알고리즘 스터디 8주차

- 완전탐색 & 시뮬레이션 –

발표자 장수현

완전탐색(Brute Force)

- 모든 경우를 탐색하는 알고리즘
- 정확도↑속도↓
- 탐색해야 할 데이터가 100만개 이하일 때 적절
- 푸는 방법?
  1. 반복/조건문(for/if)
  2. 순열, 조합
  3. 재귀함수, 백트랙킹
  4. 비트마스크
  5. DFS/BFS

완전탐색(Brute Force)

1. 반복/조건문(for/if)

- ex) 자물쇠 암호를 찾는 경우

2. 순열, 조합

```
from itertools import permutations, combinations

arr = [1, 2, 3]

result_p = list(permutations(arr, 2))
result_c = list(combinations(arr, 2))
```

```
[(1, 2), (1, 3), (2, 1), (2, 3), (3, 1), (3, 2)]
[(1, 2), (1, 3), (2, 3)]
```

완전탐색(Brute Force)

3. 재귀함수, 백트랙킹

- 재귀함수는 자기 자신을 호출하는 함수
- 백트랙킹은 재귀함수를 설계할 때 고려해야 할 아주 중요한 개념이다.
- 답을 찾아가는 도중에 답이 될 것 같지 않은 경로가 있다면 더 이상 가지 않고 back을 하는 것이다.

N-Queen

https://www.acmicpc.net/problem/9663

- 퀸은 가로, 세로, 대각선으로 움직일 수 있고 공격할 수 있다.
- 따라서 퀸은 행마다 하나만 존재할 수 있다.

N-Queen 성공

☆

5 골드 V

시간 제한	메모리 제한	제출	정답	맞힌 사람	정답 비율
10 초	128 MB	51924	26549	17381	50.510%

문제

N-Queen 문제는 크기가 N × N인 체스판 위에 퀸 N개를 서로 공격할 수 없게 놓는 문제이다.

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

입력

첫째 줄에 N이 주어진다. (1 ≤ N < 15)

출력

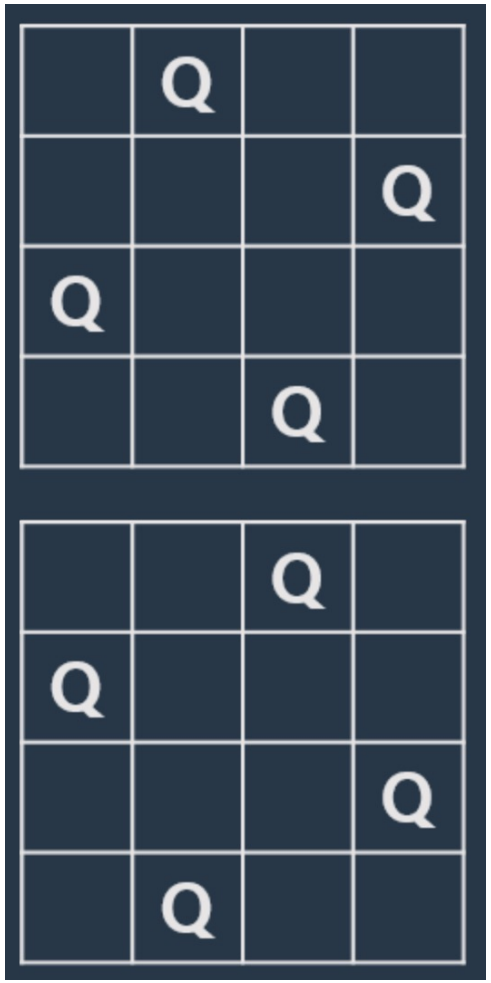
첫째 줄에 퀸 N개를 서로 공격할 수 없게 놓는 경우의 수를 출력한다.

N-Queen

[예시]

N = 4

result = 2



N-Queen

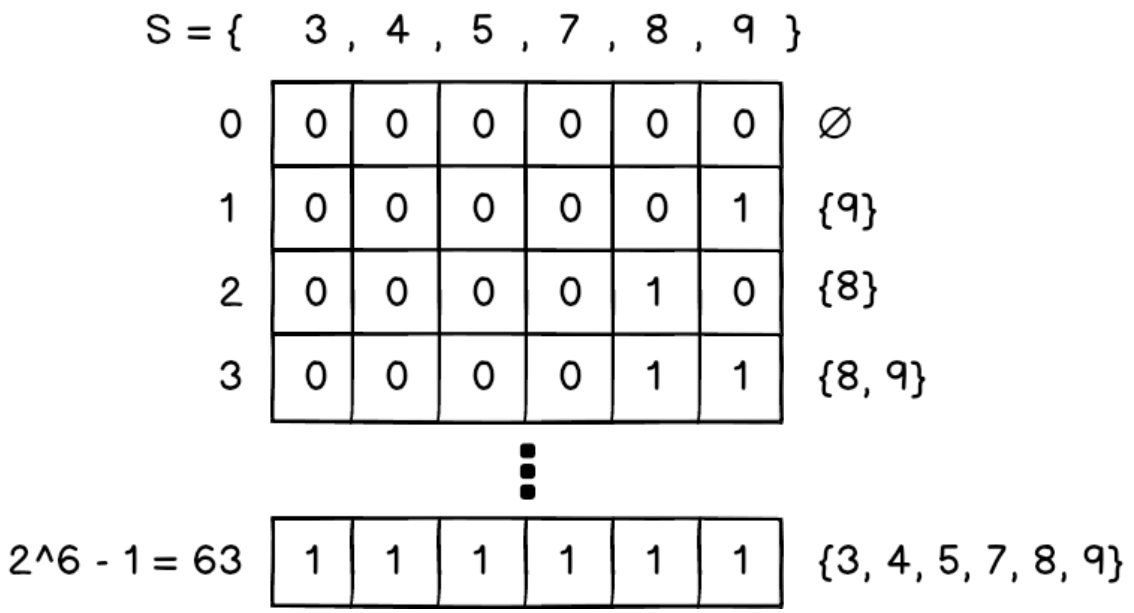
```python
def queen(n, N) :
    global result

    if n == N: # N개의 퀸이 배치되면 리턴
        result += 1
        return

    for i in range(N): # n행의 열들을 탐색(가로 탐색)
        row[n] = i # n행의 퀸 위치 변경
        for j in range(n): # 배치한 퀸의 이전 행들을 탐색
            if row[j] == row[n] or abs(row[n] - row[j]) == n - j: # 세로나 대각선에 겹치는 경우
                break
        else: # break가 실행되지 않으면 다음 퀸 배치로 넘어감
            queen(n+1, N)

N = int(input()) # 퀸의 개수
result = 0 # 경우의 수
row = [0] * N # 각 행의 퀸 위치
queen(0, N)

print(result)
```

완전탐색(Brute Force)

4. 비트마스크

- 비트연산을 통해서 부분 집합을 표현하는 방법
- 모든 경우의 수가 각각의 원소에 포함되거나 포함되지 않는 두 가지 선택으로 구성되는 경우 용이
- 쓰는 이유?

- 더 작은 메모리 사용량
- 정수 하나로 배열을 대체할 수 있기 때문에



완전탐색(Brute Force)

5. DFS/BFS

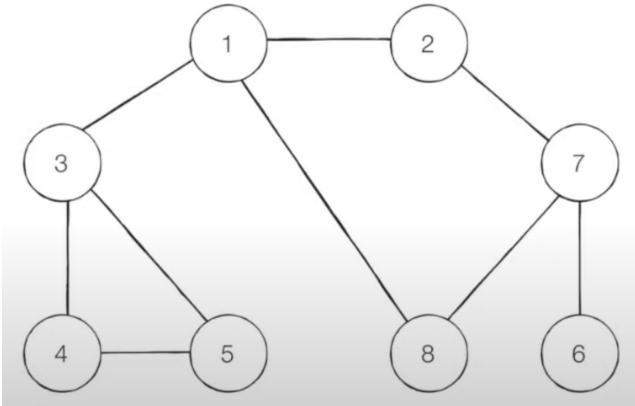
- 그래프에서 모든 정점을 탐색하기 위한 방법
- DFS - 깊이우선탐색, 스택이나 재귀로 구현
- BFS - 너비우선탐색, 큐로 구현

DFS/BFS

```python
def dfs(v):
    visited[v] = 1 # 현재 노드 방문 처리
    print(v, end=' ')

    # 현재 노드와 연결된 다른 노드 방문
    for i in graph[v]:
        if not visited[i]:
            dfs(i)

graph = [
        [],
        [2, 3, 8],
        [1, 7],
        [1, 4, 5],
        [3, 5],
        [3, 4],
        [7],
        [2, 6, 8],
        [1, 7]
]
visited = [0] * 9 # 방문확인 리스트
dfs(1)
```

```python
from collections import deque

def bfs(start):
    queue = deque([start]) # 시작노드 큐에 삽입
    visited[start] = 1 # 시작노드 방문처리

    while queue:
        v = queue.popleft()
        print(v, end=' ')
        # 해당 노드와 연결된 노드 큐에 삽입
        for i in graph[v]:
            if not visited[i]:
                queue.append(i)
                visited[i] = 1

graph = [
        [],
        [2, 3, 8],
        [1, 7],
        [1, 4, 5],
        [3, 5],
        [3, 4],
        [7],
        [2, 6, 8],
        [1, 7]
]
visited = [0] * 9 # 방문확인 리스트
bfs(1)
```

```
dfs:  1 2 7 6 8 3 4 5
bfs:  1 2 3 8 7 4 5 6
```



시뮬레이션

- 문제에서 제시한 알고리즘을 한 단계씩 차례대로 직접 수행해야 하는 문제 유형
- 2차원 공간에서의 방향벡터가 자주 활용된다

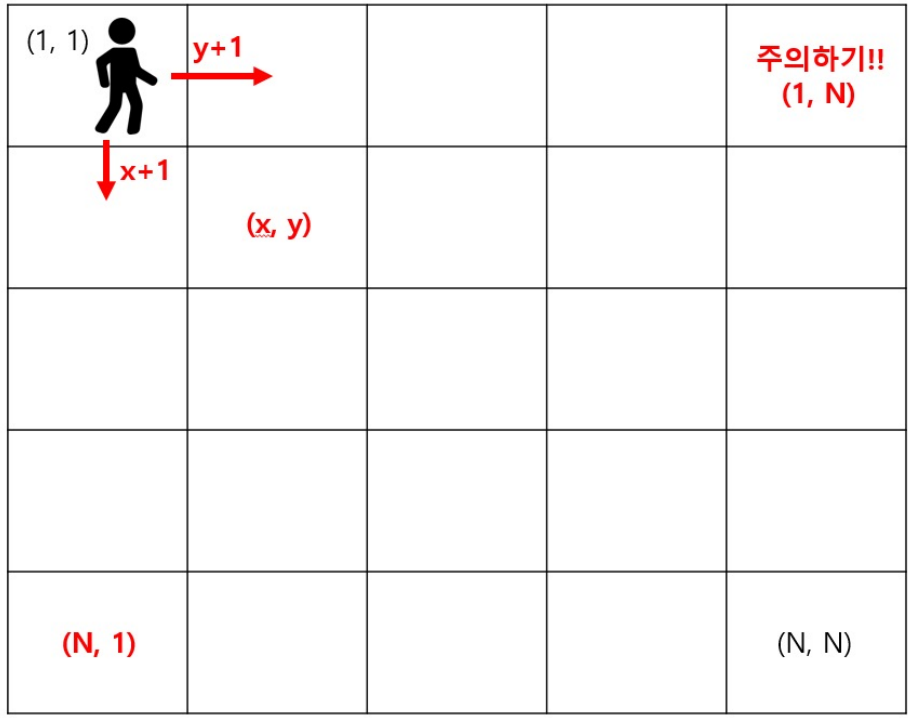
상하좌우

- 여행가 A씨는 N x N 크기의 정사각형 공간 위에 서 있습니다. 이 공간은 1 x 1 크기의 정사각형으로 나누어져 있습니다. 가장 왼쪽 위 좌표는 (1,1)이며, 가장 오른쪽 아래 좌표는 (N,N)에 해당합니다. 여행가 A는 상, 하, 좌, 우 방향으로 이동할 수 있으며, 시작 좌표는 항상 (1,1) 입니다. 우리 앞에는 여행가 A가 이동할 계획이 적힌 계획서가 놓여 있습니다.
- 계획서에는 하나의 줄에 띄어쓰기를 기준으로 하여 L, R, U, D 중 하나의 문자가 반복적으로 적혀 있다. 각 문자의 의미는 다음과 같습니다.
    - L: Left
    - R: Right
    - U: Up
    - D: Down
- 이때 여행가 A가 N x N 크기의 정사각형 공간을 벗어나는 움직임은 무시됩니다. 예를 들어 (1, 1)의 위치에서 L 혹은 U를 만나면 무시됩니다.
    - 입력 조건1: 첫째 줄에 공간의 크기를 나타내는 N이 주어집니다. (1 <= N <= 100)
    - 입력 조건2: 둘째 줄에 여행가 A가 이동할 계획서 내용이 주어집니다. (1<= 이동 횟수 <=100)
    - 출력 조건: 첫째 줄에 여행가 A가 최종적으로 도착할 지점의 좌표 (X, Y)를 공백을 기준으로 구분하여 출력합니다.

상하좌우

<입력 예시>
5
R R R U D D

<출력 예시>
3 4




```python
# 상하좌우에 따른 이동방향
move_types = ['U', 'D', 'L', 'R']
dx = [-1, 1, 0, 0]
dy = [0, 0, -1, 1]

# 시작 좌표
x, y = 1, 1

# 이동 계획에 따라 움직인다
for plan in plans:
    # 이동방향 확인
    for i in range(len(move_types)):
        if plan == move_types[i]:
            nx = x + dx[i]
            ny = y + dy[i]

    # 공간을 벗어나는 경우 무시
    if nx<1 or ny<1 or nx>n or ny>n:
        continue

    # 이동수행
    x, y = nx, ny

print(x,y)
```