JavaScript 백트래킹 알고리즘

백트래킹 문제 풀이

백트래킹 문제 풀이 | 코딩 테스트에서 자주 등장하는 백트래킹 알고리즘 이해하기

강사 나동빈

JavaScript
백트래킹 알고리즘

백트래킹 문제 풀이

N과 M (2): https://www.acmicpc.net/problem/15650

JavaScript 백트래킹
백트래킹 문제 풀이

혼자 힘으로 풀어보기

문제 제목: N과 M (2)

문제 난이도: ★★☆☆☆

문제 유형: 백트래킹, 경우의 수, 완전 탐색

추천 풀이 시간: 40분

N과 M (2): https://www.acmicpc.net/problem/15650

JavaScript 백트래킹
백트래킹 문제 풀이

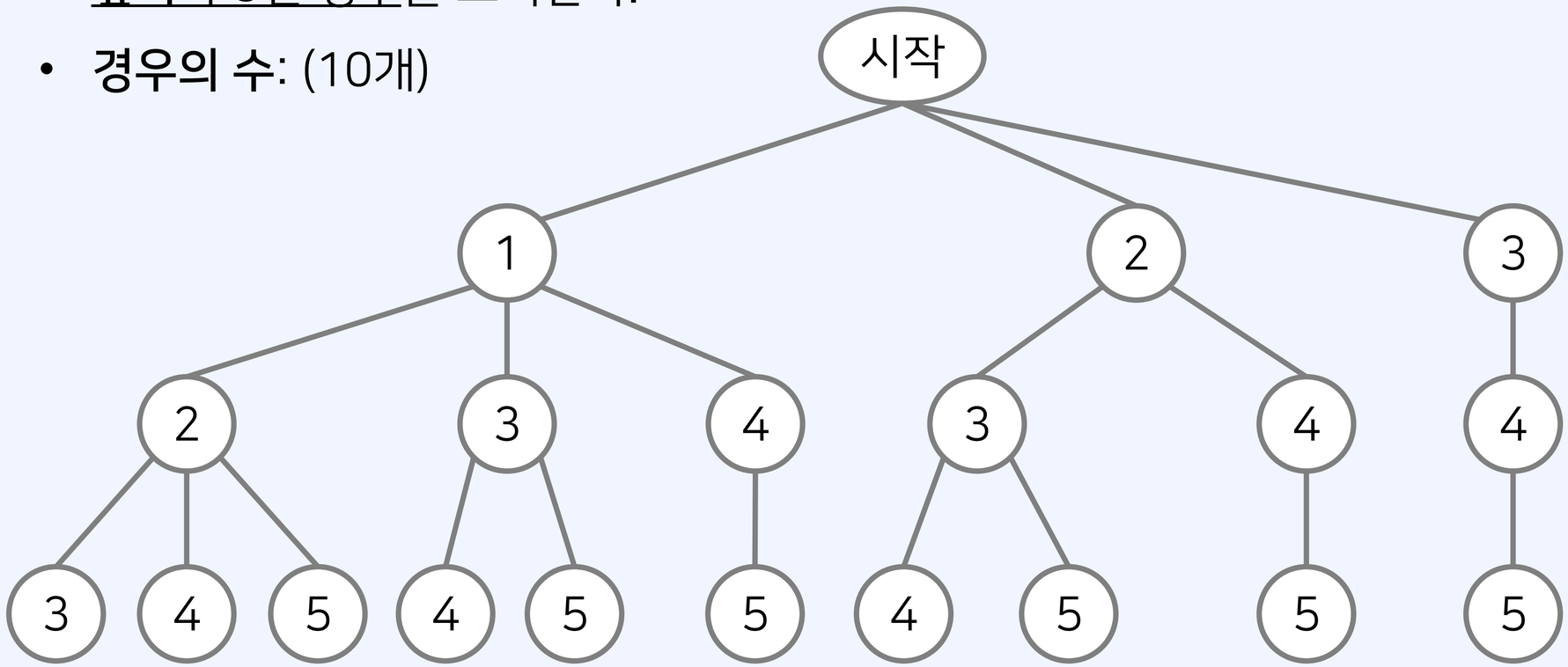
문제 해결 아이디어

- 1부터 N까지 자연수 중에서 중복 없이 M개를 고른 **모든 조합**을 계산한다.
- 오름차순이라는 점(순서가 하나만 있다는 점)에서 조합으로 이해할 수 있다.

N과 M (2): https://www.acmicpc.net/problem/15650

문제 해결 아이디어

- 5개의 수 [1, 2, 3, 4, 5]에서 3개를 고르는 **모든 조합**을 고려해 보자.
- **깊이**가 3인 경우를 고려한다.
- **경우의 수**: (10개)



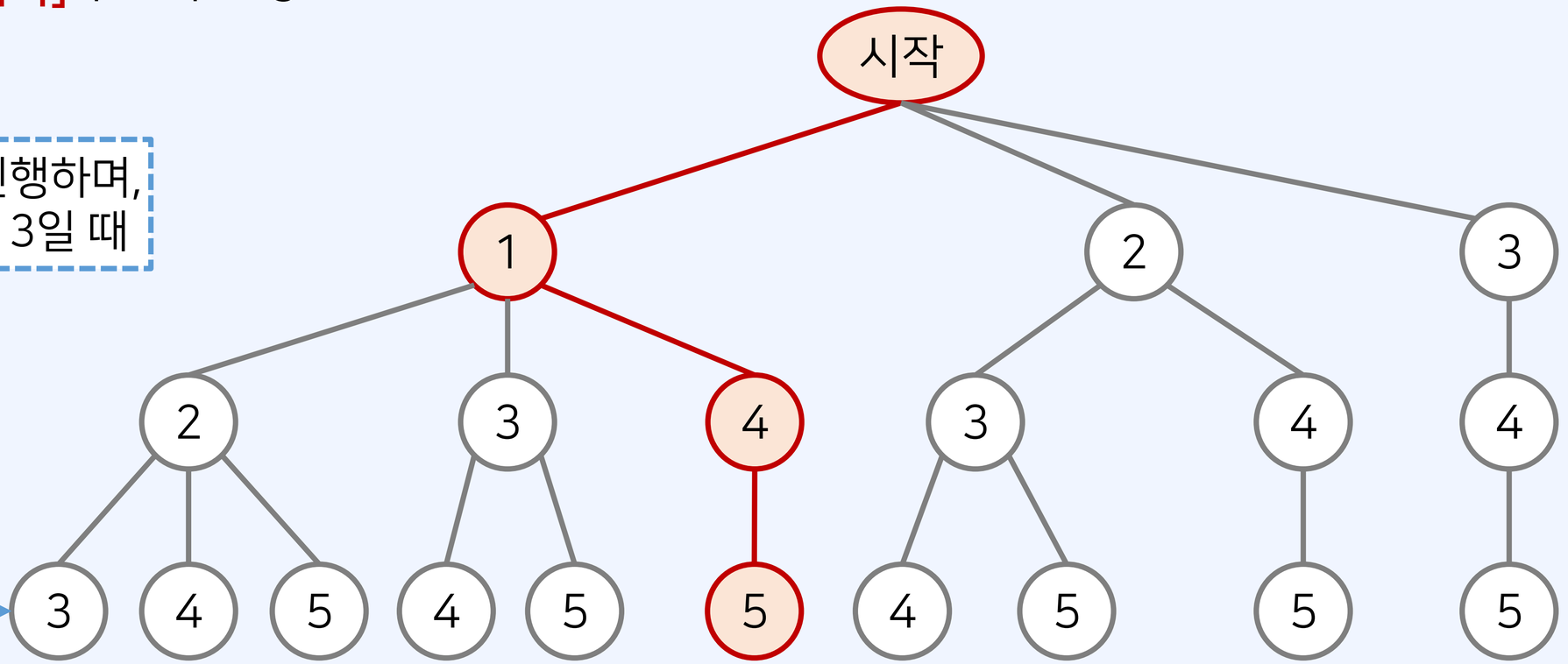
N과 M (2): https://www.acmicpc.net/problem/15650

문제 해결 아이디어

JavaScript 백트래킹
백트래킹 문제 풀이

- 5개의 수 [1, 2, 3, 4, 5]에서 3개를 고르는 **모든 조합**을 고려해 보자.
- **[예시]** 1 → 4 → 5

완전 탐색을 진행하며,
깊이(depth)가 3일 때



N과 M (2): https://www.acmicpc.net/problem/15650

JavaScript
백트래킹
백트래킹
문제 풀이

문제 해결 아이디어

- 모든 조합의 수를 고려하기 위해 **재귀 함수(백트래킹)**를 사용할 수 있다.
- 하나의 조합을 트리(tree)에서 리프 노드까지의 경로로 생각할 수 있다.

→ 이때, M개의 원소를 뽑는 조합을 고려하는 것이므로, **깊이(depth)**는 M과 같다.

- 원소를 **중복하여 선택하지 않으므로**, 방문 처리(visited) 배열을 사용한다.

→ 한 번 선택된 원소는 다음 재귀 함수에서 다시 선택되지 않는다.

[조합] 재귀 함수를 호출할 때마다(자식 노드로 갈수록) **선택되는 값이 커지도록** 하는 것이 핵심이다.

→ 순열 소스 코드와 다른 점은 $start$ **변수가 사용**된다는 점 뿐이다.

N과 M (2): https://www.acmicpc.net/problem/15650

정답 코드 예시

JavaScript
백트래킹
백트래킹
문제 풀이

```
let fs = require('fs');
let input = fs.readFileSync('/dev/stdin').toString().split('\n');

let [n, m] = input[0].split(" ").map(Number); // 1부터 N까지 자연수 중에서 중복 없이 M개를 고른 조합
let arr = []; // 조합을 계산하고자 하는 원소(element)가 담긴 배열
for (let i = 1; i <= n; i++) arr.push(i);
let visited = new Array(n).fill(false); // 각 원소 인덱스(index)별 방문 여부
let selected = []; // 현재 조합에 포함된 원소(element)의 인덱스

let answer = "";
function dfs(arr, depth, start) {
  if (depth == m) { // 모든 조합을 확인하는 부분
    let result = []; // 조합(combination) 결과 저장 테이블
    for (let i of selected) result.push(arr[i]);
    for (let x of result) answer += x + " "; // 계산된 조합을 실질적으로 처리하는 부분
    answer += "\n";
    return;
  }
  for (let i = start; i < arr.length; i++) { // start 지점부터 하나씩 원소 인덱스(index)를 확인하며
    if (visited[i]) continue; // [중복을 허용하지 않으므로] 이미 처리 된 원소라면 무시
    selected.push(i); // 현재 원소 선택
    visited[i] = true; // 현재 원소 방문 처리
    dfs(arr, depth + 1, i + 1); // 조합이므로, 재귀 함수 호출시 다음 인덱스(index)를 넣기
    selected.pop(); // 현재 원소 선택 취소
    visited[i] = false; // 현재 원소 방문 처리 취소
  }
}
dfs(arr, 0, 0);
console.log(answer);
```

N과 M (3): https://www.acmicpc.net/problem/15651

혼자 힘으로 풀어보기

JavaScript
백트래킹
백트래킹
문제 풀이

문제 제목: N과 M (3)

문제 난이도: ★★☆☆☆

문제 유형: 백트래킹, 경우의 수, 완전 탐색

추천 풀이 시간: 40분

N과 M (3): https://www.acmicpc.net/problem/15651

JavaScript
백트래킹
백트래킹
문제 풀이

문제 해결 아이디어

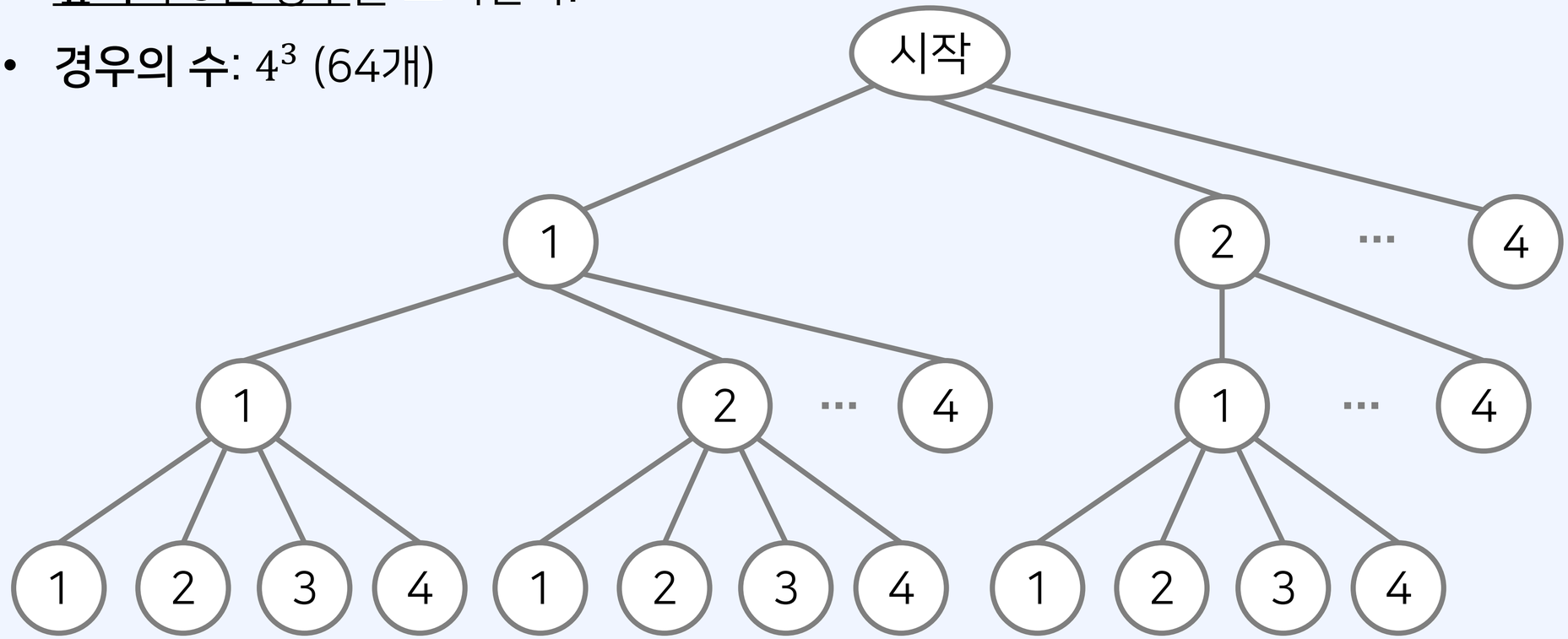
- 1부터 N까지 자연수 중에서 M개를 고른 **모든 수열**을 계산한다.

[중복 순열] 이때, 같은 수를 여러 번 골라도 된다.

N과 M (3): https://www.acmicpc.net/problem/15651

문제 해결 아이디어

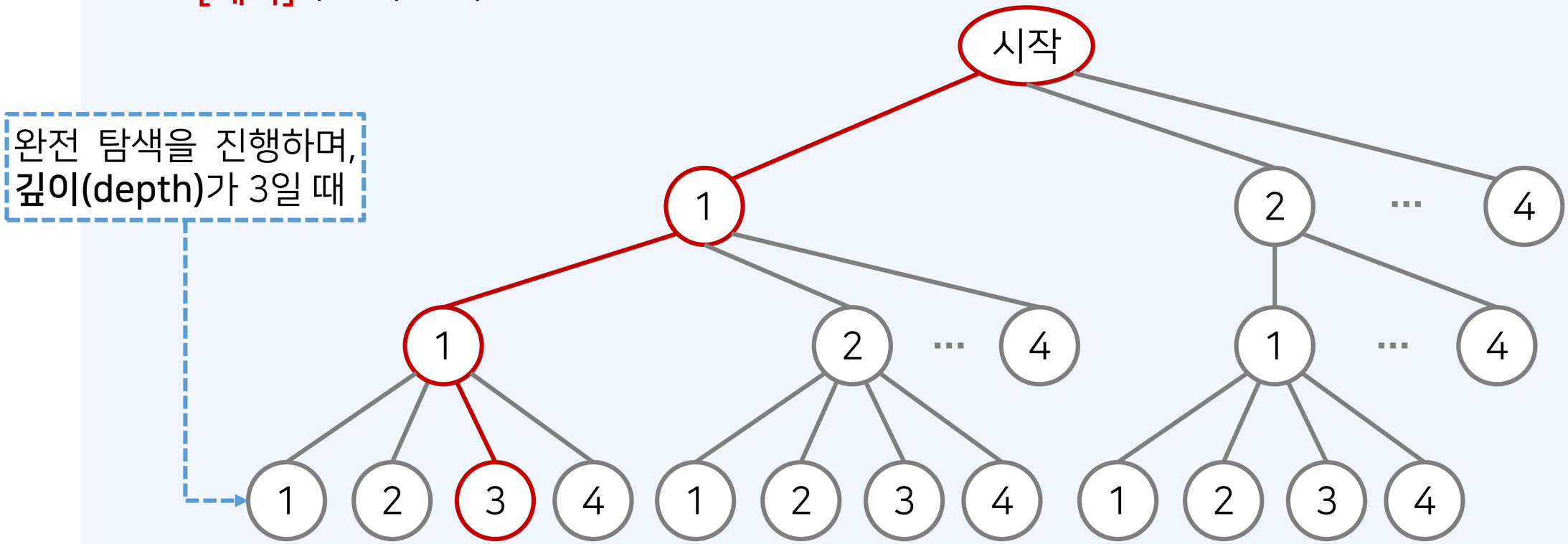
- 4개의 수 [1, 2, 3, 4]에서 3개를 고르는 **모든 중복 순열**을 고려해 보자.
- **깊이**가 3인 경우를 고려한다.
- **경우의 수**: 4^3 (64개)



N과 M (3): https://www.acmicpc.net/problem/15651

문제 해결 아이디어

- 4개의 수 [1, 2, 3, 4]에서 3개를 고르는 **모든 중복 순열**을 고려해 보자.
- **[예시]** 1 → 1 → 4



N과 M (3): https://www.acmicpc.net/problem/15651

문제 해결 아이디어

JavaScript
백트래킹
백트래킹
문제 풀이

- 모든 순열의 수를 고려하기 위해 **재귀 함수(백트래킹)**를 사용할 수 있다.
- 하나의 순열을 트리(tree)에서 리프 노드까지의 경로로 생각할 수 있다.

→ 이때, M개의 원소를 뽑는 순열을 고려하는 것이므로, **깊이(depth)**는 M과 같다.

- 원소를 **중복하여 선택하므로**, 방문 처리(visited) 배열을 사용하지 않는다.

→ 한 번 방문한(방문 처리된) 원소도 중복해서 또 방문할 수 있다.

- 따라서 기본 순열 코드에서 단순히 $visited$ **변수를 제거**한다.

N과 M (3): https://www.acmicpc.net/problem/15651

JavaScript 백트래킹
백트래킹 문제 풀이

정답 코드 예시

```
let fs = require('fs');
let input = fs.readFileSync('/dev/stdin').toString().split('\n');

let [n, m] = input[0].split(" ").map(Number); // 1부터 N까지 자연수 중에서 M개를 고른 중복 순열
let arr = []; // 중복 순열을 계산하고자 하는 원소(element)가 담긴 배열
for (let i = 1; i <= n; i++) arr.push(i);
let selected = []; // 현재 중복 순열에 포함된 원소(element)의 인덱스

let answer = "";
function dfs(arr, depth) {
  if (depth == m) { // 모든 중복 순열을 확인하는 부분
    let result = []; // 중복 순열 결과 저장 테이블
    for (let i of selected) result.push(arr[i]);
    for (let x of result) answer += x + " "; // 계산된 중복 순열을 실질적으로 처리하는 부분
    answer += "\n";
    return;
  }
  for (let i = 0; i < arr.length; i++) { // 하나씩 원소 인덱스(index)를 확인하며
    selected.push(i); // 현재 원소 선택
    dfs(arr, depth + 1); // 재귀 함수 호출
    selected.pop(); // 현재 원소 선택 취소
  }
}
dfs(arr, 0);
console.log(answer);
```

N과 M (4): https://www.acmicpc.net/problem/15652

혼자 힘으로 풀어보기

JavaScript 백트래킹
백트래킹 문제 풀이

문제 제목: N과 M (4)

문제 난이도: ★★☆☆☆

문제 유형: 백트래킹, 경우의 수, 완전 탐색

추천 풀이 시간: 40분

문제 해결 아이디어

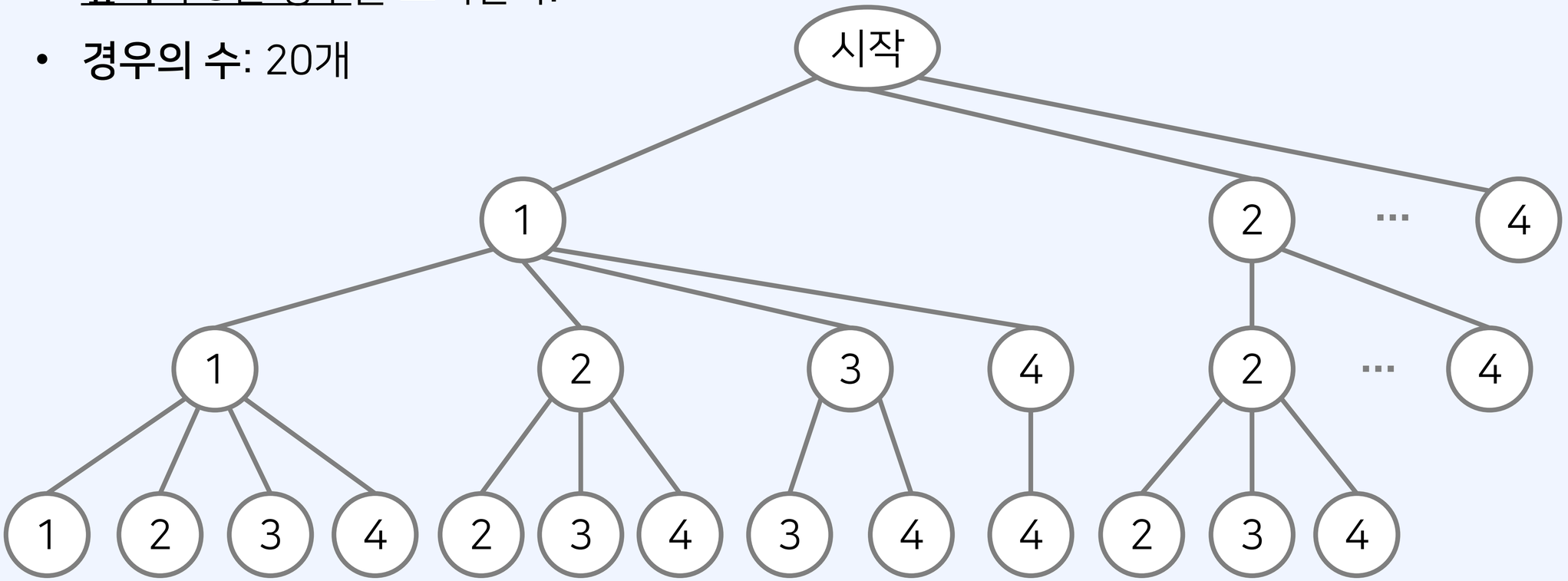
- 1부터 N까지 자연수 중에서 M개를 고른 **모든 조합**을 계산한다.
- 비내림차순이라는 점(순서가 정해진다는 점)에서 조합으로 이해할 수 있다.

[중복 조합] 이때, 같은 수를 여러 번 골라도 된다.

N과 M (4): https://www.acmicpc.net/problem/15652

문제 해결 아이디어

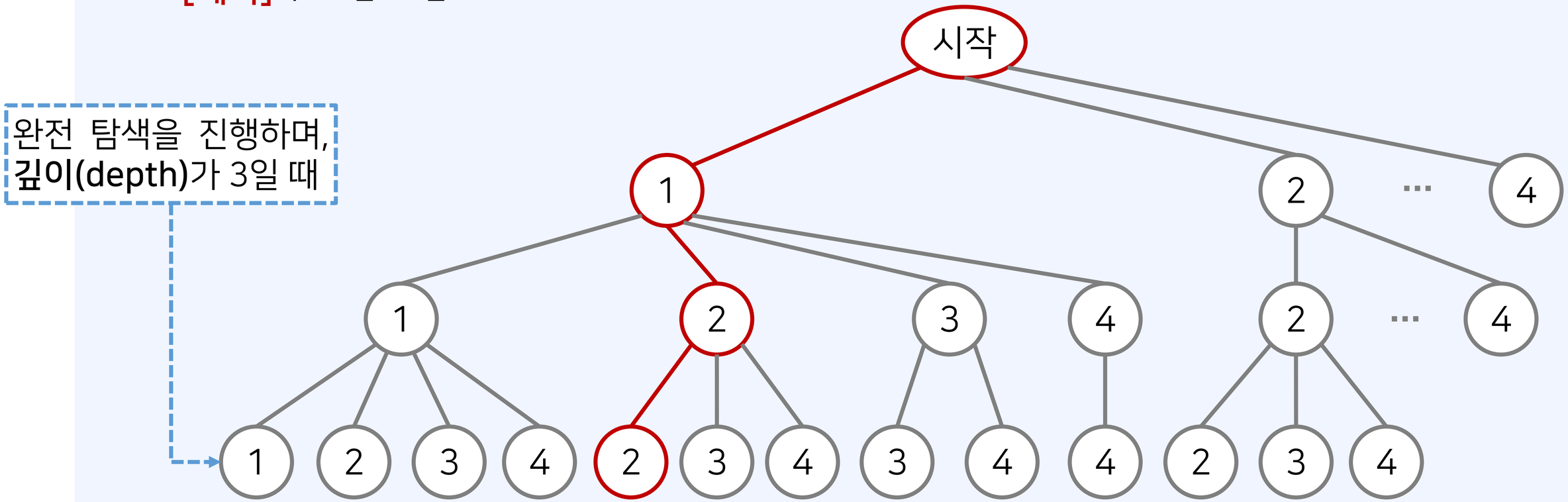
- 4개의 수 [1, 2, 3, 4]에서 3개를 고르는 **모든 중복 조합**을 고려해 보자.
- **깊이**가 3인 경우를 고려한다.
- **경우의 수**: 20개



N과 M (4): https://www.acmicpc.net/problem/15652

문제 해결 아이디어

- 4개의 수 [1, 2, 3, 4]에서 3개를 고르는 **모든 중복 조합**을 고려해 보자.
- **[예시]** 1 → 2 → 2



N과 M (4): https://www.acmicpc.net/problem/15652

문제 해결 아이디어

- 모든 조합의 수를 고려하기 위해 **재귀 함수(백트래킹)**를 사용할 수 있다.
- 하나의 조합을 트리(tree)에서 리프 노드까지의 경로로 생각할 수 있다.

→ 이때, M개의 원소를 뽑는 조합을 고려하는 것이므로, **깊이(depth)**는 M과 같다.

- 원소를 **중복하여 선택하므로**, 방문 처리(visited) 배열을 사용하지 않는다.

→ 한 번 방문한(방문 처리된) 원소도 중복해서 또 방문할 수 있다.

- 따라서 기본 순열 코드에서 $visited$ **변수를 제거**한다.

[조합] 재귀 함수를 호출할 때마다(자식 노드로 갈수록) **선택되는 값이 현재 값 이상**인 것이 핵심이다.

→ 순열 소스 코드와 달리 $start$ **변수가 사용**된다.

N과 M (4): https://www.acmicpc.net/problem/15652

JavaScript 백트래킹
백트래킹 문제 풀이

정답 코드 예시

```
let fs = require('fs');
let input = fs.readFileSync('/dev/stdin').toString().split('\n');

let [n, m] = input[0].split(" ").map(Number); // 1부터 N까지 자연수 중에서 M개를 고른 중복 조합
let arr = []; // 중복 조합을 계산하고자 하는 원소(element)가 담긴 배열
for (let i = 1; i <= n; i++) arr.push(i);
let selected = []; // 현재 중복 조합에 포함된 원소(element)의 인덱스

let answer = "";
function dfs(arr, depth, start) {
  if (depth == m) { // 모든 중복 조합을 확인하는 부분
    let result = []; // 중복 조합 결과 저장 테이블
    for (let i of selected) result.push(arr[i]); // 계산된 중복 조합을 실질적으로 처리하는 부분
    for (let x of result) answer += x + " ";
    answer += "\n";
    return;
  }
  for (let i = start; i < arr.length; i++) { // start 지점부터 하나씩 원소 인덱스(index)를 확인하며
    selected.push(i); // 현재 원소 선택
    dfs(arr, depth + 1, i); // 조합이므로, 재귀 함수 호출시 현재 인덱스(index)를 넣기
    selected.pop(); // 현재 원소 선택 취소
  }
}
dfs(arr, 0, 0);
console.log(answer);
```