Algorithms Homework 3 Report

2019-12551 박재언

이번 과제는 n-Queens 문제를 변형하여, n × n 체스 판 위에 n개의 퀸을 서로 공격하지 않게 놓는 모든 경우의 수를 구하는 것이다. 추가적으로, 체스판에는 최대 3개의 구멍(hole)이 존재할 수 있으며, 퀸은 구멍 위에 놓일 수 없고, 구멍을 통과해서 공격할 수도 없다. 본 보고서에서는 반복적(Iterative) 백트래킹과 재귀적(Recursive) 백트래킹 두 가지 방법으로 문제를 해결하였고, 각각의 알고리즘에 대해 설명하고, 효율화 방법, 실행 결과, 그리고 두 방법의 성능을 비교하였다.

반복적 백트래킹 알고리즘은 각 행마다 퀸을 놓을 수 있는 열을 찾고, 가능한 경우 다음 행으로 이동하는 방식으로 동작한다. 만약 더 이상 퀸을 놓을 수 없는 경우에는 이전 행으로 돌아가서 다음 열을 시도한다. 이 과정은 스택을 명시적으로 사용하지 않고, 배열과 인덱스를 이용하여 반복문으로 구현하였다. 퀸의 위치는 배열로 관리하여 상태 복원이 빠르며, 구멍의 위치는 최대 3개이기 때문에 매번 리스트를 순회하여 확인해도 성능에 큰 영향을 주지 않는다.

재귀적 백트래킹 알고리즘은 각 행마다 가능한 모든 열을 시도하며, 유효한 위치에 퀸을 놓고 다음 행을 재귀적으로 호출한다. 모든 행에 퀸을 놓는 데 성공하면 경우의 수를 1 증가시킨다. 이 방식 역시 퀸의 위치를 배열로 관리하고, 구멍, 열, 대각선 충돌을 빠르게 체크하여 효율적으로 동작한다.

두 알고리즘 모두 구멍의 위치를 리스트로 관리하였고, 열과 대각선 충돌은 배열과 수식을 이용해 빠르게 검사하였다. 구멍의 개수가 적기 때문에 별도의 HashSet 등으로 최적화하지 않아도 충분히 빠르게 동작한다.

실험을 위해 두 가지 입력을 사용하였다. 첫번째는 5×5 체스판에 (1,3) 위치에 구멍이 있는 경우(1.in)이고, 두번째는 8×8 체스판에 (2,3), (4,5), (6,1) 위치에 구멍이 있는 경우(2.in)이다.

5×5 보드의 경우, 두 알고리즘 모두 8가지의 해답을 출력하였다. 반복적 방식의 실행 시간은 약 0.66ms, 재귀적 방식의 실행 시간은 약 0.06ms로 측정되었다. 8×8 보드의 경우, 두 알고리즘 모두 56가지의 해답을 출력하였다. 반복적 방식의 실행 시간은 약 1.95ms, 재귀적 방식의 실행 시간은 약 0.60ms로 측정되었다.

두 입력 모두에서 재귀적 방식이 반복적 방식보다 더 빠르게 동작하는 것을 확인할 수 있었다. 특히 5×5 보드에서는 재귀적 방식이 반복적 방식보다 약 10배 빠르게 동작했고, 8×8 보드에서는 약 3배 빠르게 동작했다. 이는 재귀적 방식이 함수 호출 오버헤드가 있음에도 불구하고, 상태 관리가 더 효율적이기 때문으로 보인다.

실행 환경은 macOS, OpenJDK 23, javac, make를 사용하였다. 컴파일은 make 명령어로, 실행은 make run 명령어로 진행하였다.

이번 과제를 통해 반복적, 재귀적 백트래킹의 구현 방법과 성능 차이를 직접 비교해 볼 수 있었으며, 구멍이 있는 n-Queens 문제에서도 효율적인 백트래킹이 가능함을 확인하였다. 또한, 문제의 크기가 커질수록 재귀적 방식의 성능 이점이 더욱 두드러지는 것을 관찰할 수 있었다.