Chapter. 02

알고리즘

어떻게든 푼다. 완전 탐색 (Brute Force)

FAST CAMPUS
ONLINE
알고리즘 공채 대비반I

강사. 류호석

Life Changing Education

Chapter. 02

알고리즘
완전 탐색(Brute Force)

FAST CAMPUS
ONLINE
류호석 강사.

I 정답은 무조건 구하는 치트키

<완전 탐색>

문제를 해결하기 위해 확인해야 하는 모든 경우를 전부 탐색하는 방법

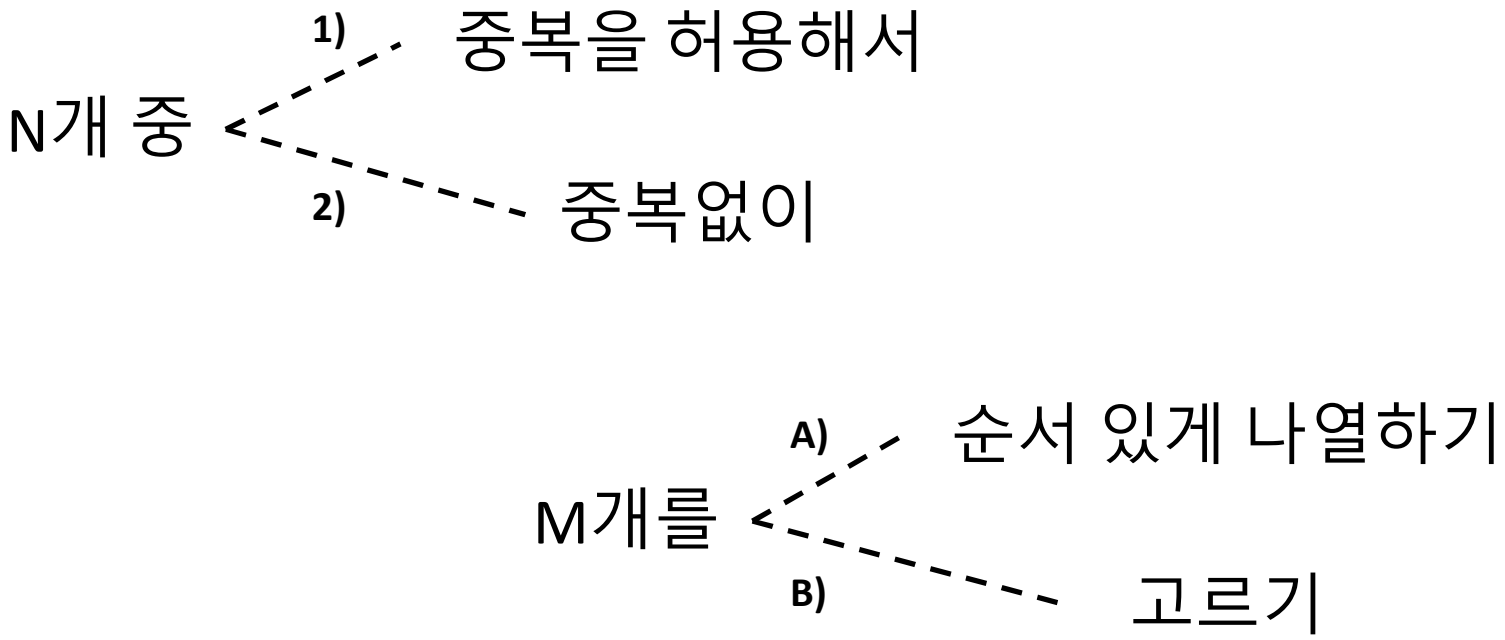
그 중에서도 백 트래킹(Back-Tracking)을 통해야 하는 상황을 해결하기!

※모든 코테 문제에서 기본적으로 접근해 봐야 한다. 많은 연습이 필요!

 : 부분점수를 얻기 좋다.

 : 전부 탐색하기에 시간 복잡도가 일반적으로 높다.

I 코테에 나오는 완전 탐색 종류

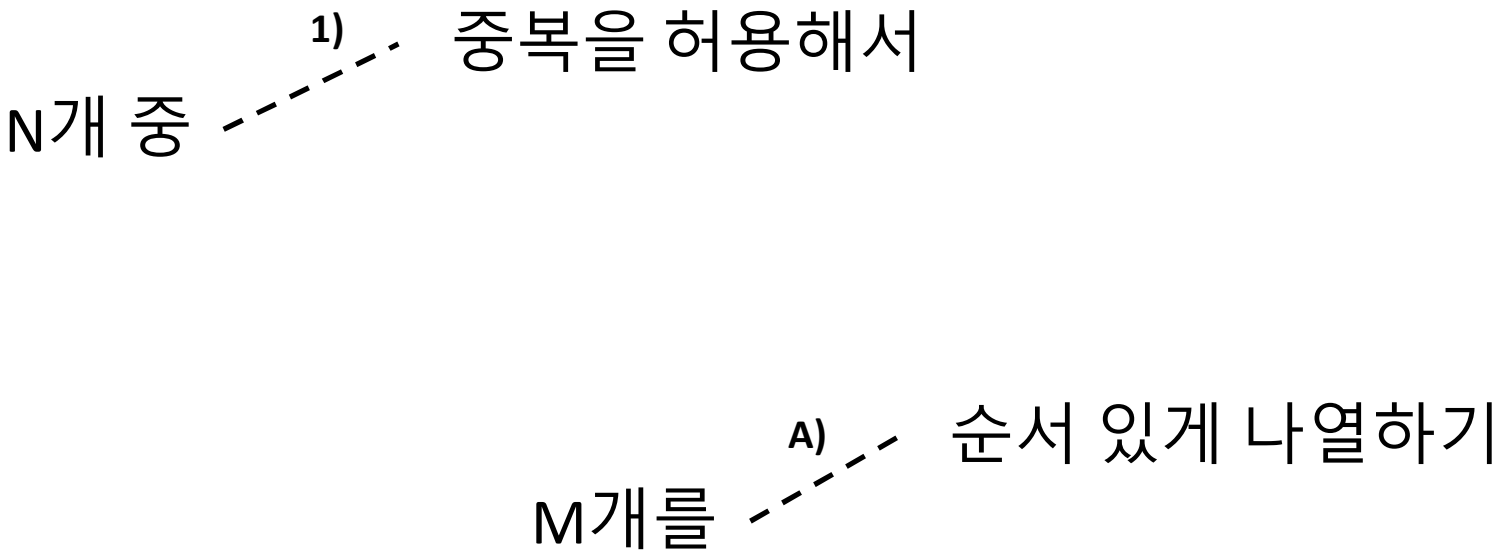


I 완전 탐색은 함수 정의가 50%


```
// Recurrence Function (재귀 함수)
// 만약 M 개를 전부 고름   => 조건에 맞는 탐색을 한 가지 성공한 것!
// 아직 M 개를 고르지 않음 => k 번째부터 M번째 원소를 조건에 맞게 고르는 모든 방법을 시도한다.
static void rec_func(int k) {}
```

```
public static void main(String[] args) {
    input();
    // 1 번째 원소부터 M 번째 원소를 조건에 맞게 고르는 모든 방법을 탐색해줘
    rec_func( k: 1);
    System.out.println(sb.toString());
}
```

I1 + A 버전



I BOJ 15651 – N과 M (3)

난이도: 2

N=4, M=3 ---- ---- ----

I 시간, 공간 복잡도 계산하기

N=4, M=3

$$\underbrace{4}_{\text{----}} \times \underbrace{4}_{\text{----}} \times \underbrace{4}_{\text{----}} = 4\text{^}3$$

시간: $O(N^M) \Rightarrow 7^7 \cong 82$만

공간: $O(M)$

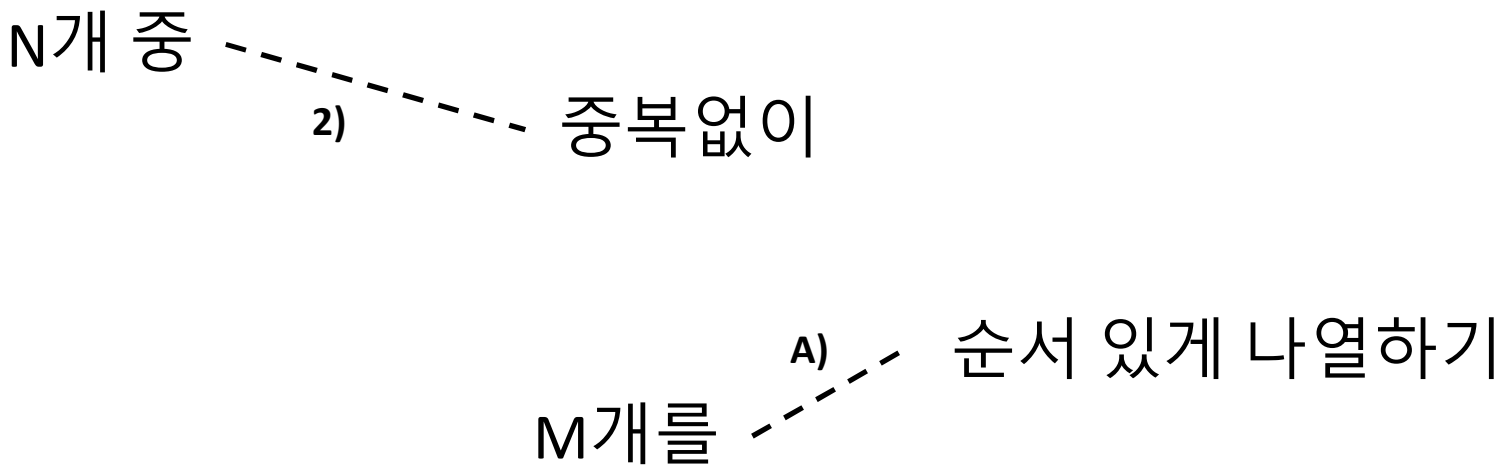
I 구현 스케치

```
static int N, M;
static int[] selected;
// Recurrence Function (재귀 함수)
// 만약 M 개를 전부 고름   => 조건에 맞는 탐색을 한 가지 성공한 것!
// 아직 M 개를 고르지 않음 => k 번째부터 M번째 원소를 조건에 맞게 고르는 모든 방법을 시도한다.
static void rec_func(int k) {
    if (k == M + 1) { // 다 골랐다!
        // selected[1...M] 배열이 새롭게 탐색된 결과
    } else {
        // 1~N 까지를 k 번 원소로 한 번씩 정하고,
        // 매번 k+1 번부터 M 번 원소로 재귀호출 해주기
    }
}
```

I 구현

```
14      static int N, M;
15      static int[] selected;
16      // Recurrence Function (재귀 함수)
17      // 만약 M 개를 전부 고름   => 조건에 맞는 탐색을 한 가지 성공한 것!
18      // 아직 M 개를 고르지 않음 => k 번째부터 M번째 원소를 조건에 맞게 고르는 모든 방법을 시도한다.
19      static void rec_func(int k) {
20          if (k == M + 1) { // 1 ~ M 번째를 전부 다 골랐다!
21              // selected[1...M] 배열이 새롭게 탐색된 결과
22              for (int i = 1; i <= M; i++) sb.append(selected[i]).append(' ');
23              sb.append('\n');
24          } else {
25              for (int cand = 1; cand <= N; cand++) {
26                  // k 번째에 cand 가 올 수 있으면
27                  selected[k] = cand;
28
29                  // k+1 번부터 M 번까지 잘 채워주는 함수를 호출해준다.
30                  rec_func( k: k + 1);
31                  selected[k] = 0;
32              }
33          }
34      }
```

I2 + A 버전



I BOJ 15649 – N과 M (1)

난이도: 2

N=4, M=3 ---- ---- ----

I 시간, 공간 복잡도 계산하기

N=4, M=3

$$\underbrace{4}_{\text{----}} \times \underbrace{3}_{\text{----}} \times \underbrace{2}_{\text{----}} = 4!$$

시간: $O({}_{M}^{N}\mathrm{P}) = O\left(\frac{N!}{(N-M)!}\right) \Rightarrow \frac{8!}{0!} = 40{,}320$

공간: $O(M)$

I 구현

```
19      static void rec_func(int k) {
20          if (k == M + 1) { // 1 ~ M 번째를 전부 다 골랐다!
21              // selected[1...M] 배열이 새롭게 탐색된 결과
22              for (int i = 1; i <= M; i++) sb.append(selected[i]).append(' ');
23              sb.append('\n');
24          } else {
25              for (int cand = 1; cand <= N; cand++) {
26                  boolean isUsed = false;
27                  for (int i=1;i<k;i++)
28                      if (cand == selected[i])
29                          isUsed = true;
30                  // k 번째에 cand 가 올 수 있으면
31                  if (!isUsed) {
32                      selected[k] = cand;
33                      rec_func( k: k + 1);
34                      selected[k] = 0;
35                  }
36              }
37          }
38      }
```

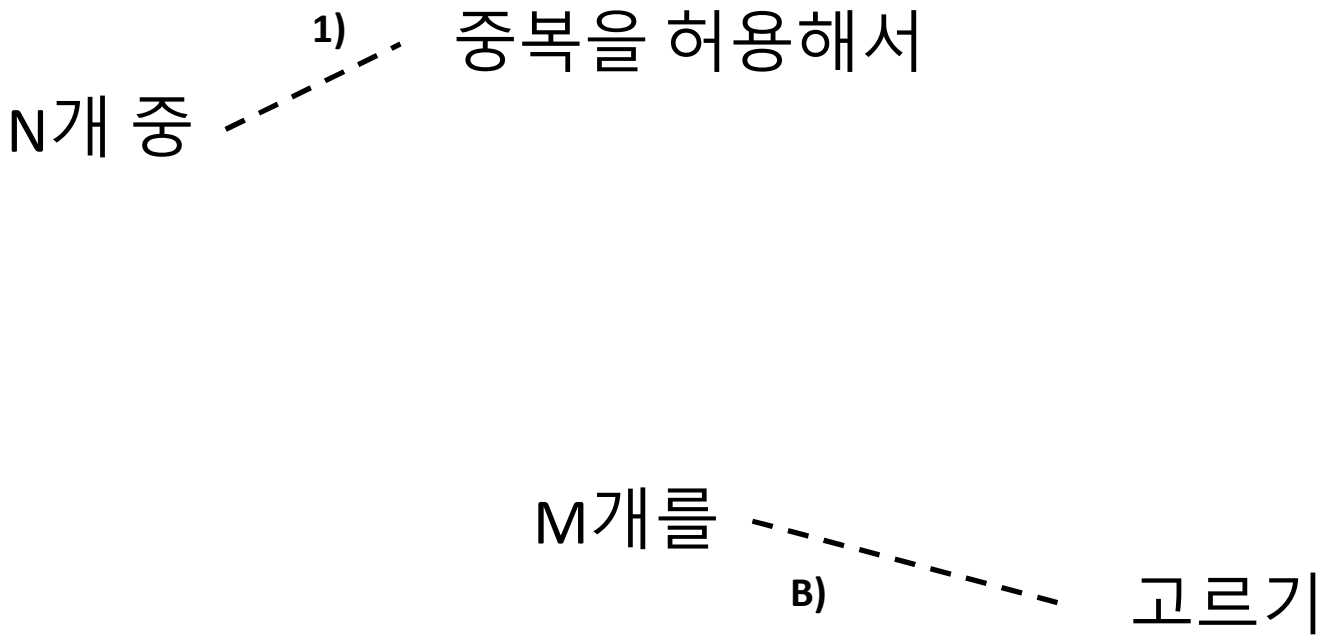
I 구현-심화

```java
static int N, M;
static int[] selected, used;
// Recurrence Function (재귀 함수)
// 만약 M 개를 전부 고름   => 조건에 맞는 탐색을 한 가지 성공한 것!
// 아직 M 개를 고르지 않음 => k 번째부터 M번째 원소를 조건에 맞게 고르는 모든 방법을 시도한다.
static void rec_func(int k) {
    if (k == M + 1) { // 1 ~ M 번째를 전부 다 골랐다!
        // selected[1...M] 배열이 새롭게 탐색된 결과
        for (int i = 1; i <= M; i++) sb.append(selected[i]).append(' ');
        sb.append('\n');
    } else {
        for (int cand = 1; cand <= N; cand++) {
            if (used[cand] == 1) continue;
            // k 번째에 cand 가 올 수 있으면
            selected[k] = cand;   used[cand] = 1;

            rec_func( k: k + 1);

            selected[k] = 0;      used[cand] = 0;
        }
    }
}
```

I1 + B 버전



I BOJ 15652 – N과 M (4)

난이도: 2

N=4, M=3 ---- ---- ----

I 시간, 공간 복잡도 계산하기

N=4, M=3

$\overbrace{4}^{\text{----}} \times \overbrace{4}^{\text{----}} \times \overbrace{4}^{\text{----}} = 4\text{^}3$ 보단 작다.

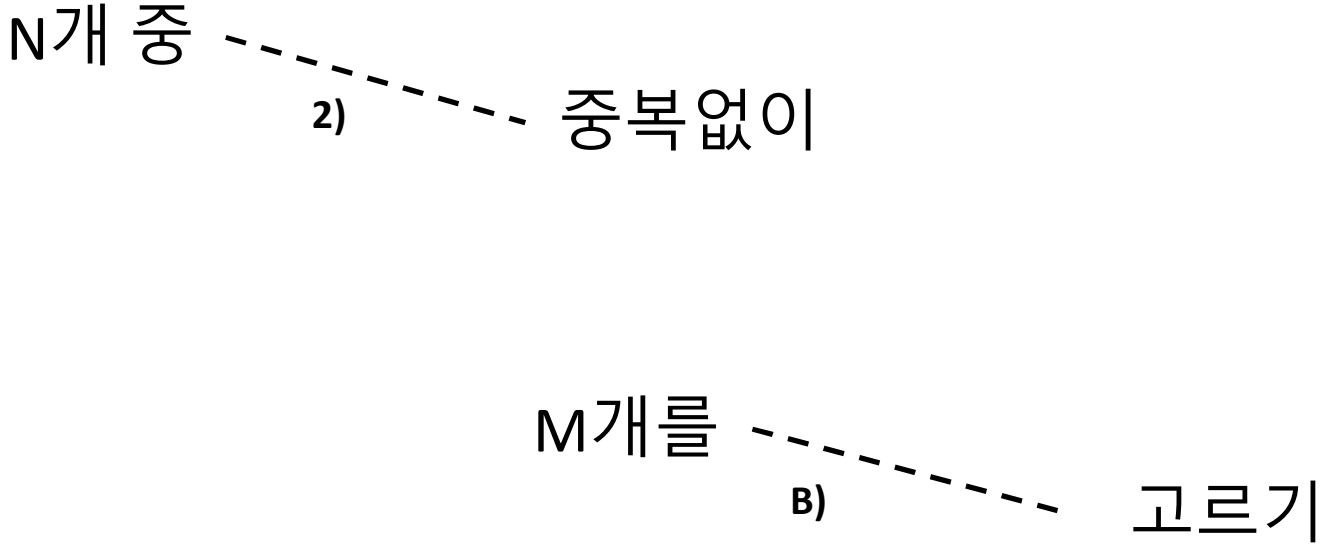
시간: $O(N^M) \Rightarrow 8^8 \cong 1677$만 보단 작다.

공간: $O(M)$

I 구현


```
19      static void rec_func(int k) {
20          if (k == M + 1) {
21              for (int i = 1; i <= M; i++) sb.append(selected[i]).append(' ');
22              sb.append('\n');
23          } else {
24              int start = selected[k-1];
25              if (start == 0) start = 1;
26              for (int cand = start; cand <= N; cand++) {
27                  // k 번째에 cand 가 올 수 있으면
28                  selected[k] = cand;
29                  rec_func( k: k + 1);
30                  selected[k] = 0;
31              }
32          }
33      }
```

I2 + B 버전



I BOJ 15650 – N과 M (2)

난이도: 2

N=4, M=3 ---- ---- ----

I 시간, 공간 복잡도 계산하기

$$\text{시간: } O({}_{M}^{N}C) = O\left(\binom{N}{M}\right) = O\left(\frac{N!}{M!(N-M)!}\right) \Rightarrow \frac{8!}{4!4!} = 70$$

$$\text{공간: } O(M)$$

I 구현


```
19      static void rec_func(int k) {
20          if (k == M + 1) {
21              for (int i = 1; i <= M; i++) sb.append(selected[i]).append(' ');
22              sb.append('\n');
23          } else {
24              for (int cand = selected[k-1] + 1; cand <= N; cand++) {
25                  selected[k] = cand;
26                  rec_func( k: k + 1);
27                  selected[k] = 0;
28              }
29          }
30      }
```

I 총 정리

중복	순서	시간 복잡도	공간 복잡도
YES	YES	$O(N^M)$	$O(M)$
NO	YES	$O({}_M^N\mathrm{P}) = O\left(\frac{N!}{(N-M)!}\right)$	$O(M)$
YES	NO	$O(N^M)$ 보단 작음	$O(M)$
NO	NO	$O({}_M^N\mathrm{C}) = O\left(\frac{N!}{M!\,(N-M)!}\right)$	$O(M)$

완전 탐색 문제를 접근할 때는,

- 고를 수 있는 값의 종류 파악하기
- **중복**을 허용하는 지
- **순서**가 중요한 지