INHA UNIVERSITY

알고리즘설계 HW #3

※ n_Queens Problem

1) Backtracking, Branch and bound

2) Genetic Algorithm

보고서 작성 서약서

1. 나는 타학생의 보고서를 베끼거나 여러 보고서의 내용을 짜깁기하지 않겠습니다.
2. 나는 보고서의 주요 내용을 인터넷사이트 등을 통해 얻지 않겠습니다.
3. 나는 보고서의 내용을 조작하지 않겠습니다.
4. 나는 보고서 작성에 참고한 문헌의 출처를 밝히겠습니다.
5. 나는 나의 보고서를 제출 전에 타학생에게 보여주지 않겠습니다.

나는 보고서 작성 시 윤리에 어긋난 행동을 하지 않고
정보통신공학인으로서 나의 명예를 지킬 것을 맹세합니다.

2017년 6월 4일

학 부	정보통신공학과
학 년	4
성 명	김민규
학 번	12091419

1. 개요

가. Backtracking이나 Branch and bound로 n-queens problem을 해결해 본다.

1) 되추적 기법으로 n-여왕 문제를 해결해 보면서 해가 도출되는 흐름을 이해하고 promising하지 않은 노드에 대해서 filtering하는 방식을 이해한다.

나. Genetic Algorithm으로 n-queens problem을 해결해 본다.

1) 유전 알고리즘을 사용해 n-여왕 문제를 해결해 보면서 유전 알고리즘의 각 부분 과정을 이해하고 각 과정에서의 기법들이 어떤 것을 사용했을 때 좀 더 효율적으로 해를 구할 수 있는지, parameter 값들을 어느 정도 설정해야 정답이 더 잘 도출되는지를 이해한다.

2. 설계내용

가. n_Queens Problem (Backtracking)

1) 체스판의 가로 크기 n을 입력받아 퀸의 위치를 저장할 배열 pos(=position)를 생성한다. n은 4가 입력되었다고 가정하고 퀸의 위치는 0-3으로 결정된다고 하자.
인덱스는 각각 체스판의 행을 의미하고 값은 퀸의 위치를 의미한다.

[0]	[1]	[2]	[3]

2) 순서대로 퀸의 위치를 계산하는데 해당 위치에 퀸을 놓았을 때 해가 될 수 있는지 가능성을 확인해서 해가 될 가능성이 있는 경우에만 즉, promising한 경우에만 다음 행에서 퀸의 위치를 검색한다.

[0]	[1]	[2]	[3]
0			

▶ promising (O)

3) 퀸이 새로 들어가야 할 위치는 이미 넣었던 퀸의 위치와 같은 열에 있지 않아야 하므로 pos[i]는 pos[0]~pos[i-1]과 같지 않아야 한다.
또한 대각선에 위치하지 않아야 하는데 비교하는 서로의 퀸의 위치 차가 행의 차와 같으면 대각선에 존재한다는 것을 이용해서 검증한다. 즉, 퀸의 위치를 검사하는 행을 row, 비교하는 행을 compare_row라 하고 검사하는 퀸의 위치를 queen_pos, 비교하는 퀸의 위치를 compare_pos라 하면 | queen_pos - compare_pos | == row - comapre_row이면 대각선에 존재하므로 non-promising하다고 한다.

[0]	[1]	[2]	[3]
0	0		

▶ non-promising (같은 열에 존재)

[0]	[1]	[2]	[3]
0	1		

▶ non-promising (대각선에 존재)

[0]	[1]	[2]	[3]
0	2		

▶ promising (O)

4) 2)와 3)을 반복한다.

[0]	[1]	[2]	[3]
0	2	0	

▶ non-promising (같은 열에 존재)

[0]	[1]	[2]	[3]
0	2	1	

▶ non-promising (대각선에 존재)

[0]	[1]	[2]	[3]
0	2	2	

▶ non-promising (대각선에 존재)

[0] [1] [2] [3]

0	2	3	

▶ non-promising (대각선에 존재)

5) 더 이상 해가 될 가능성이 있는 위치가 존재하지 않으면 즉, non promising하면 이전 단계로 돌아가(=backtracking) 확인하지 않은 해에 대해 2)와 3)을 반복한다.

[0] [1] [2] [3]

0	3		

▶ promising (O)

[0] [1] [2] [3]

0	3	0	

▶ non-promising (같은 열에 존재)

[0] [1] [2] [3]

0	3	1	

▶ promising (O)

…

[0] [1] [2] [3]

1	3	0	0

▶ non-promising (같은 열에 존재)

[0] [1] [2] [3]

1	3	0	1

▶ non-promising (같은 열에 존재)

[0] [1] [2] [3]

1	3	0	2

▶ promising (O)

6) pos 배열이 모두 채워지면 해를 구한 것이므로 알고리즘을 종료한다.

나. n_Queens Problem (Genetic Algorithm)

1) 체스판의 가로 크기 n을 입력받는다.
   가) 여기서는 n이 6이 입력되었다고 가정한다.

2) 입력된 6만큼의 공간을 가지는 유전자 배열을 생성한다.
   가) 배열에는 체스판에서의 행의 위치를 index로 참조해 퀸의 위치를 저장한다.

[0]	[1]	[2]	[3]	[4]	[5]

3) 같은 열에 2개의 퀸이 위치할 수 없으므로 중복이 되지 않게 유전자를 생성해야 한다.
   따라서 임의의 위치에 0부터 순서대로 n-1까지 넣는 방식으로 퀸의 위치의 중복을 피해 유전자를 생성한다.

[0]	[1]	[2]	[3]	[4]	[5]
1	4	0	2	5	3

4) 3)의 방식으로 구성된 유전자를 Population Size만큼 생성한다.

(Population Size)개

[0]	[1]	[2]	[3]	[4]	[5]
4	3	0	2	1	5

[0]	[1]	[2]	[3]	[4]	[5]
3	1	4	2	5	0

…

[0]	[1]	[2]	[3]	[4]	[5]
0	4	1	5	3	2

5) 생성된 유전자들의 적응도(=fitness)를 각각 계산한다.
   가) 퀸의 위치가 어느 행까지 적절하게 들어갔는지를 확인함으로써 적응도를 계산한다.
   총 행의 개수가 n개이고 (p-1)행까지 검사했을 때는 서로의 간섭이 없었고 p행에서 퀸의 위치가 서로 간섭됐다고 확인된 경우 이때의 적응도는 p/n으로 계산한다.
   모든 경우에 대해서 간섭되는 게 없으면 이때의 적응도는 n/n=1이 된다.
   적응도가 1인 유전자가 발견되면 해당 퀸의 위치를 출력하고 알고리즘을 종료한다.

   나) 예1

[0]	[1]	[2]	[3]	[4]	[5]
4	3	0	2	1	5

   fitness = 1/6
   [1]의 위치는 3이고 [0]의 위치는 4인데 이는 서로 대각선에 위치하므로 간섭된다.
   이때의 적응도는 index 1에서 간섭됐으므로 1/6이 된다.

다) 예2

[0]	[1]	[2]	[3]	[4]	[5]
3	1	4	2	5	0

fitness = 5/6

[4]까지는 간섭되는 것이 없고 [5]와 [3]이 서로 대각선에 있으므로 간섭된다.

이때의 적응도는 index 5에서 간섭됐으므로 5/6이 된다.

라) 예3

[0]	[1]	[2]	[3]	[4]	[5]
1	3	5	0	2	4

fitness = 1

모든 index에서 서로가 간섭하지 않으므로 이때의 적응도는 1이 된다.

6) 각 유전자의 적응도를 TotalFitnessScore에 합산한다.

7) 각 유전자의 적응도를 scaling하고 재계산된 값으로 TotalFitnessScore도 재계산한다.
   가) TotalFitnessScore 값과 유전자들의 각 적응도 값을 가지고 f'=f-(m-σ)의 식을 이용해서 새로운 적응도 f'를 계산하는데 f'이 음수가 되는 경우에는 0으로 설정한다.

8) 계산된 TotalFitnessScore 값에 0과 1사이의 랜덤한 유리수 값을 곱한 값을 fSlice라 하고 유전자의 fitness를 순서대로 더해 가면서 그 값이 fSlice 값을 넘어섰을 경우 마지막으로 더한 유전자를 선택한다.
   가) 이렇게 유전자를 선택하게 되면 적응도가 높은 유전자가 높은 비율로 선택된다.

9) 8)번의 과정으로 2개의 유전자를 선택한 뒤 선택된 두 유전자를 기반으로(mum, dad) 새로운 유전자 2개(baby1, baby2)를 생성하는데 일부 유전자의 보존을 위해 0에서 1사이의 임의의 값으로 생성된 유리수가 미리 설정한 CROSSOVER_RATE을 넘거나 같은 유전자가 선택된 경우 생성하는 baby 유전자는 부모 유전자를 그대로 물려받도록 설정한다.

10) 새로운 유전자를 crossover 할 때는 CROSSOVER_M_RATE(=유전자 유지 비율)만큼의 부모 유전자는 자식 유전자에 그대로 유지하고 나머지 유전자는 남은 위치에 랜덤하게 위치시킨다.
   가) 예를 들어 CROSSOVER_M_RATE = 0.5로 설정하면 부모 유전자 정보의 1/2만큼만 자식 유전자에 유지된다.

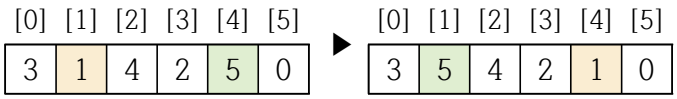
   나) CROSSOVER_M_RATE = 0.5로 설정한 경우의 예

[0]	[1]	[2]	[3]	[4]	[5]
3	1	4	2	5	0

▶

[0]	[1]	[2]	[3]	[4]	[5]
3	0	1	2	5	4

11) 10)의 과정으로 생성된 두 유전자를 변이 비율(=MUTATION_RATE)에 맞춰 변이시킨다.
   가) 퀸의 서로의 위치가 간섭하지 않아야 한다는 특수한 상황을 고려할 때 하나의 유전자만 변이가 일어날 경우는 아예 해가 될 수 없는 유전자가 생성되므로 유전자 내에서 랜덤하게 2개를 선택해서 교환하는 식으로 변이를 일으킨다.

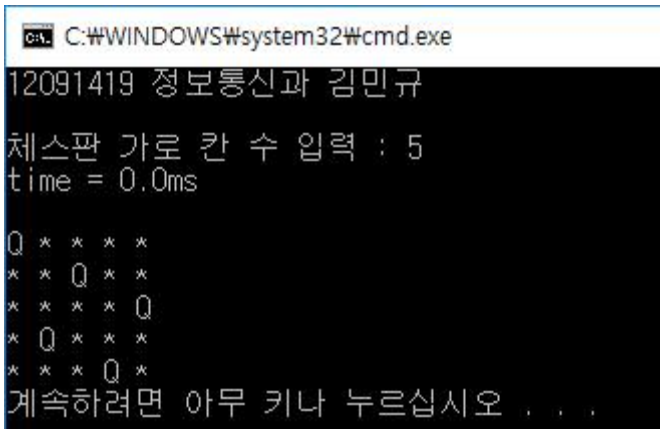


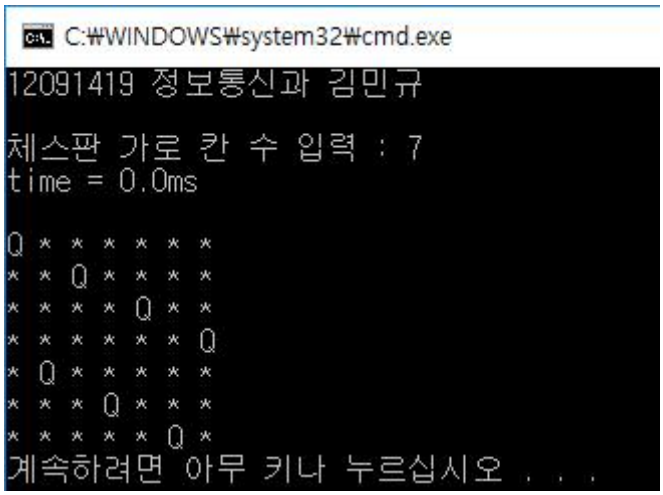
12) 변이가 끝난 baby1, baby2의 유전자를 새로운 세대에 포함시키고 8)에서 11)의 과정을 새로운 세대에 포함시킨 유전자 수가 Population Size와 같아질 때까지 반복 수행한다.

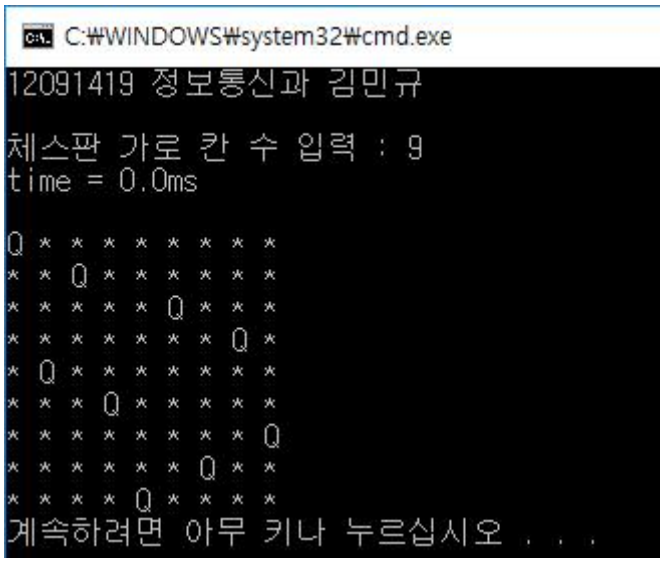
13) 5)에서 12)까지의 과정을 한 시대라고 하고 해가 출력될 때까지 반복 실행한다.

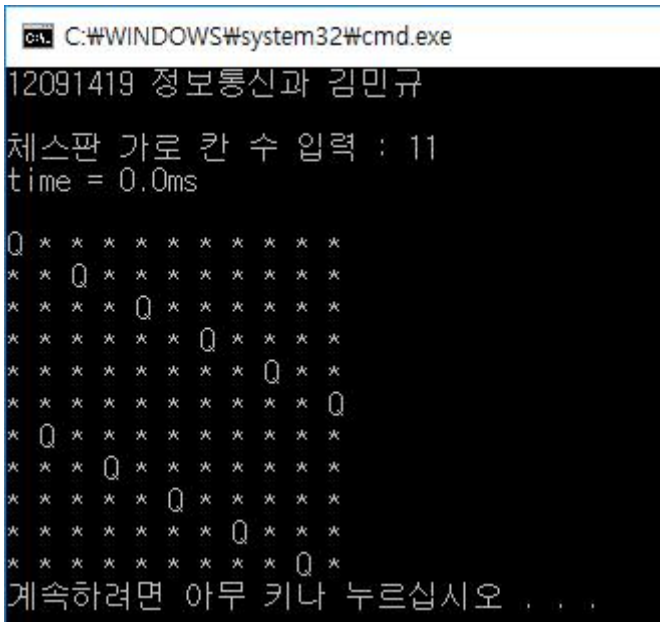
3. 실행 화면

가. Backtracking (체스판 가로 5, 7, 9, 11, 13 입력)



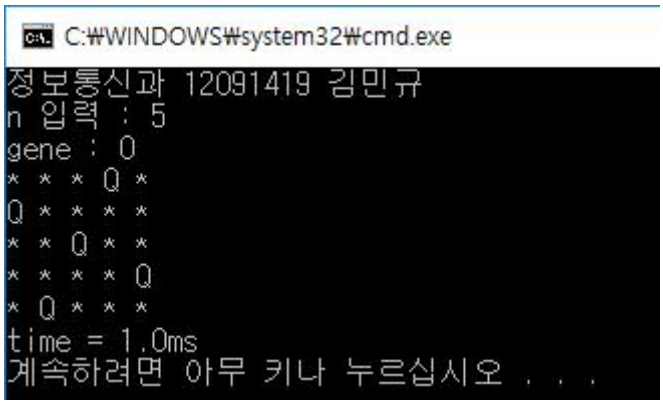


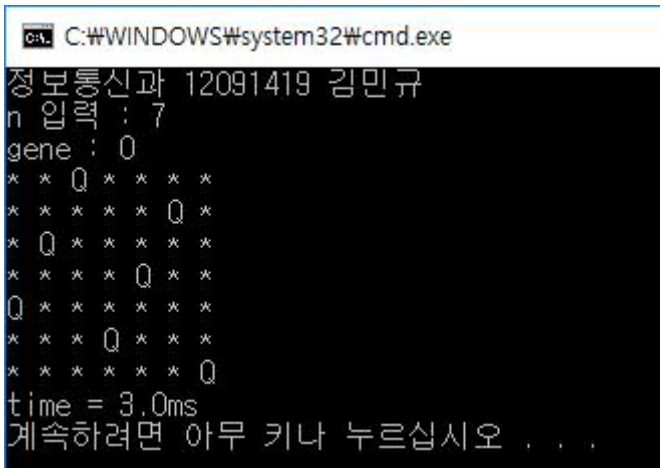


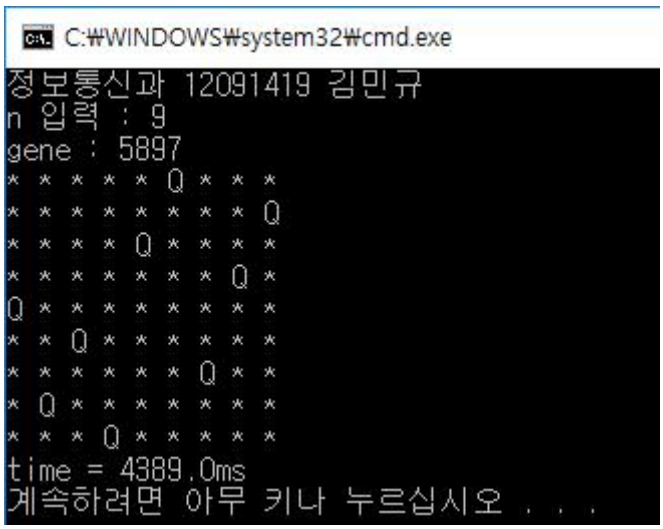

C:₩WINDOWS₩system32₩cmd.exe
12091419 정보통신과 김민규
체스판 가로 칸 수 입력 : 11
time = 0.0ms
Q * * * * * * * * * *
* * Q * * * * * * * *
* * * * Q * * * * * *
* * * * * * Q * * * *
* * * * * * * * Q * *
* * * * * * * * * * Q
* Q * * * * * * * * *
* * * Q * * * * * * *
* * * * * Q * * * * *
* * * * * * * Q * * *
* * * * * * * * * Q *
계속하려면 아무 키나 누르십시오 . . .

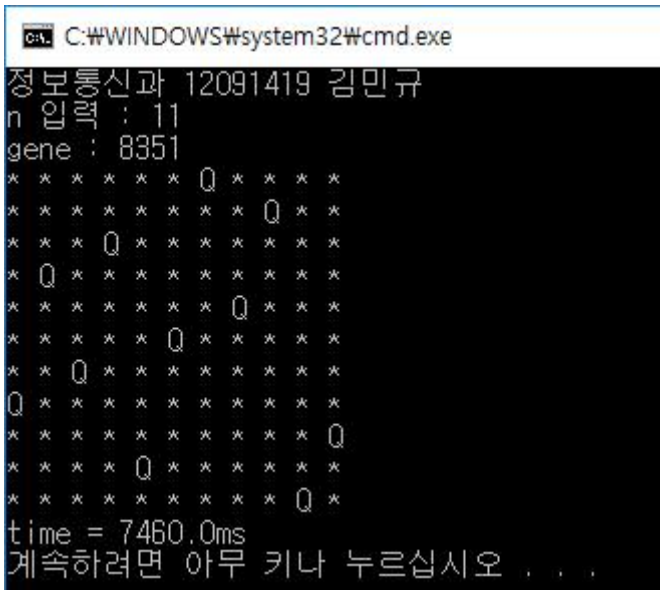

C:₩WINDOWS₩system32₩cmd.exe
12091419 정보통신과 김민규
체스판 가로 칸 수 입력 : 13
time = 0.0ms
Q * * * * * * * * * * * *
* * Q * * * * * * * * * *
* * * * Q * * * * * * * *
* Q * * * * * * * * * * *
* * * * * * * Q * * * * *
* * * * * * * * * * * Q *
* * * * * * * * * Q * * *
* * * * * * * * * * * * Q
* * * Q * * * * * * * * *
* * * * * Q * * * * * * *
* * * * * * * Q * * * * *
* * * * * * * * * * Q * *
* * * * * Q * * * * * * *
계속하려면 아무 키나 누르십시오 . . .

나. Genetic Algorithm (체스판 가로 5, 7, 9, 11, 13 입력)








C:₩WINDOWS₩system32₩cmd.exe
정보통신과 12091419 김민규
n 입력 : 11
gene : 8351
* * * * * * Q * * * *
* * * * * * * * Q * *
* * * Q * * * * * * *
* Q * * * * * * * * *
* * * * * * * Q * * *
* * * * * Q * * * * *
* * Q * * * * * * * *
Q * * * * * * * * * *
* * * * * * * * * * Q
* * * * Q * * * * * *
* * * * * * * * * Q *
time = 7460.0ms
계속하려면 아무 키나 누르십시오 . . .


C:₩WINDOWS₩system32₩cmd.exe
정보통신과 12091419 김민규
n 입력 : 13
gene : 15074
* * * * * * * Q * * * * *
* * * * * * * * * * Q * *
* * * * Q * * * * * * * *
* * Q * * * * * * * * * *
* * * * * * * * * Q * * *
* * * Q * * * * * * * * *
* * * * * * * * * * * * Q
* * * * * * * * Q * * * *
* * * * * * * * * * * Q *
Q * * * * * * * * * * * *
* * * * * Q * * * * * * *
* Q * * * * * * * * * * *
* * * * * Q * * * * * * *
time = 15784.0ms
계속하려면 아무 키나 누르십시오 . . .

4. 분석 및 결론

가. Backtracking 체스판 가로 길이에 따른 실행 시간 (단위 : ms)

가로 길이(=n)	4	7	10	13	14	16	19	22	25	28
Backtracking	0	0	0	0	2	10	3	2699	90	6726

나. Genetic Algorithm 체스판 가로 길이에 따른 실행 시간 (단위 : ms)

1) n=4

횟수	1	2	3	4	5	6	7	8	9	10
Genetic Algorithm	1	1	1	1	1	1	1	1	1	1

2) n=7

횟수	1	2	3	4	5	6	7	8	9	10
Genetic Algorithm	3	597	2	2	3	2	2	2	4	2

3) n=10

횟수	1	2	3	4	5	6	7	8	9	10
Genetic Algorithm	5895	6703	2013	X	4990	297	492	3121	2457	X

4) n=13

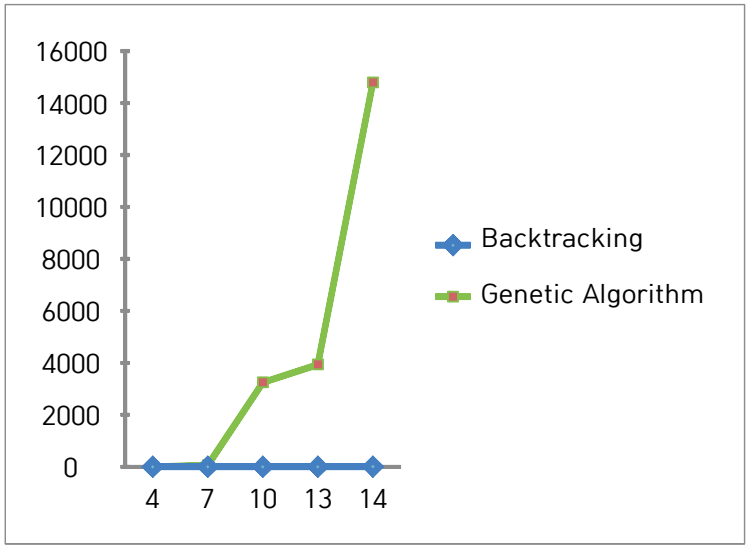
횟수	1	2	3	4	5	6	7	8	9	10
Genetic Algorithm	671	6945	X	5281	552	X	6842	6671	305	4211

5) n=14

횟수	1	2	3	4	5	6	7	8	9	10
Genetic Algorithm	12018	X	X	X	X	18188	X	14311	X	X

다. Backtracking, Genetic Algorithm 실행시간 비교

1) 체스판 가로 길이에 따른 평균 실행시간 비교



(가로 : 체스판 길이, 세로 : ms 단위의 실행시간)

우선 백트래킹과 유전 알고리즘을 별개로 봤을 때 백트래킹은 여러 번 실행해도 같은 n값에 대해서는 비슷한 실행시간을 보이는 것을 확인할 수 있었다.
유전 알고리즘은 처음 유전자 구성이 어떻게 되느냐에 따라서 시간이 들쭉날쭉한 것을 확인할 수 있었는데 처음에 유전자 자체가 해에 가까우면 빨리 해가 도출되나 아예 관계가 없는 유전자들로 개체군이 형성되면 문제 자체가 아예 해결되지 않는 경우도 있었다.
시간을 비교해 보면 backtracking이 가능한 경우의 수를 순서대로 하나씩 제거해 가는 방식이라서 같은 것은 두 번 확인하지 않는 반면에 genetic algorithm은 크로스오버 과정에서 같은 유전자가 생기기도 하므로 같은 것을 여러 번 확인하는 시간 때문에 유전 알고리즘 쪽이 더 오래 걸리는 것 같다.

라. Backtracking 결과

1) 처음에 접근할 때는 이차원 배열을 만들어서 퀸의 적당한 위치를 찾으려고 했었는데 연산의 수가 너무 많아서 위치하는 열의 번호를 기반으로 퀸의 위치를 찾는 식으로 방향을 수정하였다.
작성해 보면서 백트래킹이라는 기법 자체가 DFS와 차이가 없다는 것을 알 수 있었다.
체스판 가로 길이 29까지는 실행시간을 확인했는데 가로 길이 30부터는 시간이 오래 걸려 확인할 수가 없었다.

마. Genetic Algorithm 결과

1) 개체군의 수(=POP_SIZE)는 500, 기존 유전자 유지 비율(=CROSSOVER_RATE)은 0.9, 변이 비율(=MUTATION_RATE)은 0.01, 기존 코드에 주어진 값 외 크로스오버 수행 시 유전자 유지 비율(=CROSSOVER_M_RATE)은 0.6을 설정하고 실험을 진행하였다.
population을 1000 정도로 설정하고 n에 작은 값을 입력했을 때는 한 세대가 지나기도 전에 처음 생성한 유전자에 해답이 포함되어 있는 경우가 많아서 n이 작은 값일 경우에는 세대를 거치지 않고도 바로 해를 구할 수 있었다.

처음 유전자를 설정할 때는 이진수로 작성을 시작했었는데 crossover할 때 임의로 선택되는 지점에서 새로운 유전자를 생성하는 데서 새로이 생성된 유전자에 한 행에 퀸이 2개가 있거나 하나의 퀸도 들어가 있지 않는 문제가 발견되었다. 그래서 유전자를 구성하는 형태를 이진수에서 한 행에서의 열의 번호를 저장하는 식으로 변형하였다.
crossover를 수행할 때는 처음에는 임의의 위치를 잡아서 mum, dad의 유전자를 교차시키는 방식으로 baby1, 2를 만들었는데 이는 높은 빈도로 유전자 안에 퀸의 위치를 중복시키는 문제가 있어서 오히려 답을 구하는 데 더 시간을 오래 잡아먹게 만드는 요인으로 작용하는 것을 확인할 수 있었다. 따라서 crossover 기법을 Position-based Crossover 기법으로 바꾸어 유전자 유지 비율(=CROSSOVER_M_RATE)만큼은 유전자를 그대로 유지하고 남은 나머지의 값들은 임의의 위치로 배치하는 식으로 변경해서 문제를 해결하였다.
mutate를 수행할 때는 처음에는 하나의 값만 다른 값으로 변경하는 식으로 접근을 했었는데 처음부터 각 행이 겹치지 않도록 유전자를 구성했던 것이라 아예 해가 될 가능성이 없는 유전자가 생성되는 문제가 여기서 또한 발생하였다. 따라서 변이 과정을 2개의 임의의 위치를 선택해서 두 위치를 교환하는 식으로 수정해서 이 문제를 해결하였다.
결과를 놓고 보면 유전 알고리즘은 되추적 알고리즘이 결과를 쉽게 출력하는 것과는 다르게 높은 빈도로 해가 도출되지 않기도 하는 것을 확인할 수 있었다. 대략적으로 n 입력이 4~10 정도로 낮을 경우에는 쉽게 답이 나오고 그 이상인 경우에는 답이 도출되지 않는 경우가 많았다.
가장 해가 잘 구해질 때는 변이 비율을 높였을 경우였다. 적응도를 구하는 방식이 앞에서부터 안 맞는 위치가 나올 때까지의 행의 수를 기준으로 설정하는 방식이다 보니 세대가 거듭될수록 초반부 행에 설정된 퀸의 위치는 해의 가능성이 없더라도 고정되는 경우가 많고 이는 해에서 점점 멀어지는 결과를 가져오므로 이에 따라 이것을 변이시키는 비율을 높일수록 해에 가까워지기 때문인 것 같다.