#6 백트래킹

2019 SCSC Summer Coding Workshop

서강대학교 컴퓨터공학과 박수현

me@shiftpsh.com

백트래킹

가능한 상태들을 그래프로 생각하고 DFS를 하되, 가능한 후보 상태만 탐색하는 기법

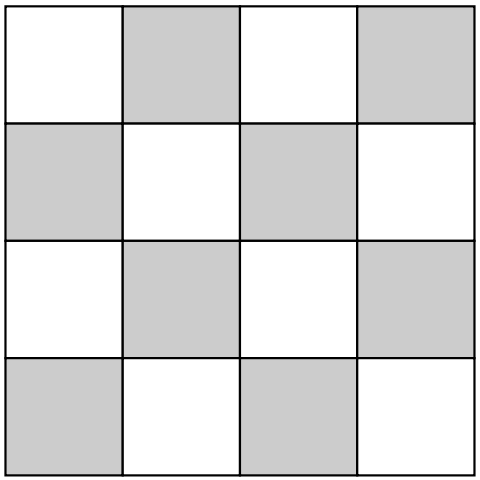
- 즉 불가능한 상태를 탐색하지 않음으로서 시간을 아낄 수 있다

N-Queen 문제

체스에서 퀸은 가로, 세로, 대각선 방향으로 몇 칸이라도 움직일 수 있는 말이다.
크기가 $N \times N$인 체스판 위에 퀸 N개를 서로 공격할 수 없게 놓는다면 몇 개를 올려놓을 수 있을까?

N-Queen 문제

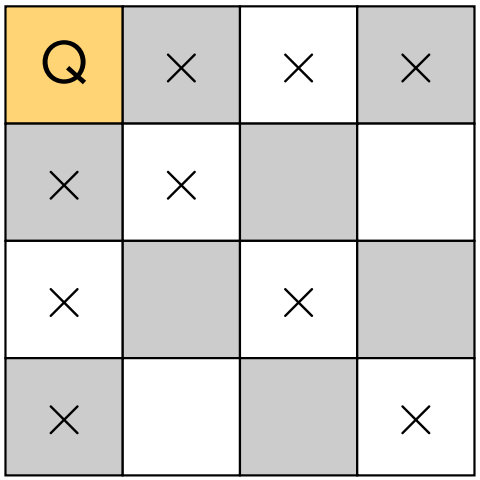
$N = 4$일 때를 예로:



한 열에 하나씩 놓을 수 있으니 맨 왼쪽 열부터 놓아 보자

N-Queen 문제

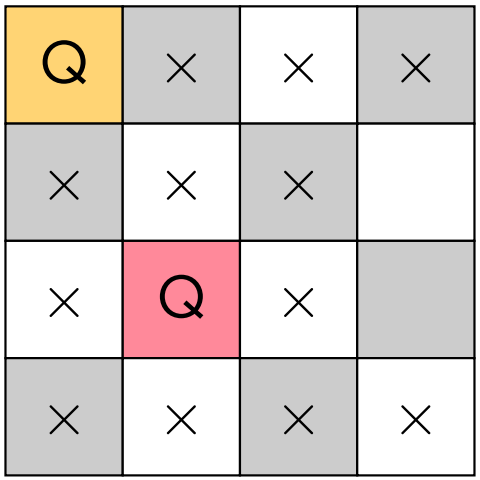
$N = 4$일 때를 예로:



한 열에 하나씩 놓을 수 있으니 맨 왼쪽 열부터 놓아 보자

N-Queen 문제

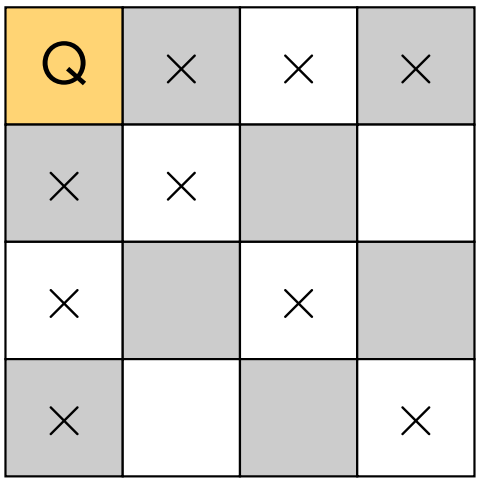
$N = 4$일 때를 예로:



놓을 수 없는 칸은 고려하지 않음

N-Queen 문제

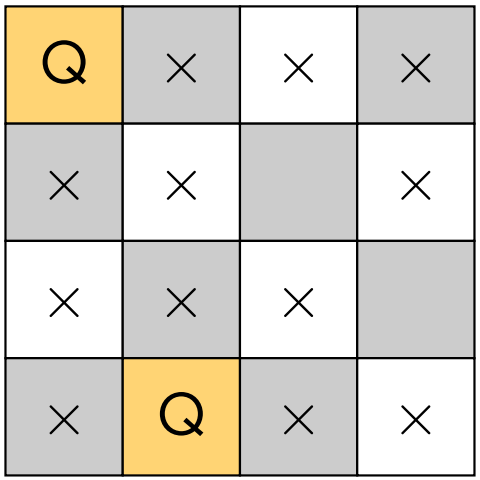
$N = 4$일 때를 예로:



세 번째 열에 놓을 수 있는 칸이 없으니 퇴각backtrack 한다

N-Queen 문제

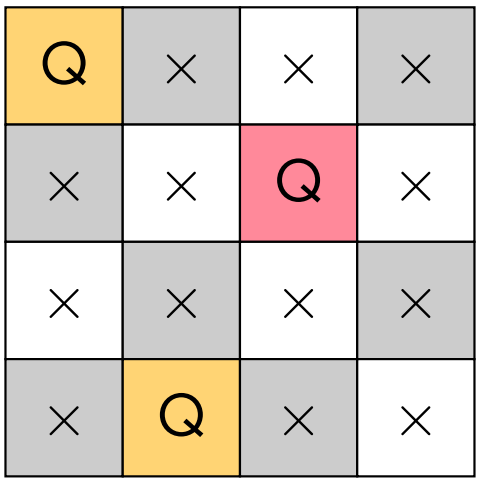
$N = 4$일 때를 예로:



DFS 진행

N-Queen 문제

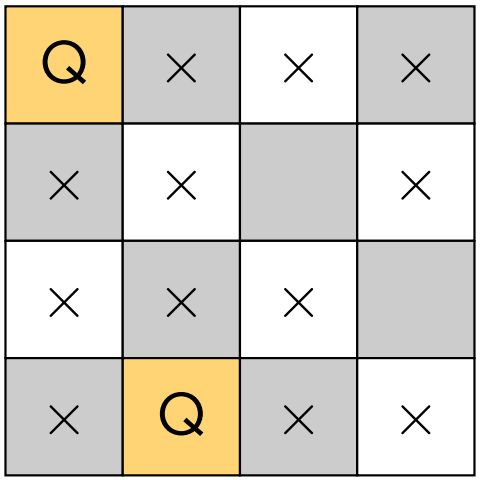
$N = 4$일 때를 예로:



세번째 열에서도 역시 불가능한 칸은 고려하지 않음

N-Queen 문제

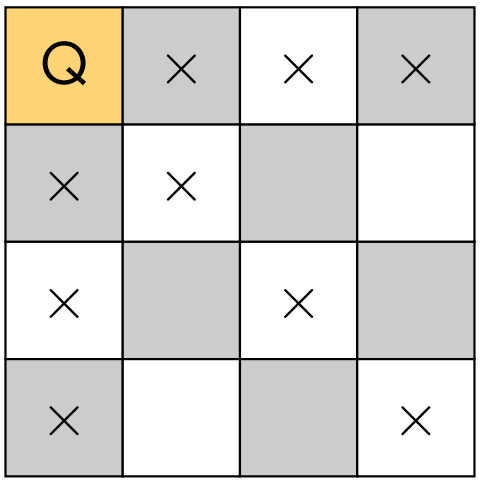
$N = 4$일 때를 예로:



네번째 열에 둘 칸이 없으니 퇴각한다

N-Queen 문제

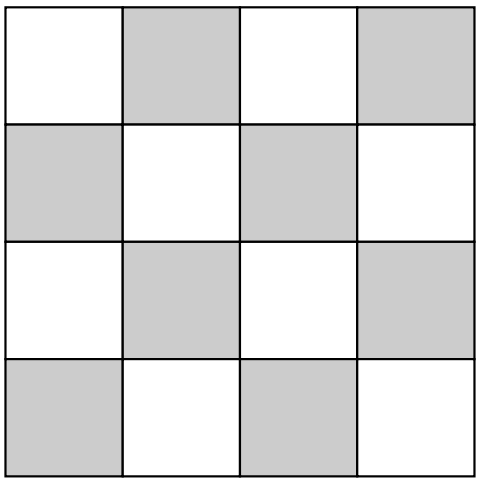
$N = 4$일 때를 예로:



계속 퇴각한다

N-Queen 문제

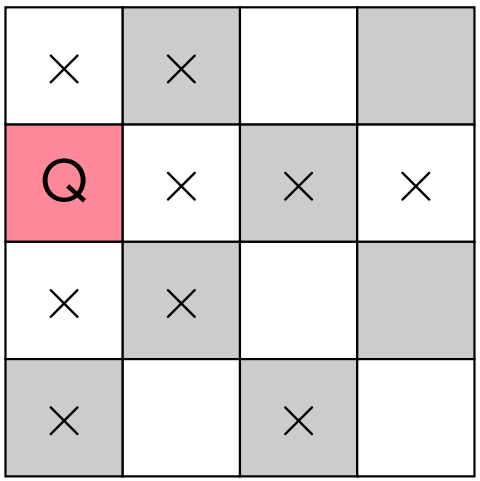
$N = 4$일 때를 예로:



맨 왼쪽 맨 위에 퀸을 놓으면 안 된다는 것을 알았다

N-Queen 문제

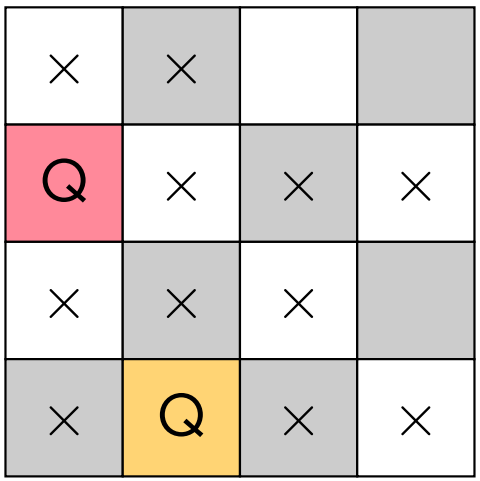
$N = 4$일 때를 예로:



여기에 놓으면 어떨까?

N-Queen 문제

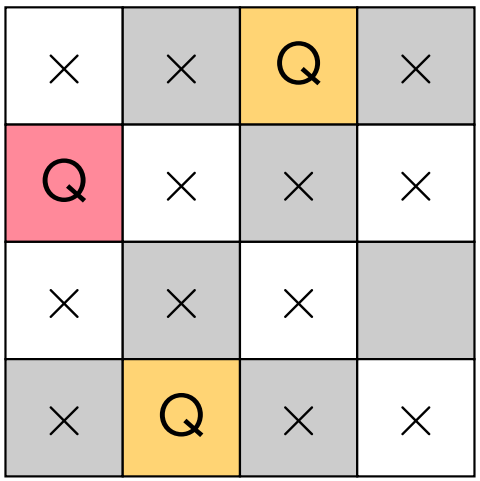
$N = 4$일 때를 예로:



두번째 열 – 불가능한 칸은 고려하지 않음

N-Queen 문제

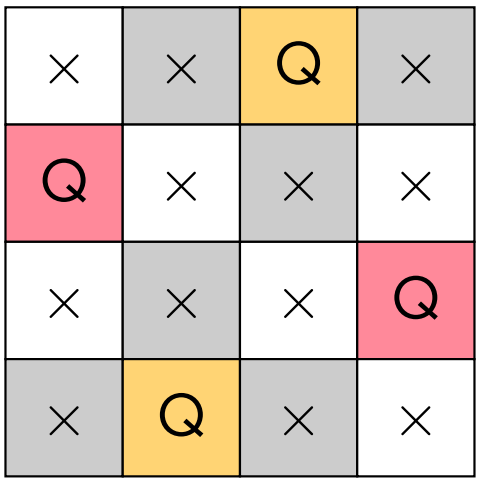
$N = 4$일 때를 예로:



세번째 열 – 불가능한 칸은 고려하지 않음

N-Queen 문제

$N = 4$일 때를 예로:



네번째 열까지 다 둘 수 있었다 $\rightarrow$ 성공

N-Queen BOJ #9663

N이 주어졌을 때, 퀸을 놓는 방법의 수를 구하는 프로그램을 작성하시오.

- $N < 15$

N-Queen BOJ #9663

```
6  int used[16], n, ans = 0;
7
8  void dfs(int col) {
9      if (col == n) {
10         ans++;
11         return;
12     }
13     bool disabled[16];
14     memset(disabled, 0, sizeof(disabled));
15     for (int i = 0; i < col; i++) {
16         int j = used[i];
17         disabled[j] = true;
18         int up = j - i + col, down = j + i - col;
19         if (0 <= up && up < n) disabled[up] = true;
20         if (0 <= down && down < n) disabled[down] = true;
21     };
22     for (int i = 0; i < n; i++) {
23         if (disabled[i]) continue;
24         used[col] = i;
25         dfs(col + 1);
26         used[col] = -1;
27     }
28 }
```

N-Queen BOJ #9663

```
8  void dfs(int col) {
9      if (col == n) {
10         ans++;
11         return;
12     }
```

col: 현재 열 번호. 이게 n이라면 경우의 수를 하나 찾은 것이다

N-Queen BOJ #9663

```
13      bool disabled[16];
14      memset(disabled, 0, sizeof(disabled));
15      for (int i = 0; i < col; i++) {
16          int j = used[i];
17          disabled[j] = true;
18          int up = j - i + col, down = j + i - col;
19          if (0 ⩽ up && up < n) disabled[up] = true;
20          if (0 ⩽ down && down < n) disabled[down] = true;
21      };
```

disabled: 현재 열에서 놓을 수 없는 칸 번호들.

전 열들에 어떤 칸에 퀸이 놓였나를 체크하면서 현재 열에 놓일 수 없는 곳들을 마킹

N-Queen BOJ #9663

```
22      for (int i = 0; i < n; i++) {
23          if (disabled[i]) continue;
24          used[col] = i;
25          dfs(col + 1);
26          used[col] = -1;
27      }
```

놓일 수 없는 칸들을 무시하고 DFS를 돌린다

N-Queen BOJ #9663

```
22      for (int i = 0; i < n; i++) {
23          if (disabled[i]) continue;
24          used[col] = i;
25          dfs(col + 1);
26          used[col] = -1;
27      }
```

퇴각하는 경우를 고려해 다음 열들을 본 후엔 이 열에서 퀸을 없애준다

N-Queen BOJ #9663

```
30  int main() {
31      cin.tie(nullptr);
32      cout.tie(nullptr);
33      ios_base::sync_with_stdio(false);
34
35      memset(used, -1, sizeof(used));
36
37      cin >> n;
38      dfs(0);
39      cout << ans;
40
41      return 0;
42  }
```

로또 BOJ #6603

독일 로또는 $\{1, 2, \cdots, 49\}$에서 수 6개를 고른다.
로또 번호를 선택하는데 사용되는 가장 유명한 전략은 49가지 수 중 $k(k > 6)$개의 수를 골라 집합 S를 만든 다음 그 수만 가지고 번호를 선택하는 것이다.
집합 S와 k가 주어졌을 때, 수를 고르는 모든 방법을 구하는 프로그램을 작성하시오.

로또 BOJ #6603

$\{1, 2, 3, 4, 5, 6, 7\}$ 중에서 6개 뽑기

- 1, 2, 3, 4, 5, 6
- 1, 2, 3, 4, 5, 7
- 1, 2, 3, 4, 6, 7
- 1, 2, 3, 5, 6, 7
- 1, 2, 4, 5, 6, 7
- 1, 3, 4, 5, 6, 7
- 2, 3, 4, 5, 6, 7

로또 BOJ #6603

- 앞에서 사용된 수들을 어느 배열에 저장해 두고
- 이미 사용된 수라면 다시 사용하지 않도록 하여
- 백트래킹

로또 BOJ #6603

```
int k, a[13];
bool used[13];

void dfs(int i, int from) {
    if (i == 6) {
        for (int j = 0; j < k; j++) {
            if (!used[j]) continue;
            cout << a[j] << ' ';
        }
        cout << '\n';
        return;
    }

    for (int j = from + 1; j < k; j++) {
        if (used[j]) continue;
        used[j] = true;
        dfs(i + 1, j);
        used[j] = false;
    }
}
```

로또 BOJ #6603

```
9  void dfs(int i, int from) {
10     if (i == 6) {
11         for (int j = 0; j < k; j++) {
12             if (!used[j]) continue;
13             cout << a[j] << ' ';
14         }
15         cout << '\n';
16         return;
17     }
```

6개의 수를 사용했다면 사용된 수들을 출력

로또 BOJ #6603

```
19      for (int j = from + 1; j < k; j++) {
20          if (used[j]) continue;
21          used[j] = true;
22          dfs(i + 1, j);
23          used[j] = false;
24      }
25  }
```

이미 사용된 수라면 넘어가고 DFS

로또 BOJ #6603

```
27  int main() {
28      cin.tie(nullptr);
29      cout.tie(nullptr);
30      ios_base::sync_with_stdio(false);
31
32      while (true) {
33          cin >> k;
34          if (!k) break;
35          for (int i = 0; i < k; i++) {
36              cin >> a[i];
37          }
38          dfs(0, -1);
39          cout << '\n';
40      }
41
42      return 0;
43  }
```

N과 M 시리즈(15649-15666)가 이런 류의 백트래킹 연습하는 데 좋다

스도쿠 BOJ #2580

이 게임은 가로, 세로 각각 9개씩 총 81개의 작은 칸으로 이루어진 정사각형 판 위에서 이뤄지는데, 몇 몇 칸에는 1부터 9까지의 숫자 중 하나가 쓰여 있다.
나머지 빈 칸을 채우는 방식은 다음과 같다.

1. 각각의 가로줄과 세로줄에는 1부터 9까지의 숫자가 한 번씩만 나타나야 한다.
2. 굵은 선으로 구분되어 있는 3×3 정사각형 안에도 1부터 9까지의 숫자가 한 번씩만 나타나야 한다.

게임 시작 전 스도쿠 판에 쓰여 있는 숫자들의 정보가 주어질 때 모든 빈 칸이 채워진 최종 모습을 출력하는 프로그램을 작성하시오.

스도쿠 BOJ #2580

- 빈 칸들에 순서대로 번호를 붙이고
- 각각의 빈 칸에 들어갈 수 있는 숫자들을 계산한 후
- 가능한 경우만 대입하면서 백트래킹

스도쿠 BOJ #6603

```
8  int a[9][9], n; // incomplete square count
9  pii incomplete[81]; // incomplete squares
10
11 bitset<10> calc_possible(int i, int j) {
12     bitset<10> possible;
13     possible.flip();
14     int x = (i / 3) * 3;
15     int y = (j / 3) * 3;
16     for (int k = 0; k < 9; k++) {
17         possible[a[i][k]] = false;
18         possible[a[k][j]] = false;
19     }
20     for (int k = x; k < x + 3; k++) {
21         for (int l = y; l < y + 3; l++) {
22             possible[a[k][l]] = false;
23         }
24     }
25     return possible;
26 }
```

bitset: bool 배열 같은 거

스도쿠 BOJ #6603

```
11  bitset<10> calc_possible(int i, int j) {
12      bitset<10> possible;
13      possible.flip();
14      int x = (i / 3) * 3;
15      int y = (j / 3) * 3;
16      for (int k = 0; k < 9; k++) {
17          possible[a[i][k]] = false;
18          possible[a[k][j]] = false;
19      }
20      for (int k = x; k < x + 3; k++) {
21          for (int l = y; l < y + 3; l++) {
22              possible[a[k][l]] = false;
23          }
24      }
25      return possible;
26  }
```

가로, 세로에서 이미 나온 수를 제외

스도쿠 BOJ #6603

```
11  bitset<10> calc_possible(int i, int j) {
12      bitset<10> possible;
13      possible.flip();
14      int x = (i / 3) * 3;
15      int y = (j / 3) * 3;
16      for (int k = 0; k < 9; k++) {
17          possible[a[i][k]] = false;
18          possible[a[k][j]] = false;
19      }
20      for (int k = x; k < x + 3; k++) {
21          for (int l = y; l < y + 3; l++) {
22              possible[a[k][l]] = false;
23          }
24      }
25      return possible;
26  }
```

3×3 정사각형에서 이미 나온 수를 제외

스도쿠 BOJ #6603

```
28  bool dfs(int i) {
29      if (i == n) {
30          for (int x = 0; x < 9; x++) {
31              for (int y = 0; y < 9; y++) {
32                  cout << a[x][y] << ' ';
33              }
34              cout << '\n';
35          }
36          return true;
37      }
38      int x = incomplete[i].first;
39      int y = incomplete[i].second;
40      bitset<10> possible = calc_possible(x, y);
41      for (int z = 1; z <= 9; z++) {
42          if (possible[z]) {
43              a[x][y] = z;
44              bool b = dfs(i + 1);
45              a[x][y] = 0;
46              if (b) return true;
47          }
48      }
49      return false;
50  }
```

스도쿠 BOJ #6603

```
28  bool dfs(int i) {
29      if (i == n) {
30          for (int x = 0; x < 9; x++) {
31              for (int y = 0; y < 9; y++) {
32                  cout << a[x][y] << ' ';
33              }
34              cout << '\n';
35          }
36          return true;
37      }
```

모든 칸이 채워졌다면 완성된 스도쿠를 출력하고 종료

스도쿠 BOJ #6603

```
38      int x = incomplete[i].first;
39      int y = incomplete[i].second;
40      bitset<10> possible = calc_possible(x, y);
41      for (int z = 1; z ⩽ 9; z++) {
42          if (possible[z]) {
43              a[x][y] = z;
44              bool b = dfs(i + 1);
45              a[x][y] = 0;
46              if (b) return true;
47          }
48      }
49      return false;
50  }
```

다음 칸에 넣을 수 있는 숫자들을 계산, 가능한 숫자들에 대해서만 탐색

스도쿠 BOJ #6603

```
38      int x = incomplete[i].first;
39      int y = incomplete[i].second;
40      bitset<10> possible = calc_possible(x, y);
41      for (int z = 1; z ≤ 9; z++) {
42          if (possible[z]) {
43              a[x][y] = z;
44              bool b = dfs(i + 1);
45              a[x][y] = 0;
46              if (b) return true;
47          }
48      }
49      return false;
50  }
```

true가 나왔다는 뜻은 성공한 보드가 있었다는 뜻이므로 바로 종료

스도쿠 BOJ #6603

```
52  int main() {
53      cin.tie(nullptr);
54      cout.tie(nullptr);
55      ios_base::sync_with_stdio(false);
56
57      for (int i = 0; i < 9; i++) {
58          for (int j = 0; j < 9; j++) {
59              cin >> a[i][j];
60              if (!a[i][j]) {
61                  incomplete[n] = pii(i, j);
62                  n++;
63              }
64          }
65      }
66      dfs(0);
67      return 0;
68  }
```

빈 칸 위치 전처리

문제 풀어보고, 질문하는 시간 (-17시까지)