알고리즘 특강 완전탐색

모든 경우의 수를 탐색하는 기법입니다. 당연히 아이디어는 단순하지만,
실제로 많이 사용되기 때문에 한번쯤 고민해봐야 합니다.

Brute Force

- 단순히 **가능한 모든 경우**를 확인하는 기법.
- **테스트 케이스의 크기**에 따라 소요 시간이 엄청나게 커질 수 있음.
- 그러나, 떠올릴 수 있는 **가장 쉬운 방법**이기 때문에, 문제를 풀 때 가장 먼저 고민해야 하는 방법이기도 함.
- **표본의 수와 시간복잡도를 전체적으로 고려해** 선택하는 것이 아주 중요한 기법.

매직 넘버

100,000,000

- 총 연산수가 약 1억 회 이하인 경우, 충분히 **완전탐색으로 접근할 수 있음.**
- 실제와는 거리가 있지만, 대략적인 완전탐색 가능 기준으로 잡으면 유용함.

간단한 예시부터..

Silver 5 - 영화감독 숌 (#1436)

요약

- 종말의 숫자란 어떤 수에 6이 적어도 3개 이상 연속으로 들어가는 수를 말한다.
- 즉, 666, 1666, 2666, 3666 … 은 종말의 숫자이다.
- 이때, N번째 종말의 숫자를 구하는 프로그램을 작성하시오.

제약조건

- N의 범위는 1 <= N <= 10,000 이다.

간단한 예시부터..

Silver 5 - 영화감독 숌

요약

- 종말의 숫자란 어떤 수에 6이 적어도 3개 이상 연속으로 들어가는 수를 말한다.
- 즉, 666, 1666, 2666, 3666 … 은 종말의 숫자이다.
- 이때, N번째 종말의 숫자를 구하는 프로그램을 작성하시오.

제약조건

- N의 범위는 1 <= N <= 10,000 이다.

접근

- 666 앞과 뒤에 숫자를 계속 붙이고, 정렬해볼까?

→ 충분히 가능. 다른 방법은 없을까?

간단한 예시부터..

Silver 5 - 영화감독 숌

요약

- 종말의 숫자란 어떤 수에 6이 적어도 3개 이상 연속으로 들어가는 수를 말한다.
- 즉, 666, 1666, 2666, 3666 … 은 종말의 숫자이다.
- 이때, N번째 종말의 숫자를 구하는 프로그램을 작성하시오.

제약조건

- N의 범위는 1 <= N <= 10,000 이다.

접근

- 666 앞과 뒤에 숫자를 계속 붙이고, 정렬해볼까?
  - → 충분히 가능. 다른 방법은 없을까?
- 1부터 수를 계속 올리면서 666 있는지 판단 하는 건?
  - → 10000번째 종말의 숫자는 6,669,999 보다 작음.
    (Why? 6,660,000 ~ 6,669,999: 1만개)
  - → 6,669,999 < 100,000,000 이므로,
    충분히 **루프를 돌려 해결할 수 있음.**
- 다른 풀이가 있을 수 있지만,
  - 완전탐색 풀이가 더 간단한 경우가 많음.
  - 시험장에서 다른 풀이가 떠오르지 않을 수 있음.
  - → **완전탐색을 먼저 떠올려 보는 것이 유리하다!**

난이도를 살짝 올렸습니다.

Silver 4 - 퇴사 (#14501)

요약

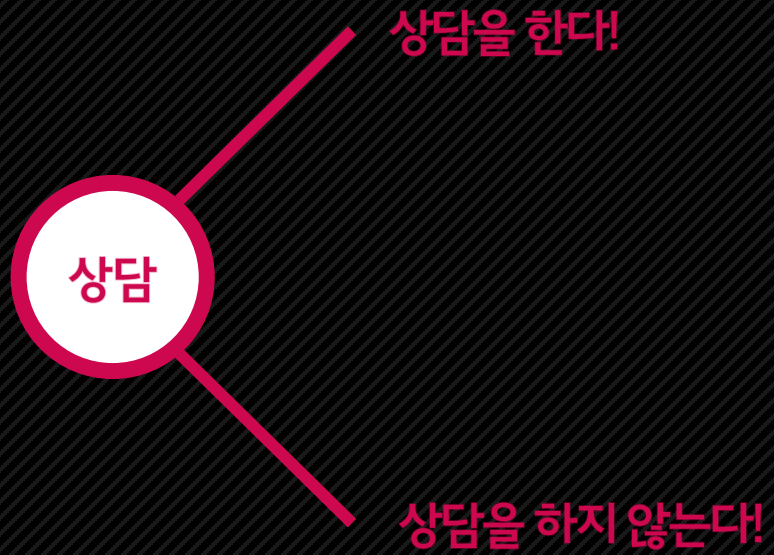
- N + 1번째 날 퇴사를 하기 위해 N일 동안 최대한 많은 상담을 하려고 한다.
- 1 ~ N 일 까지 매일 상담 스케쥴이 있는데, 각각의 상담은 상담을 완료하는데 걸리는 시간 T_i와 보수 P_i가 존재한다.
- 상담을 적절히 했을 때, 얻을 수 있는 최대 수익을 구하는 프로그램을 작성하시오.

제약조건

- 퇴사까지 남은 날은 $1 <= N <= 15$ 이다.
- 시간 T_i와 P_i의 범위는 각각 $1 <= T_i <= 5$, $1 <= P_i <= 1,000$ 이다.

완탐이 되는거 맞아요?

Silver 4 - 퇴사 (#14501)



물론…

Silver 4 - 퇴사 (#14501)

모든 T_i가 1이라면?

Silver 4 - 퇴사 (#14501)

재귀함수 써보기

Silver 4 - 퇴사 (#14501)

```
function recursive(day, remain, money) do
    if day == N
        answer = max(answer, money)
        return
    if remain <= 0
        recursive(day + 1, T[day], money + P[day])
    recursive(day + 1, remain - 1, money)
```

복잡해보이지만…
엄청 단순합니다.

Silver 3 - 마인크래프트 (#18111)

요약

- 마인크래프트는 1 * 1 * 1 크기의 블록들로 이루어진 세계에서 땅을 파거나 집을 지을 수 있는 게임이다.
- 게임 상에서 특정 구간의 땅을 평평하게 하는 작업을 하려고 한다.
- 블록을 하나 제거하는데 1초, 인벤토리에서 블록을 하나 꺼내 설치하는데 2초가 걸린다.
- 각각의 땅의 높이와 인벤토리 내 블록의 수가 주어질 때, 땅을 고르는데 걸리는 최소 시간을 구하는 프로그램을 작성하시오.

제약조건

- 땅의 가로와 세로의 길이는 $1 <= N, M <= 500$ 이다.
- 인벤토리 내 블록의 수의 범위는 $0 <= B <= 6.4 * 10^7$ 이다.
- 땅의 높이의 범위는 $1 <= H <= 256$ 이다.

그래서 우리가 구해야하는게 뭔데?

Silver 3 - 마인크래프트 (#18111)

- 각각의 땅의 높이와 인벤토리 내 블록의 수가 주어질 때, **땅을 고르는데 걸리는 최소 시간**을 구하는 프로그램을 작성하시오.

그래서 우리가 구해야하는게 뭔데?

Silver 3 - 마인크래프트 (#18111)

- 각각의 땅의 높이와 인벤토리 내 블록의 수가 주어질 때, **땅을 고르는데 걸리는 최소 시간**을 구하는 프로그램을 작성하시오.

↓

특정 높이가 정해지면, 최소 시간이 나오겠지…

그래서 우리가 구해야하는게 뭔데?

Silver 3 - 마인크래프트 (#18111)

- 각각의 땅의 높이와 인벤토리 내 블록의 수가 주어질 때, **땅을 고르는데 걸리는 최소 시간**을 구하는 프로그램을 작성하시오.

↓

특정 높이가 정해지면, 최소 시간이 나오겠지…

↓

높이는 1 ~ 256이니까, **그냥 다 시도하면 해결…**

Backtracking

- 완전탐색처럼 모든 경우를 탐색하나, 중간 중간에 **조건에 맞지 않는 케이스를 가지치기** 하여 탐색 시간을 줄이는 기법.
- 일반적으로 완전탐색에 비해 **시간적으로 효율적임**.
- 탐색 중 조건에 맞지 않는 경우 이전 과정으로 돌아가야 하기 때문에, **재귀를 사용하는 경우가 많음**.
- 조건을 **어떻게 설정하고, 틀렸을 시 어떤 시점으로 돌아가야 할지** 설계를 잘 해야 함.

문제를 한 번 봅시다…

Silver 3 - N과 M (1)

요약

- 자연수 N과 M이 주어졌을 때, 1부터 N까지 자연수 중 **중복없이 M개를 고른 수열**을 **모두** 구하는 프로그램을 작성하시오.

제약조건

- N과 M의 범위는 1 <= M <= N <= 8 이다.

문제를 한 번 봅시다…

Silver 3 - N과 M (1)

요약

- 자연수 N과 M이 주어졌을 때, 1부터 N까지 자연수 중 중복없이 M개를 고른 수열을 모두 구하는 프로그램을 작성하시오.

제약조건

- N과 M의 범위는 1 <= M <= N <= 8 이다.

접근

- 어떤 제약 조건이 있을까?
  - → 수열 내부에는 중복이 있으면 안된다.
  - → 수열 크기는 M이어야 한다.

문제를 한 번 봅시다…

Silver 3 - N과 M (1)

요약

- 자연수 N과 M이 주어졌을 때, 1부터 N까지 자연수 중 중복없이 M개를 고른 수열을 모두 구하는 프로그램을 작성하시오.

제약조건

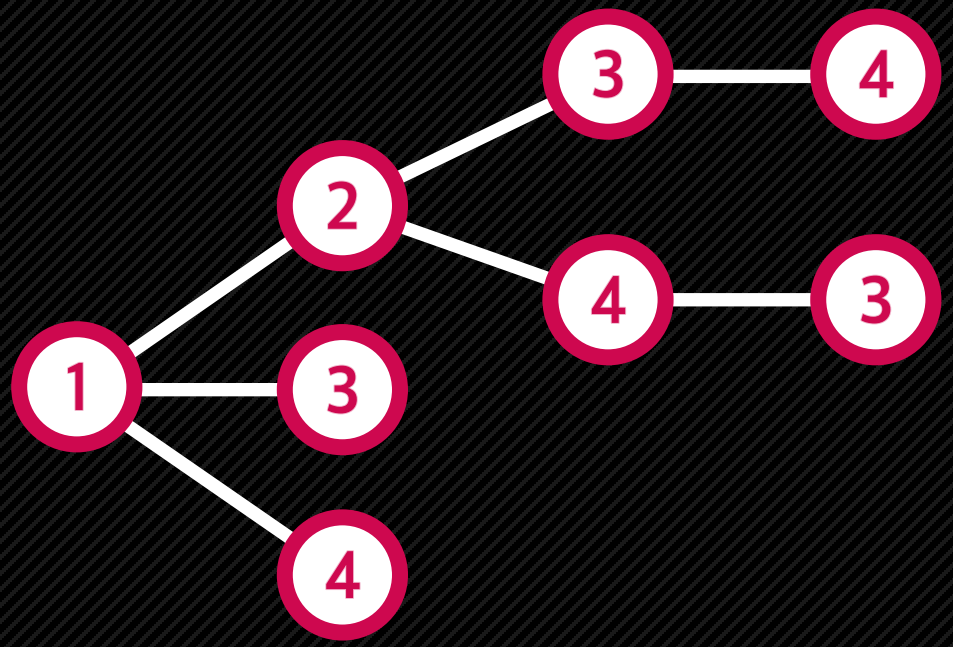
- N과 M의 범위는 1 <= M <= N <= 8 이다.

접근

- 어떤 제약 조건이 있을까?
  - → 수열 내부에는 중복이 있으면 안된다.
  - → 수열 크기는 M이어야 한다.
- 재귀함수를 설계 해보자.
  - → 각 수를 넣으려고 할 때, 이미 수열 내에 있으면 스킵.
  - → 수열 크기가 M이 되면 리턴.

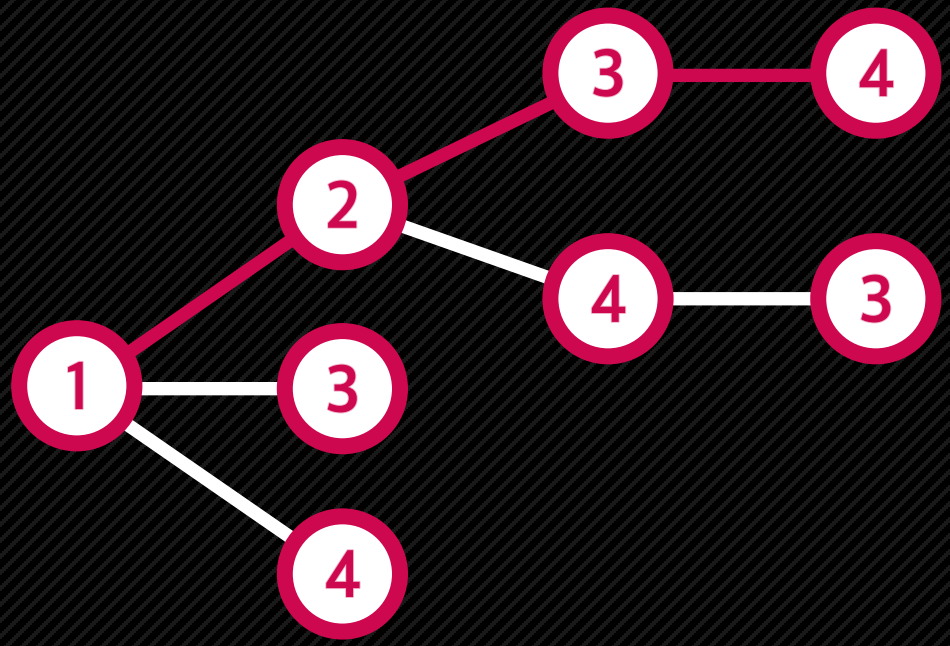
재귀로 문제 풀때…

Silver 3 - N과 M (1)



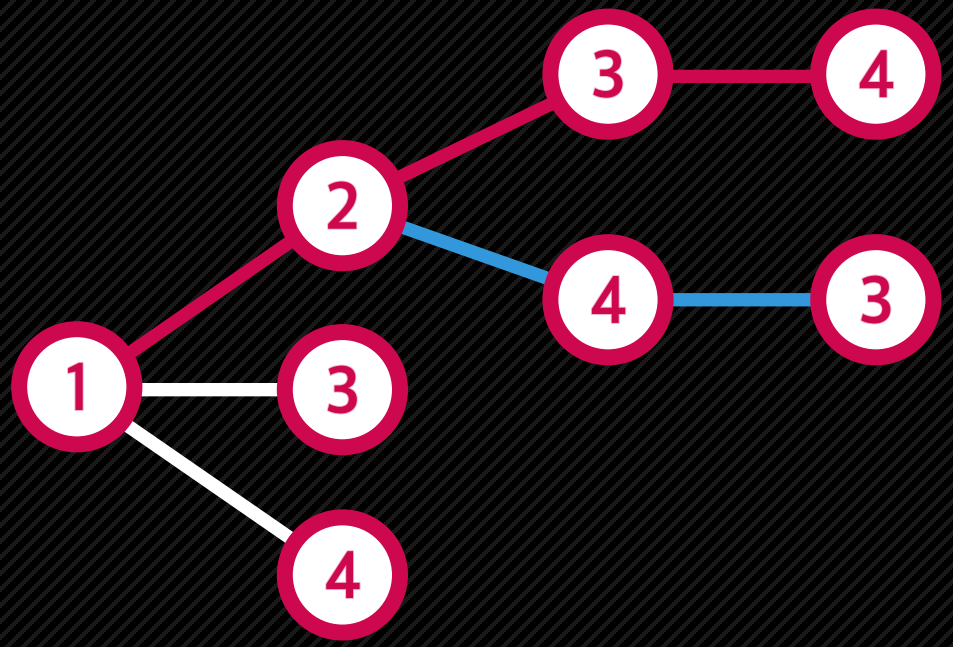
재귀로 문제 풀때…

Silver 3 - N과 M (1)



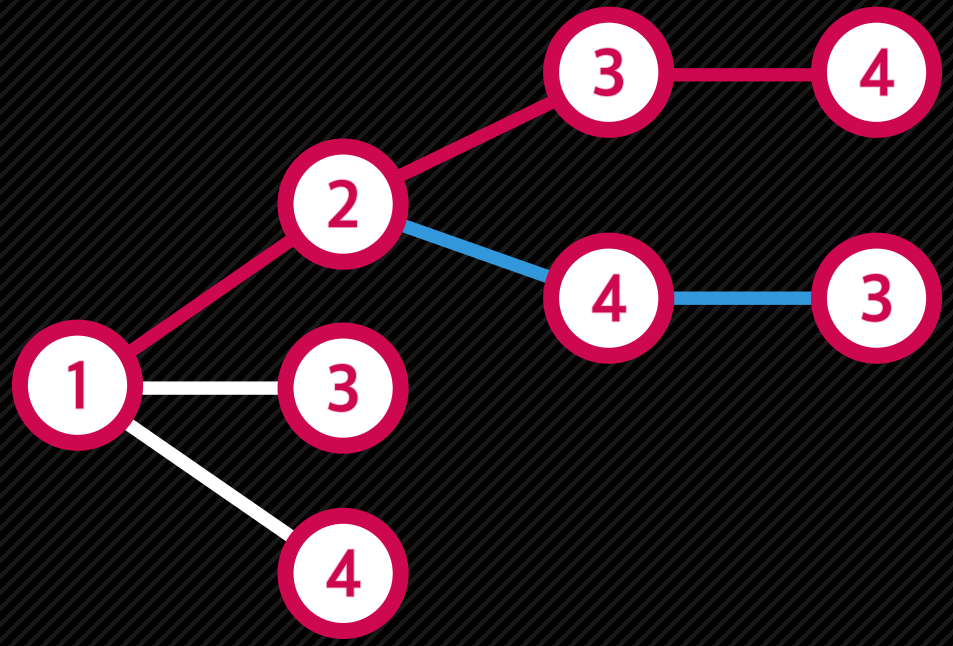
재귀로 문제 풀때…

Silver 3 - N과 M (1)



재귀로 문제 풀때…

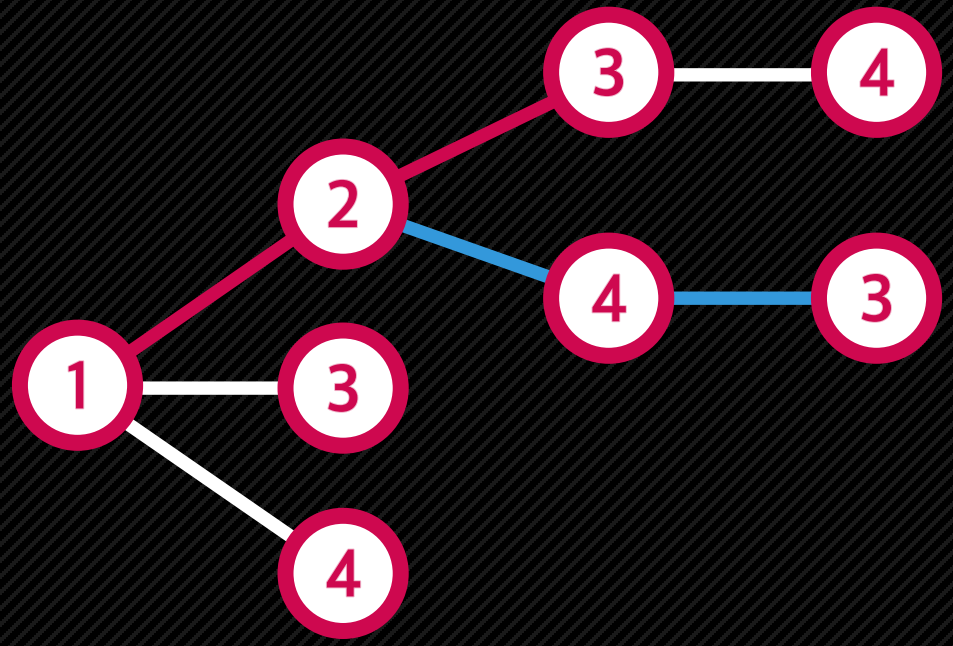
Silver 3 - N과 M (1)




```
function nm(list)
    if size of list == M
        // Do Something
        return
    for i = 1 ... N
        if i not in list
            add i to list
            nm(list)
            remove i to list
```

재귀로 문제 풀때…

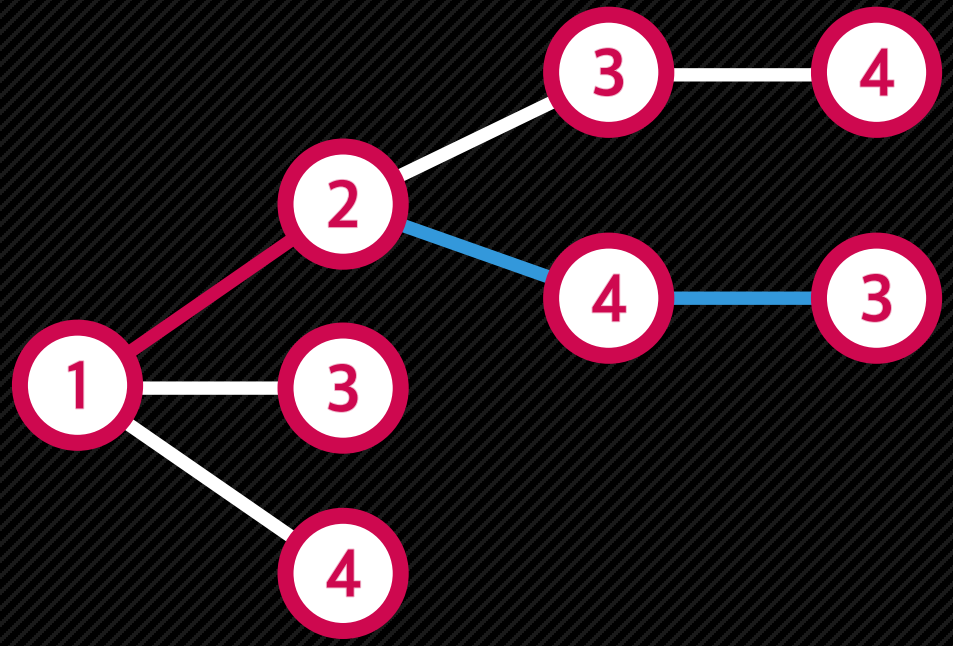
Silver 3 - N과 M (1)




```
function nm(list)
    if size of list == M
        // Do Something
        return
    for i = 1 ... N
        if i not in list
            add i to list
            nm(list)
            remove i to list
```

재귀로 문제 풀때…

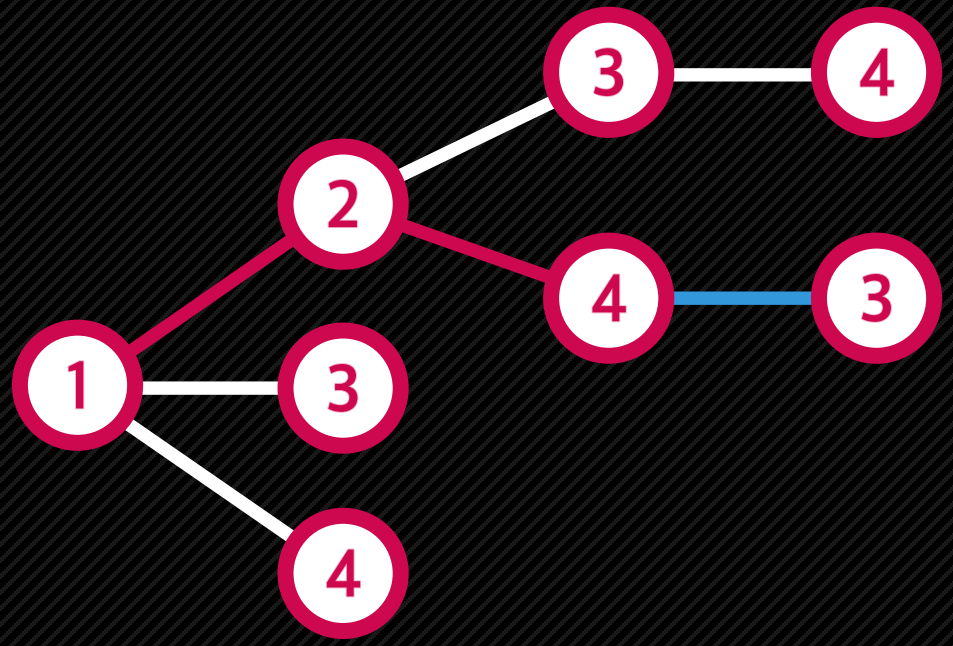
Silver 3 - N과 M (1)




```
function nm(list)
    if size of list == M
        // Do Something
        return
    for i = 1 ... N
        if i not in list
            add i to list
            nm(list)
            remove i to list
```

재귀로 문제 풀때…

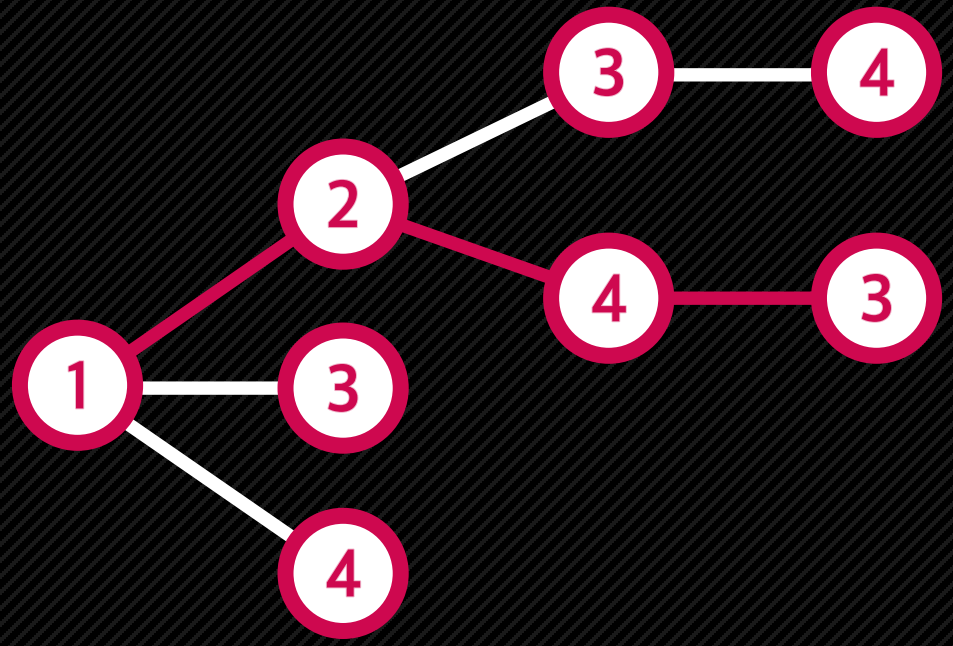
Silver 3 - N과 M (1)




```
function nm(list)
    if size of list == M
        // Do Something
        return
    for i = 1 ... N
        if i not in list
            add i to list
            nm(list)
            remove i to list
```

재귀로 문제 풀때…

Silver 3 - N과 M (1)




```
function nm(list)
    if size of list == M
        // Do Something
        return
    for i = 1 ... N
        if i not in list
            add i to list
            nm(list)
            remove i to list
```

문제를 두 번 봅시다…

Silver 3 - N과 M (2)

요약

- 자연수 N과 M이 주어졌을 때, 1부터 N까지 자연수 중 **중복없이 M개를 고른 수열**을 **모두** 구하는 프로그램을 작성하시오.
- 단, 수열은 **오름차순**이다.

제약조건

- N과 M의 범위는 **1 <= M <= N <= 8** 이다.

1과 무슨 차이가 있을까?

접근

- 어떤 제약 조건이 있을까?
  - → 수열 내부에는 **중복이 있으면 안된다.**
  - → 수열 크기는 **M**이어야 한다.
- 재귀함수를 설계 해보자.
  - → 각 수를 넣으려고 할 때, 이미 **수열 내에 있으면 스킵.**
  - → 수열 크기가 **M이 되면 리턴.**

단, 수열은 **오름차순**이다. →

?

1과 무슨 차이가 있을까?

접근

- 어떤 제약 조건이 있을까?
  - → 수열 내부에는 **중복이 있으면 안된다.**
  - → 수열 크기는 **M**이어야 한다.
- 재귀함수를 설계 해보자.
  - → 각 수를 넣으려고 할 때, 이미 **수열 내에 있으면 스킵.**
  - → 수열 크기가 **M이 되면 리턴.**

단, 수열은 **오름차순**이다.

- 오름차순은 어떻게 걸러낼까?
  - → 마지막 원소를 확인하여 **(해당 값 + 1) 부터 반복문 수행**

복잡해 보여도, 차근차근!

Gold 5 (Level 3) - N-Queen

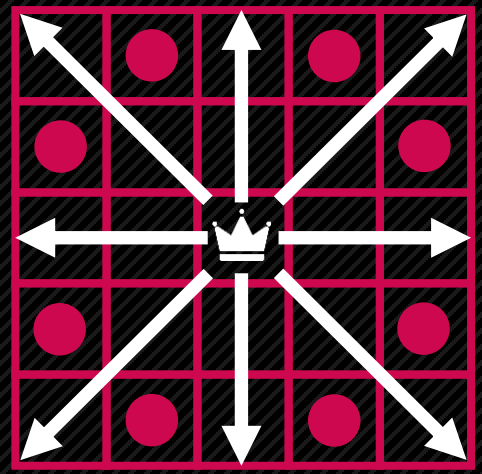
요약

- N * N 인 체스판 위에 퀸 N개가 서로 공격할 수 없게 놓는 경우의 수를 구하는 프로그램을 작성하시오.

제약조건

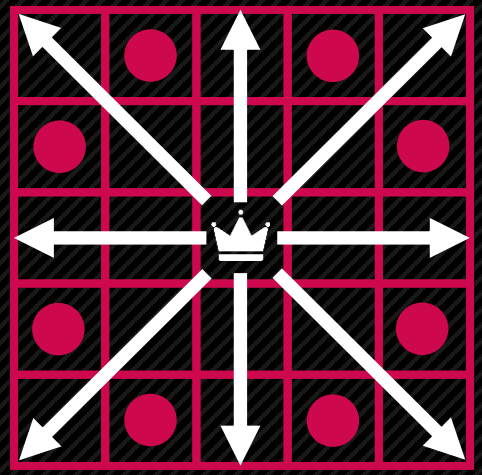
- N의 범위는 1 < N <= 15 이다.

문제를 분석해보자.



- 퀸이 놓이면, 가로, 세로, 대각선에 위치한 곳엔 말을 둘 수 없다.

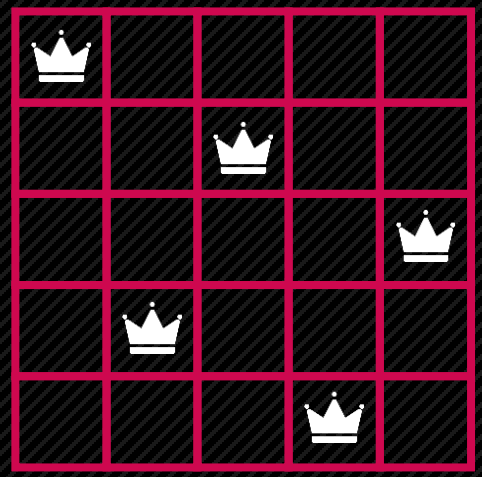
문제를 분석해보자.



- 퀸이 놓이면, 가로, 세로, 대각선에 위치한 곳엔 말을 둘 수 없다.

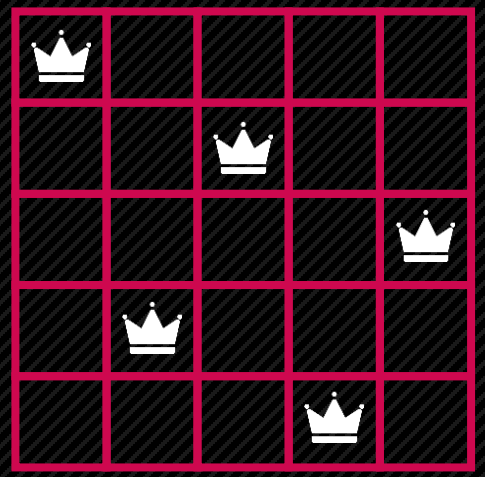
- 완전탐색은 어떨까?
  - → ${}_{255}C_{15} > 10^{30}$. 택도 없다!
- 가지치기 할 방법은 없을까?
  - → 어차피 각각 가로와 세로에는 1개만 들어감.

문제를 분석해보자.



- 어차피 각각 가로와 세로에는 1개만 들어감.

문제를 분석해보자.



- 어차피 각각 가로와 세로에는 1개만 들어감.

Silver 3 - N과 M (1)

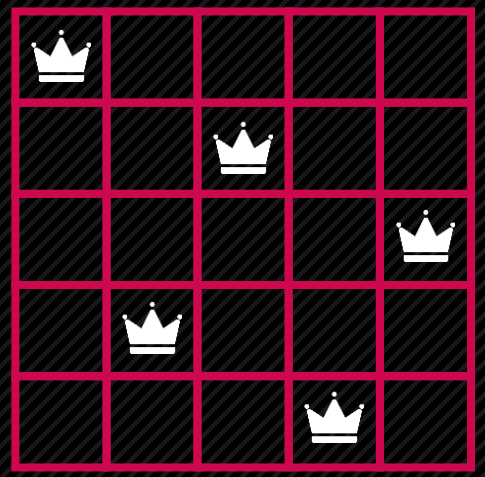
요약

- 자연수 N과 M이 주어졌을 때, 1부터 N까지 자연수 중 중복없이 M개를 고른 수열을 모두 구하는 프로그램을 작성하시오.

제약조건

- N과 M의 범위는 1 <= M <= N <= 8 이다.

문제를 분석해보자.



- 어차피 각각 가로와 세로에는 1개만 들어감.

Silver 3 - N과 M (1)

요약

- 자연수 N과 M이 주어졌을 때, 1부터 N까지 자연수 중 중복없이 M개를 고른 수열을 모두 구하는 프로그램을 작성하시오.

제약조건

- N과 M의 범위는 1 <= M <= N <= 8 이다.

→ 결국, 시도할 수 있는 경우는 N!
그럼에도 15! > 10^{11}…

→ 하지만, 대각선 가지치기를 하면 충분히 걸러 낼 수 있을 것!

문제를 분석해보자.

Silver 3 - N과 M (1)

요약

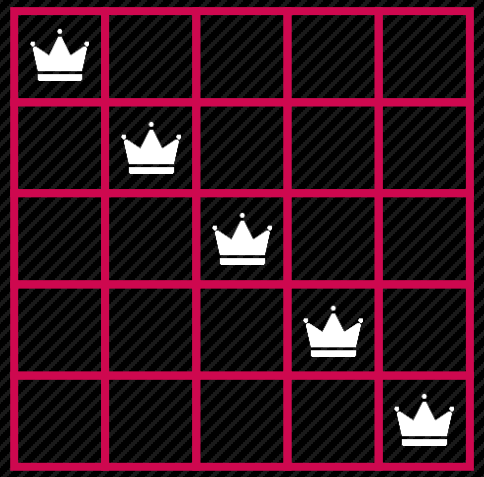
- 자연수 N과 M이 주어졌을 때, 1부터 N까지 자연수 중 중복없이 M개를 고른 수열을 모두 구하는 프로그램을 작성하시오.

제약조건

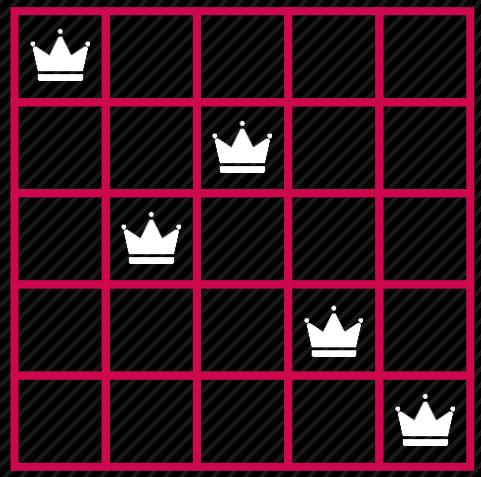
- N과 M의 범위는 1 <= M <= N <= 8 이다.

1, 2, 3, 4, 5, 6, 7, 8
1, 3, 2, 4, 5, 6, 7, 8
...

문제를 분석해보자.

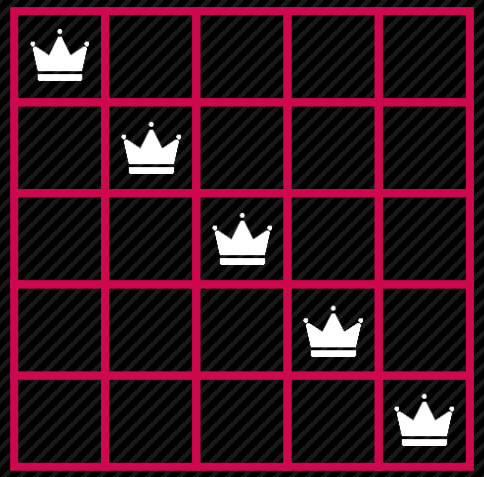


1, 2, 3, 4, 5

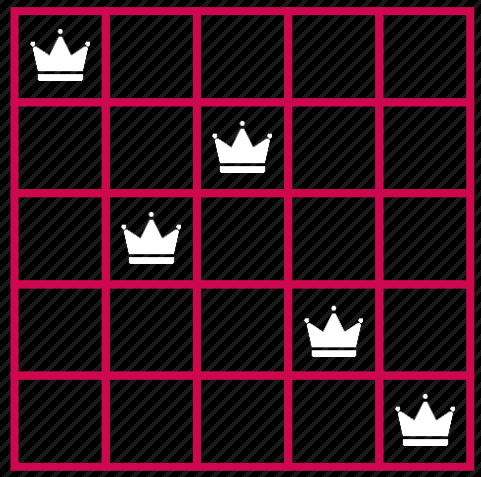


1, 3, 2, 4, 5

문제를 분석해보자.



```
if(abs(arr[i] - arr[j]) == j - i)
```



1, 2, 3, 4, 5

1, 3, 2, 4, 5

백트래킹은 중요하니,
한 문제 더!

Gold 5 - 비밀번호 (#13908)

요약

- N자리의 숫자로 된 비밀번호가 있고, 해당 비밀번호를 풀기 위해 모든 경우를 시도하려고 한다.
- 선견지명을 발휘해 비밀번호에 들어있는 일부 숫자를 알 수 있다.
- 알게된 단서를 기반으로, 최대 몇 번 시도해야 하는지 구하는 프로그램을 작성하시오.

제약조건

- 비밀번호의 길이는 1 <= N <= 7 이다.
- 선견지명으로 알게 된 숫자의 개수는 0 <= M <= N 이다.

익숙한 문제죠?

Silver 3 - 모든 순열

요약

- N이 주어졌을 때, 1부터 N까지의 수로 이루어진 순열을 사전순으로 출력하는 프로그램을 작성하시오.

제약조건

- N의 범위는 1 <= N <= 8 이다.

Itertools!

```
import itertools

iter = itertools.combinations('12345678', 5)
for data in iter:
    print(' '.join(map(str, data)))
```

- Iterable한 데이터들의 조합/순열/합성곱을 쉽게 구해주는 라이브러리.
- 모든 경우를 고려해야 하는 문제에서 상당히 유용하게 사용됨!

Itertools!

```python
import itertools
```

Itertools 를 미리 import 하고 사용.

```python
iter = itertools.combinations('1234', 2)
# 12 13 14 23 24 34
for data in iter:
    print(' '.join(map(str, data)))
```

combinations: 주어진 데이터 중 M개를 뽑아 가능한 모든 조합 리턴

```python
iter = itertools.permutations('1234', 2)
# 12 13 14 21 23 24 31 32 34 41 42 43
for data in iter:
    print(*data)
```

permutations: 주어진 데이터 중 M개를 뽑아 가능한 모든 순열 리턴

```python
iter = itertools.combinations_with_replacement('1234', 2)
# 11 12 13 14 22 23 24 33 34 44
for data in iter:
    x, y = data
    print(x, y)
```

combinations_with_replacement:
주어진 데이터 중 중복을 포함하여 M개를 뽑아 가능한 모든 조합 리턴

```python
iter = itertools.product('1234', repeat=2)
# 11 12 13 14 21 22 23 24 31 32 33 34 41 42 43 44
for data in iter:
    print('{} {}'.format(*data))
```

combinations_with_replacement:
주어진 데이터 목록의 Catesian Product 리턴

Itertools를 활용한 분할

Silver 3 - 스타트와 링크 (삼성 SW 역량테스트 기출) (#14888)

요약

- 짝수 인원 N명이 있을 때, N/2 명으로 구성된 두 팀을 만들려고 한다.
- I번째 사람과 J번째 사람이 같은 팀을 했을 시, 얻을 수 있는 능력치는 $S_{IJ} + S_{JI}$ (S_{IJ} 와 S_{JI} 는 다를 수 있음.) 이다.
- 게임의 밸런스를 위해, 주어진 능력치를 기반으로 두 팀의 능력치의 합을 최소로 하려고 한다.
- 차이의 최솟값을 출력하는 프로그램을 작성하시오.

제약조건

- N의 범위는 1 <= N <= 20 이다.
- S_{II} 는 항상 0이며, S_{IJ} 의 범위는 1 <= S_{IJ} <= 100 이다.

Itertools를 활용한 분할

Silver 3 - 스타트와 링크 (삼성 SW 역량테스트 기출) (#14888)

Itertools를 활용한 분할

Silver 3 - 스타트와 링크 (삼성 SW 역량테스트 기출) (#14888)

- combinations: 주어진 데이터 중 M개를 뽑아 가능한 모든 조합 리턴

Itertools를 활용한 분할

Silver 3 - 스타트와 링크 (삼성 SW 역량테스트 기출) (#14888)

-	2	3	5	1	4
3	-	7	5	2	6
1	2	-	1	2	3
3	4	8	-	4	5
4	4	5	2	-	2
5	6	8	1	3	-

다른 방법?

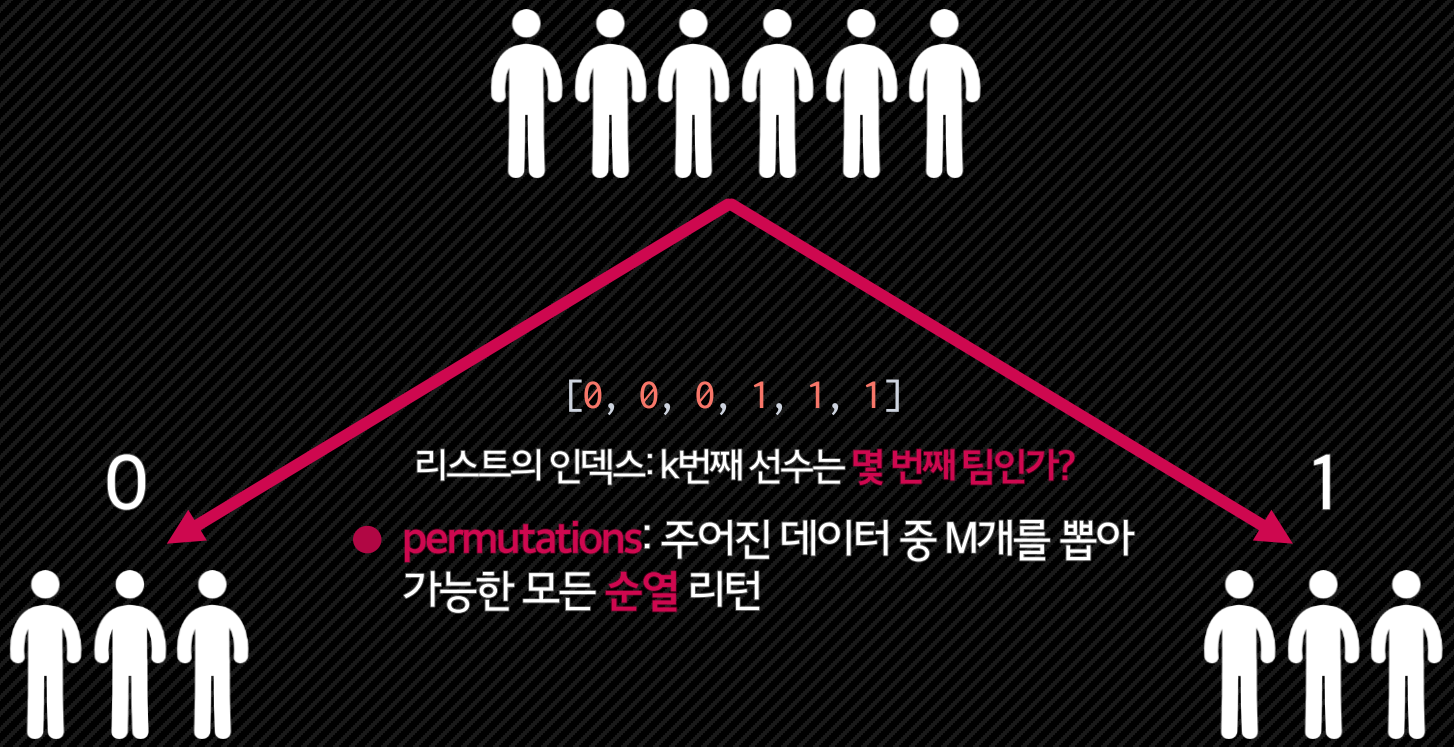
Silver 3 - 스타트와 링크 (삼성 SW 역량테스트 기출) (#14888)

다른 방법?

Silver 3 - 스타트와 링크 (삼성 SW 역량테스트 기출) (#14888)

다른 방법?

Silver 3 - 스타트와 링크 (삼성 SW 역량테스트 기출) (#14888)



- permutations: 주어진 데이터 중 M개를 뽑아 가능한 모든 순열 리턴

심화된 문제!

Silver 1 - 연산자 끼워넣기 (삼성 SW 역량테스트 기출) (#14889)

요약

- N개의 수들이 주어지고, 수와 수 사이에 끼워 넣을 수 있는 N - 1 개의 연산자가 주어진다.
- 수와 연산자가 주어졌을 때, 만들 수 있는 식의 결과가 최대인 것과 최소인 것을 구하는 프로그램을 작성하시오.

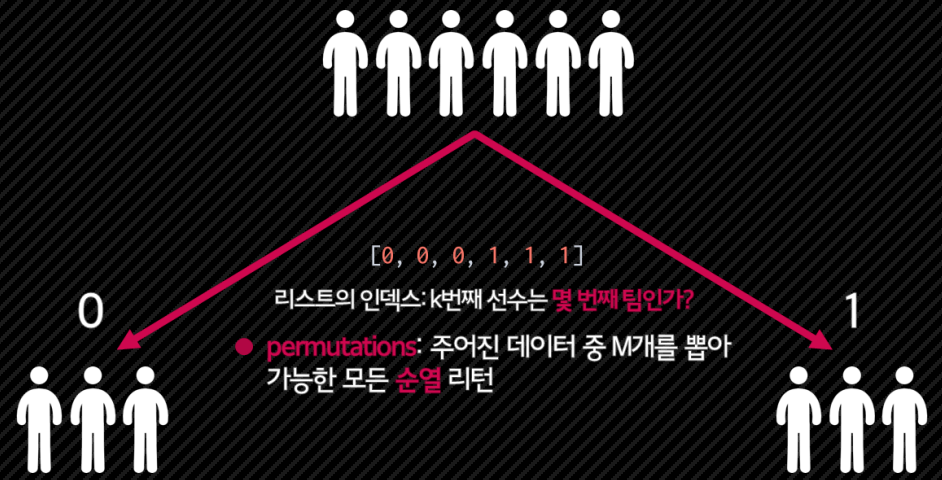
제약조건

- N의 범위는 $2 <= N <= 11$ 이다.
- 각각의 수의 범위는 $1 <= A_i <= 100$ 이다.

심화된 문제!

Silver 1 - 연산자 끼워넣기 (삼성 SW 역량테스트 기출) (#14889)

- combinations: 주어진 데이터 중 M개를 뽑아 가능한 모든 조합 리턴

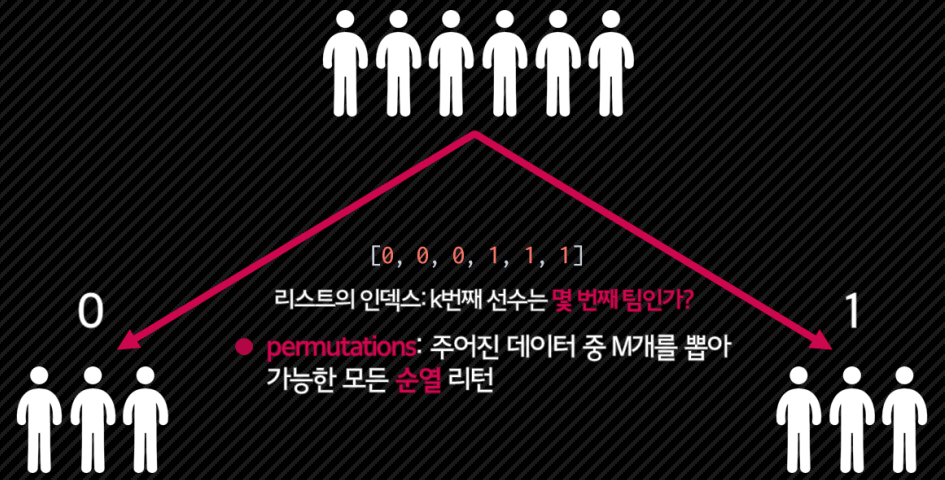


- permutations: 주어진 데이터 중 M개를 뽑아 가능한 모든 순열 리턴

두번째 방법을 설명한 이유!

Silver 1 - 연산자 끼워넣기 (삼성 SW 역량테스트 기출) (#14889)

- combinations: 주어진 데이터 중 M개를 뽑아 가능한 모든 조합 리턴



- permutations: 주어진 데이터 중 M개를 뽑아 가능한 모든 순열 리턴

“Any Question?”