완전 탐색

23.01.27

완전 탐색?

- Exhaustive Search

- 컴퓨터의 빠른 계산 능력을 이용하여 가능한 경우의 수를 일일이 나열하면서 답을 찾는 방법

즉, 무식하게 가능한 것 다 해보겠다!

1. 사용된 알고리즘이 적절한가? (문제를 해결할 수 있는가)

2. 효율적으로 동작하는가? (시간 복잡도)

백준 1912 - 연속합 문제

n개의 정수로 이루어진 임의의 수열이 주어질 때 이 중 연속된 몇 개의 수를 선택해 구할 수 있는 합의 최대(수는 한 개 이상 선택)
(1 <= n <= 100,000)

ex_ [10, −4, 3, 1, 5, 6, −35, 12, 21, −1]

- dp[n] = max(dp[n−1] + arr[n], arr[n]) →O(N)
- 완전 탐색 → O(N^2)

▶ O(N)

- N <= 1억
- 1억 개 경우의 수 연산까지 1초

▶ O(N log N)

- N <= 500만
- 5백만 개 경우의 수 연산까지 1초

▶ O(N^2)

- N <= 1만
- 1만 개 경우의 수 연산까지 1초

▶ O(N!)

- N <= 11
- 11! = 39,916,800 (12! = 479,001,600)

1. 단순 Brute-Force ✓

2. 비트마스크(Bitmask) ✓

3. 순열 (Permutation) ✓

4. 재귀 함수(백트래킹) ✓

5. BFS / DFS

단순 Brute-Force

- 어느 기법을 사용하지 않고 단순히 for문과 if문 등으로 모든 case들을 만들어 답을 구하는 방법

- 이는 아주 기초적인 문제에서 주로 이용되거나, 전체 풀이의 일부분으로 이용하며, 따라서 대회나 코딩 테스트에서는 이 방법만 이용한 문제는 거의 나오지 않음

- ex_ 0000~9999 자물쇠 비밀번호 풀기, 생일 맞추기...

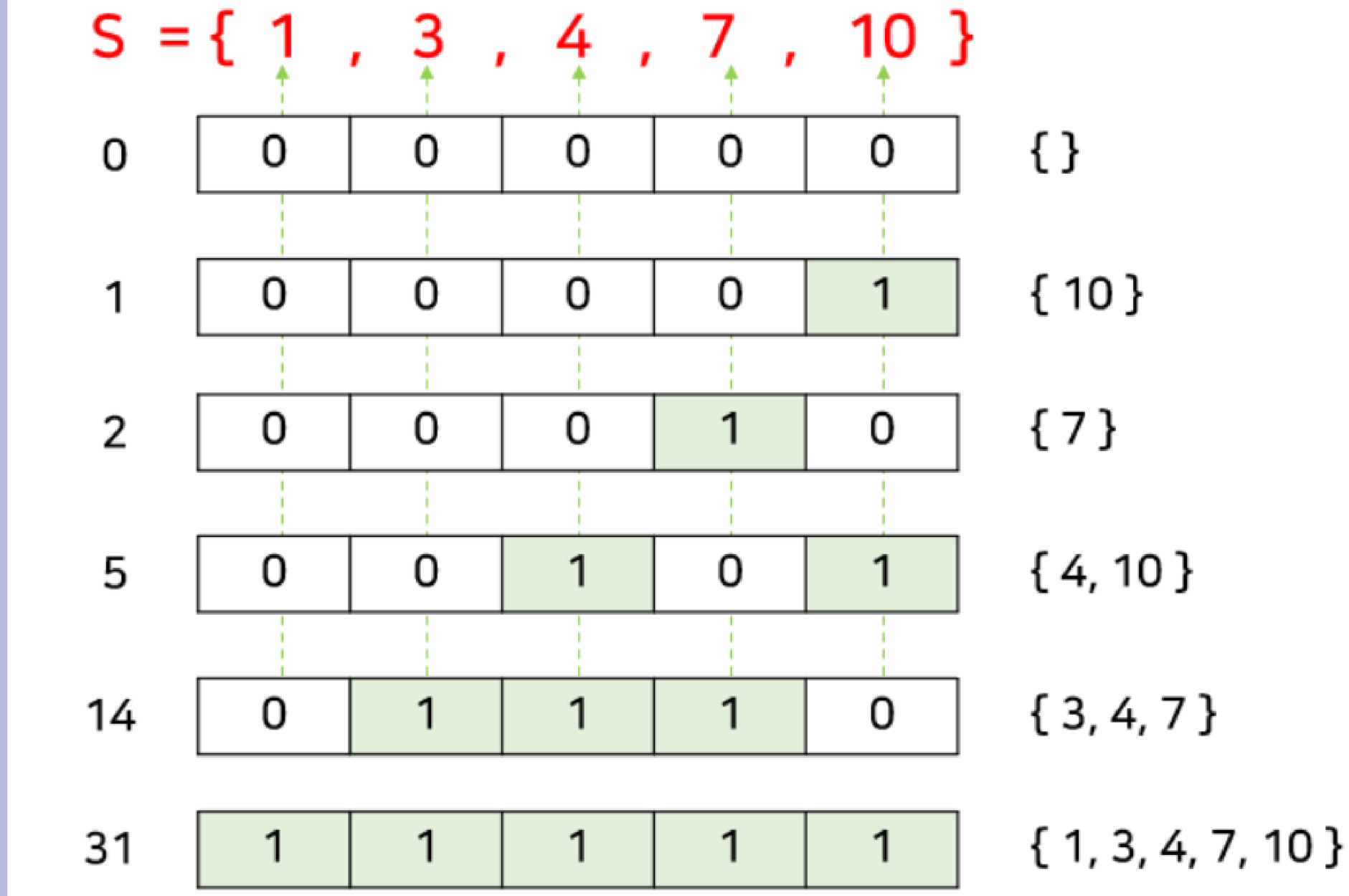
비트마스크(Bitmask)

- 2진수를 이용하는 컴퓨터의 연산을 이용하는 방식

- 모든 경우의 수가 각각의 원소가 포함되거나, 포함되지 않는 두 가지 선택으로 구성되는 경우에 사용

- ex_ 주어진 일부 배열의 원소가 전체 배열(길이=5)에서 포함 여부 확인

- 5자리 이진수 (0 ~ 31)를 이용하여 각 원소의 포함 여부를 체크 가능


S = { 1 , 3 , 4 , 7 , 10 }
0
0 0 0 0 0
{ }
1
0 0 0 0 1
{ 10 }
2
0 0 0 1 0
{ 7 }
5
0 0 1 0 1
{ 4, 10 }
14
0 1 1 1 0
{ 3, 4, 7 }
31
1 1 1 1 1
{ 1, 3, 4, 7, 10 }

순열 (Permutation) - nPr

- 서로 다른 n개의 원소에서 r개를 중복을 허용하지 않고
  순서에 상관있게 선택

- 서로 다른 N개를 일렬로 나열하는 순열의 경우의 수는 N!

→ 순열에 완전 탐색을 이용하기 위해서는 N이 충분히 작아야 함

ex_ [1, 2, 3]에 대해 순열을 만든다면 6가지 순열 가능

1 2 3
1 3 2
2 1 3
2 3 1
3 1 2
3 2 1

```python
from itertools import permutations

arr = [1, 2, 3]
for i in permutations(arr):
    print(i)
```

```
(1, 2, 3)
(1, 3, 2)
(2, 1, 3)
(2, 3, 1)
(3, 1, 2)
(3, 2, 1)
```

(arr, 2) :

```
(1, 2)
(1, 3)
(2, 1)
(2, 3)
(3, 1)
(3, 2)
```

조합 (Combination) – nCr

ex_ [1, 2, 3]에서 2가지를 뽑아 조합을 만든다면, 3가지 조합 가능

1 2
1 3
2 3

```python
from itertools import combinations

arr = [1, 2, 3]
for i in combinations(arr, 2):
    print(i)
```

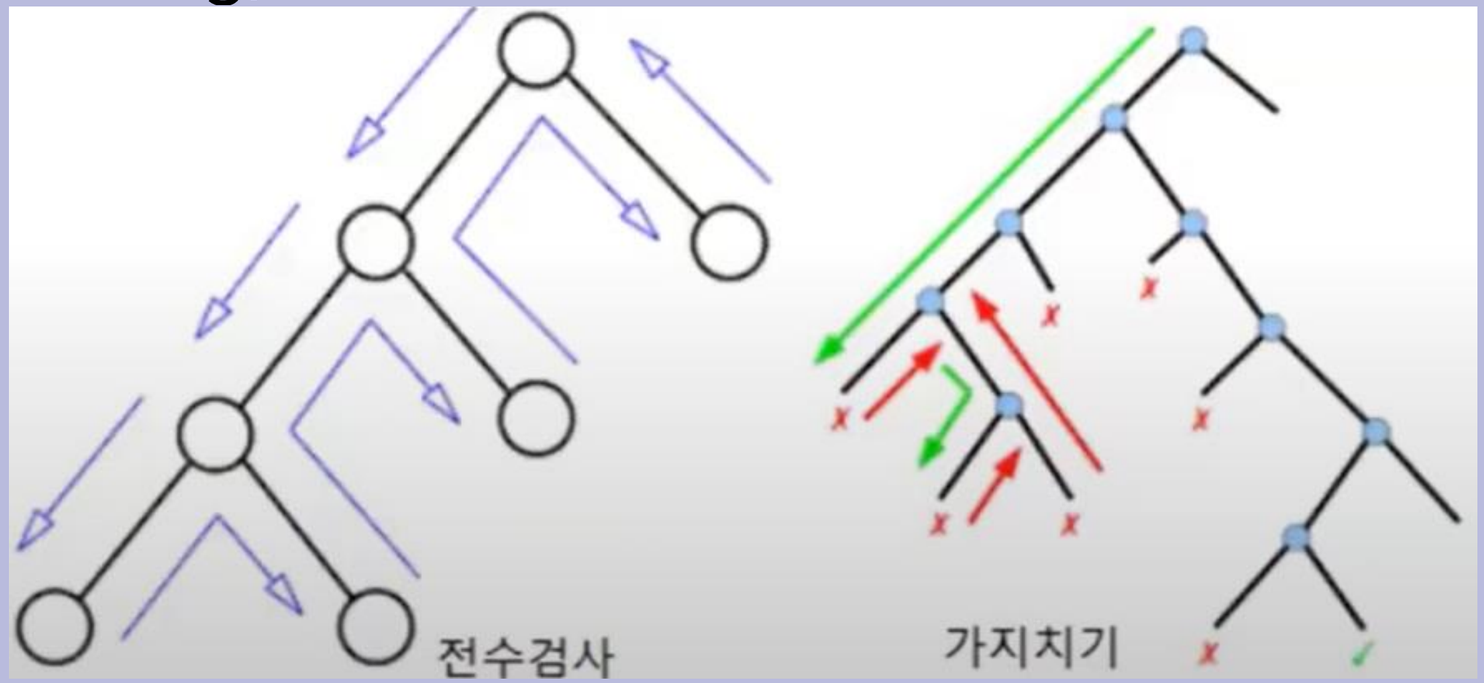
```
(1, 2)
(1, 3)
(2, 3)
```

백트래킹

- 현재 상태에서 가능한 모든 후보군을 따라 들어가며 해결책에 대한 후보를 구축해 나아가다 가능성이 없다고 판단되면 즉시 후보를 포기하면서 돌아가 다시 정답을 찾아가는 방법

- 후보를 포기 = 가지 치기(Pruning)

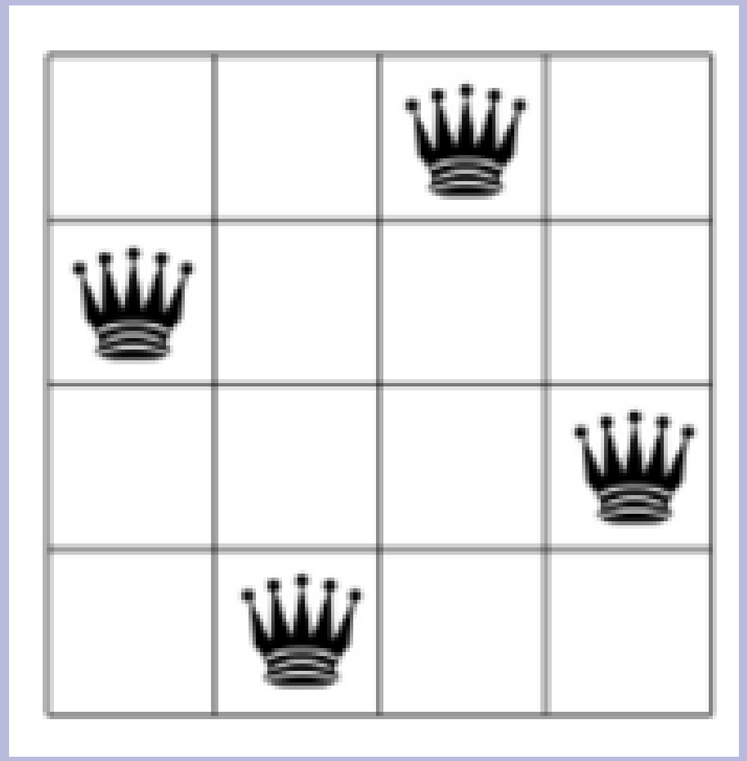
- ex_ N-Queen 문제

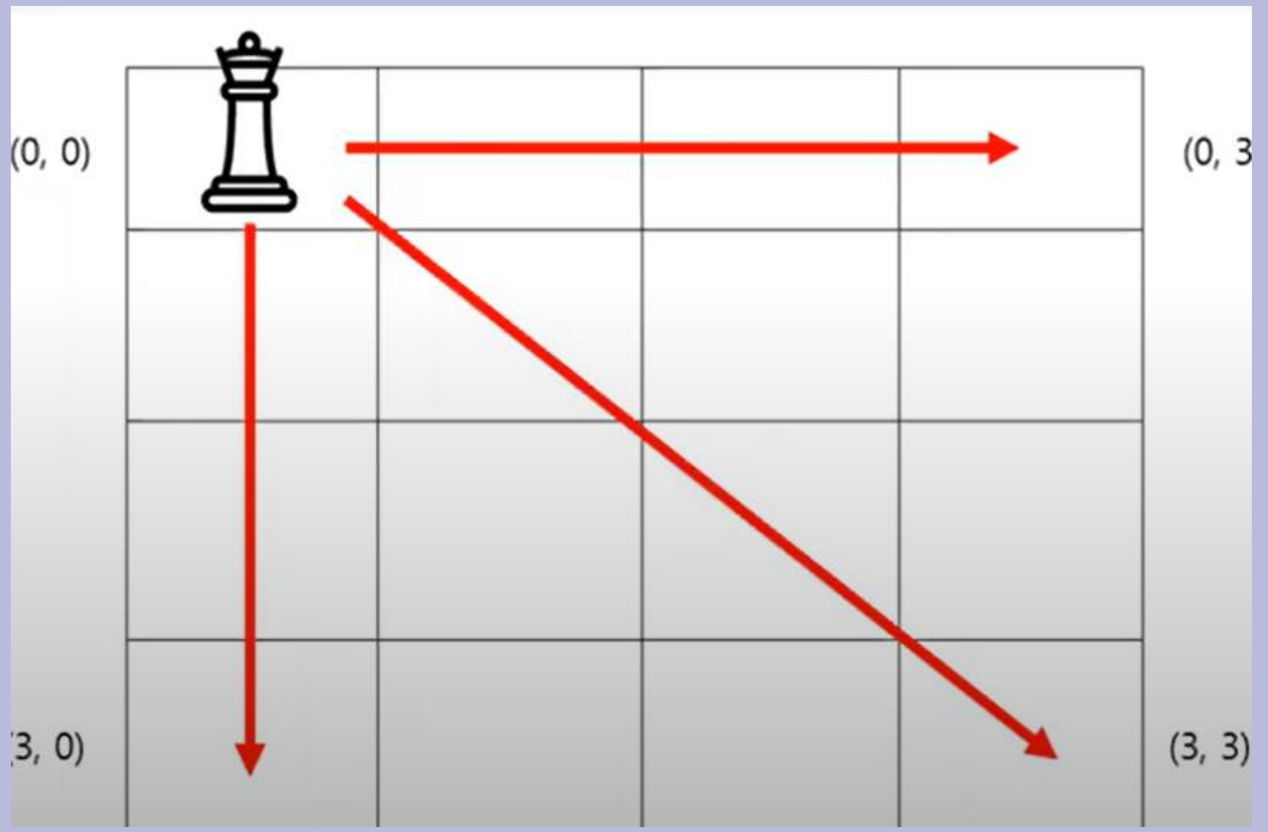


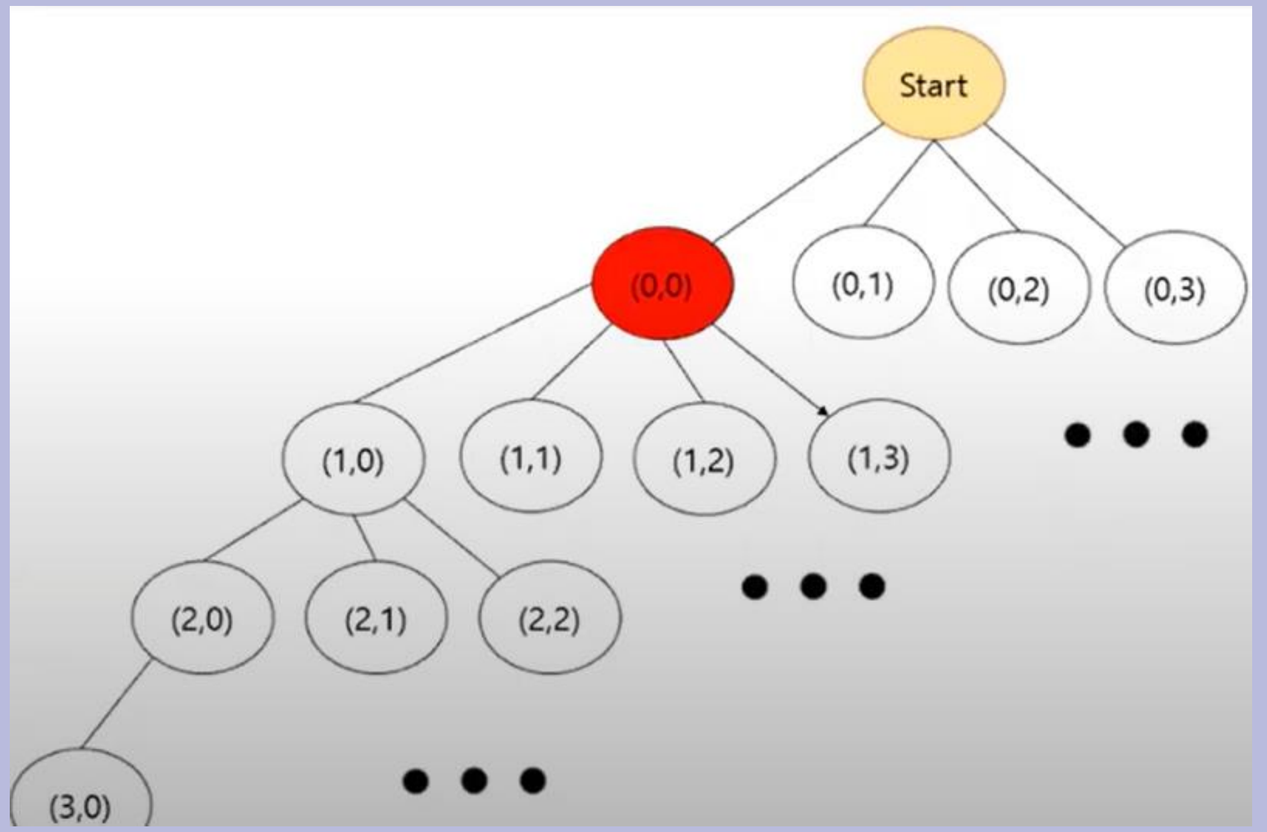
N-Queen

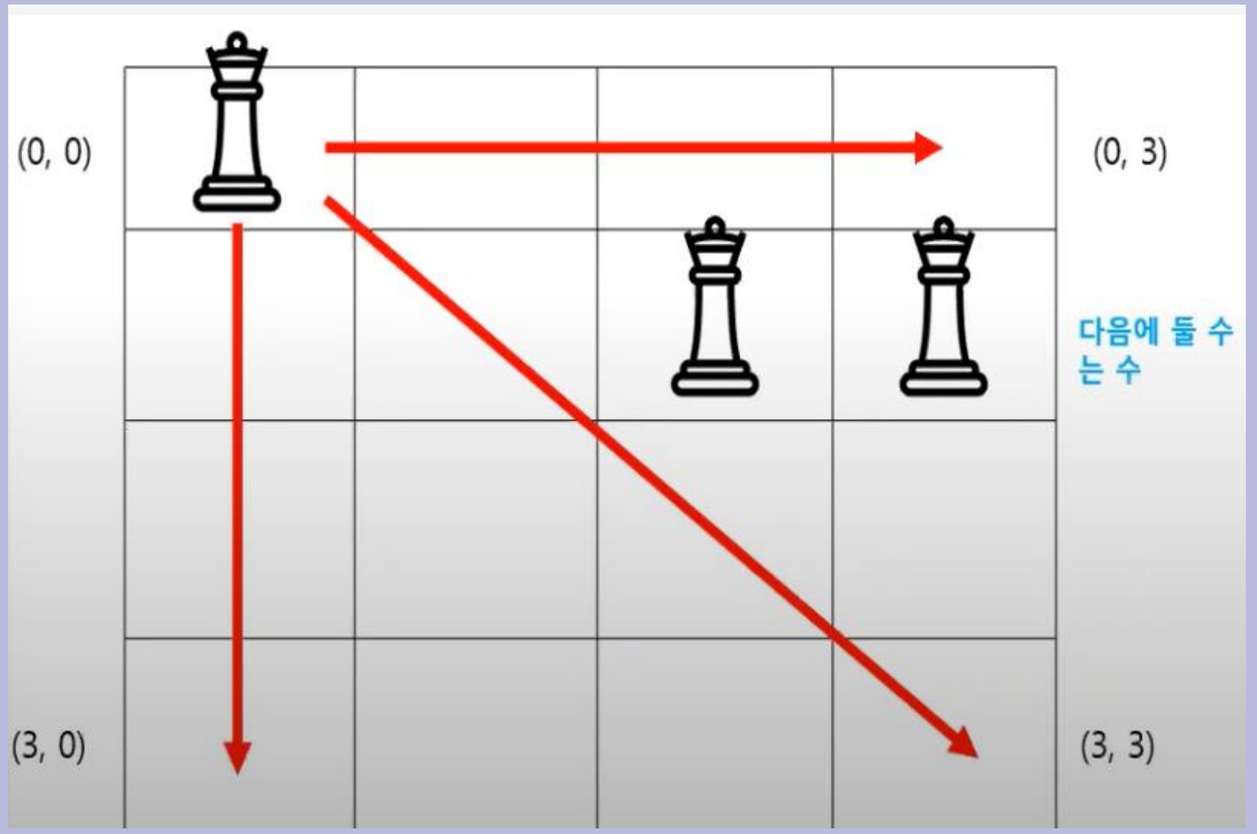
(백준 9663, 3344 등 다수 존재)

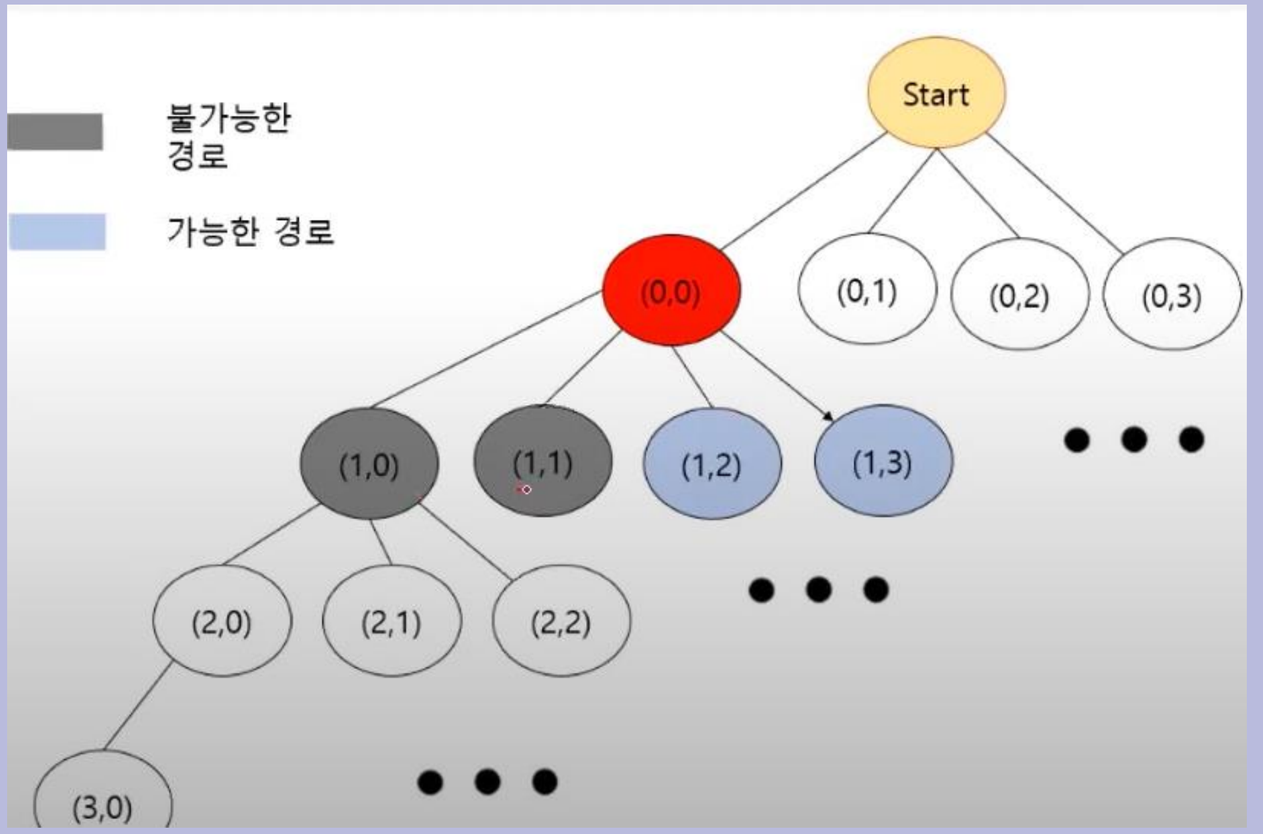
크기가 N * N 인 체스판 위에 퀸 N 개를 서로 공격할 수 없게 놓는 문제
N이 주어졌을 때, 퀸을 놓는 방법의 수를 구하는 프로그램을 작성하시오.
(퀸은 오와 열, 대각선 이동이 가능한 말이다.)

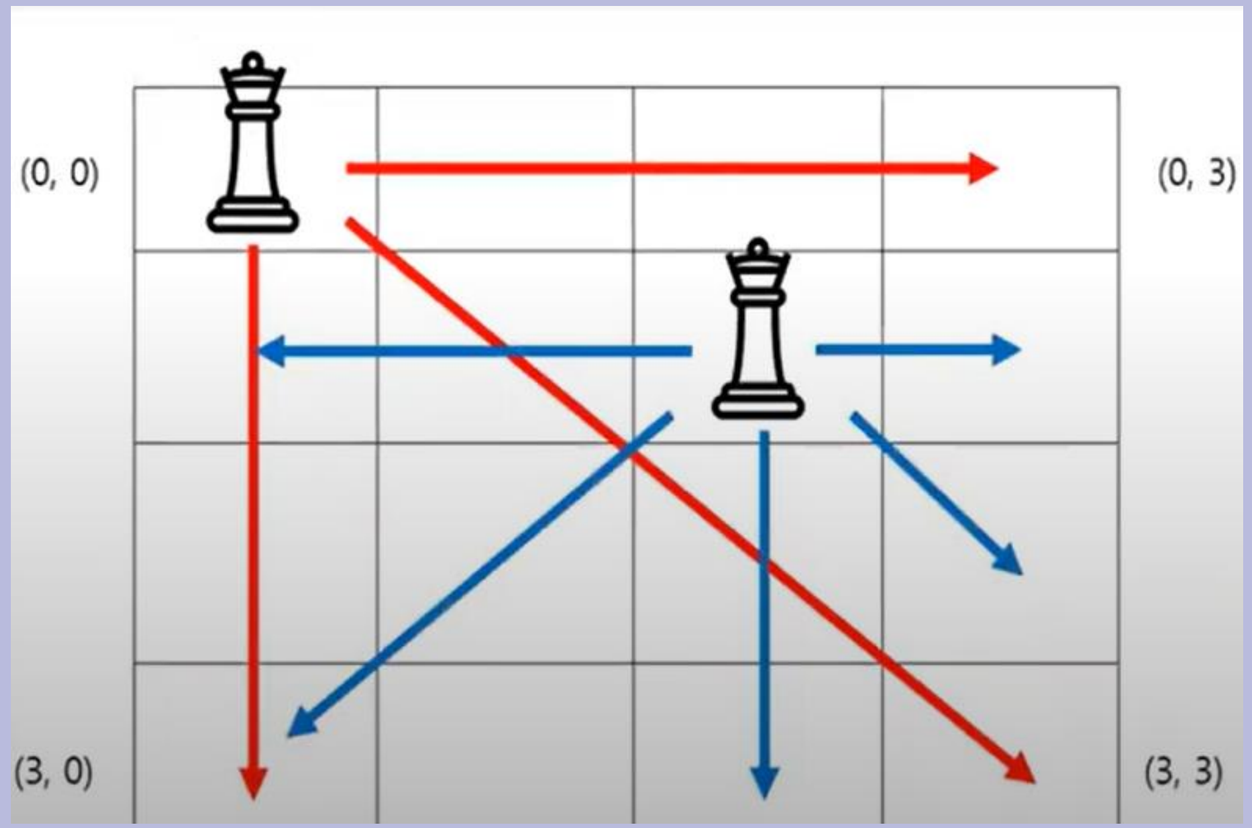


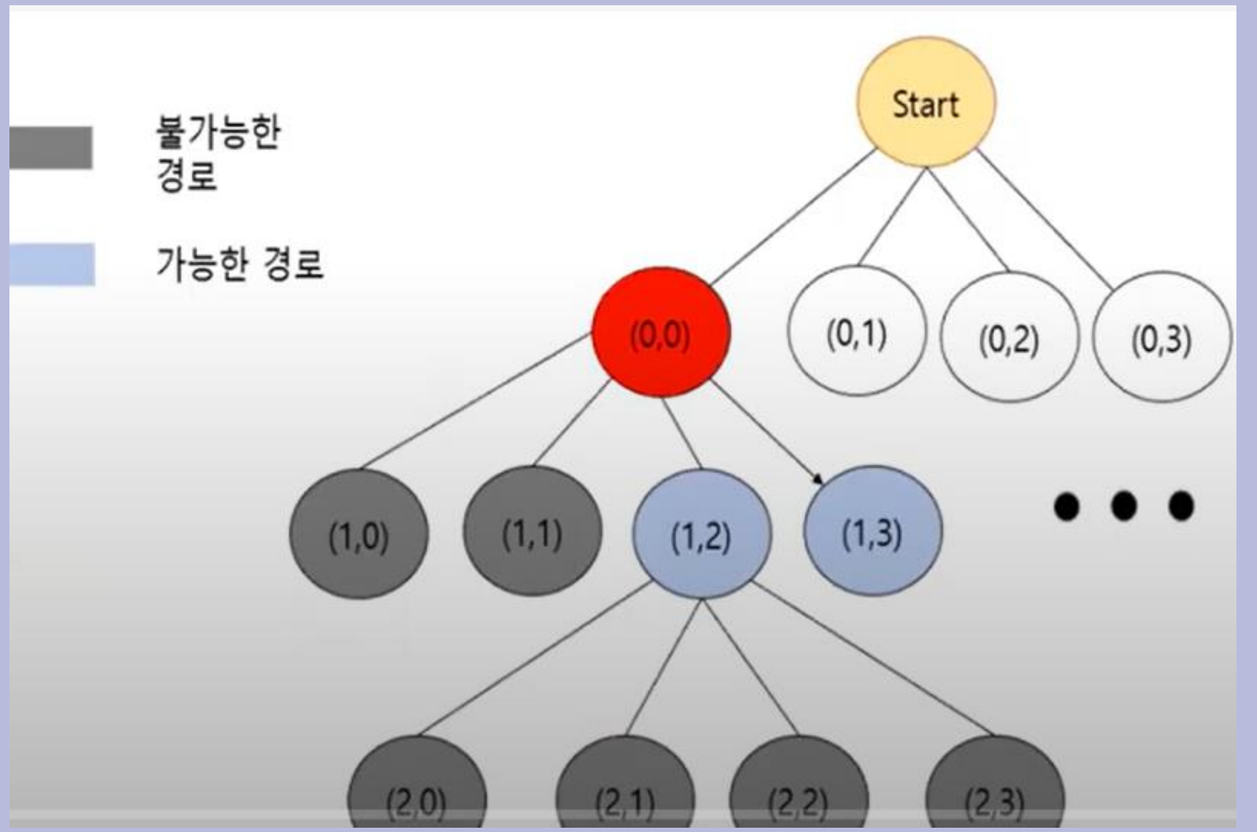

(0, 0)
(0, 3
3, 0)
(3, 3)

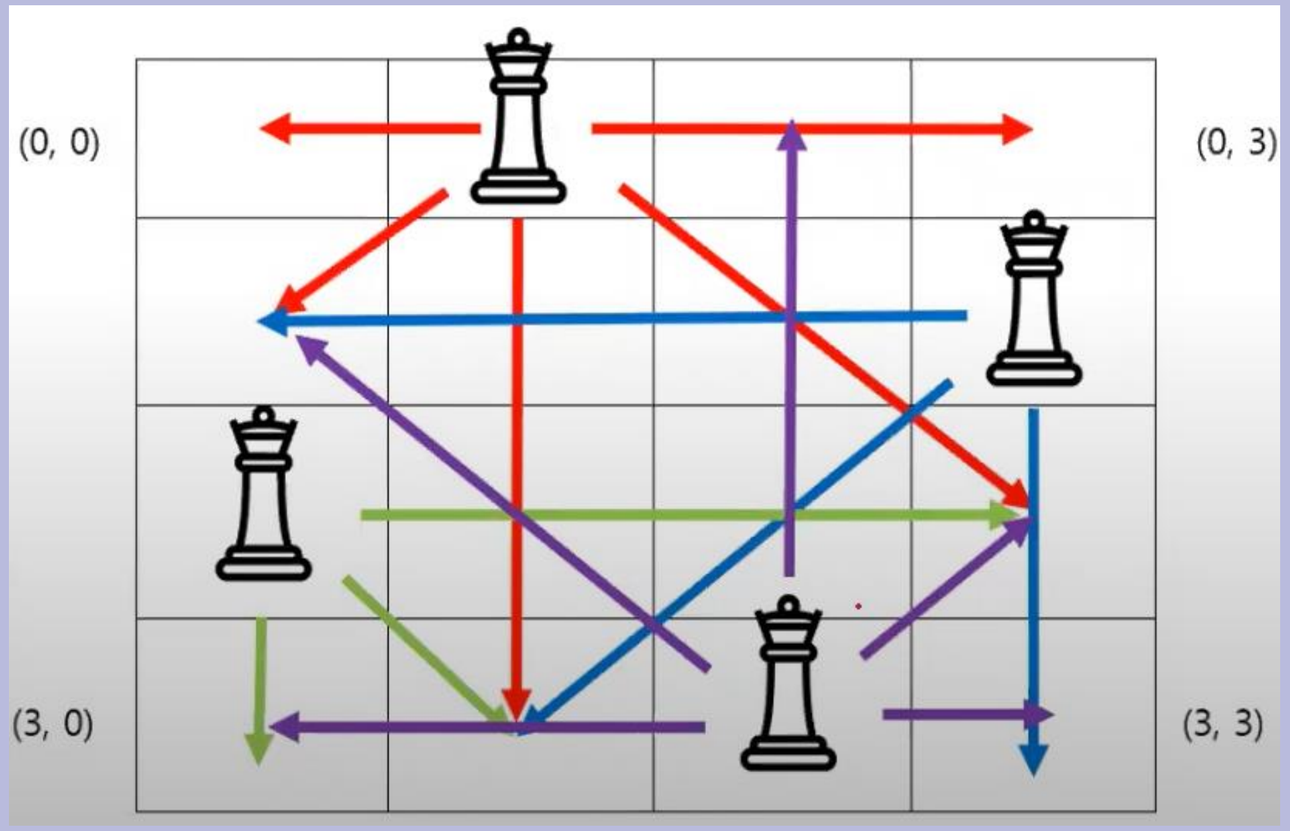

Start
(0,0)
(0,1)
(0,2)
(0,3)
(1,0)
(1,1)
(1,2)
(1,3)
(2,0)
(2,1)
(2,2)
(3,0)

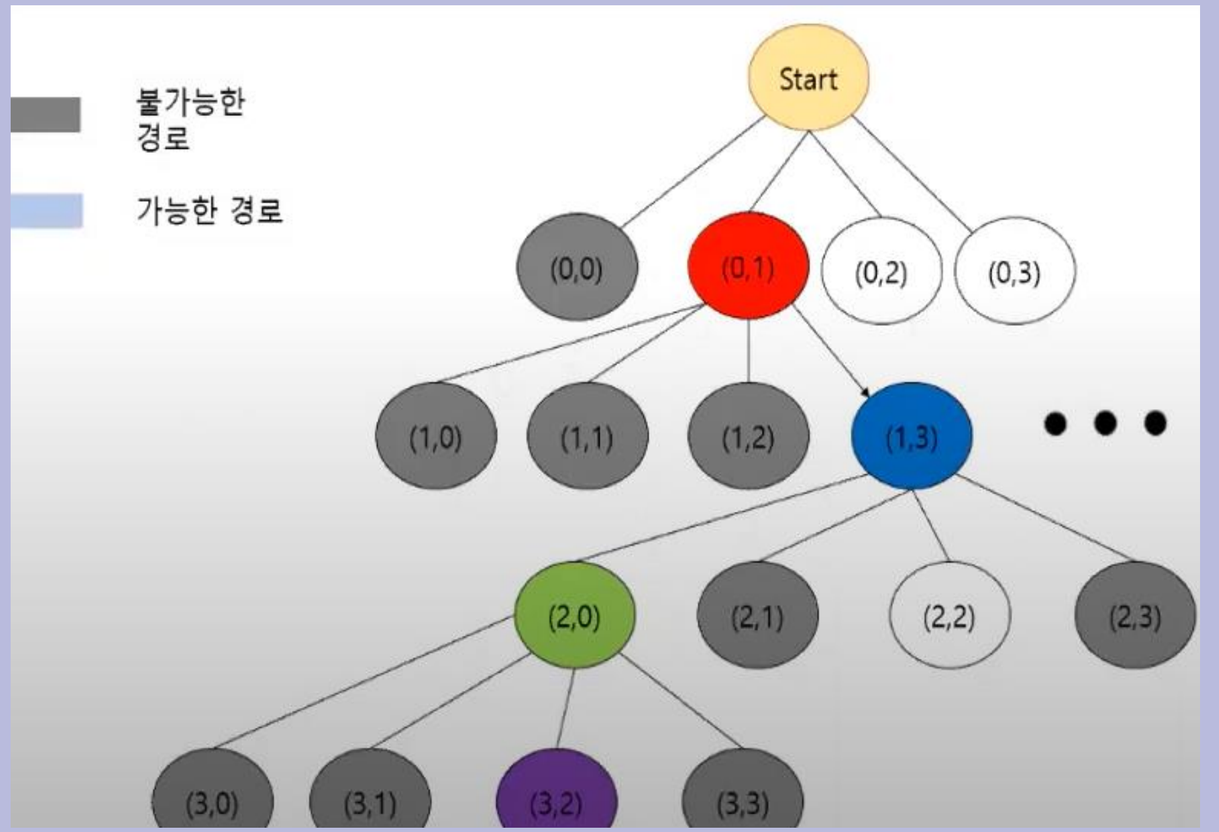

(0, 0)
(0, 3)
다음에 둘 수 는 수
(3, 0)
(3, 3)


불가능한 경로
가능한 경로
Start
(0,0)
(0,1)
(0,2)
(0,3)
(1,0)
(1,1)
(1,2)
(1,3)
(2,0)
(2,1)
(2,2)
(3,0)


(0, 0)
(0, 3)
(3, 0)
(3, 3)


불가능한
경로
가능한 경로
Start
(0,0)
(0,1)
(0,2)
(0,3)
(1,0)
(1,1)
(1,2)
(1,3)
(2,0)
(2,1)
(2,2)
(2,3)


(0, 0)
(0, 3)
(3, 0)
(3, 3)


불가능한
경로
가능한 경로
Start
(0,0)
(0,1)
(0,2)
(0,3)
(1,0)
(1,1)
(1,2)
(1,3)
(2,0)
(2,1)
(2,2)
(2,3)
(3,0)
(3,1)
(3,2)
(3,3)

※ DFS와 백트래킹(Backtracking)

- DFS는 기본적으로 그래프 형태 자료구조에서 모든 정점을 탐색할 수 있는 알고리즘
  깊이를 우선적으로 탐색하기에 재귀 또는 스택을 이용한다.

- 재귀를 이용해 깊이 탐색을 수행한다는 부분에서 완전탐색의 백트래킹(Backtracking)과 유사

- DFS는 기본적으로 모든 노드를 탐색하는 것이 목적이지만 상황에 따라서 백트래킹 기법을 혼용하여 불 필요한 탐색을 그만두는 것이 가능

- 글로는 백트래킹이 더 좋은 알고리즘 같지만, 각 문제 별 유용하게 사용할 수 있는 방식이
  다른 것 뿐 더 좋고 나쁨 X

- 재귀 방식을 사용한다는 점에서 항상 종료 조건을 꼭 확인하기

완전 탐색 자체가 알고리즘은 아니기 때문에 완전 탐색 방법을 이용하기 위해서 앞의 여러 알고리즘 기법이 이용됨

이런 방법들도 익혀둬 잘 사용하면 까다로운 문제에서도 좀 더 효율적인 코드 작성 가능

항상 사용하려는 방법의 시간 복잡도를 먼저 계산해 제한 시간 내에 통과 할 수 있는 방법인지 먼저 생각해보기

THANK YOU!