Chapter 06.
재귀#4 퇴각 검색

Clip 01 | [1987] 알파벳
퇴각(Backtracking) 검색이란?
최장거리 탐색과 가지치기
비트마스크

Clip 02 | [17136] 색종이 붙이기
영역의 분할과 가지치기

Clip 03 | [10597] 순열 장난
중복을 허용하지 않는 순열 생성

Clip 04 | [2661] 좋은 수열
부분 수열 검사와 가지치기

Clip 05 | [9663] N-Queen
체스의 퀸을 서로 공격 불가능하도록 배치

Ch06. 재귀#4 퇴각 검색

1. [1987] 알파벳

퇴각 검색(Backtracking) - 이론

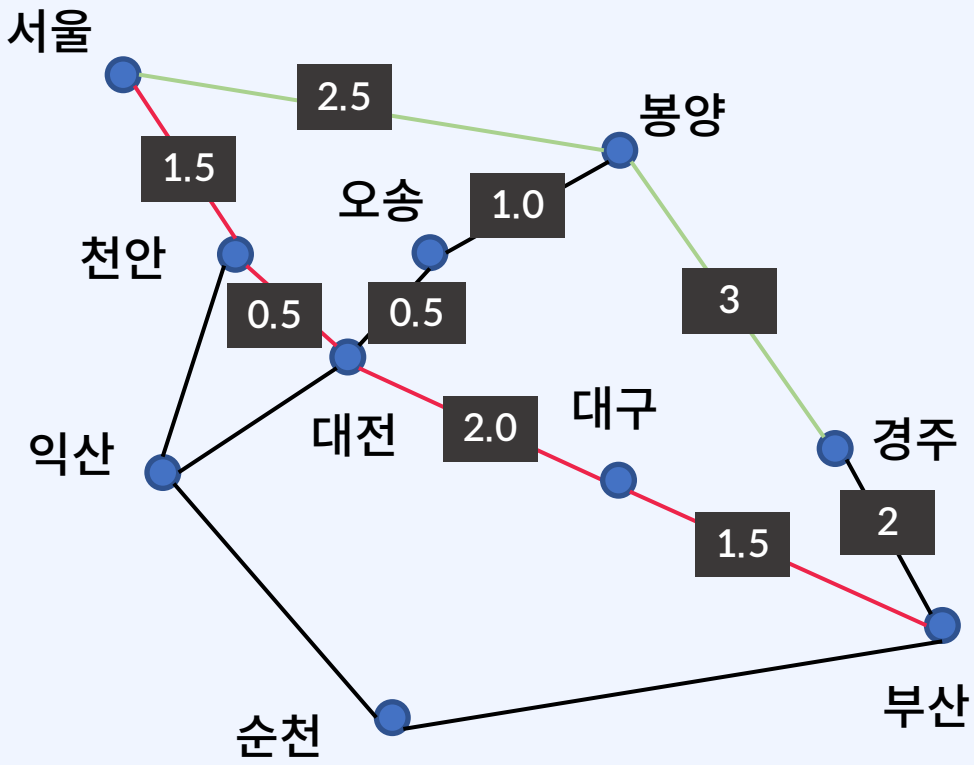
재귀를 이용해 모든 가능한 경우를 탐색하는 알고리즘

일반적인 재귀랑 다른 점은?

- 답을 찾는 도중에 최적해의 가능성이 없어지면 탐색을 중단
- 가능성이 있는 경우를 '유망하다' 라고 표현함
- 최악의 경우 완전탐색이 되지만, 랜덤성이 있는 데이터에서는 평균적인 시간 소요가 낮아진다

퇴각 검색(Backtracking) - 이론

ex) 서울에서 부산을 가는 빠른 방법



서울 → 천안 → 대전 → 대구 → 부산
: 5.5h

서울 → 제천 → 경주
: 5.5h (탐색 중단)

BOJ1987: 알파벳

문제 요약

- 세로 R칸, 가로 C칸의 보드에 대문자 알파벳 입력
  - 1 <= R, C <= 20
- 좌측 상단에서 (1행 1열) 탐색 시작
- 같은 알파벳은 두 번 지나갈 수 없음
- 최대로 지나갈 수 있는 칸 수는?

BOJ1987: 알파벳

입력 데이터
2 4 CAAB ADCB

R C
보드알파벳

```
for(int i = 0; i < r; i++) {
    String line = sc.next();
    for(int j = 0; j < c; j++) {
        board[i][j] = line.charAt(j) - 'A';
    }
}
```

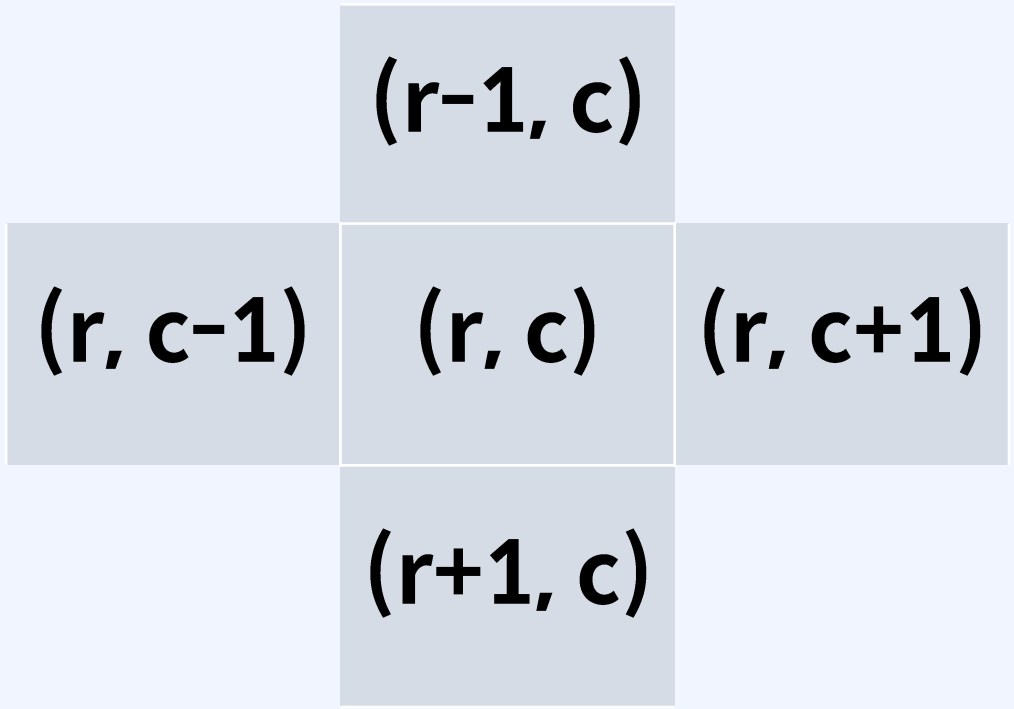
'A': 0
'B': 1
…
'Z': 25

출력 데이터
3

최대 칸 수

BOJ1987: 알파벳

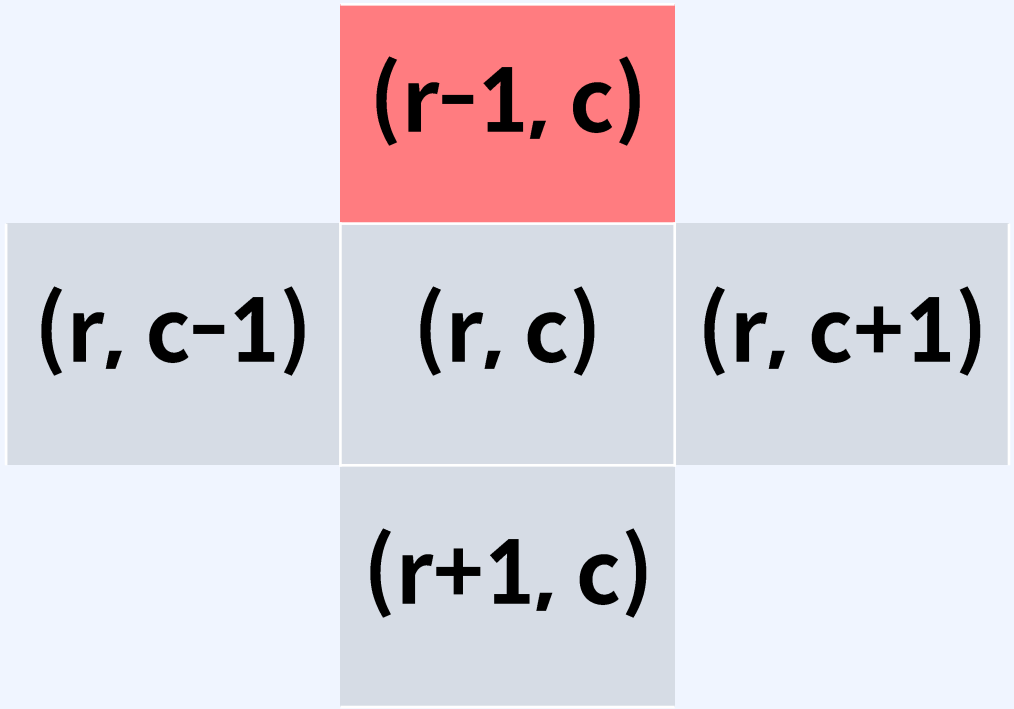
2차원 행렬 탐색 팁




```
public static int[] dr = {0, 0, -1, 1};
public static int[] dc = {-1, 1, 0, 0};
```

BOJ1987: 알파벳

2차원 행렬 탐색 팁



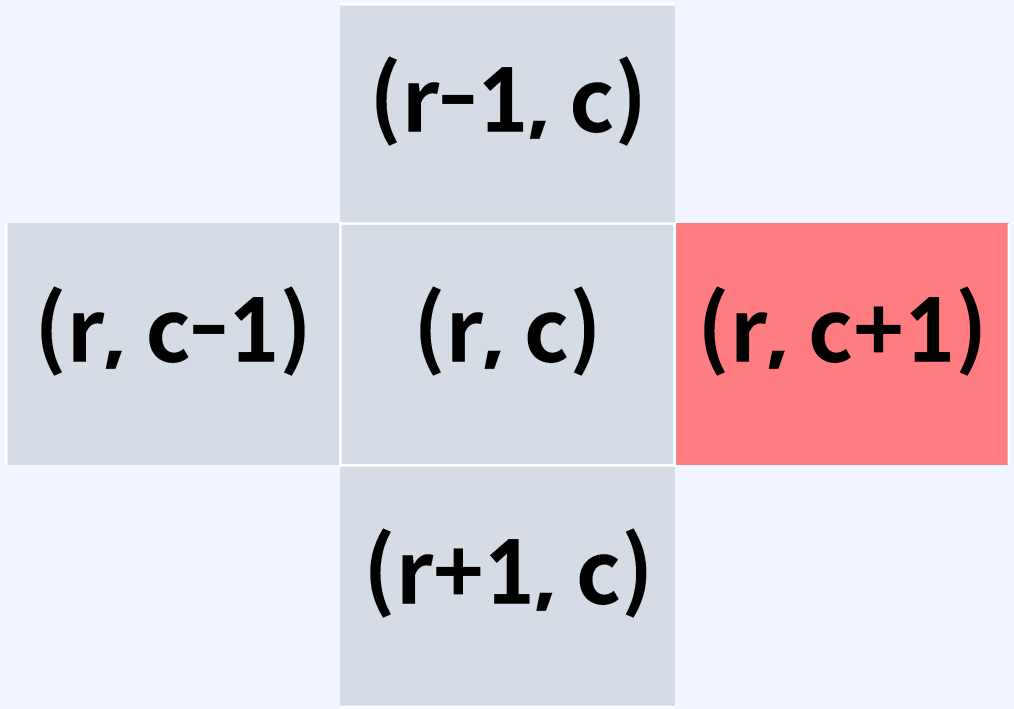

```
public static int[] dr = {-1, 0, 1, 0};
public static int[] dc = {0, 1, 0, -1};
```

r + dr[0] → r-1

c+ dc[0] → c

BOJ1987: 알파벳

2차원 행렬 탐색 팁



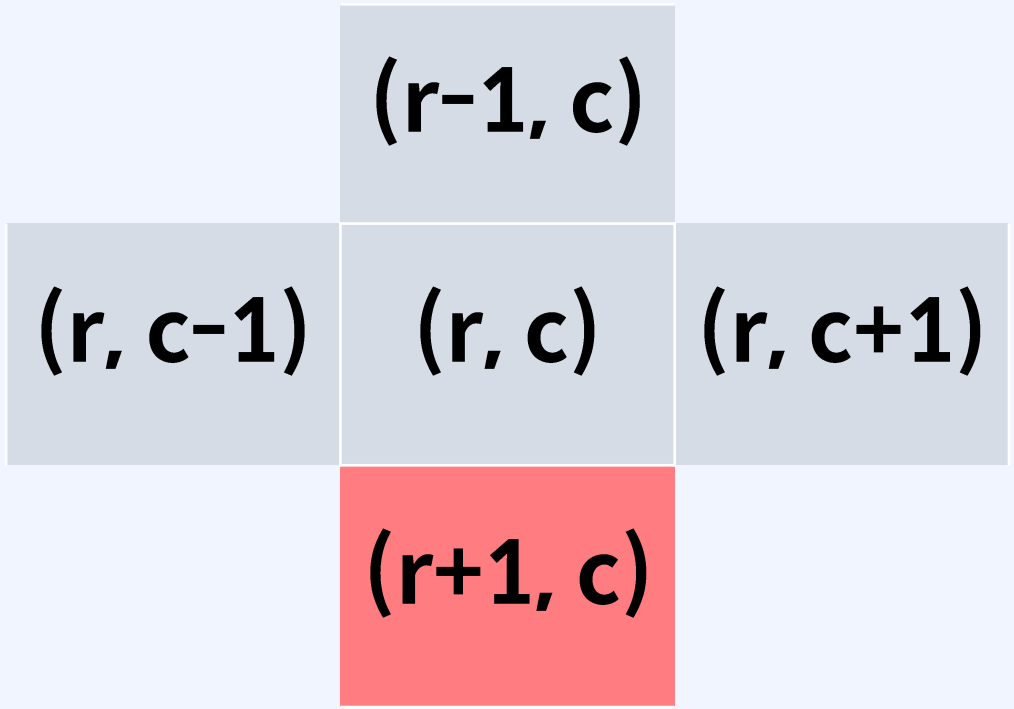

```
public static int[] dr = {-1, 0, 1, 0};
public static int[] dc = {0, 1, 0, -1};
```

r + dr[1] → r

c+ dc[1] → c + 1

BOJ1987: 알파벳

2차원 행렬 탐색 팁



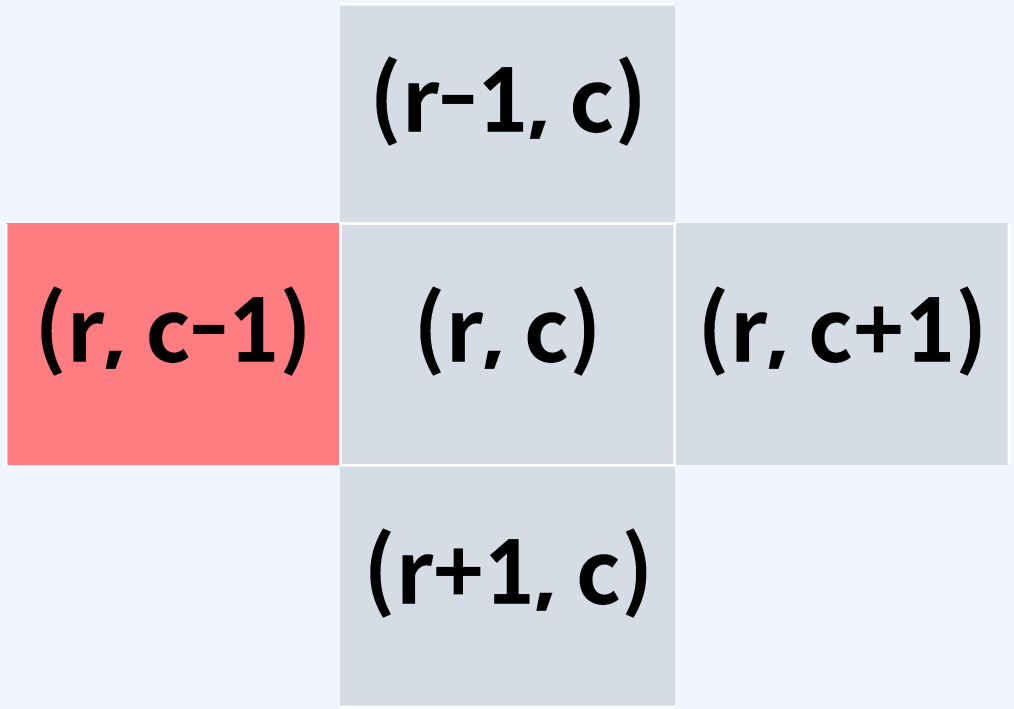

```
public static int[] dr = {-1, 0, 1, 0};
public static int[] dc = {0, 1, 0, -1};
```

r + dr[2] → r+1

c+ dc[2] → c

BOJ1987: 알파벳

2차원 행렬 탐색 팁



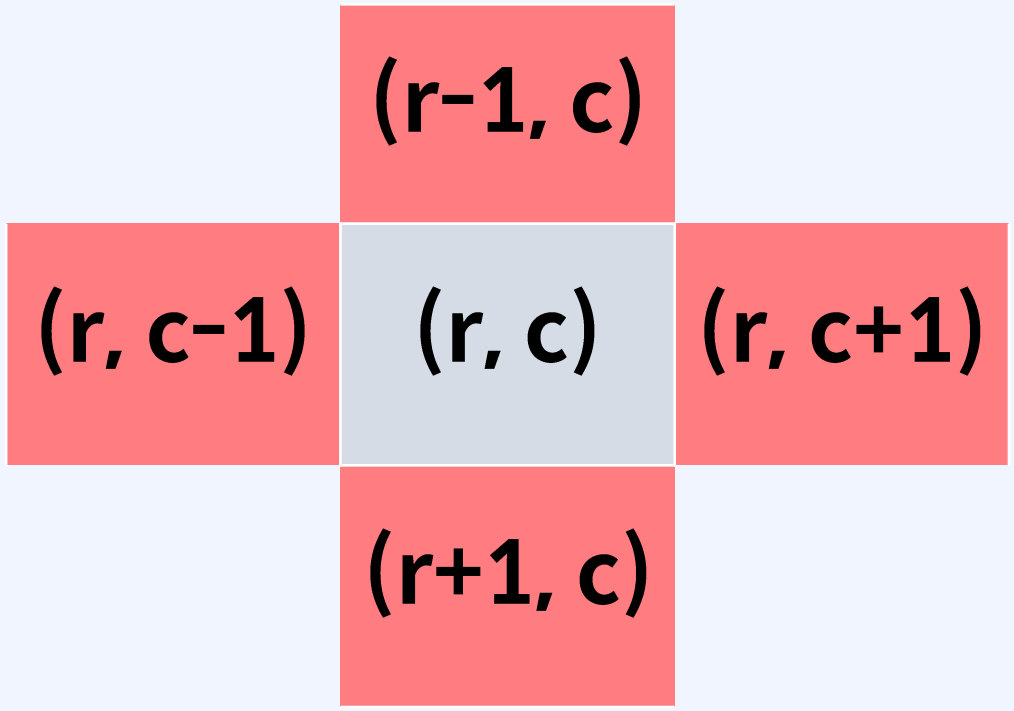

```
public static int[] dr = {-1, 0, 1, 0};
public static int[] dc = {0, 1, 0, -1};
```

r + dr[3] → r

c+ dc[3] → c-1

BOJ1987: 알파벳

2차원 행렬 탐색 팁




```
public static int[] dr = {-1, 0, 1, 0};
public static int[] dc = {0, 1, 0, -1};
```

```
for(int i = 0; i < 4; i++) {
    int nr = row + dr[i];
    int nc = col + dc[i];
     // do something
}
```

BOJ1987: 알파벳

주의할 점

```
for(int i = 0; i < 4; i++) {
    int nr = row + dr[i];
    int nc = col + dc[i];

    if(isOutOfBound(nr, nc, r, c))
        continue;
}
```

```
boolean isOutOfBound(int row, int col, int boundR, int boundC) {
    return row < 0 || row >= boundR || col < 0 || col >= boundC;
}
```

새로 만든 nr, nc의 좌표가 배열의 범위를 벗어날 수 있다
해당 케이스는 코드가 수행되지 않도록 예외 처리가 필요하다

BOJ1987: 알파벳

Base Case

- 더 이상 이동할 수 있는 방향이 없는 경우
- 현재 위치가 범위를 벗어난 경우

Recursive Case

- 상 / 하 / 좌 / 우 방향으로 이동하며 사용한 알파벳을 기록

BOJ1987: 알파벳

1. 새로 탐색할 좌표가 범위 내인지 체크 1
2. 사용되지 않은 알파벳인지 체크 2
3. 다음 좌표를 재귀를 통해 탐색 3

```
public static int solve(int row, int col) {
    int result = 0;
    for(int i = 0; i < 4; i++) {
        int nr = row + dr[i], nc = col + dc[i];
        if(isOutOfBound(nr, nc, r, c)) continue; 1
        int next = board[nr][nc];
        if(check[next]) continue; 2

        check[next] = true;
        result = Math.max(result, solve(nr, nc)); 3
        check[next] = false;
    }
    return result + 1;
}
```

BOJ1987: 알파벳

```
public static int solve(int row, int col) {
    int result = 0;
    for(int i = 0; i < 4; i++) {
        int nr = row + dr[i], nc = col + dc[i];
        if(check[next]) continue;  2

        check[next] = true;
        result = Math.max(result, solve(nr, nc));  3
        check[next] = false;
    }
    return result + 1;
}
```

문제	결과	메모리	시간	언어
1987	맞았습니다!!	18104 KB	976 ms	Java 15 / 수정

1. 새로
2. 사용되지 않은 알파벳인지 체크 2
3. 다음 좌표를 재귀를 통해 탐색

정답이 나오긴 하지만, 백트래킹을 적용해 최적화를 해보자

BOJ1987: 알파벳

아이디어

- {r, c}의 좌표에 임의의 알파벳 집합을 사용하여 도착하였다면
  - 다음번에 동일한 알파벳 집합으로 방문한다면 생략해도 된다

A	B	…	…
B	C	…	…
…	…	…	…
…	…	…	Z

A	B	…	…
B	C	…	…
…	…	…	…
…	…	…	Z

BOJ1987: 알파벳

아이디어

- {r, c}의 좌표에 임의의 알파벳 집합을 사용하여 도착하였다면
  - 다음번에 동일한 알파벳 집합으로 방문한다면 생략해도 된다

선택한 알파벳 조합을 어떻게 기록할까?
- boolean used[row][col][2^26] <- 메모리 초과

그렇다면?
- 26개의 알파벳에 대해 true / false만 기록하면 된다
- 4바이트 변수는 31개의 비트로 이루어져 있다

→ 4바이트 변수에 비트로 true(1) / false(0)를 기록한다

BOJ1987: 알파벳

아이디어

- {r, c}의 좌표에 임의의 알파벳 집합을 사용하여 도착하였다면
  - 다음번에 동일한 알파벳 집합으로 방문한다면 생략해도 된다

ex)
board[1][1] 에 A, B, C 를 사용하여 도달한 경우
배열로 표현하면?

[0]: A	[1]: B	[2]: C	…	[25]: Z
1	1	1		0

비트로 표현하면?
0000_0000_0000_0000_0000_0000_0000_0111(2) == 7

BOJ1987: 알파벳

비트마스크

- 조회: & (1 << n)
- true 대입: |= (1 << n)
- false 대입: &= ~(1 << n)

BOJ1987: 알파벳

배열 vs 비트마스크 – 값 조회

boolean arr[] = {1, 0, 0, 1, 0, 1}

- 3번째 인덱스의 값은?
  - arr[3] (왼쪽부터 0번째)

101001(2)

- 3번째 비트는? (오른쪽부터 0번째)
  - 101001(2) & (1 << 3)
  - 101001(2) & 001000(2)

BOJ1987: 알파벳

배열 vs 비트마스크 – true 대입

arr[] = {1, 0, 0, 1, 0, 1}

- 4번째 인덱스를 1로 변경?
  - arr[4] = 1

101001(2)

- 4번째 비트를 1로 변경
  - 101001(2) |= (1 << 4)
  - 101001(2) |= 010000(2)
  - 111001(2)

BOJ1987: 알파벳

배열 vs 비트마스크 – false 대입

arr[] = {1, 0, 0, 1, 0, 1}

- 5번째 인덱스를 0으로 변경?
  - arr[5] = 0

101001(2)

- 5번째 비트를 0으로 변경
  - 101001(2) &= ~(1 << 5)
  - 101001(2) &= 01111(2)
  - 011001(2)

BOJ1987: 알파벳

1. 다음 경로에 사용한 알파벳을 {route} 변수에 비트로 추출 (1)
2. 다음 방문할 좌표에 ({현재 경로 알파벳} | {route}) 로 방문한 이력이 있는지 체크 (2)
3. 방문하지 않았다면 재귀를 통한 탐색 (3)

```
for(int i = 0; i < 4; i++) {
    int next = board[nr][nc];
1   int route = 1 << next;

2   if(visited[nr][nc] == (visited[row][col] | route)) continue;
    if(check[next]) continue;

    visited[nr][nc] = visited[row][col] | route;
    check[next] = true;
3   result = Math.max(result, solve(nr, nc));
    check[next] = false;
}
```

Ch06. 재귀#4 퇴각 검색

2. [17136] 색종이 자르기

BOJ17136: 색종이 붙이기

문제 요약

- 한 변의 길이가 [1, 5] 인 정사각형 색종이 5종류가 각각 5장씩
- 10*10 위의 지정된 위치(1)에 색종이를 붙임
- 모든 칸을 붙이는데 필요한 색종이의 최소 개수 출력
- 덮는 것이 불가능한 경우에는 -1을 출력

BOJ17136: 색종이 붙이기

입력 데이터
0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

출력 데이터
5

BOJ17136: 색종이 붙이기

입력 데이터
0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

출력 데이터
5

BOJ17136: 색종이 붙이기

Base Case

- 색종이를 더 이상 붙일 공간이 없는 경우

Recursive Case

- {row, col} 좌표에 색종이를 붙일 수 있는 경우
  - (1*1) : (5*5) 의 색종이를 붙일 수 있다면 재귀 수행

Backtrack

- 현재 붙인 색종이 수가 탐색해본 최솟값보다 큰 경우
- 색종이를 다 쓴 경우
- 색종이를 붙이기에 공간이 좁은 경우

BOJ17136: 색종이 붙이기

유틸리티성 기능을 함수로 작성

색종이 부착 가능 여부 검사

```
public static boolean isValid(int row, int col, int size) {
    if (row + size > 10 || col + size > 10) return false;
    for (int r = 0; r < size; r++) {
        for (int c = 0; c < size; c++) {
            if (board[row + r][col + c] == 0) return false;
        }
    }
    return true;
}
```

색종이 부착

```
public static void fill(int row, int col, int size, int color) {
    for (int r = 0; r < size; r++) {
        for (int c = 0; c < size; c++) {
            board[row + r][col + c] = color;
        }
    }
}
```

BOJ17136: 색종이 붙이기

유틸리티성 기능을 함수로 작성

```
public static void findNext(int row, int col) {
    for (int r = row; r < 10; r++) {
        for (int c = 0; c < 10; c++) {
            if (board[r][c] == 1) {
                nextRow = r;
                nextCol = c;
                return;
            }
        }
    }
    nextRow = -1;
    nextCol = -1;
}
```

(row, col) 부터
비어 있는 좌표 획득

빈 공간이 없다면
(-1, -1)

BOJ17136: 색종이 붙이기

색종이의 최대 개수는?

(1 * 1) ~ (5 * 5) 크기별로 5장씩 사용하므로
최대 25개의 색종이를 붙일 수 있다.

따라서 초기 값은 26 이상의 정수로 세팅하고 로직을 수행해도 무관하다
→ 재귀를 수행해도 여전히 26이상이 나온다면, -1을 출력

```
public static int result = 26;
```

```
solve(0, 0, 0);
if (result == 26) System.out.println(-1);
```

BOJ17136: 색종이 붙이기

Base Case

- 색종이를 더 이상 붙일 공간이 없는 경우

```
public static void solve(int row, int col, int cnt) {
    if (result <= cnt) return;
    // TODO: Backtrack: 붙인 색종이 수가 최솟값보다 큰 경우
      findNext(row, col);
    if (nextRow == -1 && nextCol == -1) {
        result = cnt;
        return;
    }
}
```

BOJ17136: 색종이 붙이기

Recursive Case

- {row, col} 좌표에 색종이를 붙일 수 있는 경우
  - (1*1) : (5*5) 의 색종이를 붙일 수 있다면 재귀 수행

```
int r = nextRow, c = nextCol;
for (int size = 1; size <= 5; size++) {
    // TODO: Backtrack: 색종이를 다 쓴 경우
    // TODO: Backtrack: 색종이를 붙일 공간이 없는 경우
    paper[size]--;
    fill(r, c, size, 0);
    solve(r, c, cnt + 1);
    paper[size]++;
    fill(r, c, size, 1);
}
```

BOJ17136: 색종이 붙이기

Backtrack

- 현재 붙인 색종이 수가 탐색해본 최솟값보다 큰 경우
- 색종이를 다 쓴 경우
- 색종이를 붙이기에 공간이 좁은 경우

```
if (result <= cnt) return;
```

```
if (paper[size] == 0) continue;
if (!isValid(r, c, size)) continue;
```

Ch06. 재귀#4 퇴각 검색

3. [10597] 순열 장난

BOJ10597: 순열 장난

문제 요약

- 1부터 N까지 수로 이루어진 순열 (순서 무작위)
- 공백이 모두 제거된 상태
- 중복되는 숫자가 없도록 순열을 복구
  - 정답이 여러 개라면 한 개만 출력 (스페셜 저지)

BOJ10597: 순열 장난

입력 데이터
4111109876532

출력 데이터
4 1 11 10 9 8 7 6 5 3 2

BOJ10597: 순열 장난

문제 분석

- 1부터 N까지 수로 이루어진 순열 (순서 무작위)

- N이 무슨 값인지 알 수 있을까?

[1:9] : 일의자리 수 (숫자 1개)

[10:50]: 십의자리 수 (숫자 2개)

→ 문자열의 길이가 9 이하: (문자열의 길이) == (숫자의 개수)
→ 문자열의 길이가 10 이상: (문자열의 길이 - 9) / 2 + 9

BOJ10597: 순열 장난

입력 데이터
4111109876532

총 13글자
문자열의 길이가 10 이상: (문자열의 길이 - 9) / 2 + 9
(13 – 9) / 2 + 9 == 11

따라서 수열은 [1:11] 범위의 숫자로 구성되어 있다

```
n = input.length > 9 ? 9 + (input.length - 9) / 2 : input.length;
```

BOJ10597: 순열 장난

Base Case

- 정수로 파싱 중인 문자의 인덱스가 끝에 도달했을 때
  - 파싱에 성공한 정수를 출력하고 프로그램 종료

Recursive Case

- 현재의 인덱스 위치에서 1개의 숫자로 정수를 생성
- 현재의 인덱스 위치에서 2개의 숫자로 정수를 생성

Backtrack

- 생성하려는 정수가 이전에 사용된 적이 있는 경우
- N보다 큰 정수를 생성하려는 경우
- 생성하려는 정수가 원본 문자열 인덱스를 초과하는 경우

BOJ10597: 순열 장난

Base Case

- 정수로 파싱 중인 문자의 인덱스가 끝에 도달했을 때
  - 파싱에 성공한 정수를 출력하고 프로그램 종료

```
public static void solve(int index) {
    if(index >= input.length) {
        for (Integer integer : answer) {
            System.out.print(integer + " ");
        }
        System.out.println();
        System.exit(0);
    }
}
```

BOJ10597: 순열 장난

Recursive Case

- 현재의 인덱스 위치에서 1개의 숫자로 정수를 생성
- 현재의 인덱스 위치에서 2개의 숫자로 정수를 생성

```
int target1 = atoi(input, index, 1);

answer.add(target1);
solve(index + 1);
```

```
int target2 = atoi(input, index, 2);

answer.add(target2);
solve(index + 2);
```

BOJ10597: 순열 장난

Backtrack

- 생성하려는 정수가 이전에 사용된 적이 있는 경우
- N보다 큰 정수를 생성하려는 경우
- 생성하려는 정수가 원본 문자열 인덱스를 초과하는 경우

```
if(target1 <= n && check[target1] == 0)
```

```
if(index + 1 >= input.length) return;
```

Ch06. 재귀#4 퇴각 검색

4. [2661] 좋은 수열

BOJ2661: 좋은 수열

문제 요약

- 숫자 1, 2, 3 으로만 이루어지는 수열
- 임의의 길이로 인접한 두개의 부분 수열이 같다: 나쁜 수열
  - 그렇지 않다: 좋은 수열
- 길이가 N인 좋은 수열 중에 가장 작은 수
  - 1 <= N <= 80

BOJ2661: 좋은 수열

나쁜 수열의 예시

33
32121323
123123213

좋은 수열의 예시

2
32
32123
1232123

BOJ2661: 좋은 수열

좋은 / 나쁜 수열 판별 법

- 가장 뒤에 숫자를 추가 할 때 마다 검사를 한다
- 인접한 부분 수열이 2번 등장해야 하므로
  {마지막 인덱스 + 1} / 2까지 검사한다

length == 1

1	2	1	3	1	2	1

length == 2

1	2	1	3	1	2	1

length == 3

1	2	1	3	1	2	1

BOJ2661: 좋은 수열

좋은 / 나쁜 수열 판별 법

- 가장 뒤에 숫자를 추가 할 때 마다 검사를 한다
- 인접한 부분 수열이 2번 등장해야 하므로
  {마지막 인덱스 + 1} / 2까지 검사한다

```
for (int i = 1; i <= (endIndex + 1) / 2; i++) {
    boolean isSame = true;
    for (int j = 0; j < i; j++) {
        if (numbers[endIndex - j] != numbers[endIndex - i - j]) {
            isSame = false;
            break;
        }
    }
```

BOJ2661: 좋은 수열

Base Case

- 생성하는 수열의 길이가 N에 도달
  - 1 → 2 → 3 순으로 수열 생성을 시도하여
    가장 먼저 발견된 수열이 오름차순으로 가장 빠른 수열

Recursive Case

- 기존에 생성한 수열의 가장 마지막에 ‘1’, ‘2’, ‘3’ 을
  각각 추가하며 재귀 수행

Backtrack

- 마지막 인덱스에 추가한 숫자가 나쁜 수열을 만들면 중단

BOJ2661: 좋은 수열

Base Case

- 생성하는 수열의 길이가 N에 도달
  - 1 → 2 → 3 순으로 수열 생성을 시도하여, 가장 먼저 발견된 수열이 오름차순으로 가장 빠른 수열

```
if (endIndex == n) {
    for (int i = 0; i < n; i++) {
        System.out.print(numbers[i]);
    }
    return true;
}
```

BOJ2661: 좋은 수열

Recursive Case

- 기존에 생성한 수열의 가장 마지막에
  '1', '2', '3' 을 각각 추가하며 재귀 수행

Backtrack

- 마지막 인덱스에 추가한 숫자가 나쁜 수열을 만들면 중단

```
for (int i = 1; i <= 3; i++) {
    numbers[endIndex] = i;
    if (!isBad(endIndex)) {
        if (solve(endIndex + 1)) return true;
    }
}
```

Ch06. 재귀#4 퇴각 검색

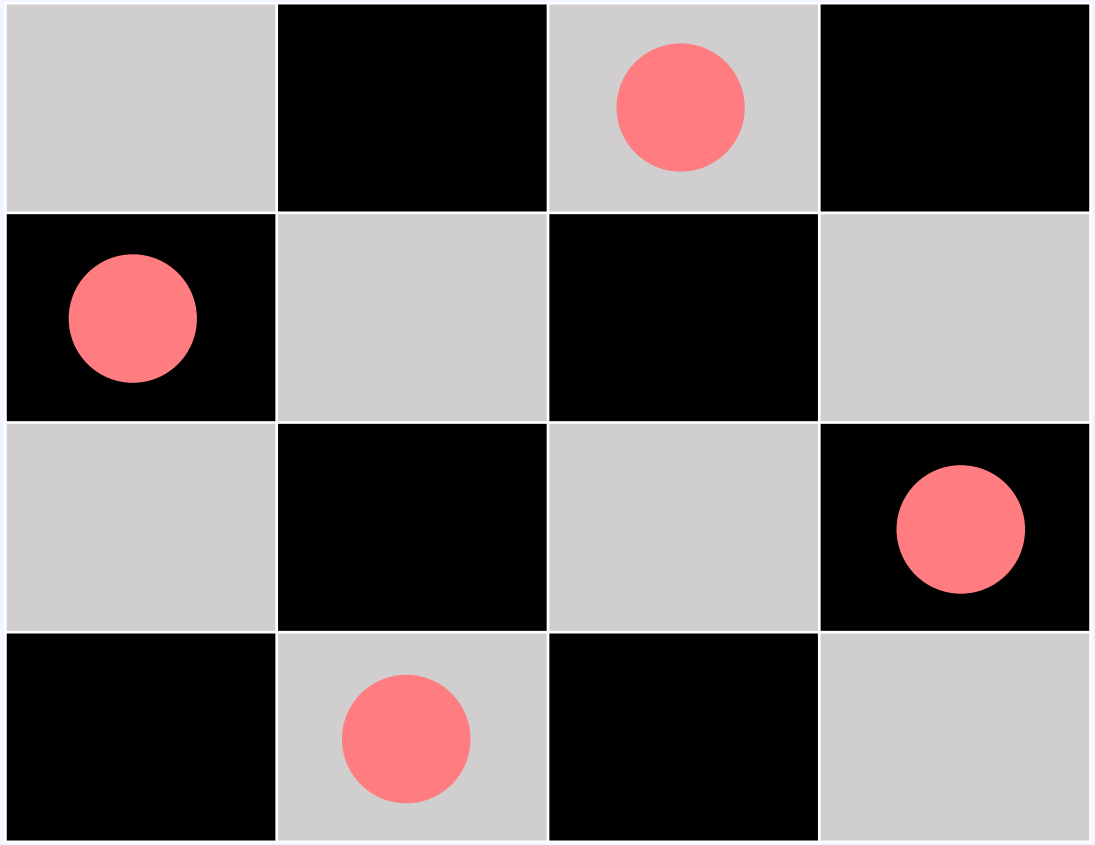
5. [9663] N-Queen

BOJ9663: N-Queen

문제 요약

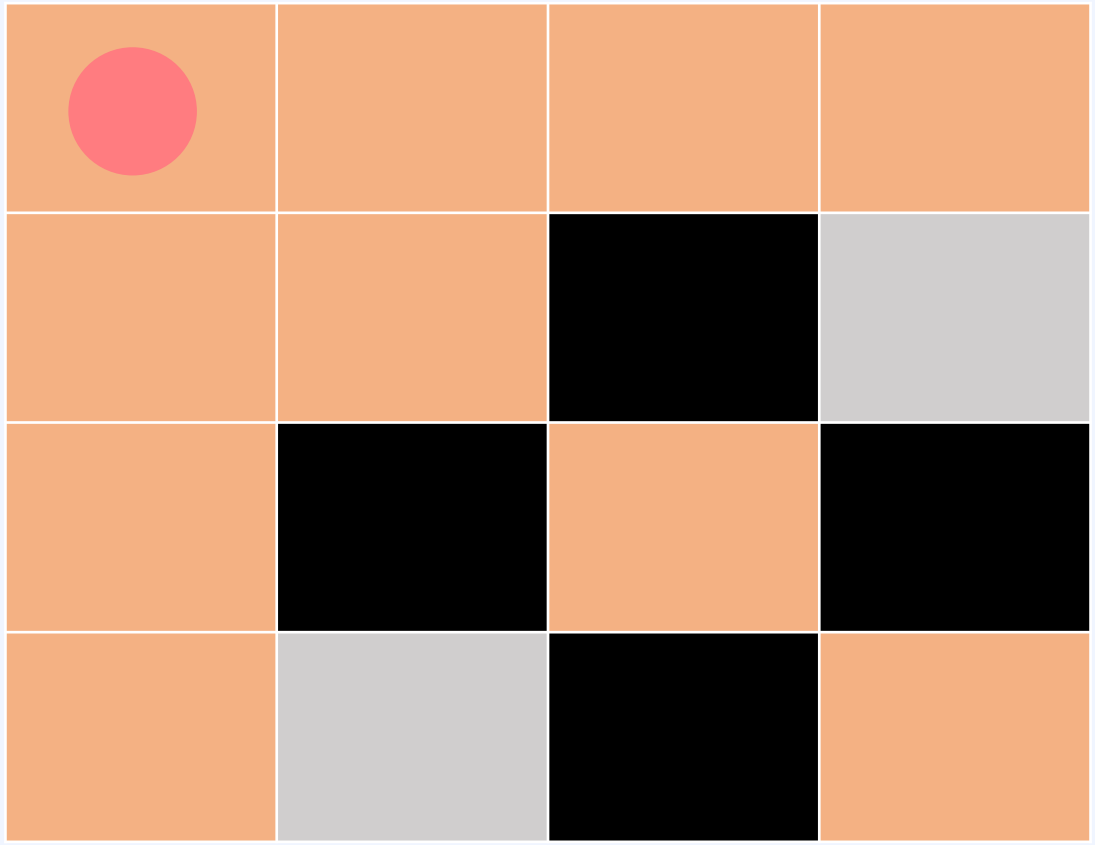
- 행렬에 퀸을 놓았을 때, 서로를 공격할 수 없는 상태가 되도록 배치하는 문제
  - 퀸이 놓여있는 좌표에 직선과 대각선이 만들어지는 구간에 아무것도 없어야 한다
- 1 <= N <= 15
- 전체 경우의 수를 출력

BOJ9663: N-Queen



4 * 4 체스판에서 N-Queen 조건을 만족하는 배치

BOJ9663: N-Queen



퀸을 배치하면, 다음 퀸이 배치가능한 공간에 제약이 생긴다

BOJ9663: N-Queen

문제 분석

- 퀸은 하나의 행에 하나만 존재할 수 있다
  - 재귀함수 탐색의 기준(depth)으로 사용할 수 있다
- 마지막 행에 퀸 배치를 성공하면,
  유망한 경우의 수가 하나 증가한다
- 불가능한 경우는 탐색을 중단하여 가지치기 한다

BOJ9663: N-Queen

체스 판의 표현

한 행에 1개의 퀸만 존재할 수 있다

따라서 1차원 배열만으로 퀸의 위치를 표현할 수 있다

queen[행번호] = 열번호

ex)
queen[2]=3
→ 체스판의 2행 3열에 퀸을 배치할 수 있다

BOJ9663: N-Queen

Base Case

- 재귀 함수가 마지막 행에 퀸을 배치한 경우
  - 0번째 행부터 순서대로 모든 경우의 수를 가지 치며 카운트

Recursive Case

- [0, n) 열을 돌아다니며 퀸 배치를 시도

Backtrack

- 체스 룰에 위배된다면 퀸을 배치할 수 없도록 막는다

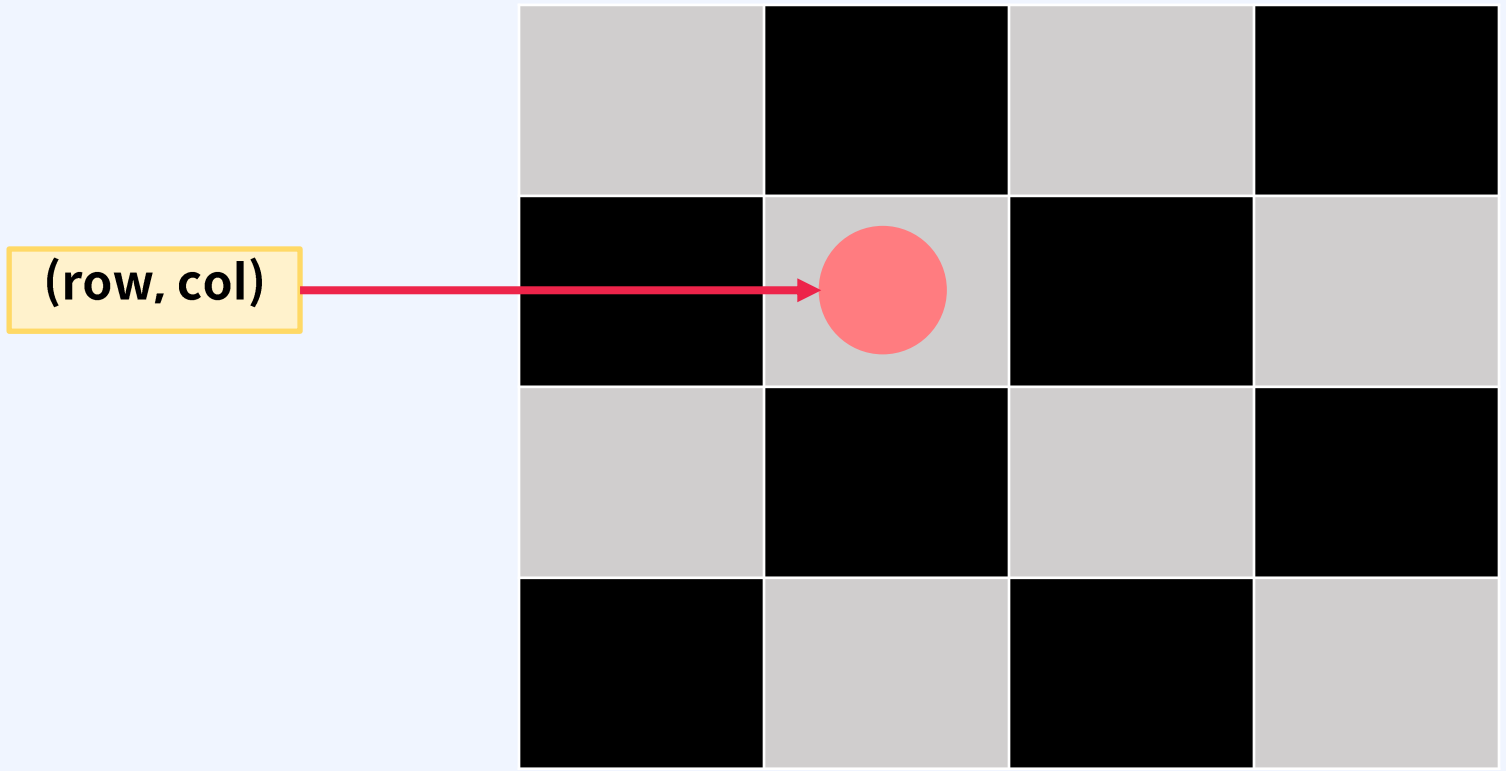
BOJ9663: N-Queen

Backtrack
- 체스 룰에 위배된다면 퀸을 배치할 수 없도록 막는다

```
public static boolean isValid(int row, int col) {
        for (int i = 0; i < row; i++) {
            if (queen[i] == col) return false;
            if (Math.abs(i - row) == Math.abs(queen[i] - col)) return false;
        }
        return true;
    }
}
```

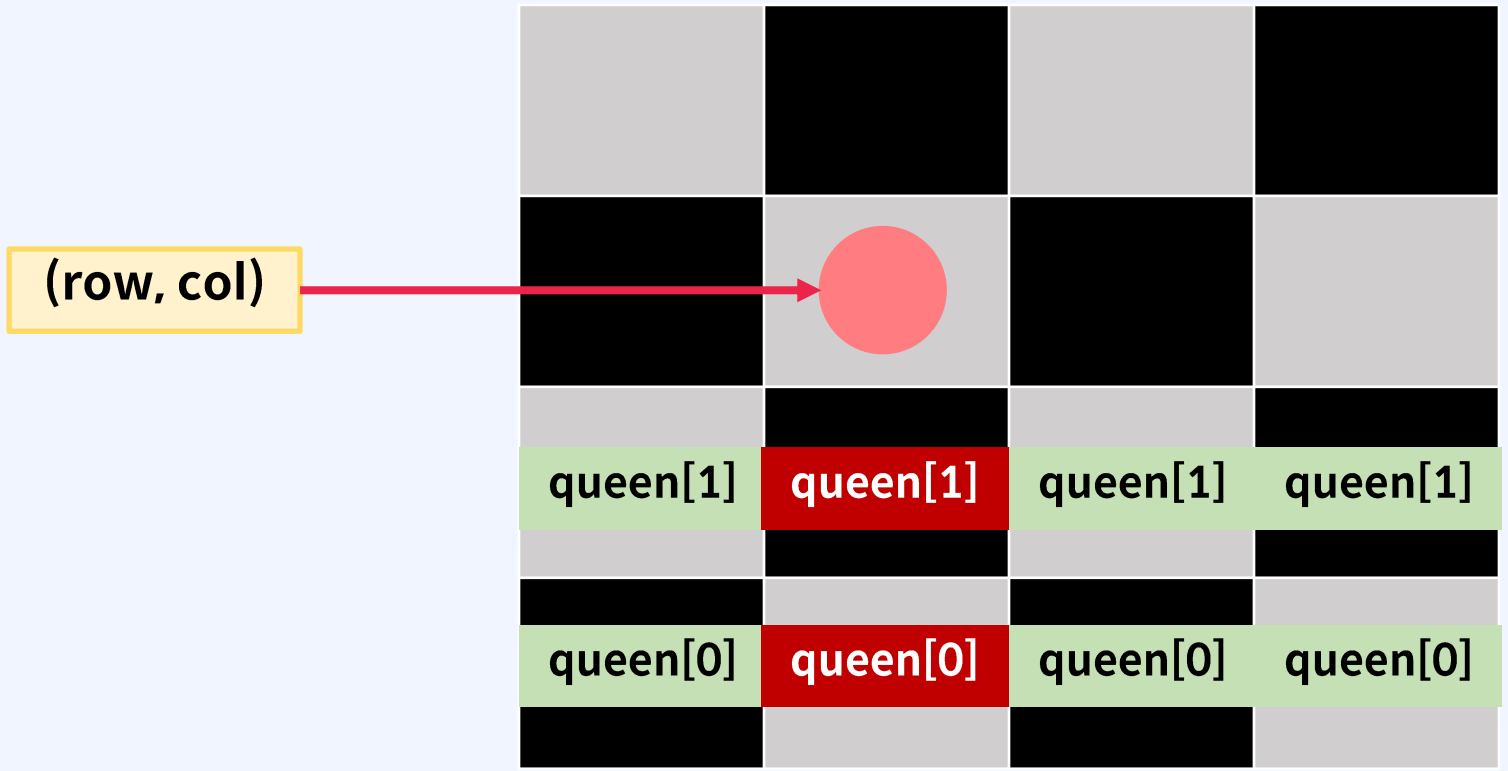
BOJ9663: N-Queen

```
for (int i = 0; i < row; i++) {
    if (queen[i] == col) return false;
    if (Math.abs(row - i) == Math.abs(col - queen[i])) return false;
}
```



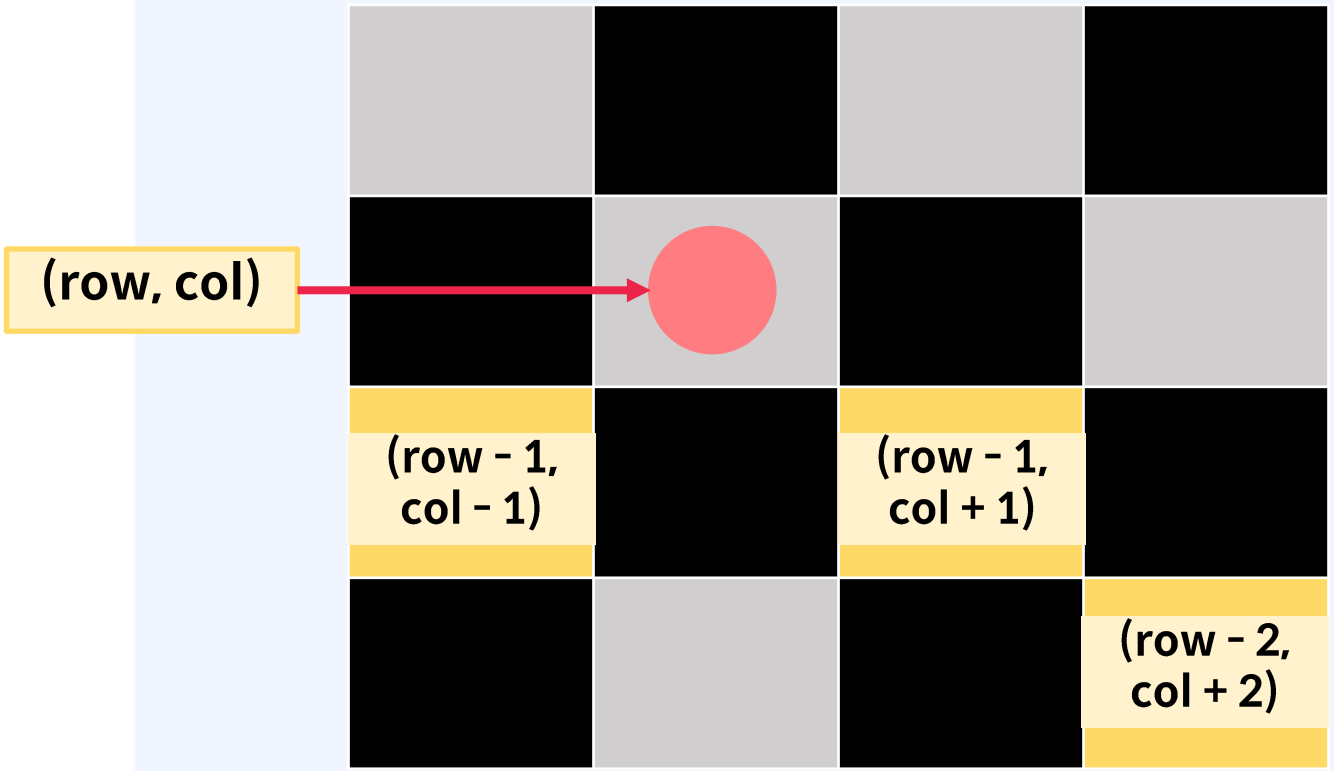
BOJ9663: N-Queen

```
for (int i = 0; i < row; i++) {
    if (queen[i] == col) return false;
    if (Math.abs(row - i) == Math.abs(col - queen[i])) return false;
}
```



BOJ9663: N-Queen

```
for (int i = 0; i < row; i++) {
    if (queen[i] == col) return false;
    if (Math.abs(row - i) == Math.abs(col - queen[i])) return false;
}
```



대각선의 좌표는 (row, col) 로 부터
같은 스칼라만큼 떨어져 있다 (방향은 자유)

따라서 절댓값을 씌워 행과 열의 차이가 같은지 체크하면, 대각선상 인지 확인할 수 있다

BOJ9663: N-Queen

Base Case
- 재귀 함수가 마지막 행에 퀸을 배치한 경우
  - 0번째 행부터 순서대로 모든 경우의 수를 가지 치며 카운트

Recursive Case
- [0, n) 열을 돌아다니며 퀸 배치를 시도

```
public static int solve(int n, int row) {
    int count = 0;
    if(row == n) return 1;
    for(int col = 0; col < n; col++) {
        if(isValid(row, col)) {
            queen[row] = col;
            count += solve(n, row + 1);
        }
    }
    return count;
}
```

Base Case

Recursive Case