수치해석 선택형과제

Queen Problem

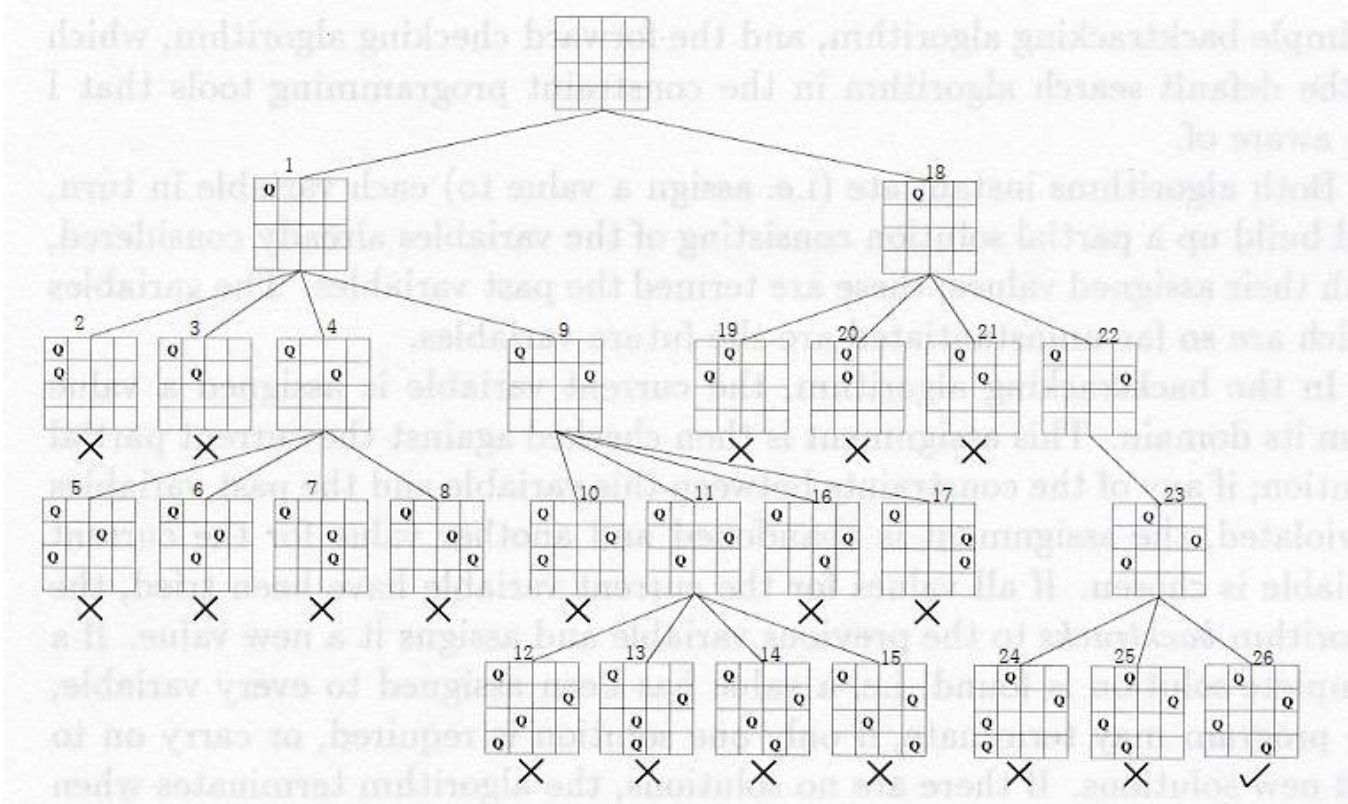
1분반 20181653 이강희

1. 깊이 우선 탐색의 백트래킹 과정

백트래킹 (퇴각 검색법) 이란 가능한 모든 해답을 찾는 알고리즘이다. 비어 있는 해로 탐색을 시작하고, 단계마다 해를 확장해간다. 만약 해를 만들어가는 과정에서 불가능한 후보가 있다면 퇴각 (Backtrack) 한다고 해서 백트래킹 알고리즘이라고 한다.

백트래킹의 구현 방법 중에 깊이우선탐색이 있다. **깊이우선탐색** (Depth First Search, DFS) 이란 트리 또는 그래프를 탐색하는 알고리즘이며, 가능한 깊이 탐색하는 것을 우선한다. 더 이상 탐색이 불가능하면 퇴각하고 다른 루트를 탐색한다. 스택이나 스택과 비슷한 재귀로 많이 구현한다. Queen Problem에서는 텅 빈 보드를 루트로 하고, 행에 퀸을 하나씩 놓는 경우를 자식으로 하는 가상의 트리가 있다고 가정하고 탐색 알고리즘을 적용한다.

다음 사진은 4x4 Queen Problem의 정답을 DFS로 찾는 일부 과정이다. 적힌 숫자 순서대로 탐색한다. 한 단계마다 한 행에 퀸을 놓는다. 최대한 깊이 탐색하는 것을 우선시하는 것을 볼 수 있다. 불가능한 경우에는 다시 퇴각하여 다른 가능한 경우의 수를 찾으며, 모든 경우의 수를 검사해봤으면 종료한다.



뒷장에는 DFS 알고리즘 코드와 주석이 있다.

```
61  def dfs(board, col, size):
62      # 기저 사례, 더이상 탐색이 불가능하므로 탐색을 중지한다.
63      if col >= size:
64          return
65
66      # 모든 열에 놓는 경우의 수를 고려하기 위한 반복문
67      for i in range(size):
68          # board[i][col]에 놓는게 가능하다면 탐색을 진행한다.
69          if is_safe(board, i, col, size):
70              board[i][col] = 1
71              # 만약 N개의 퀸을 모두 놓았으면 정답에 추가하고 퇴각한다.
72              if col == size - 1:
73                  add_solution(board)
74                  board[i][col] = 0
75                  return
76              # 다음 단계로 탐색을 진행한다.
77              dfs(board, col + 1, size)
78              # board[i][col]에 대한 탐색이 끝났으므로 퀸을 다시 뺀다.
79              board[i][col] = 0
```

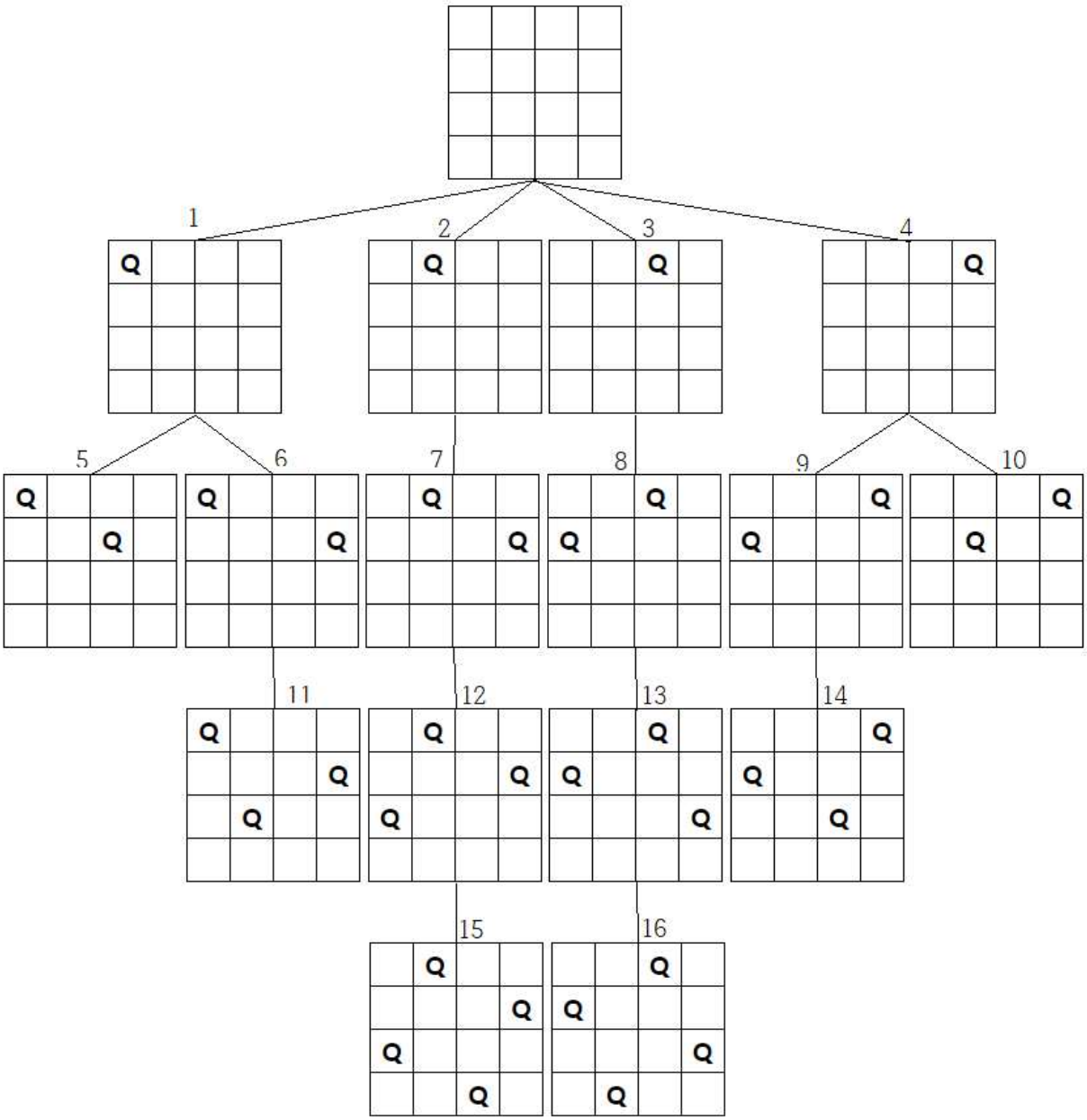
원래코드에 주석만 단 코드이다.
재귀함수 형태로 DFS가 구현된 것을 볼 수 있다.

그리고 사진에서는 행마다 퀸을 놓는 방식으로 한단계씩 진행했지만, 코드에서는 열마다 퀸을 놓는 방식으로 구현되었는데, 별 문제가 되지 않는다.

2. 너비 우선 탐색에 대한 코드와 풀이과정

너비우선탐색 (Breadth First Search, BFS) 이란 DFS와 같이 트리 또는 그래프를 탐색하는 알고리즘이며, DFS와는 반대로 넓게 탐색하는 것을 우선한다. 같은 단계(높이)의 노드들을 모두 탐색하고 다음 단계의 노드로 이동하는 알고리즘이다. BFS는 그 특성 때문에 큐로 많이 구현한다.

다음 사진은 4x4 Queen Problem의 정답을 BFS로 찾는 과정이다. 적힌 숫자 순서대로 탐색한다. 같은 높이의 노드들을 먼저 탐색하는 것을 볼 수 있다. 탐색하면서 다음 단계로 넘어갈 수 있으면 큐에 넣는 방식으로 구현한다. 정답은 4개의 퀸을 놓은 마지막 2개가 된다.



뒷장에는 BFS 알고리즘 코드와 주석이 있다.

```
81  # queue 모듈을 불러옴
82  import queue
83
84  def bfs(board, size):
85      # 큐를 선언한다.
86      # 처음 큐에는 텅 빈 baord와 퀸의 개수를 알려주는 0이 들어있다.
87      Q = queue.Queue()
88      Q.put((board, 0))
89      # 큐가 빌때까지 진행 (모든 경우의수를 탐색할때까지)
90      while Q.qsize() > 0:
91          board, col = Q.get()
92          # 만약 size개의 퀸이 있으면 정답임
93          if col == size:
94              add_solution(board)
95              continue
96          # 모든 열에 놓는 경우의 수를 고려하기 위한 반복문
97          for i in range(size):
98              # board[i]][col]에 놓는게 가능하다면 큐에 넣는다
99              if is_safe(board, i, col, size):
100                 next_board = copy.deepcopy(board)
101                 next_board[i][col] = 1
102                 Q.put((next_board, col+1))
```

큐 자료구조를 사용하여 BFS를 구현하였다.
큐에는 board와 퀸의 개수 쌍을 넣어준다.

소감

과제로 알고리즘 문제를 해결하는 것이 재미있었다. 또한, C++, Java에서만 써보았던 자료구조 큐를 파이썬에서 써보았는데, push, pop 메소드가 아닌 put, get 라는 이름으로 되어있던게 생소했지만 큰 문제가 되지는 않았다. 만약 다음에 파이썬으로 자료구조를 다루게 될 일이 있다면 좀 더 능숙하게 다룰 수 있을 것 같다.

지금까지 Queen Problem은 DFS로만 풀 수 있는 문제인 줄 알았는데, BFS로 이 문제를 해결하려니 어려운 점이 많았다. 인터넷에서도 Queen Problem을 BFS로 다루는 경우는 없어서 스스로 생각해내야 했다. 색다른 시각에서 Queen Problem을 풀이해 보고 직접 그림을 그려봄으로써 이 문제에서 경우의 수들이 어떤 식으로 탐색되는지 더 잘 이해할 수 있었다.

Queen Problem을 BFS로 풀이하는 것은 그렇게 효율적인 알고리즘은 아닌 것 같지만, 다른 관점에서 이 문제를 풀이해보는 것은 문제해결능력을 기르는데 도움이 될 것 같다.