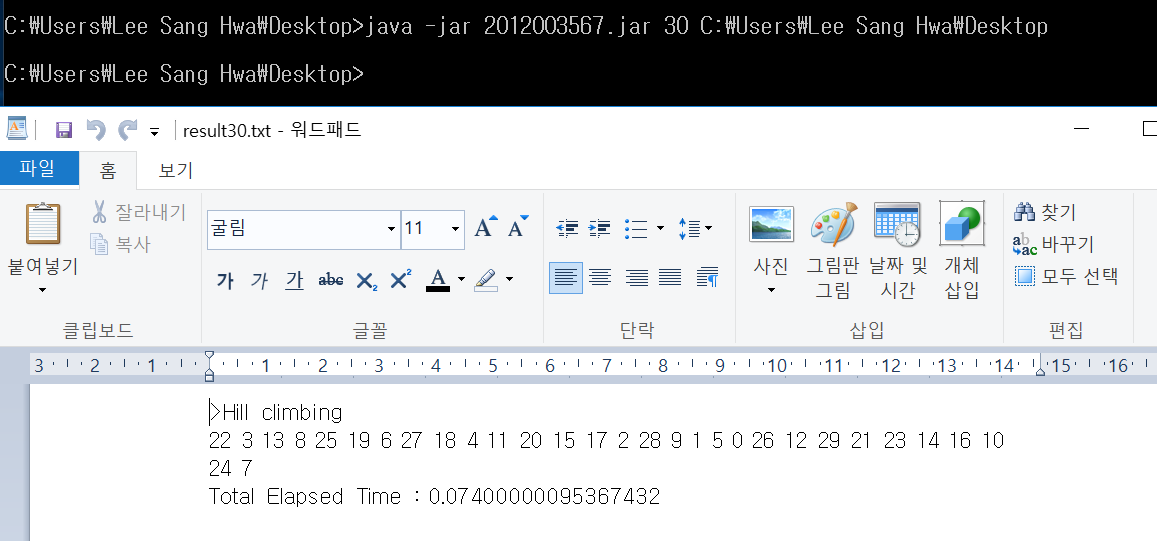
successor를 전부 순회하면서 min heuristic 값을 추출한 뒤에 min값이 0이라는 의미는 그 successor state가 solution이라는 의미이므로 그 state를 return 해준다.

만약 min heuristic이 이전 state의 heuristic보다 작아졌다면 optimal 값은 아니지만 더 정답에 가까운 successor state를 발견했다는 의미이므로 successor\_best를 인자로 hill\_climbing을 다시 호출한다.

만약 min heuristic 값에 변화가 없다면 Local optimum에 갇힌 경우이므로 클래스의 멤버변수인 cur\_location[]을 랜덤 값으로 다시 세팅해준 뒤에 hill\_climbing을 호출 한다. StackOverflowError를 throw 해준 이유는 N이나 2나 3일때는 답이 없으므로 무한루프에 빠져서 StackOverflow 에러를 발생시키는 것을 확인했다. 이때를 따로 처리해 주기 위해서 throw 해주었다.

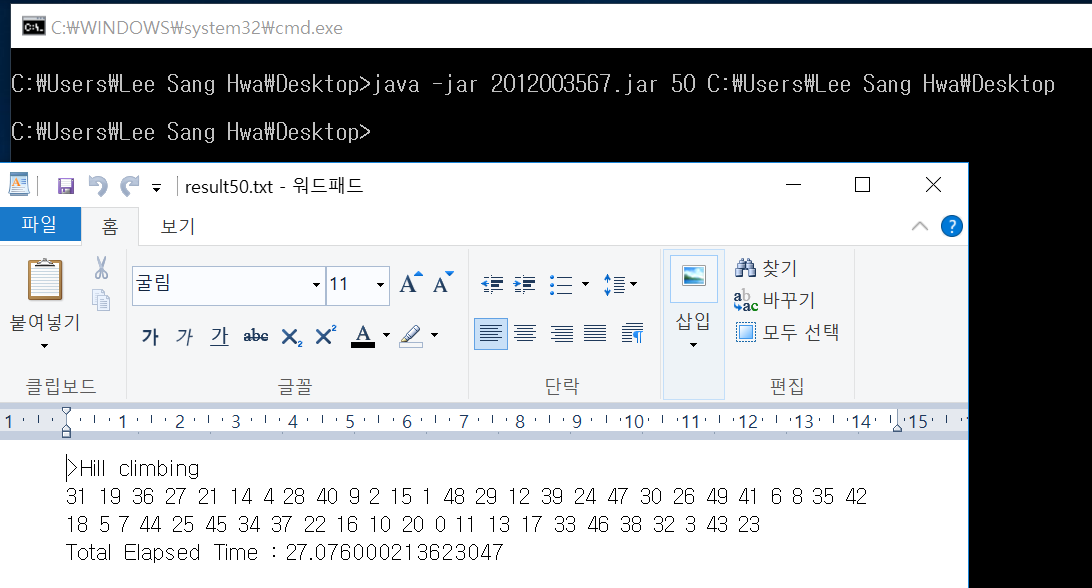
**4. 실행 결과**

1) N=30 일 때



0.07초정도만에 답을 구했다. DFS,BFS,DFID는 N이 10만되도 답을 구하지 못했었는데 N이 무려 30이나 되도 빠른 시간에 답을 구했다.

2) N=50 일 때



N이 무려 50이나 되는데도 27초만에 답을 찾아냈다.