3주차

DFS, BFS, Backtracking

신지섭

오늘 할 내용은?

0. 그래프: 비선형 자료구조인 그래프에 대해 알아봅시다

1. DFS, BFS: 그래프를 완전 탐색해봅시다

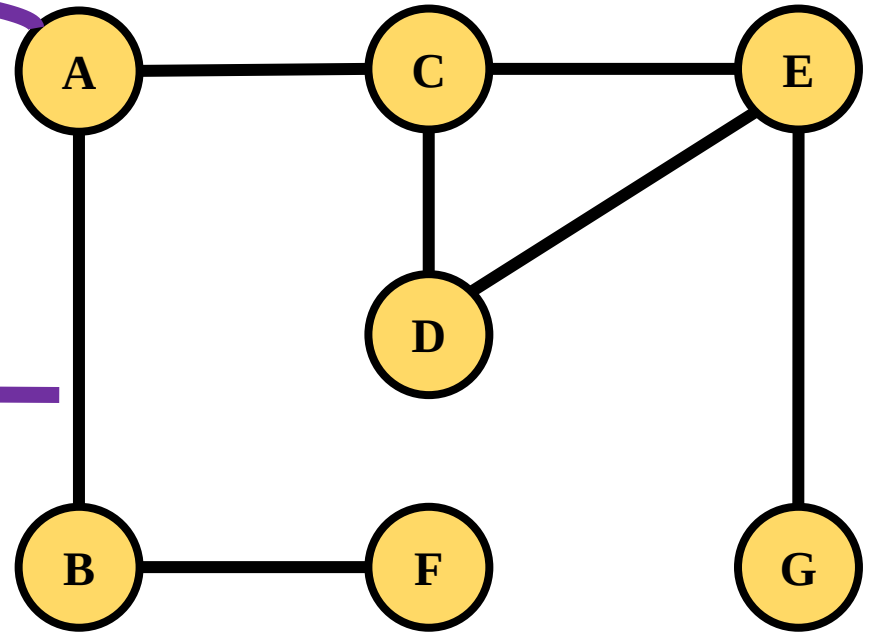
2. Backtracking: DFS로 탐색하면서 ‘가지치기’로 최적화 해봅시다

그래프 Graph

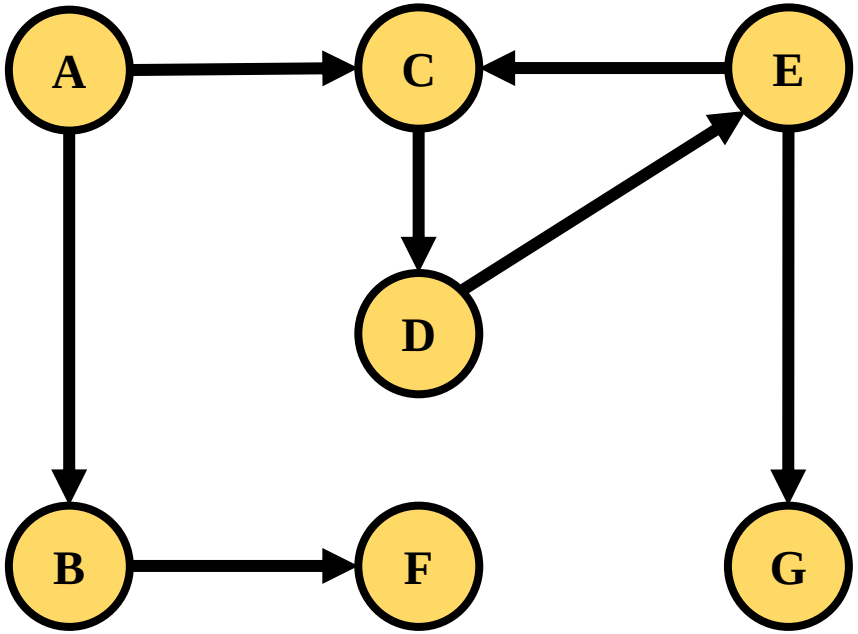
정점 간의 관계를 나타내는 비선형 자료구조

노드(node): 정점(vertex)

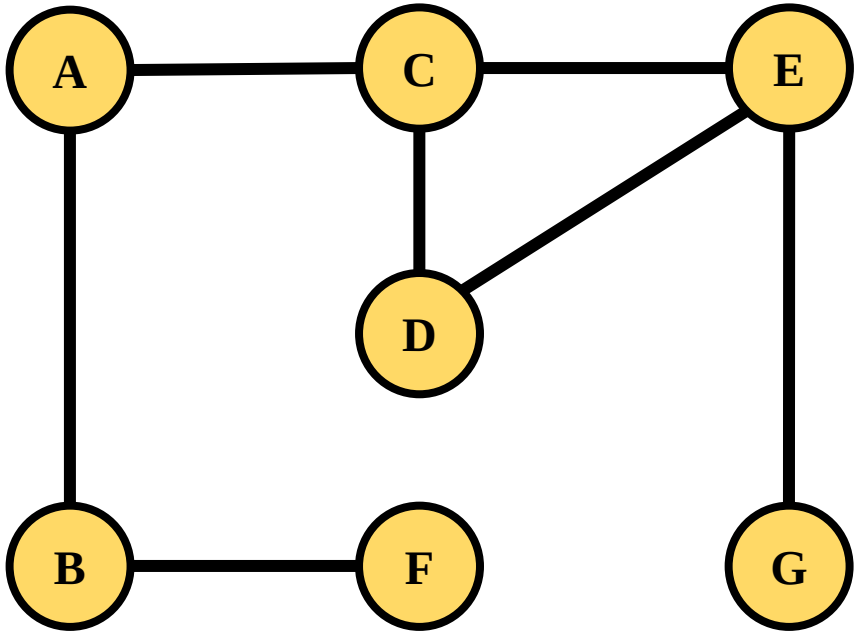
간선(edge): 노드의 관계를 나타낸 선



그래프 Graph



Directed Graph
방향 그래프



Undirected Graph
무방향 그래프

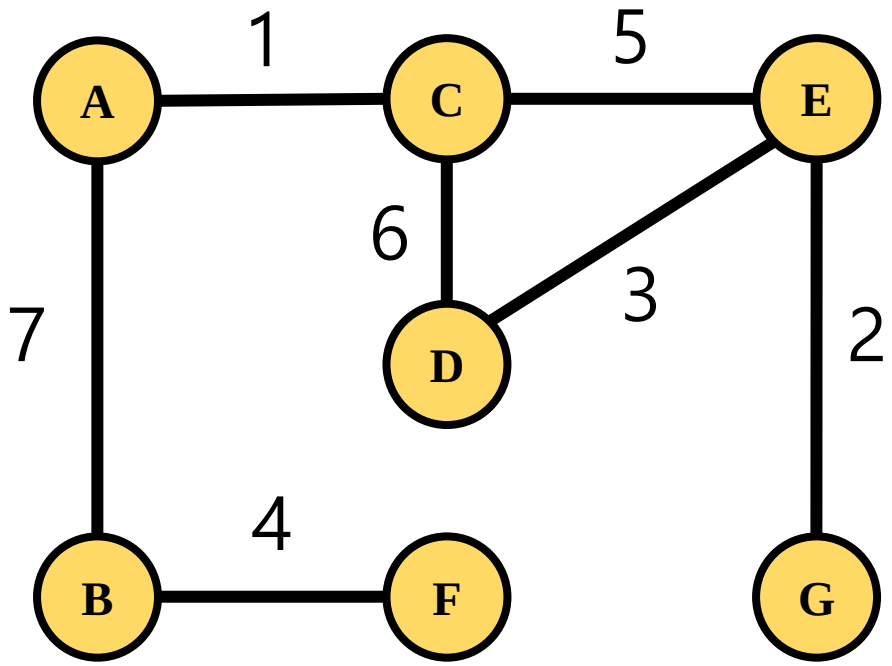
그래프 Graph

가중치 그래프(Weight Graph)

그래프의 간선에 가중치가 부여된 그래프

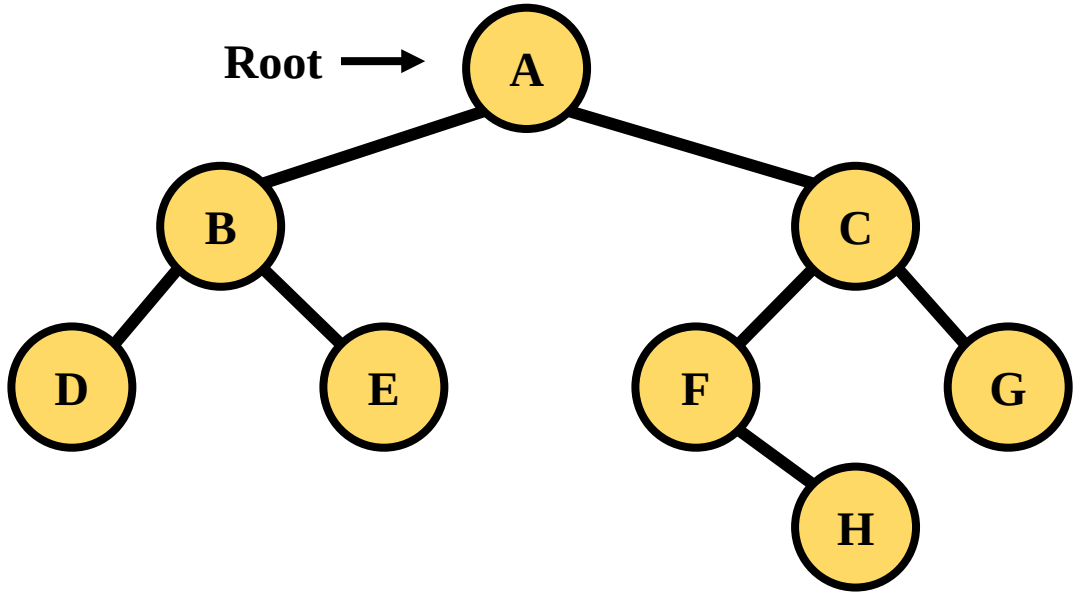
network라고도 불림

ex) 이동하는 비용, 도로의 길이 등



그래프 Graph

트리(Tree): 그래프의 일종



※ 특징

1. 사이클이 없고 하나의 방향성을 가짐
2. 각 노드는 깊이(level)을 가진다
   ※ Root 노드는 level 0
3. 간선의 개수 == 정점의 개수 - 1

그래프 Graph - 구현

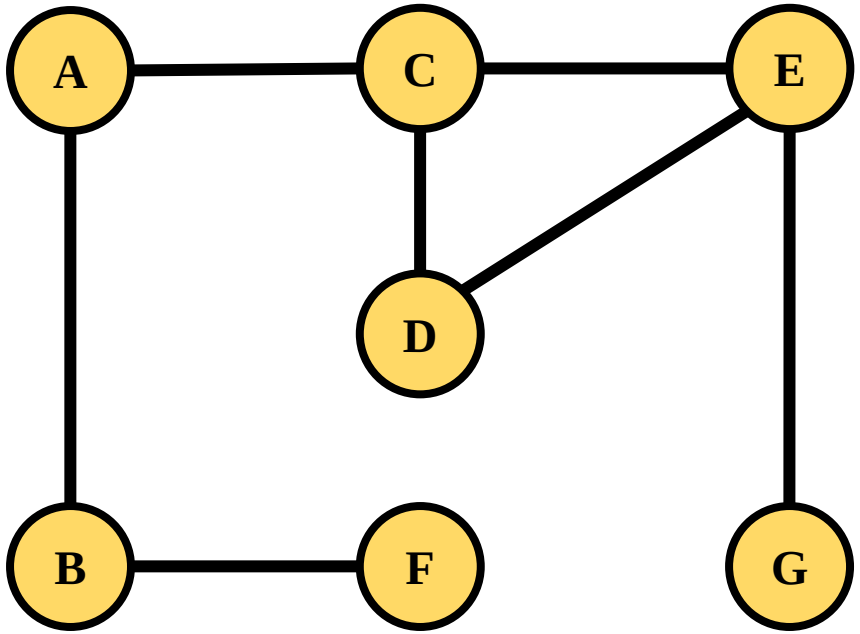
1. 인접행렬 → 2차원 배열로 구현

2. 인접리스트 → vector로 구현

그래프 Graph - 인접행렬

간선이 존재하면 ‘1’, 그렇지 않으면 ‘0’

i \ j	A	B	C	D	E	F	G
A	0	1	1	0	0	0	0
B	1	0	0	0	0	1	0
C	1	0	0	1	1	0	0
D	0	0	1	0	1	0	0
E	0	0	1	1	0	0	1
F	0	1	0	0	0	1	0
G	0	0	0	0	0	0	0

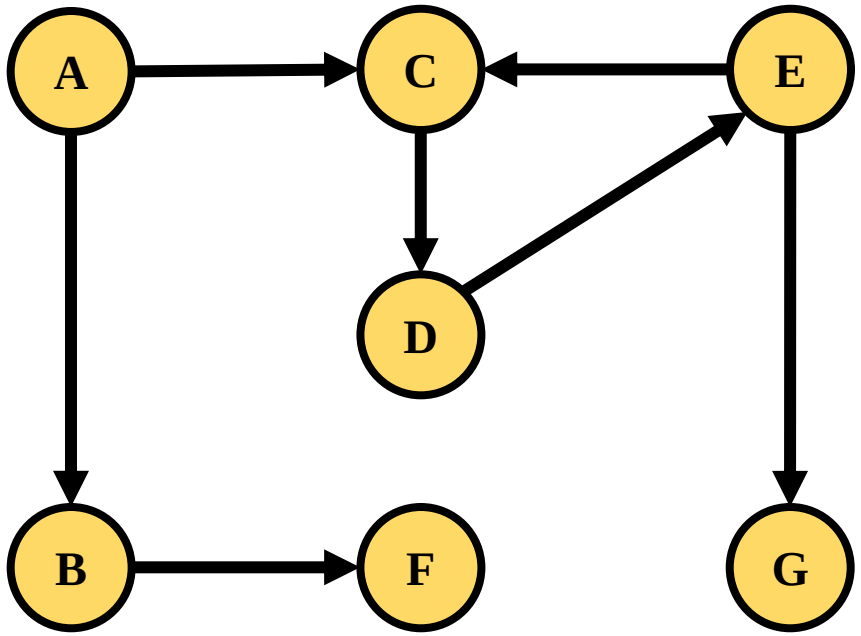


무방향 그래프

그래프 Graph - 인접행렬

i → j인 간선이 존재하면 ‘1’, 그렇지 않으면 ‘0’

i \ j	A	B	C	D	E	F	G
A	0	1	1	0	0	0	0
B	0	0	0	0	0	1	0
C	0	0	0	1	0	0	0
D	0	0	0	0	1	0	0
E	0	0	1	0	0	0	1
F	0	0	0	0	0	0	0
G	0	0	0	0	0	0	0

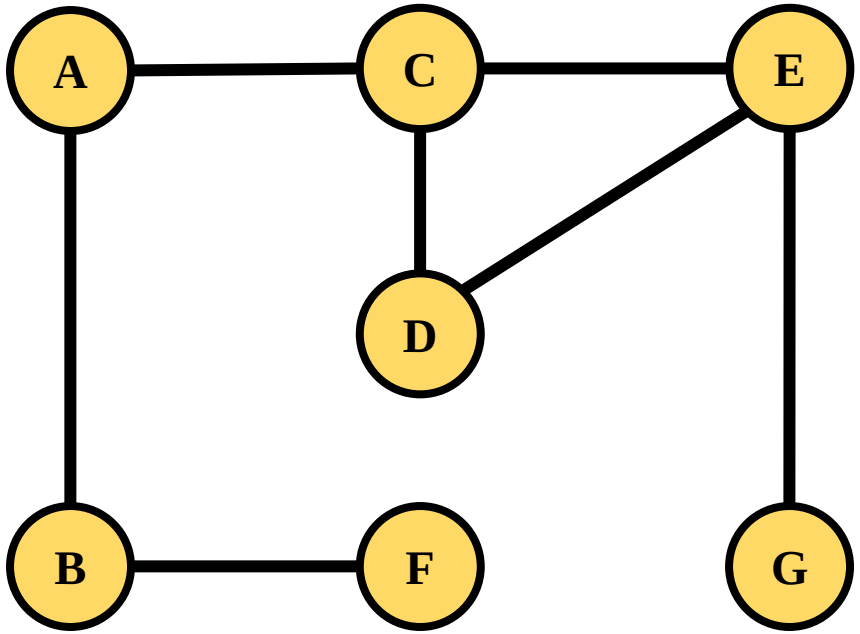


방향 그래프

그래프 Graph - 인접리스트

하나의 정점에 대해 연결된 정점을 vector에 push_back

A	B	C		
B	A	F		
C	A	D	E	
D	C	E		
E	C	D	G	
F	B			
G	E			

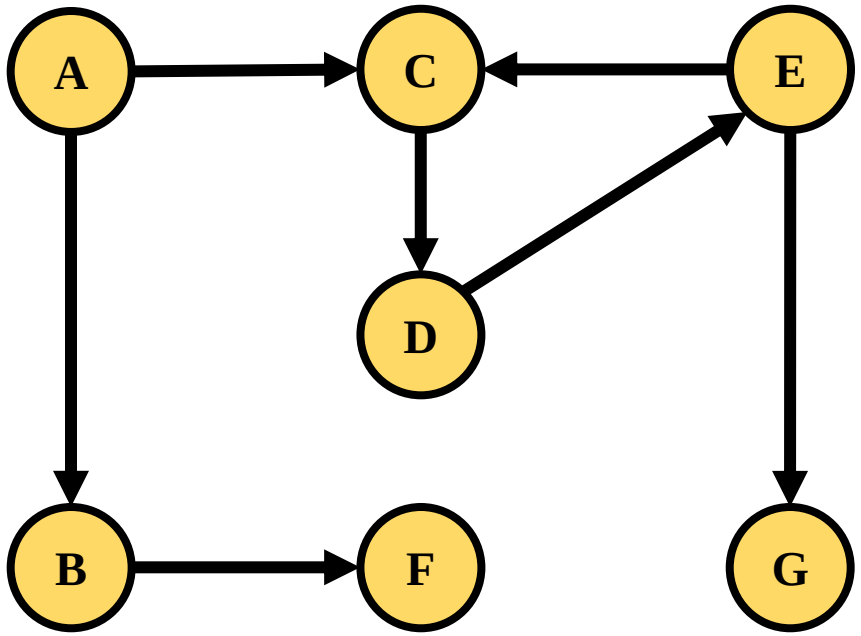


무방향 그래프

그래프 Graph - 인접리스트

하나의 정점에 대해 갈 수 있는 정점을 vector에 push_back

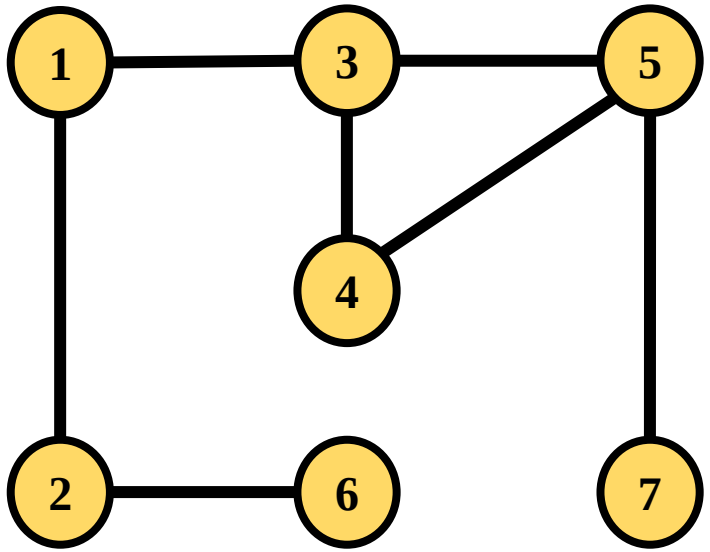
A	B C
B	F
C	D
D	E
E	C G
F	empty
G	empty



방향 그래프

그래프 Graph - 인접리스트

1	2	3		
2	1	6		
3	1	4	5	
4	3	5		
5	3	4	7	
6	2			
7	5			




```cpp
#include <vector>
#include <iostream>
using namespace std;

// 그래프 데이터
pair<int, int> edge[7] = { {1, 2}, {1, 3}, {2, 6}, {3, 4}, {3,5}, {4,5}, {5, 7} };

int main() {
    vector<int> ve[105];
    int n = 7;
    for (int i = 0; i < n; i++) {
        ve[edge[i].first].push_back(edge[i].second);
        ve[edge[i].second].push_back(edge[i].first); // 양방향이므로 두번 삽입
    }

    return 0;
}
```

DFS 깊이 우선 탐색

현재 정점에서 갈 수 있는 정점들까지 들어가며 탐색하는데

깊이를 우선시하며 탐색!

한 번 방문한 정점은 다시 방문하지 않는다

=> visit 배열을 만들어 방문여부를 확인하자!

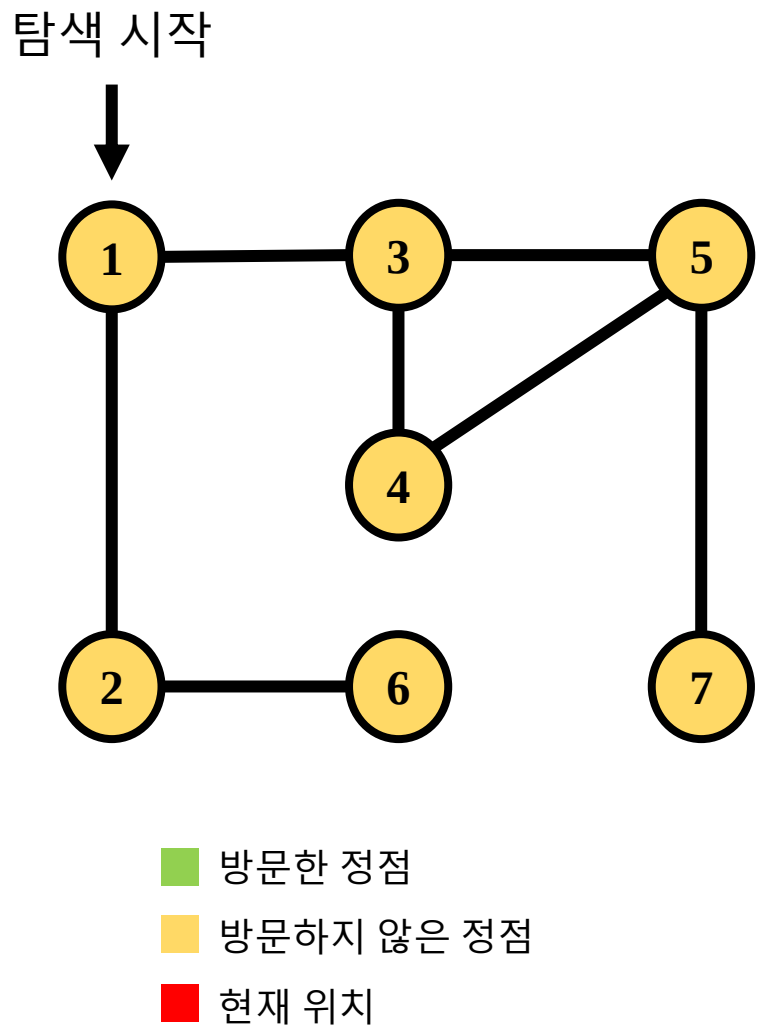
재귀함수나 **스택(stack)**으로 구현

DFS 깊이 우선 탐색

현재 위치: **not started**

방문 순서

	1	2	3	4	5	6	7
visit	0	0	0	0	0	0	0



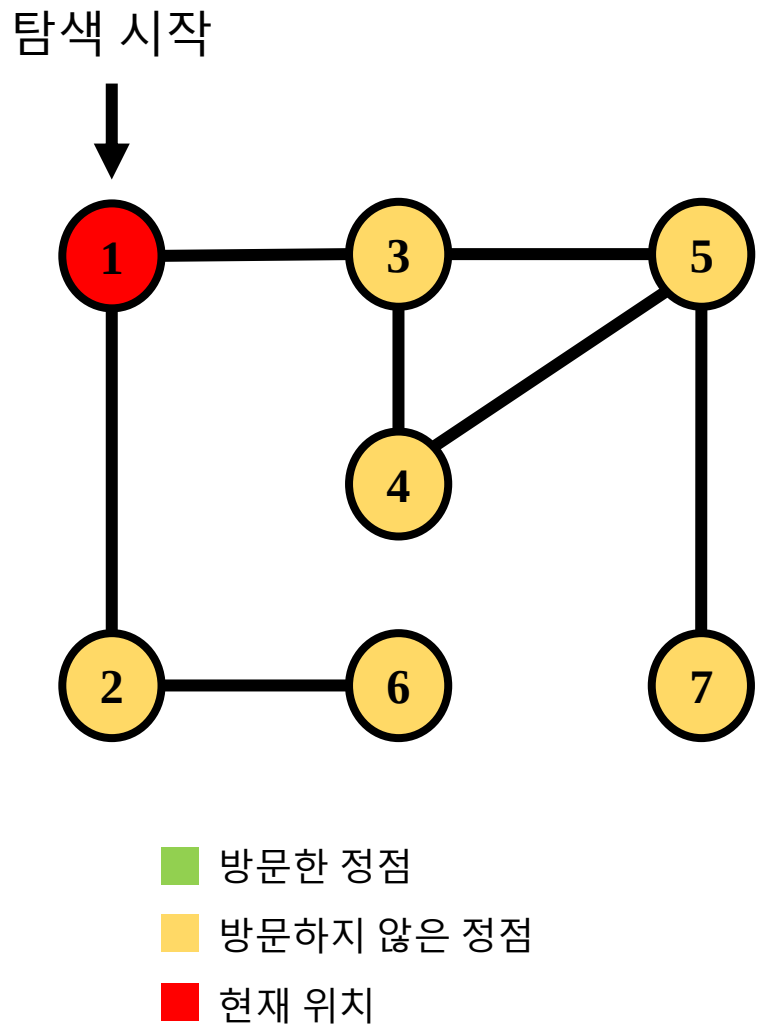
DFS 깊이 우선 탐색

현재 위치: 1

방문 순서

1

	1	2	3	4	5	6	7
visit	1	0	0	0	0	0	0



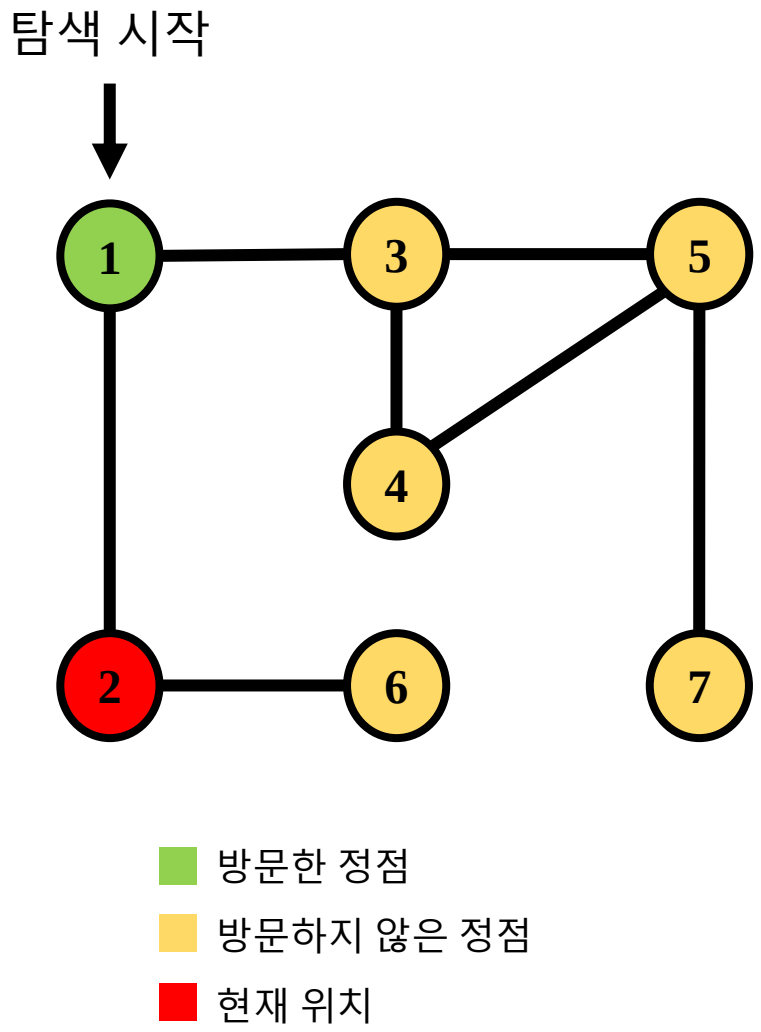
DFS 깊이 우선 탐색

현재 위치: 2

방문 순서

1 → 2

	1	2	3	4	5	6	7
visit	1	1	0	0	0	0	0



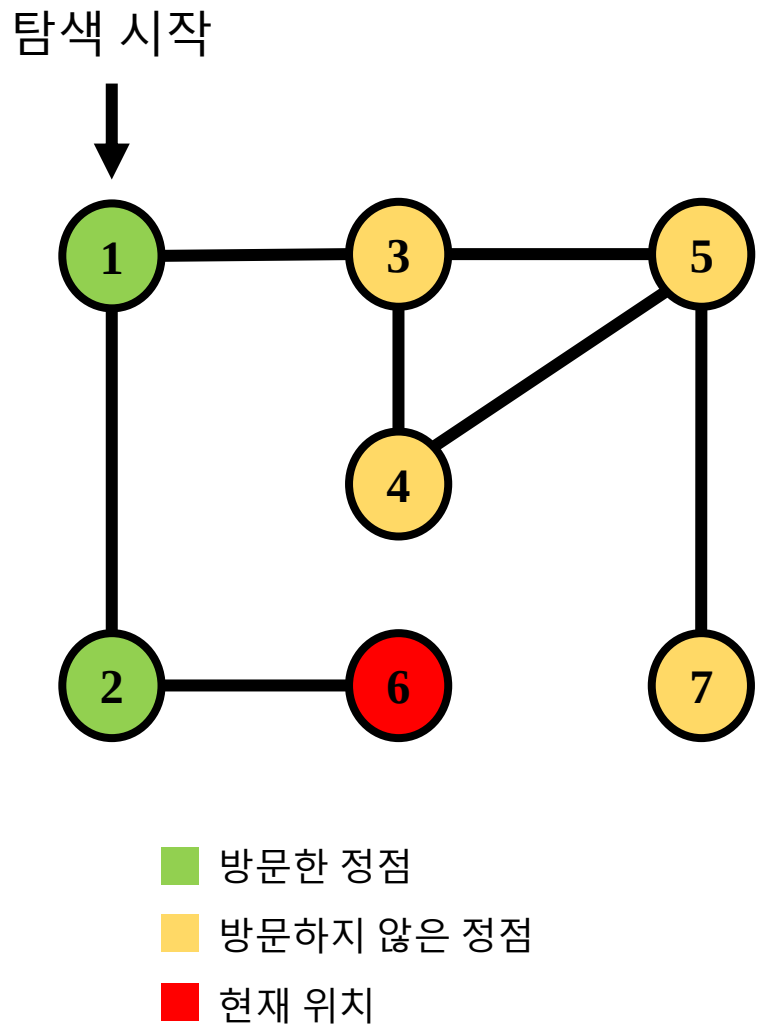
DFS 깊이 우선 탐색

현재 위치: **6**

방문 순서

1 → 2 → 6

	1	2	3	4	5	6	7
visit	1	1	0	0	0	1	0



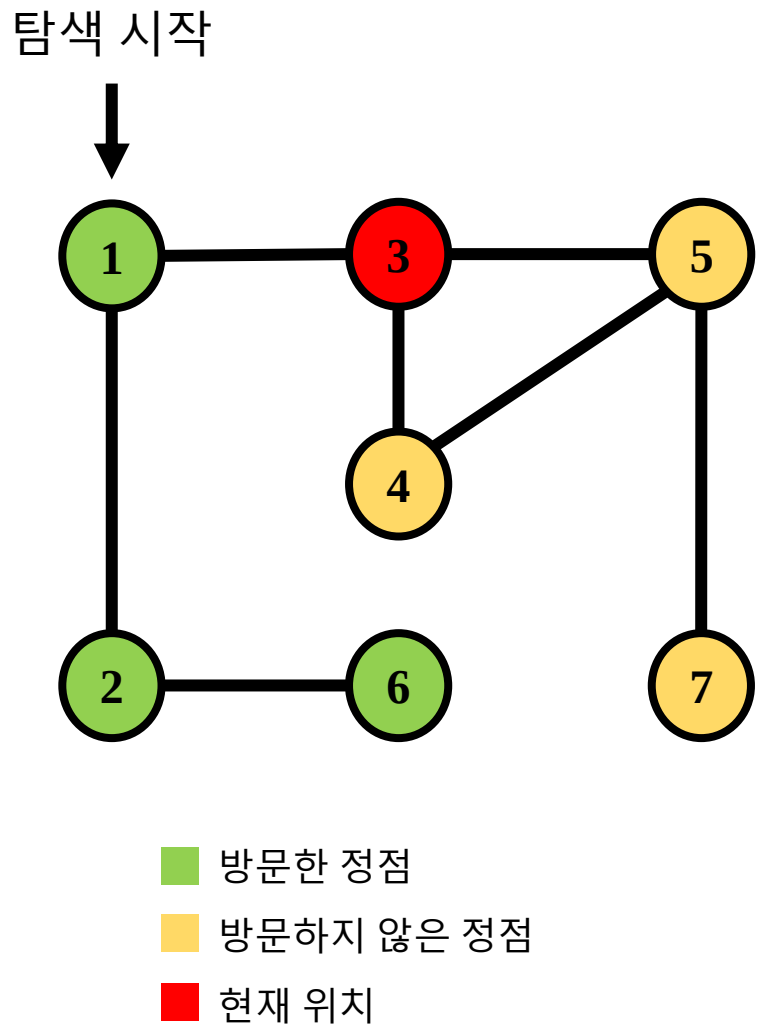
DFS 깊이 우선 탐색

현재 위치: **3**

방문 순서

1 → 2 → 6 → 3

	1	2	3	4	5	6	7
visit	1	1	1	0	0	1	0



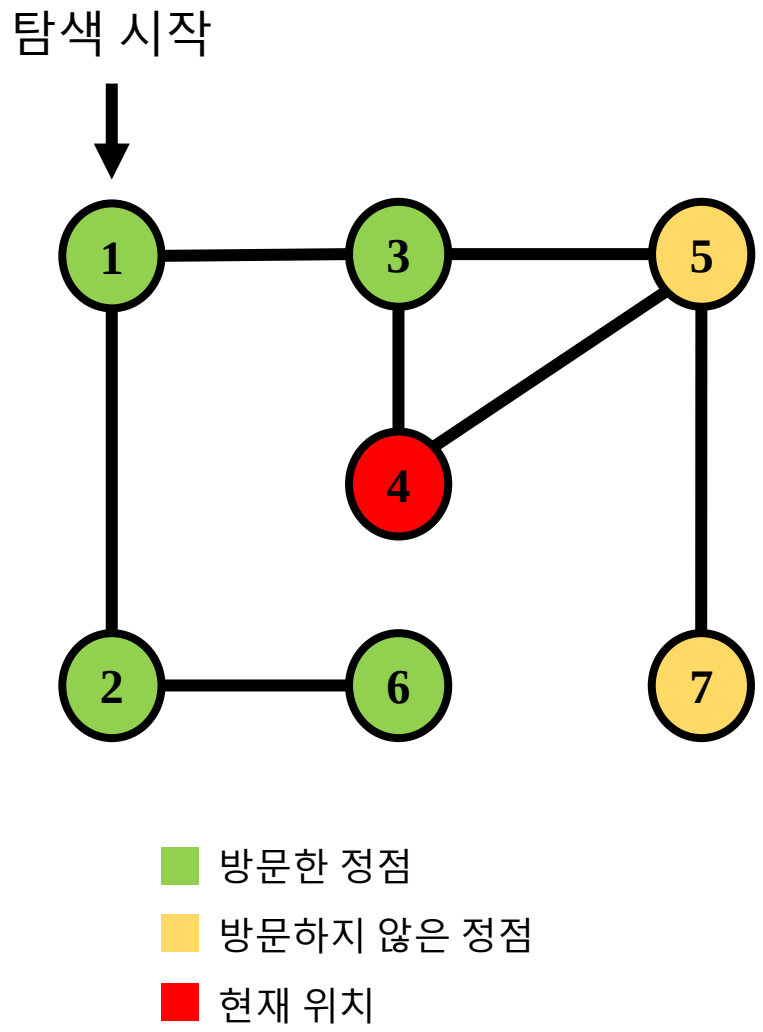
DFS 깊이 우선 탐색

현재 위치: **4**

방문 순서

1 → 2 → 6 → 3 → 4

	1	2	3	4	5	6	7
visit	1	1	1	1	0	1	0



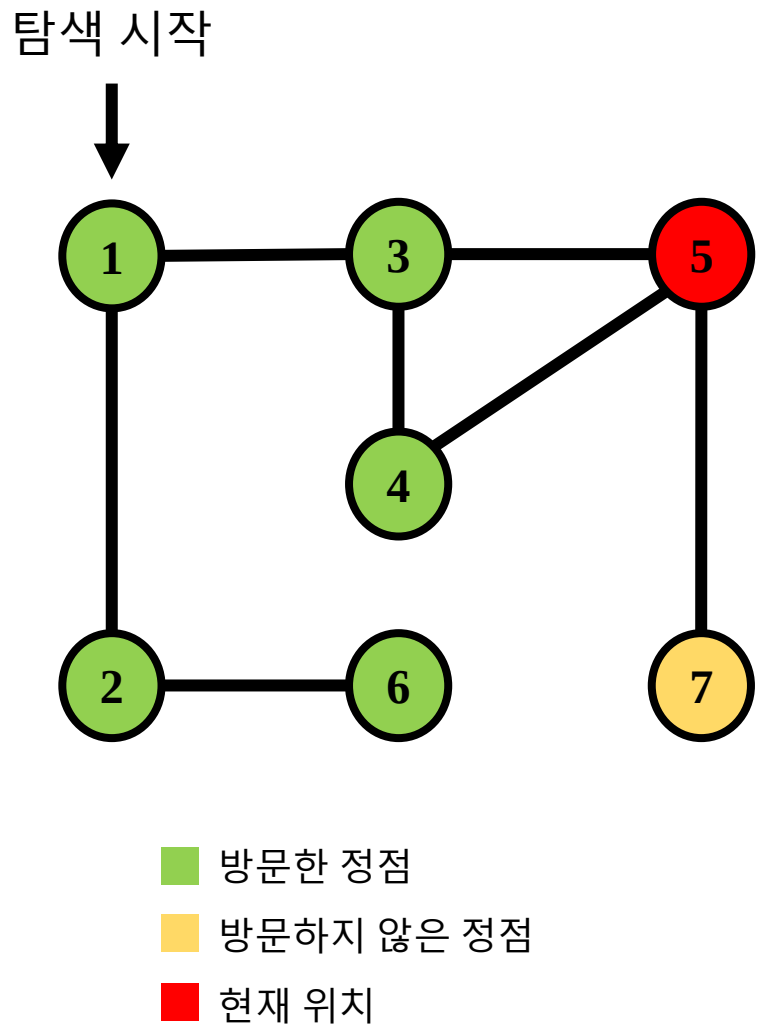
DFS 깊이 우선 탐색

현재 위치: 5

방문 순서

1 → 2 → 6 → 3 → 4 → 5

	1	2	3	4	5	6	7
visit	1	1	1	1	1	1	0



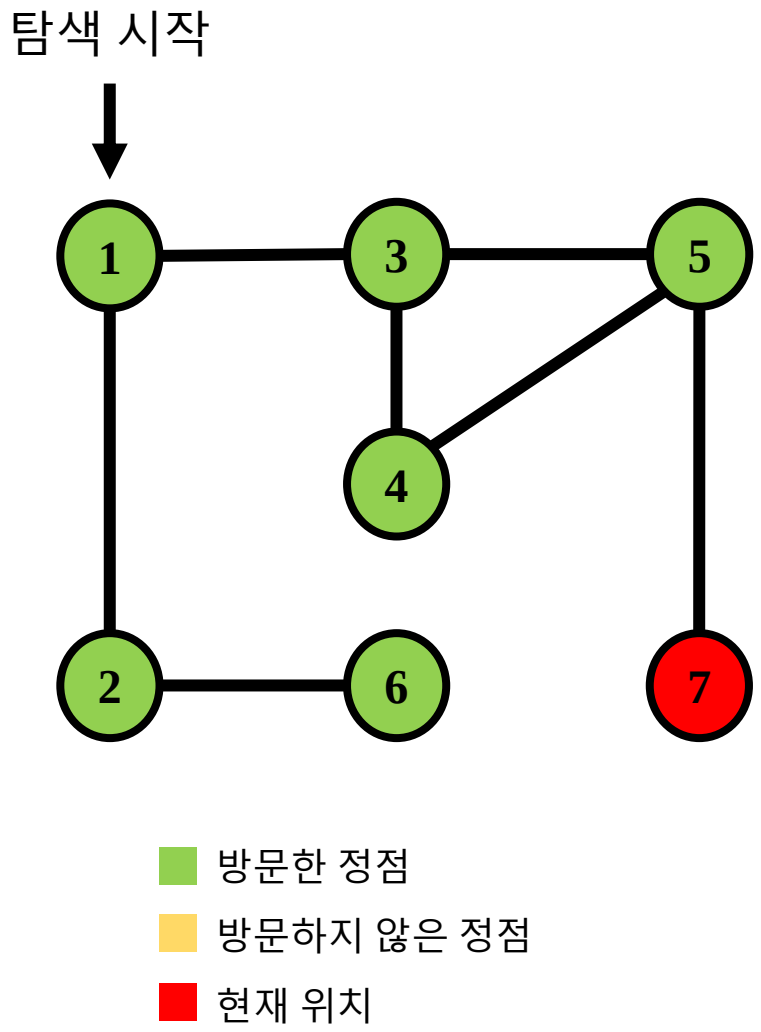
DFS 깊이 우선 탐색

현재 위치: 7

방문 순서

1 → 2 → 6 → 3 → 4 → 5 → 7

	1	2	3	4	5	6	7
visit	1	1	1	1	1	1	1



BFS 너비 우선 탐색

현재 정점에서 갈 수 있는 정점들까지 들어가며 탐색하는데

너비를 우선시하며 탐색!

→ 먼 정점은 가장 나중에 방문한다

한 번 방문한 정점은 다시 방문하지 않는다

=> visit 배열을 만들어 방문여부를 확인하자!

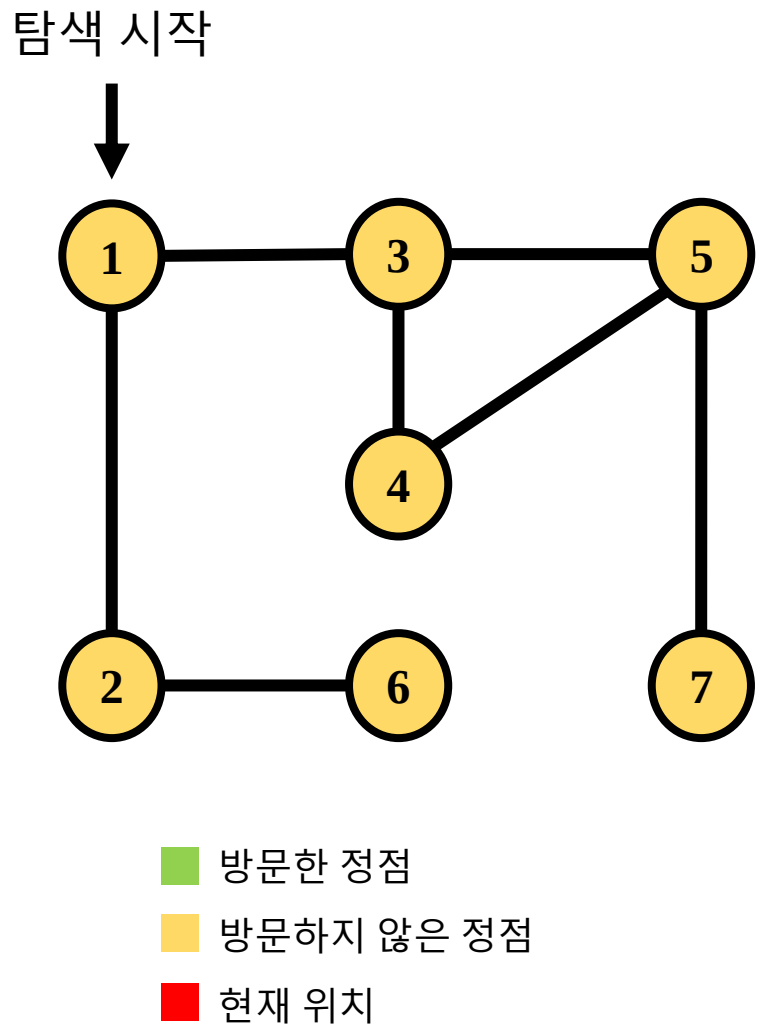
큐(queue)를 통해 구현

BFS 너비 우선 탐색

현재 위치: **not started**

방문 순서

	1	2	3	4	5	6	7
visit	1	0	0	0	0	0	0
Queue	1						



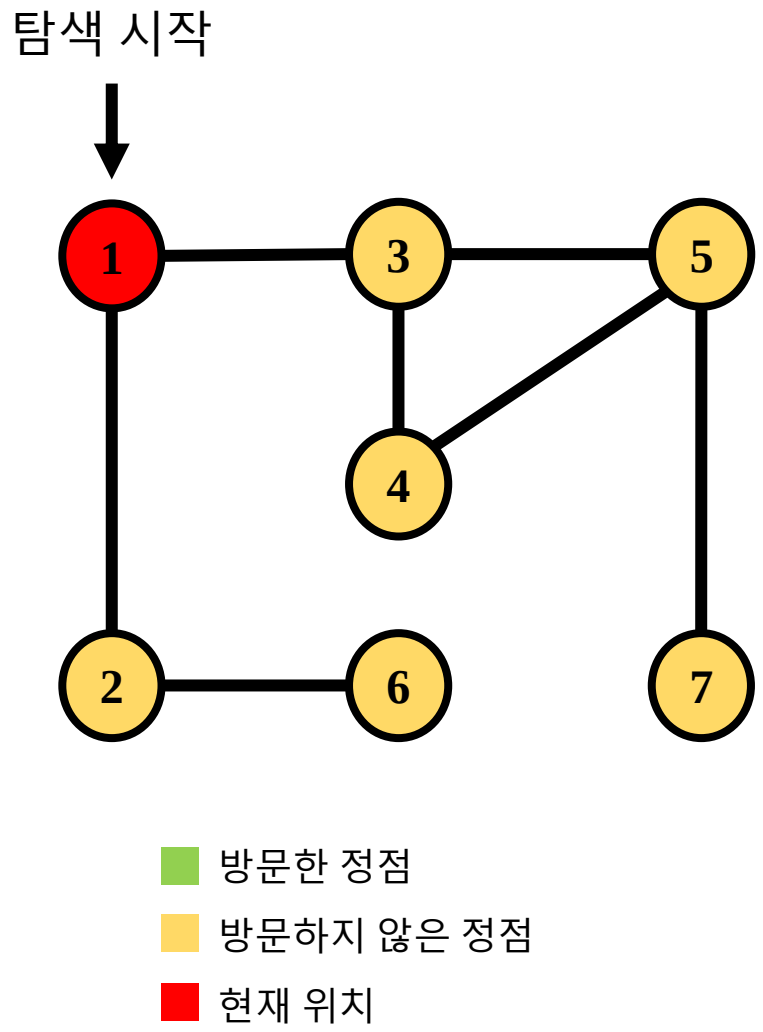
BFS 너비 우선 탐색

현재 위치: 1

방문 순서

1

	1	2	3	4	5	6	7
visit	1	1	1	0	0	0	0
Queue	2	3					



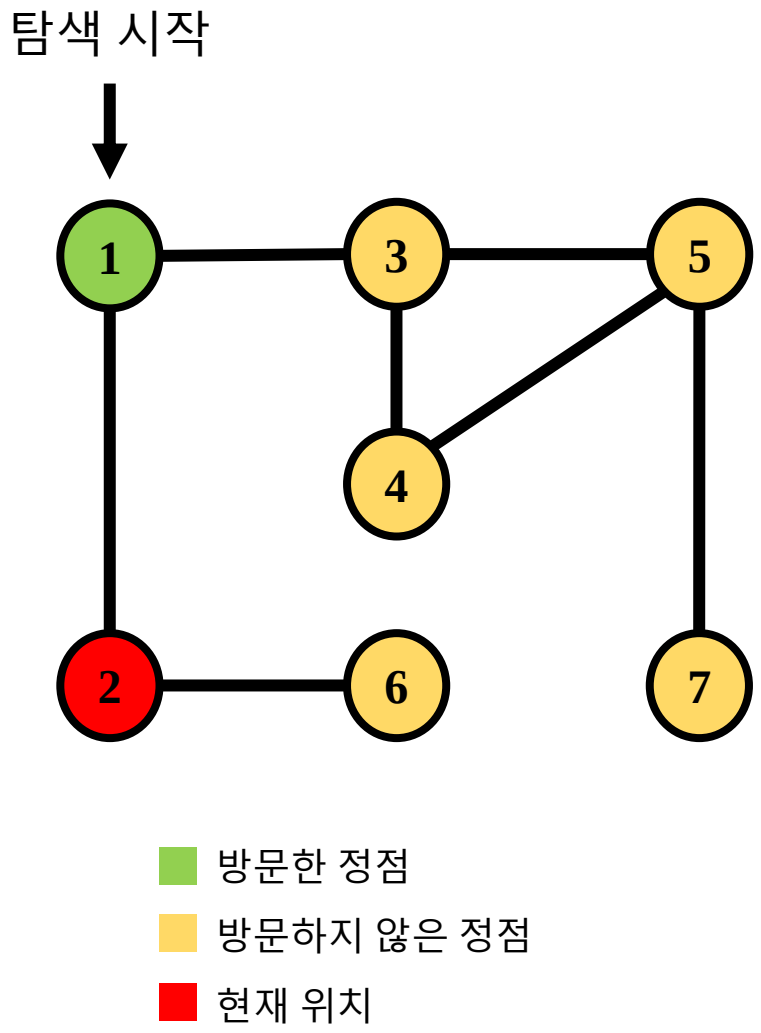
BFS 너비 우선 탐색

현재 위치: 2

방문 순서

1 → 2

	1	2	3	4	5	6	7
visit	1	1	1	0	0	1	0
Queue	3	6					



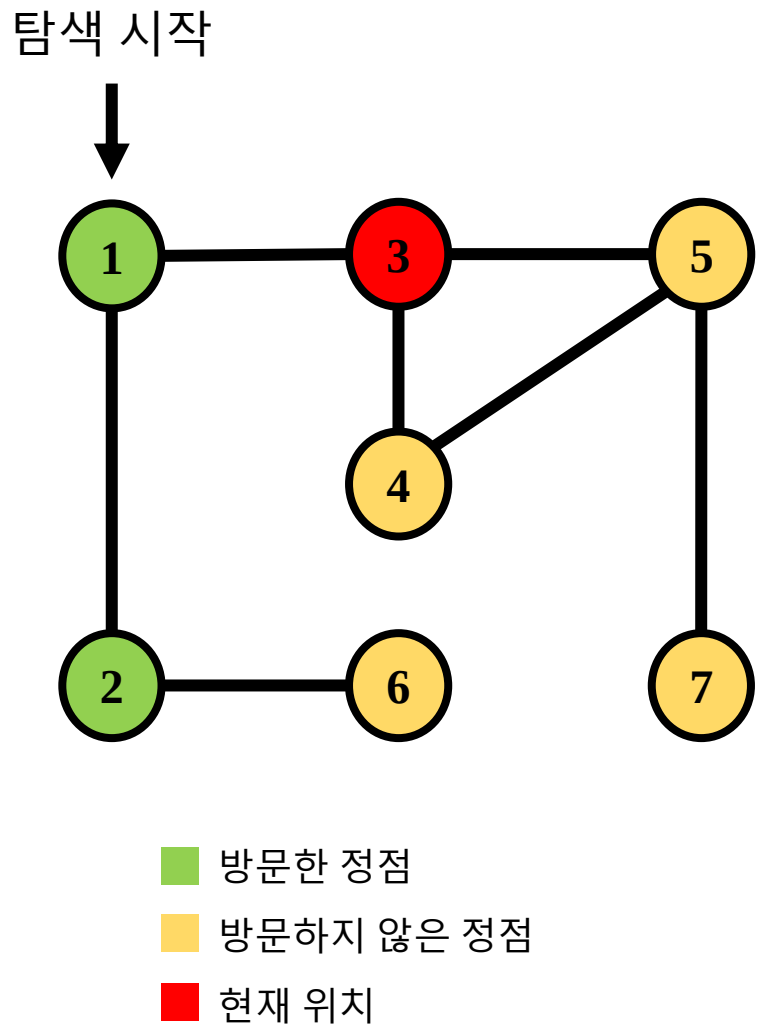
BFS 너비 우선 탐색

현재 위치: 3

방문 순서

1 → 2 → 3

	1	2	3	4	5	6	7
visit	1	1	1	1	1	1	0
Queue	6	4	5				



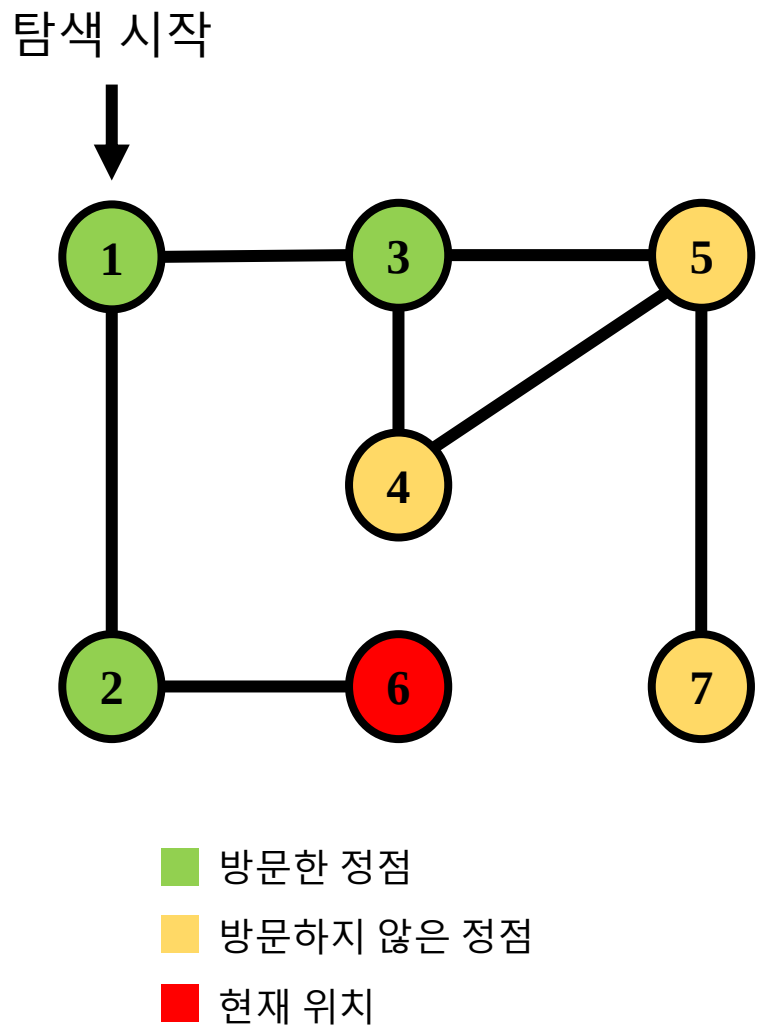
BFS 너비 우선 탐색

현재 위치: **6**

방문 순서

1 → 2 → 3 → 6

	1	2	3	4	5	6	7
visit	1	1	1	1	1	1	0
Queue	4	5					



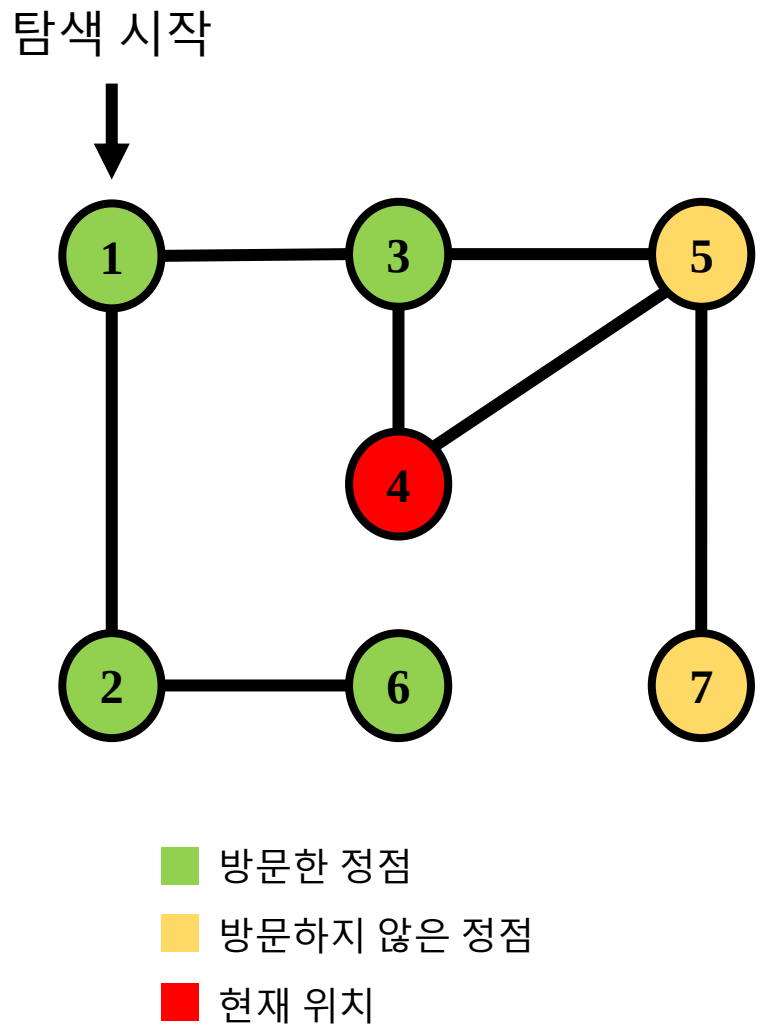
BFS 너비 우선 탐색

현재 위치: **4**

방문 순서

1 → 2 → 3 → 6 → 4

	1	2	3	4	5	6	7
visit	1	1	1	1	1	1	0
Queue	5						



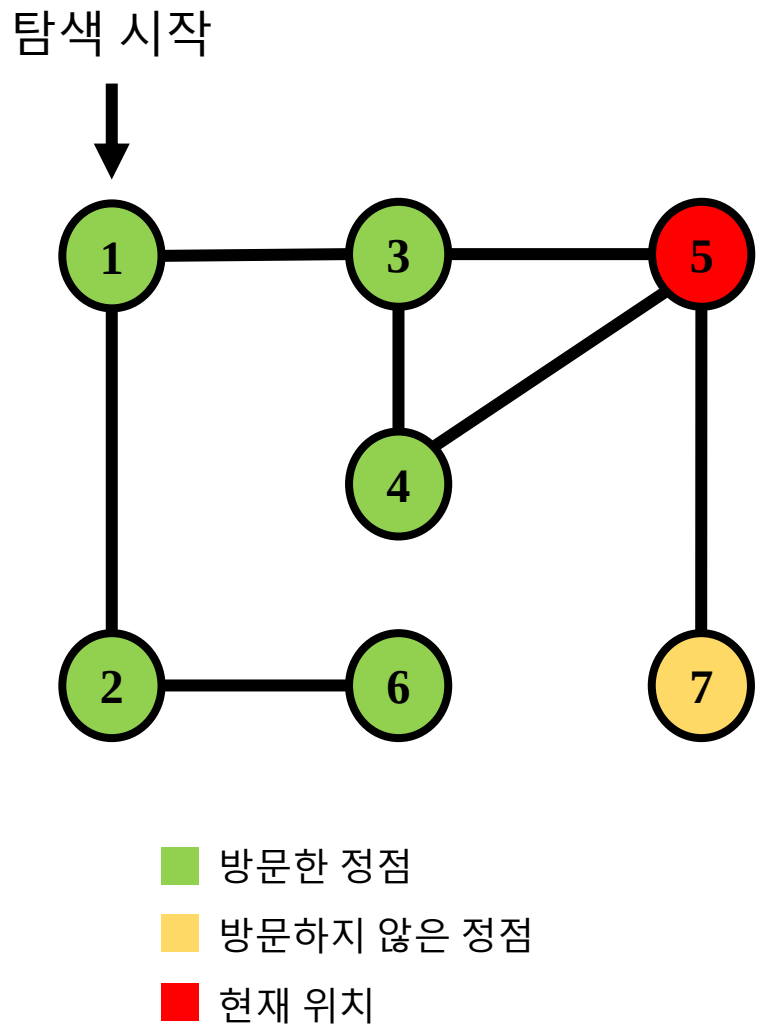
BFS 너비 우선 탐색

현재 위치: 5

방문 순서

1 → 2 → 3 → 6 → 4 → 5

	1	2	3	4	5	6	7
visit	1	1	1	1	1	1	1
Queue	7						



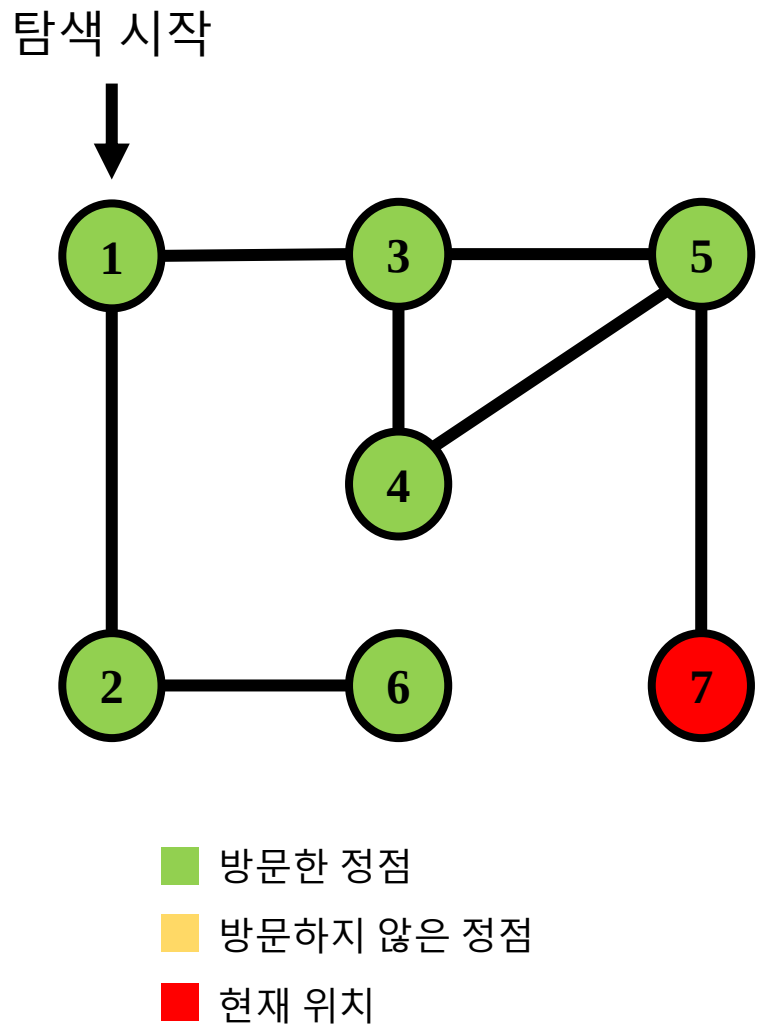
BFS 너비 우선 탐색

현재 위치: 7

방문 순서

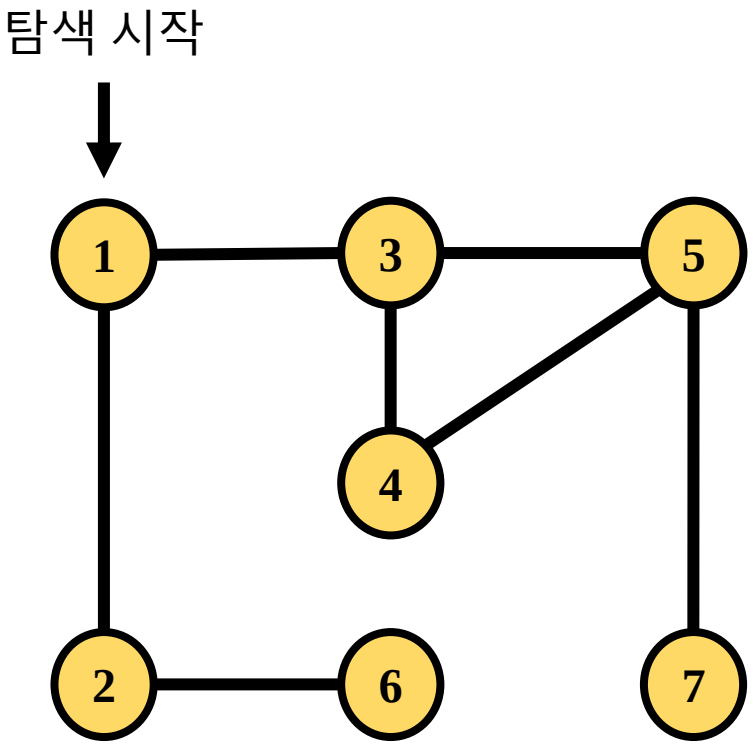
1 → 2 → 3 → 6 → 4 → 5 → 7

	1	2	3	4	5	6	7
visit	1	1	1	1	1	1	1
Queue							



DFS와 BFS 정리

	DFS	BFS
방식	현재 갈 수 있는 점을 먼저 탐색	인접한 정점을 모두 먼저 탐색
방문 순서	1 → 2 → 3 → 6 → 4 → 5 → 7	1 → 2 → 6 → 3 → 4 → 5 → 7
구현	재귀, 스택	큐
특징	구현이 간단, 경로의 특징 저장 가능	거리를 구할 수 있음
방문여부	방문 시 확인	큐에 넣을 때 확인



DFS, BFS 문제풀이

DFS와 BFS를 구현해 봅시다!

백준 1260
DFS와 BFS

1260 DFS와 BFS

문제: 그래프를 DFS로 탐색한 결과와 BFS로 탐색한 결과를 출력

방문 할 수 있는 정점이 많으면 오름차순 정렬 출력

⇒ 인접리스트로 구현한다면 main에서 미리 정렬해주자!!

DFS와 BFS 코드

```
13  void dfs(int x) {
14      visit[x] = 1;
15      cout << x << " ";
16      for (int i : ve[x]) {//인접한 곳 탐색
17          if (visit[i] == 0) { // 미방문 시
18              dfs(i);
19          }
20      }
21  }
```

dfs 함수

```
23  void bfs() {
24      while (!q.empty()) { //queue가 빌때까지
25          int con = q.front();
26          q.pop();
27          cout << con << " ";
28          for (int i : ve[con]) {
29              if (visit[i] == 0) {
30                  visit[i] = 1; // queue에 넣을 때 visit한 것!!
31                  q.push(i);
32              }
33          }
34      }
35  }
```

bfs 함수

DFS와 BFS 코드

```cpp
#include <iostream>
#include <vector>
#include <cstring>
#include <queue>
#include <algorithm>
using namespace std;

int n, m, v;
vector<int> ve[1005]; //인접리스트로 구현
queue<int> q;
int visit[1005];
```

헤더 파일과 전역변수

```cpp
int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    cout.tie(NULL);
    cin >> n >> m >> v;
    for (int i = 0; i < m; i++) {
        int x, y; cin >> x >> y;
        ve[x].push_back(y);
        ve[y].push_back(x); //양방향 그래프
    }
    for (int i = 1; i <= n; i++) {
        if (!ve[i].empty())
            sort(ve[i].begin(), ve[i].end()); // 작은 것 먼저 방문하기 위해
    }
    dfs(v);
    cout << "\n";
    memset(visit, 0, sizeof(visit)); //visit 배열 초기화
    visit[v] = 1;
    q.push(v);
    bfs();
    return 0;
}
```

Main 함수

DFS, BFS 문제풀이

하나 더 풀어볼까요?

백준 1012
유기농 배추

1012 유기농 배추

문제: 배추(1)를 먹는 배추흰지렁이의 최소 마리 수를 출력

한 마리의 배추흰지렁이는 인접한 배추만 먹을 수 있다

⇒ 각 칸을 정점, 상하좌우의 정점이 간선으로 이어져 있는 그래프라 생각하면

visit 배열을 초기화하지 않고 몇 번 DFS를 쓸 수 있는지 세면 된다!

1012 유기농 배추

위치: 0행 0열

⇒ dfs 탐색!

1	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
0	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
0	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**
0	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**
0	**0**	**1**	**1**	**0**	**0**	**0**	**1**	**1**	**1**
0	**0**	**0**	**0**	**1**	**0**	**0**	**1**	**1**	**1**

table 배열

0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**

visit 배열

answer: 1

1012 유기농 배추

위치: 0행 1열

⇒ pass

1	1	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0
0	0	1	1	0	0	0	1	1	1
0	0	0	0	1	0	0	1	1	1

table 배열

1	1	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

visit 배열

answer: 1

이 과정을 반복하면 됩니다!

1012 유기농 배추

위치: 2행 4열

⇒ dfs 탐색!

1	1	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0
0	0	1	1	0	0	0	1	1	1
0	0	0	0	1	0	0	1	1	1

table 배열

1	1	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

visit 배열

answer: 2

1012 유기농 배추

위치: 5행 9열

⇒ pass

⇒ **종료**

1	1	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0
0	0	1	1	0	0	0	1	1	1
0	0	0	0	1	0	0	1	1	1

table 배열

1	1	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0
0	0	1	1	0	0	0	1	1	1
0	0	0	0	1	0	0	1	1	1

visit 배열

answer: 5

유기농배추 코드

```cpp
int t, n, m, k;
int table[55][55];
int visit[55][55];
int an = 0;
int dy[4] = { 1, -1, 0,0 }; // 방향 배열
int dx[4] = { 0,0,1,-1 }; // 방향 배열

void bfs(int y, int x) {
    visit[y][x] = 1;
    for (int i = 0; i < 4; i++) {
        int xx = x + dx[i];
        int yy =  y + dy[i];
        if (yy >= n || yy < 0 || xx < 0 || xx >= m) // 범위 밖이면
            continue;
        //방문해야하는데 방문하지 않았으면 방문
        if(visit[yy][xx] == 0 && table[yy][xx] == 1)
            bfs(yy, xx);
    }
}
```

```cpp
int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    cout.tie(NULL);
    cin >> t;
    for (int test = 0; test < t; test++) {
        memset(visit, 0, sizeof(visit)); //초기화
        memset(table, 0, sizeof(table));
        an = 0;
        cin >> m >> n >> k;
        for (int i = 0; i < k; i++) {
            int a, b;
            cin >> a >> b;
            table[b][a] = 1;
        }
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
                // 방문해야하는데 방문하지 않았으면
                if (table[i][j] == 1 && visit[i][j] == 0) {
                    bfs(i, j);
                    an++; // 정답 증가
                }
            }
        }

        cout << an << "\n";
    }
    return 0;
}
```

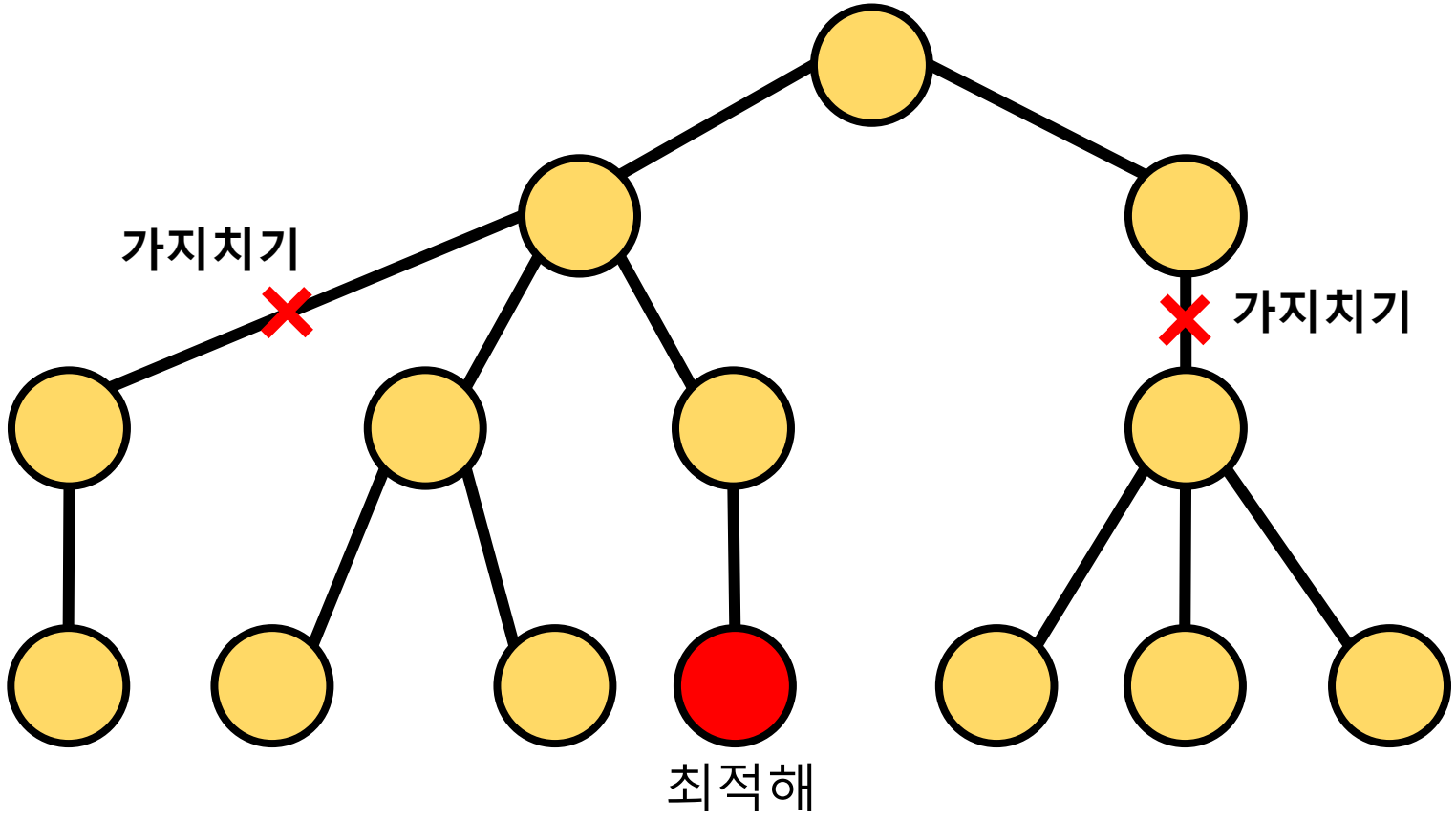
Backtracking 백트래킹

Brute Force: 모든 경우의 수를 탐색하는 알고리즘

Backtracking: **조건을 만족하는** 모든 경우의 수를 탐색하는 알고리즘

탐색을 하다가 조건에 해당하지 않으면 가지치기!

Backtracking 백트래킹



답이 될 가능성이 없는 해는 가지치기

Backtracking 백트래킹 – skeleton code

```
// 루트는 depth가 0
void back(int depth) {
    if (depth == n) {
        if (IsOK()) {//최적해인지 검사
            Dosomething();
        }
        return;
    }
    for (int i = 0; i <= 1; i++) {
        solution[depth] = i;
        back(depth + 1); //자식노드로 이동
    }
}
```

방법1: 해를 만들고 **최적해인지 검사**

```
// 루트는 depth가 0
void back(int depth) {
    if (depth == n) {
        Dosomething();
        return;
    }
    for (int i = 0; i <= 1; i++) {
        if (IsOK()) { //해일 가능성이 있는지 검사
            solution[depth] = i;
            back(depth + 1); //자식노드로 이동
        }
    }
}
```

방법2: 가능성이 있는 해인지 매번 검사 → **가지치기**

자세한 건 문제를 풀면서 알아봅시다!

Backtracking 문제풀이

백준 1182
부분수열의 합

1182 부분수열의 합

문제: 부분수열의 합이 S가 되는 경우의 수를 구하기

■ **예**를 통해서 알아봅시다

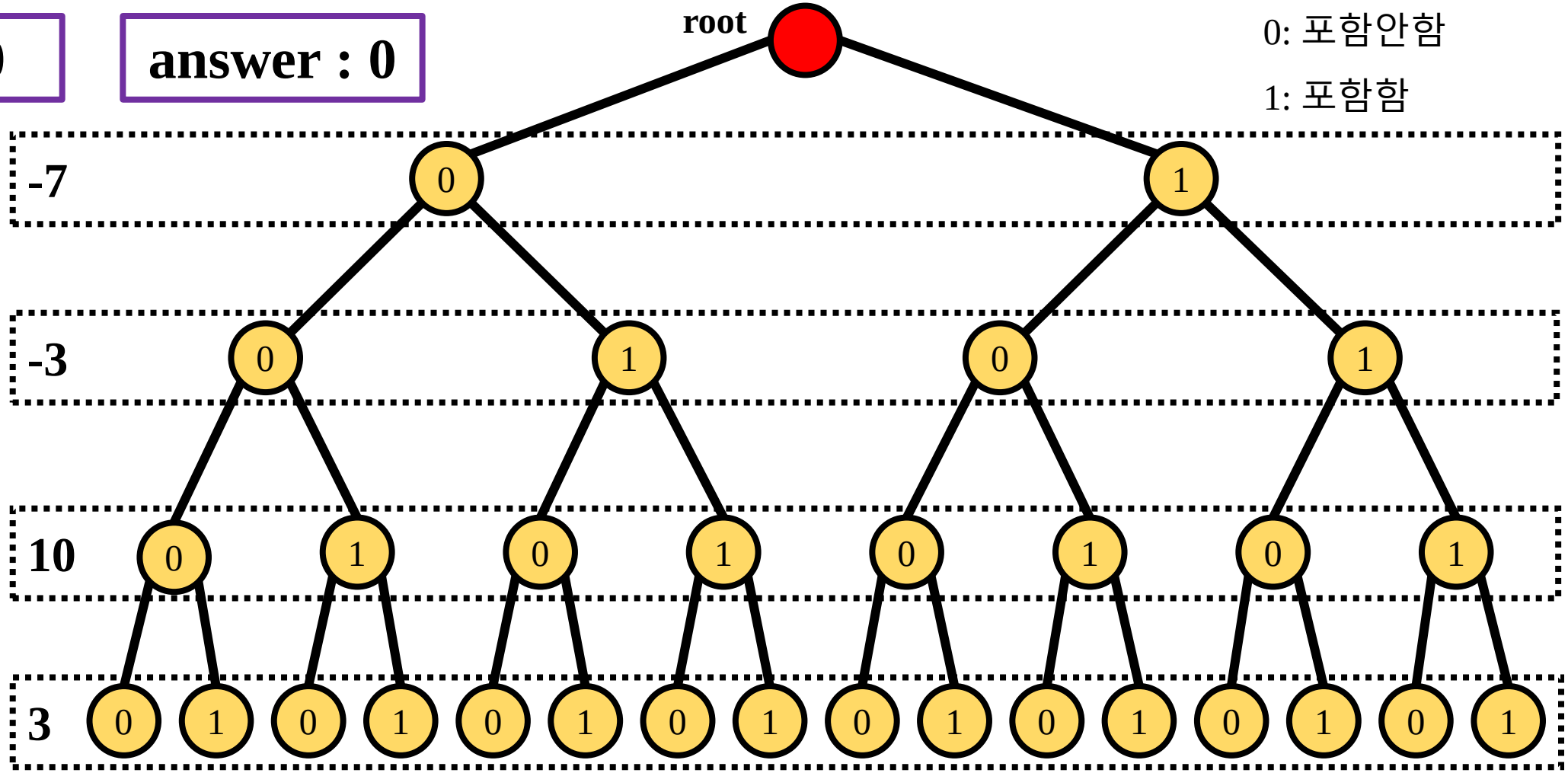
수열이 **{ -7, -3, 10, 3}** 일때 **합이 0**이 되는 경우 찾기

답: 2 ({ -7, -3, 10}, {-3, 3})

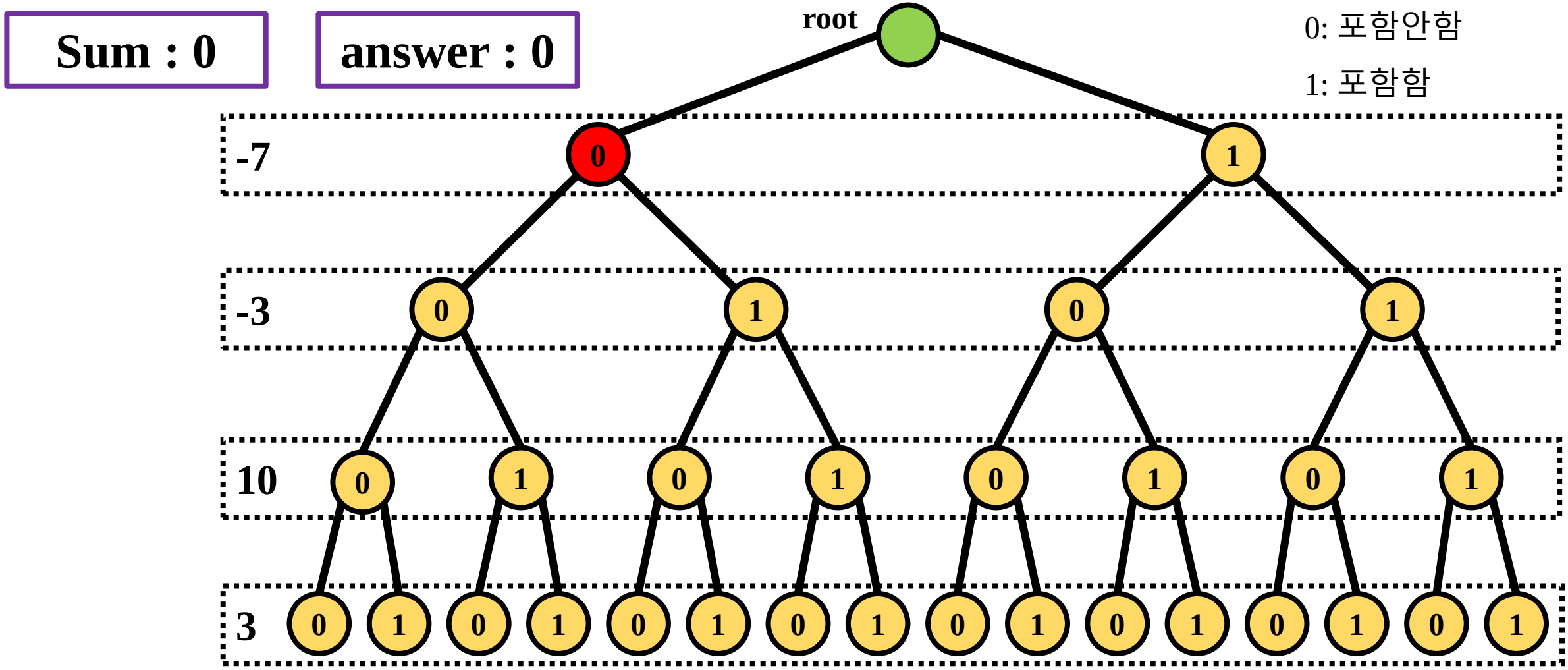
1182 부분수열의 합

Sum : 0

answer : 0



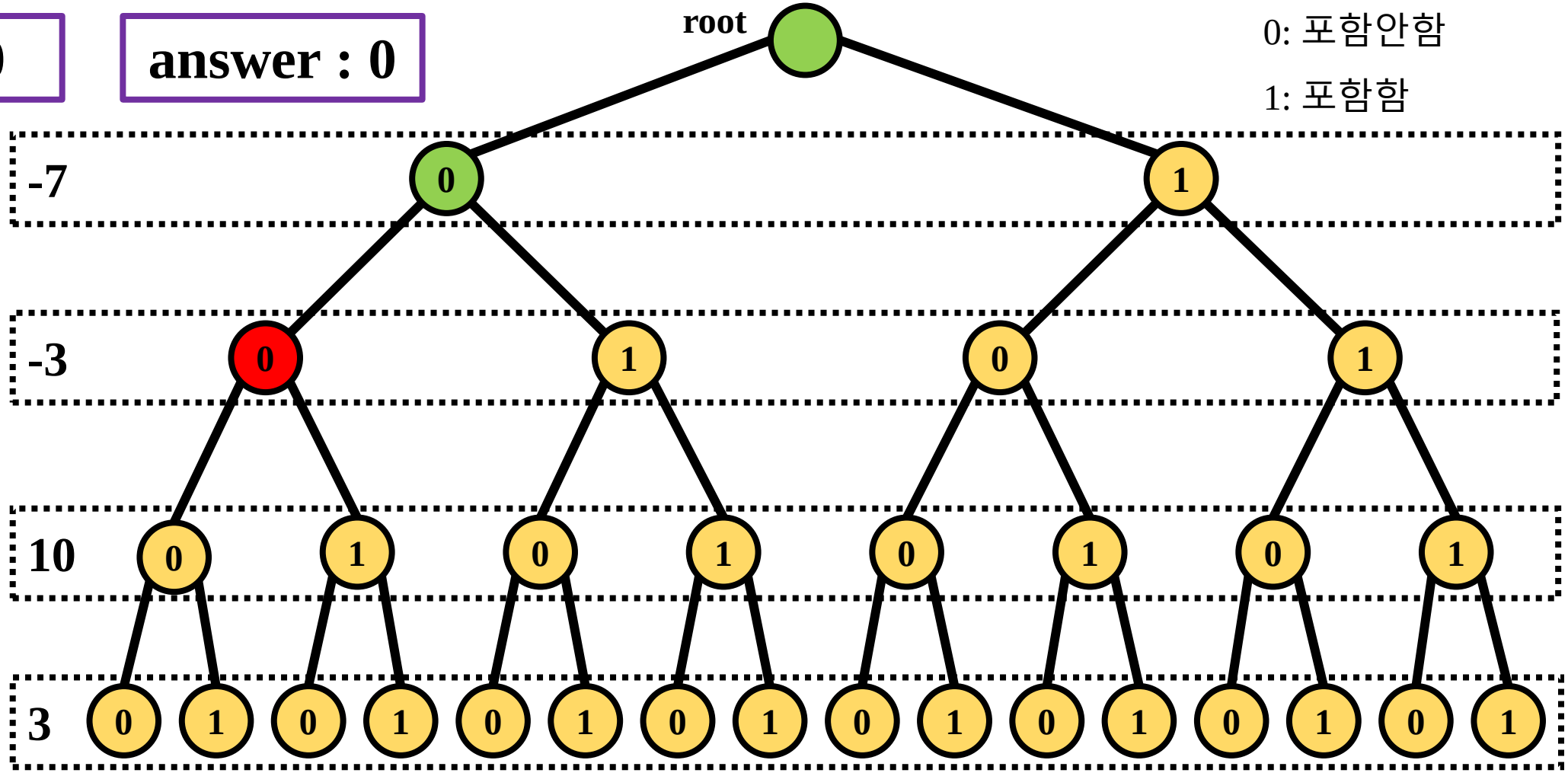
1182 부분수열의 합



1182 부분수열의 합

Sum : 0

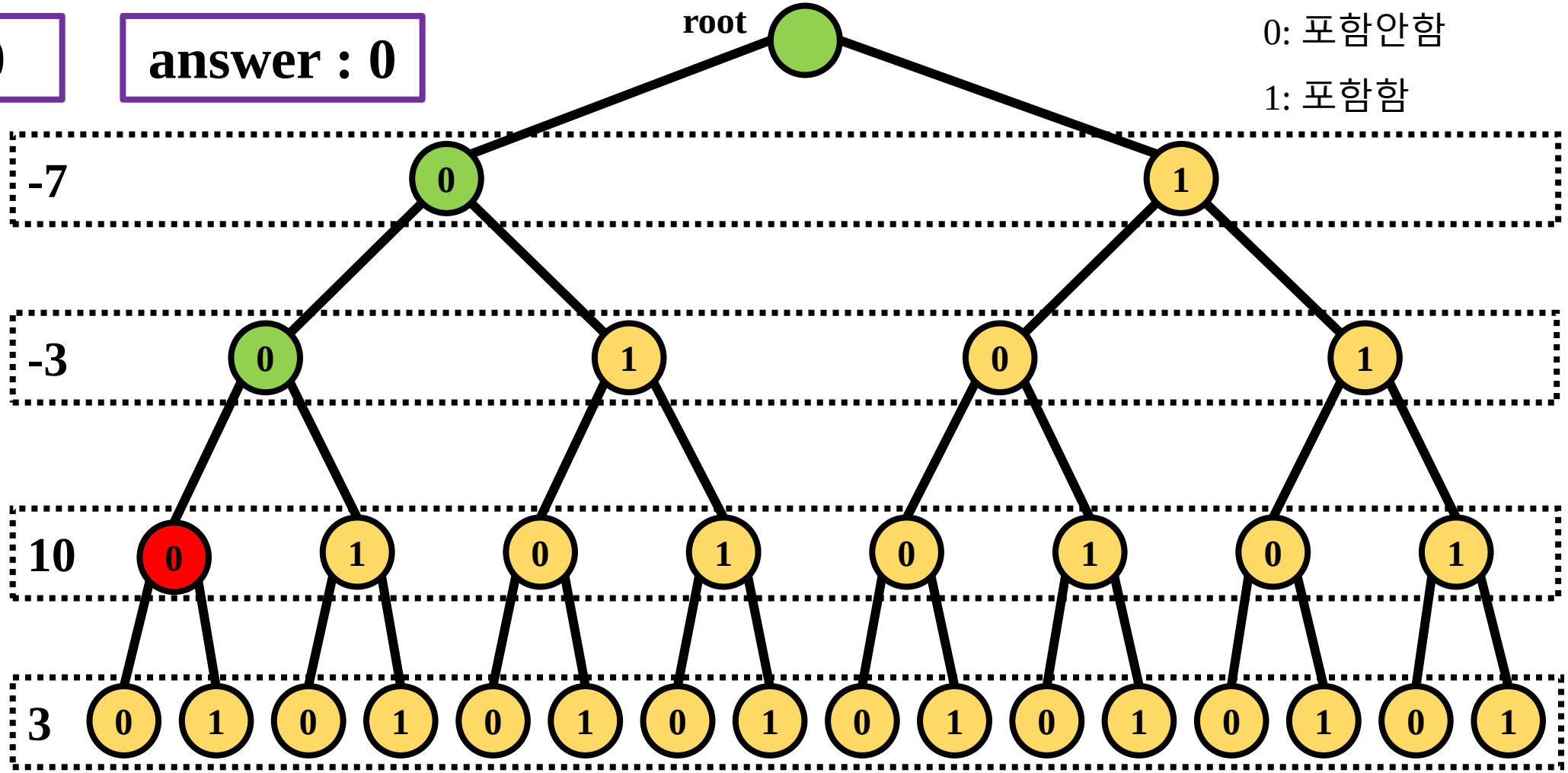
answer : 0



1182 부분수열의 합

Sum : 0

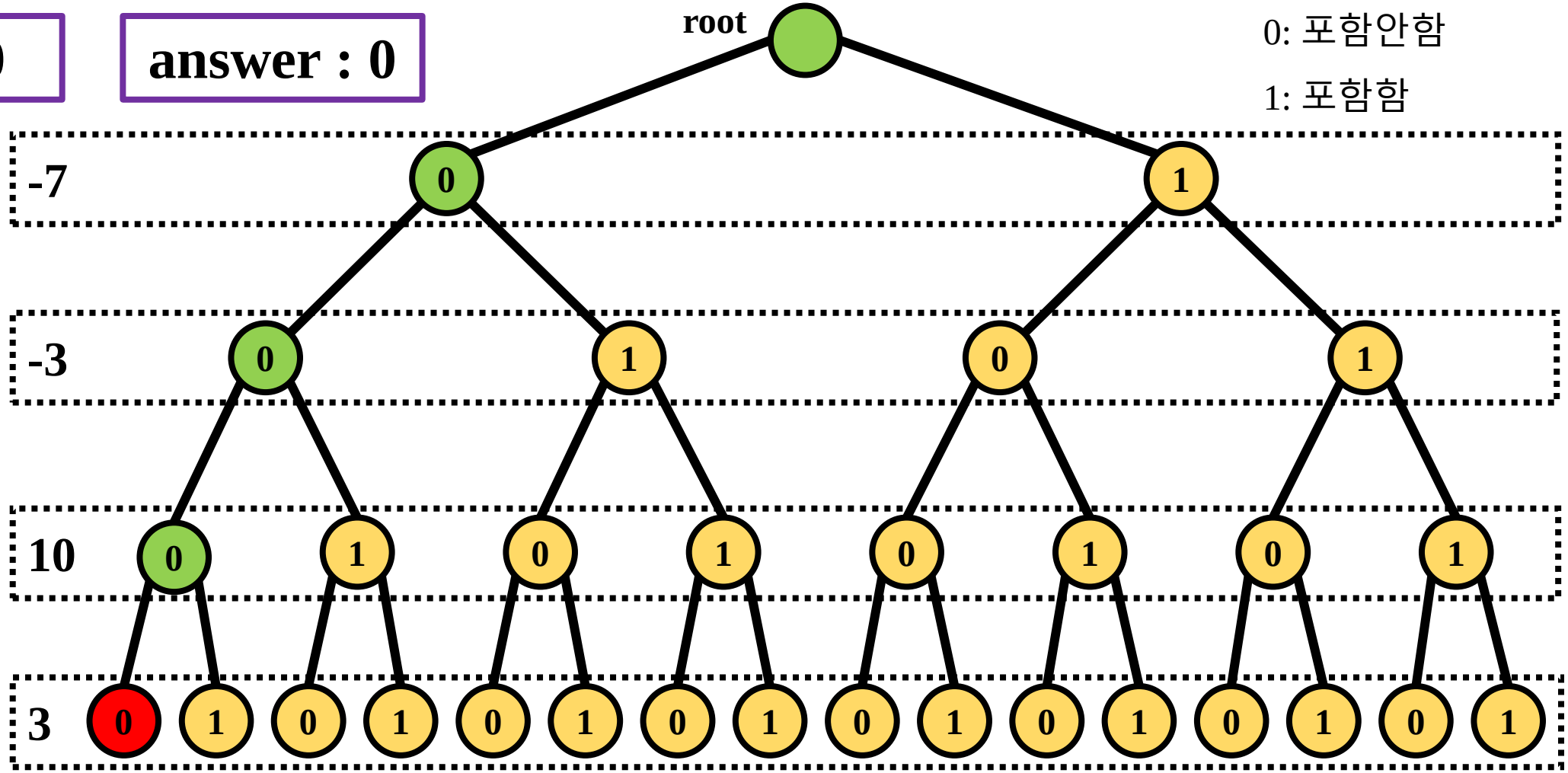
answer : 0



1182 부분수열의 합

Sum : 0

answer : 0

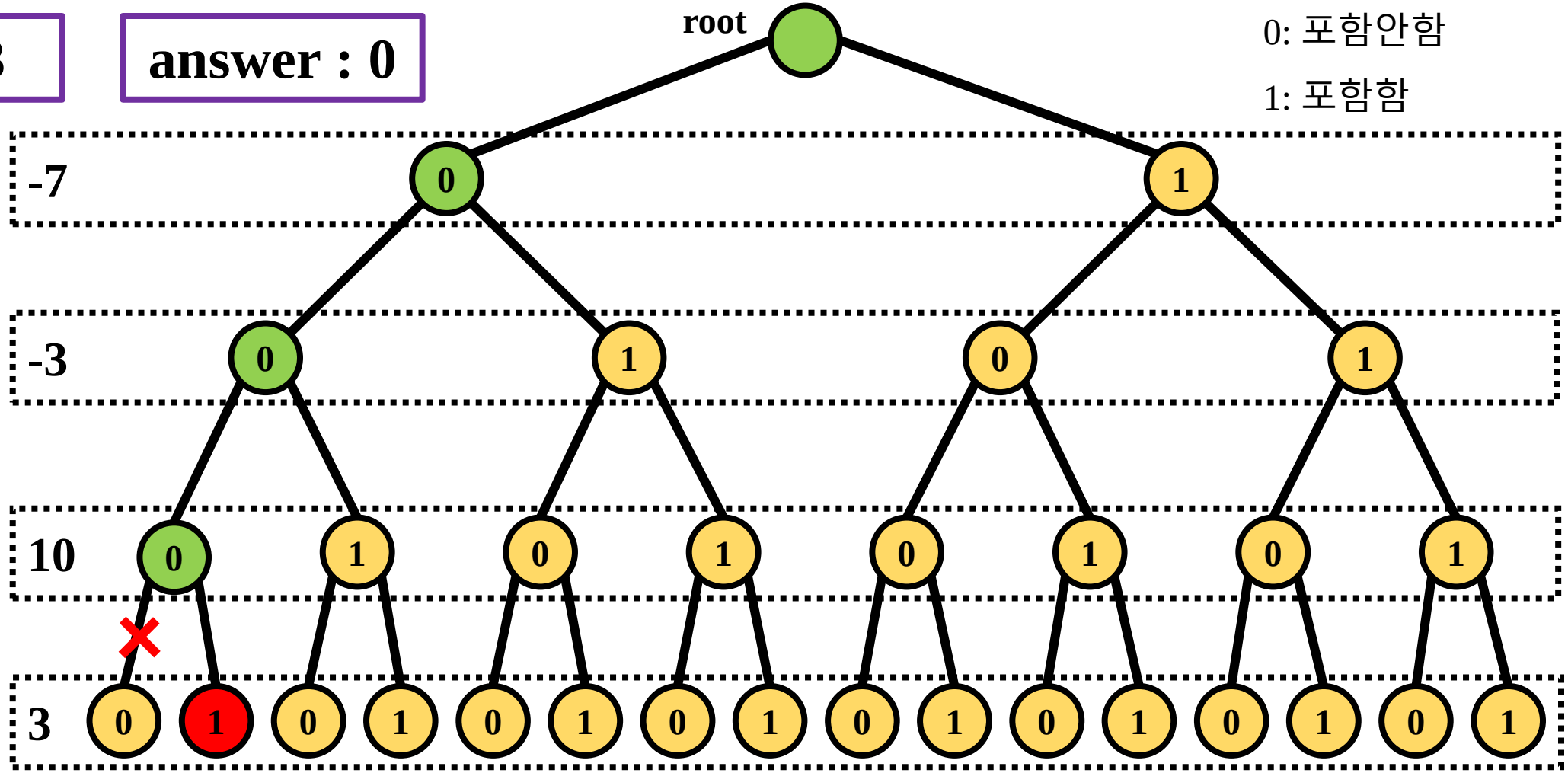


Sum값은 0이지만 **크기가 0**이라 해당 X

1182 부분수열의 합

Sum : 3

answer : 0

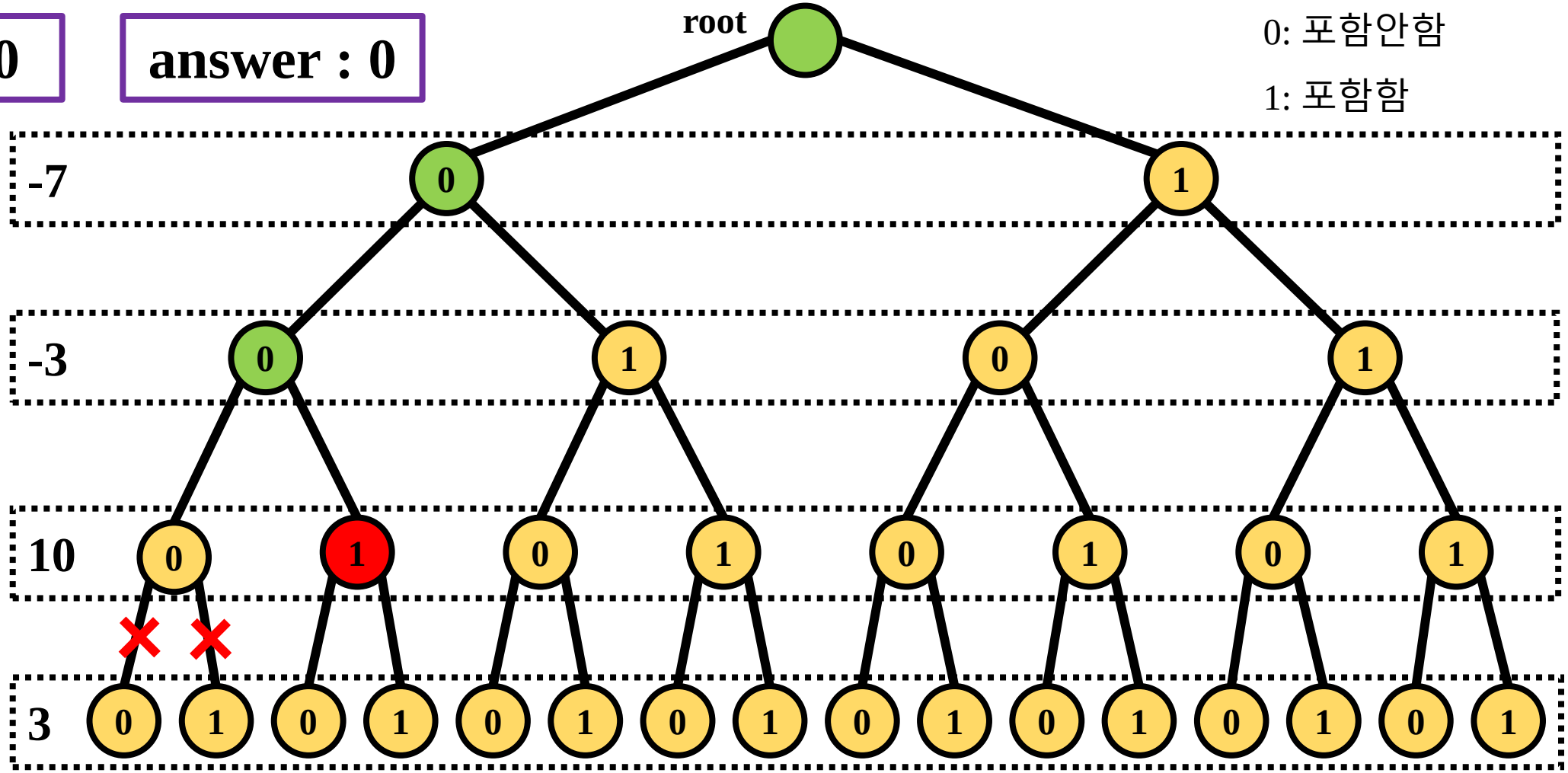


Sum값은 3이라 해당 X

1182 부분수열의 합

Sum : 10

answer : 0



Sum값은 3이라 해당 X

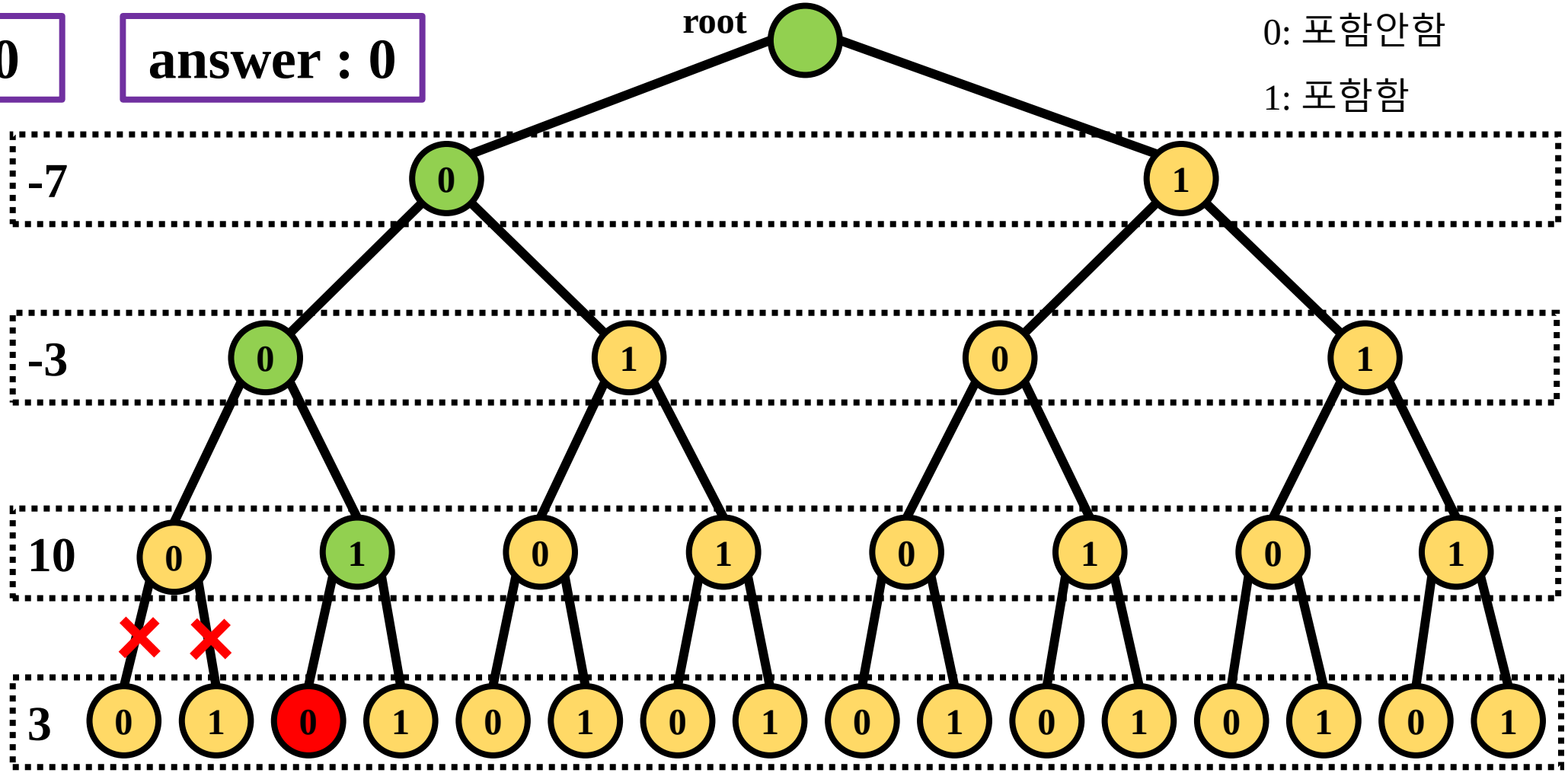
1182 부분수열의 합

Sum : 10

answer : 0

0: 포함안함

1: 포함함



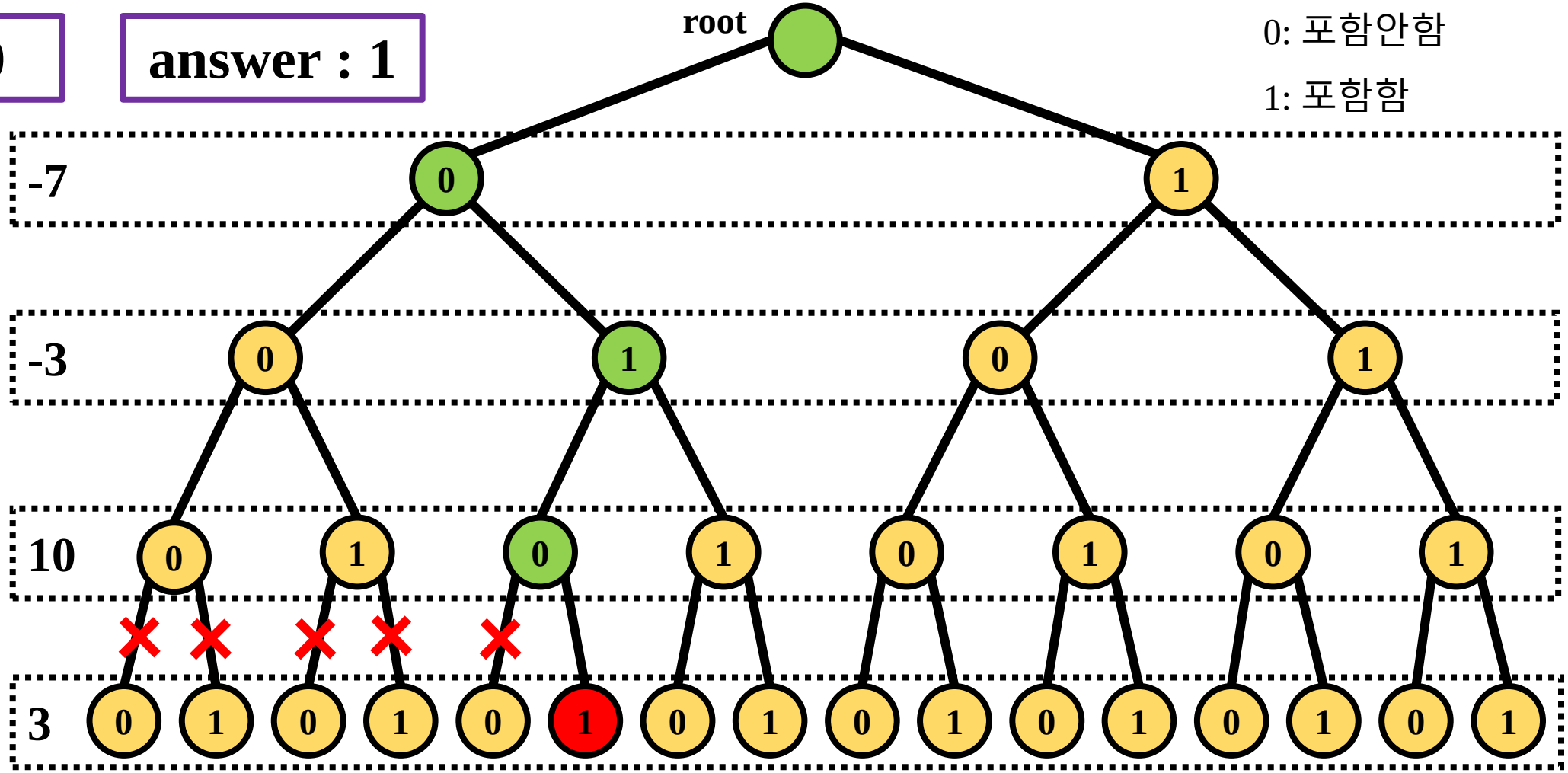
이 과정을 반복!!

Sum값은 10이라 해당 X

1182 부분수열의 합

Sum : 0

answer : 1

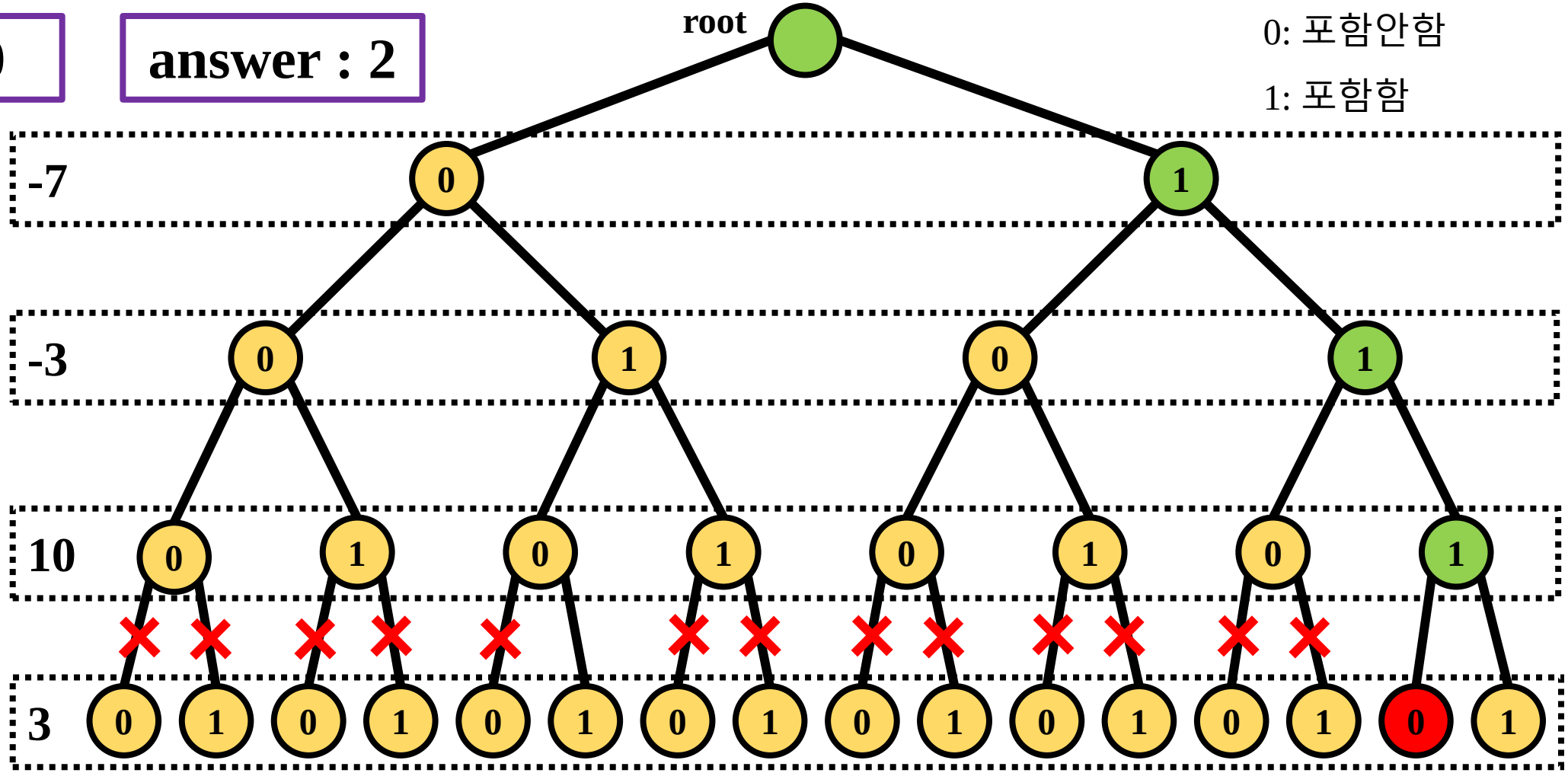


Sum값은 0 => **answer 증가!!**

1182 부분수열의 합

Sum : 0

answer : 2

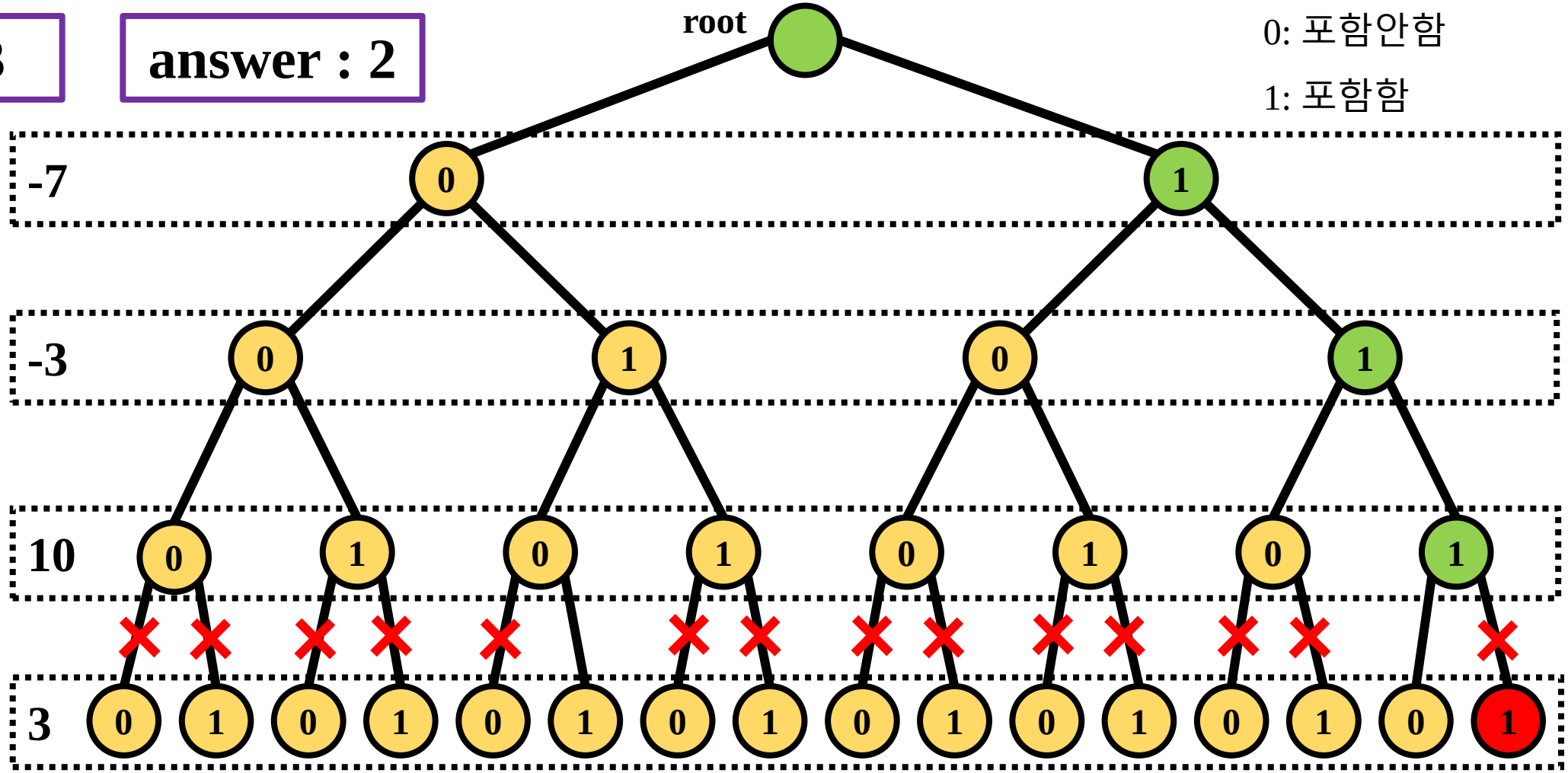


Sum값은 0 => **answer 증가!!**

1182 부분수열의 합

Sum : 3

answer : 2



Sum값은 3이라 해당 X

부분수열의 합 코드

```cpp
#include <iostream>
using namespace std;

int n, s;
int num[25];
int an = 0;

void back(int depth, int sum) {
    if (depth == n) { // 기저조건에 도달
        if (sum == s) // 합과 같으면
            an++; //정답 증가
        return;
    }
    back(depth + 1, sum); // 숫자 포함안함
    back(depth + 1, sum + num[depth]); // 숫자 포함함
}
```

```cpp
int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    cout.tie(NULL);
    cin >> n >> s;
    for (int i = 0; i < n; i++) {
        cin >> num[i];
    }
    back(0, 0);
    if (s == 0) //크기가 0이면서 답이되는 경우는 s가 0일때
        cout << an - 1;
    else cout << an;
    return 0;
}
```

Backtracking 문제풀이

Backtracking의 정석문제!!

백준 9663
N-Queen

9663 N-Queen

문제: N × N 체스판에 퀸 N개를 놓을 수 있는 경우의 수를 구하기

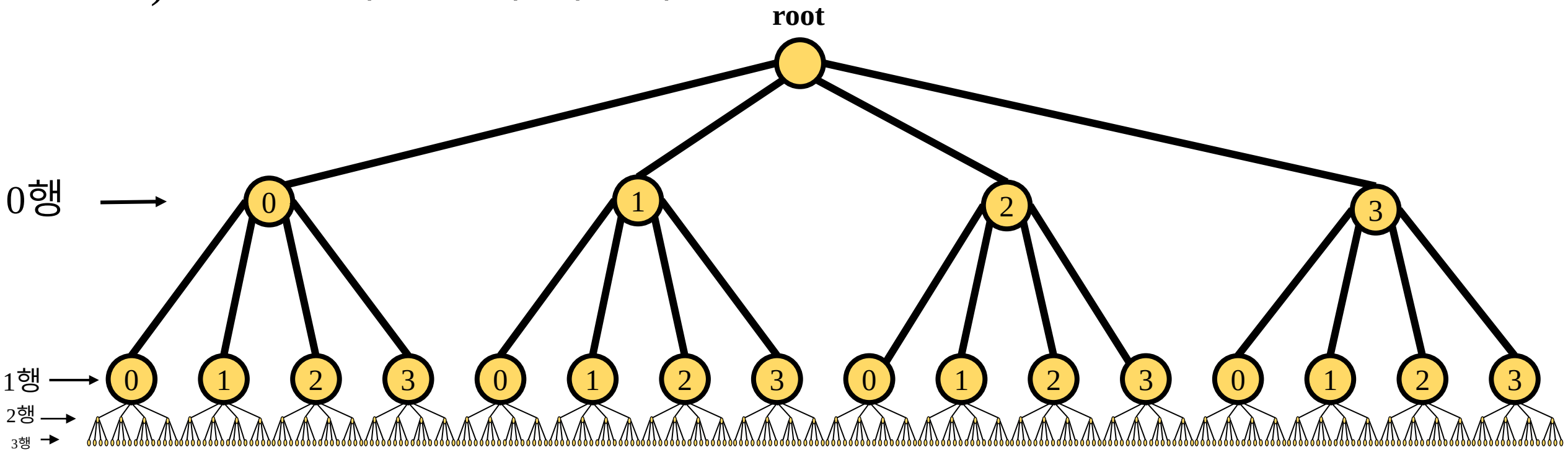
퀸은 가로, 세로, 대각선으로 이동 가능하다

⇒ **하나의** 가로, 세로, 대각선에는 **하나의 퀸**만 존재해야 한다!

⇒ **한 행에 하나의 퀸**만 놓는다 생각하고 재귀함수로 구현하자

9663 N-Queen

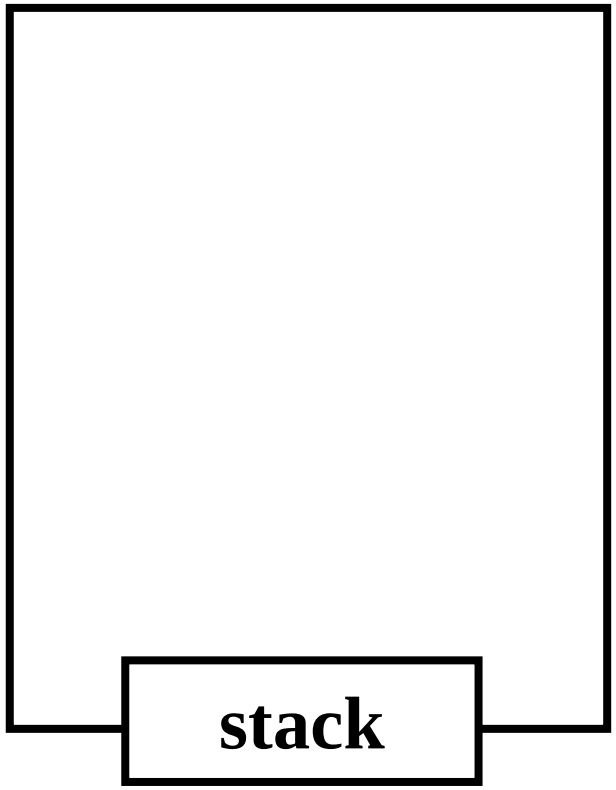
ex) n = 4 일때 답을 구해보자



⇒ n = 4만 해도 경우의 수가 4^4라 너무 많다 트리 그림으로 설명하기엔 쉽지 않다...

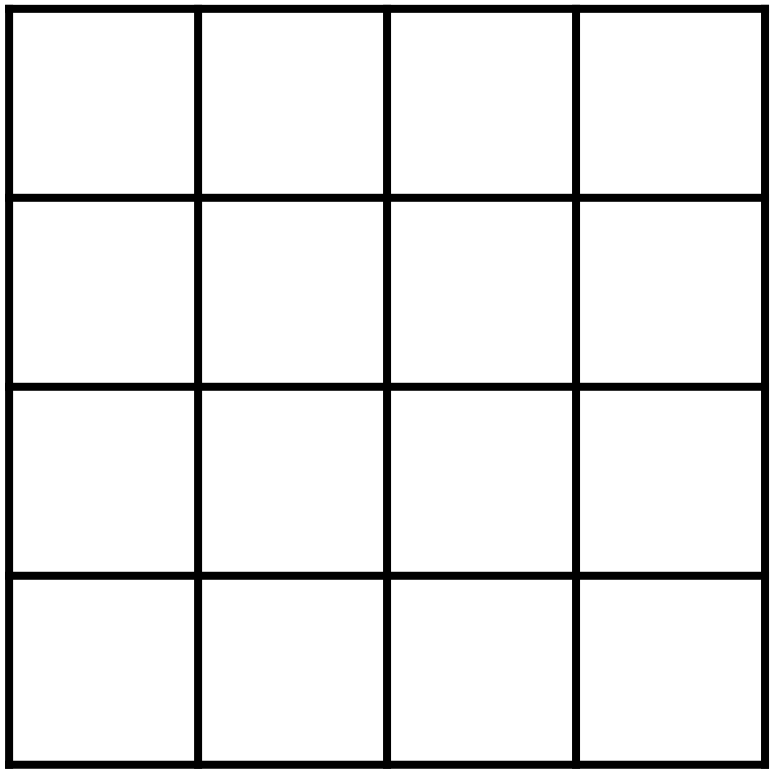
⇒ 해 공간에 도달하기 전에 **가지치기**를 하자!

9663 N-Queen



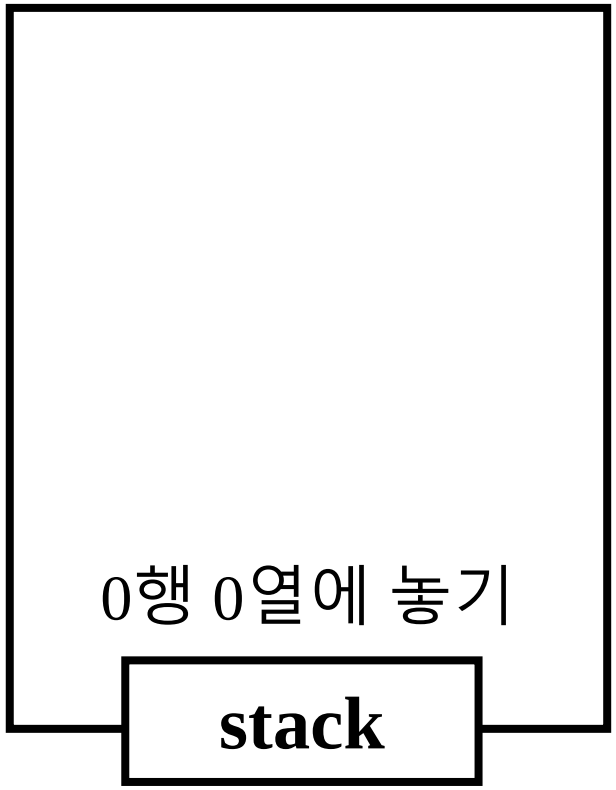
answer: **0**

n = 4 체스판 (X는 퀸)

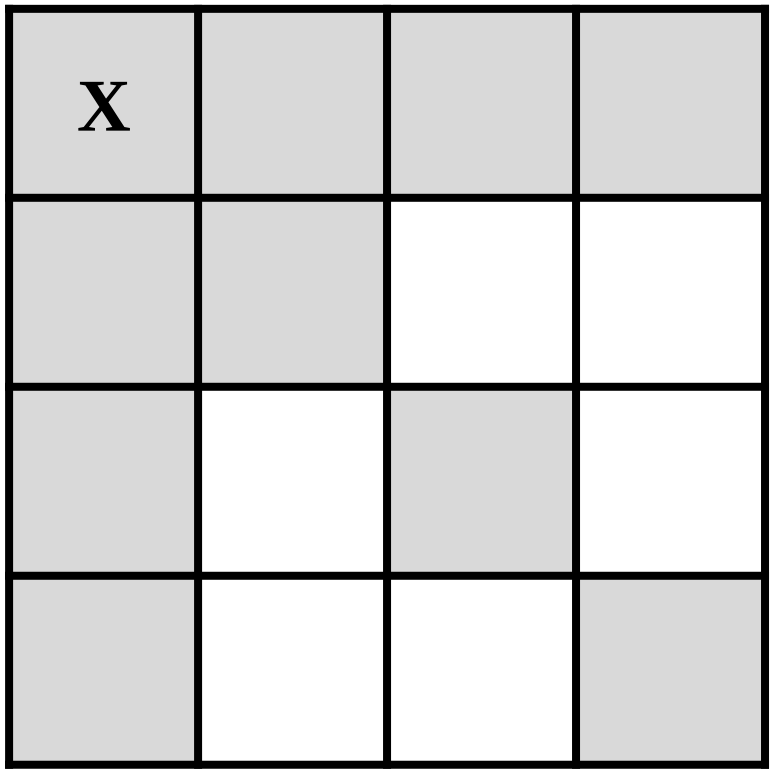


회색 칸은 가지치기해야 하는 경우입니다!

9663 N-Queen

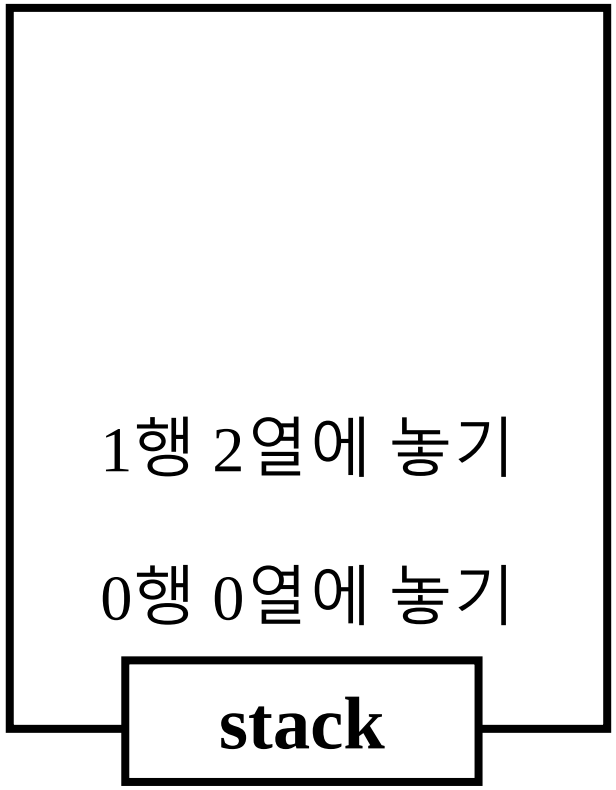


n = 4 체스판 (X는 퀸)



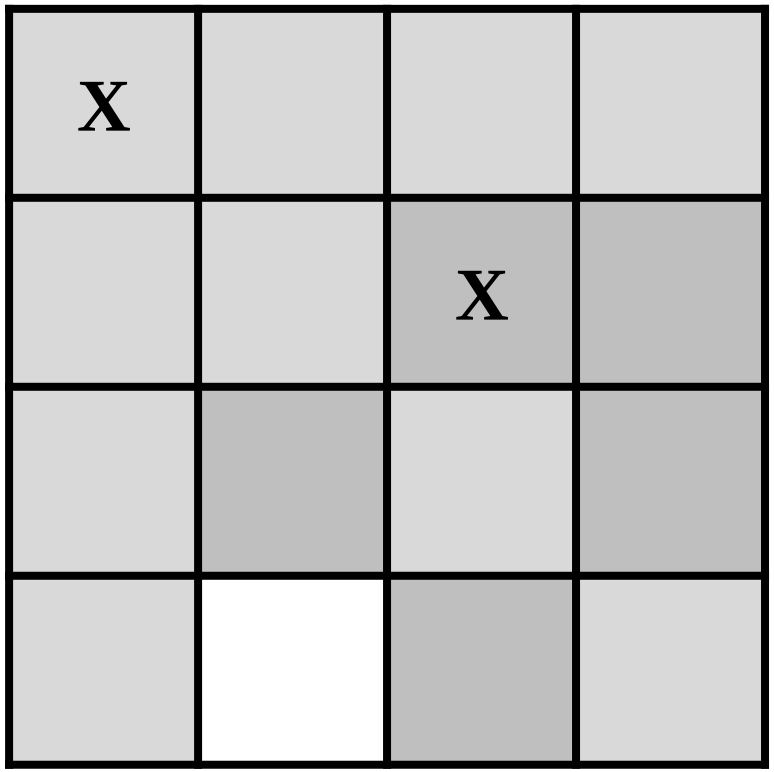
answer: **0**

9663 N-Queen



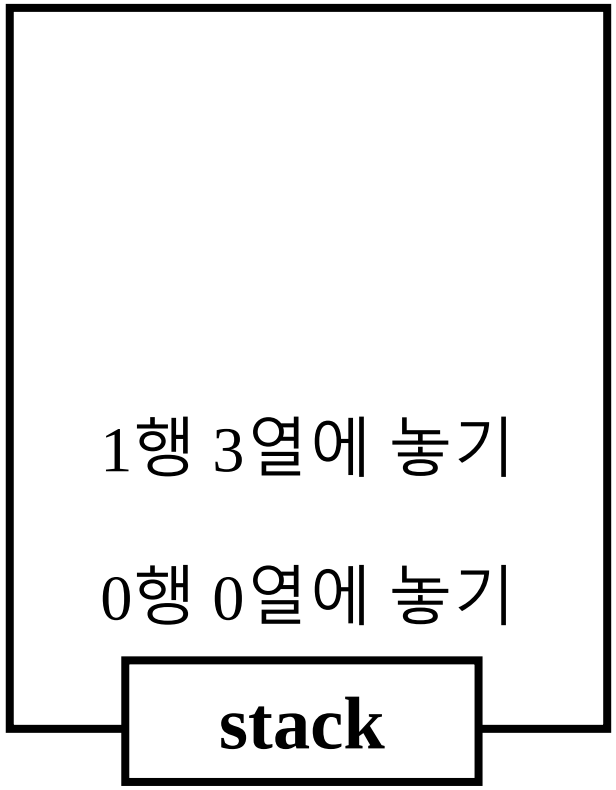
answer: **0**

n = 4 체스판 (X는 퀸)

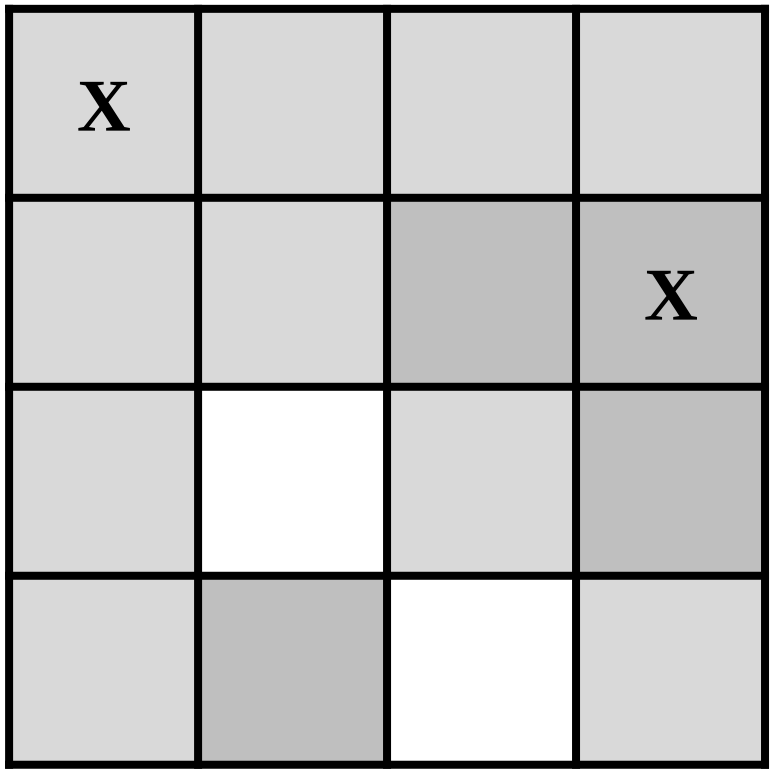


2행에 놓을 칸이 없으므로 **return** → 1행

9663 N-Queen

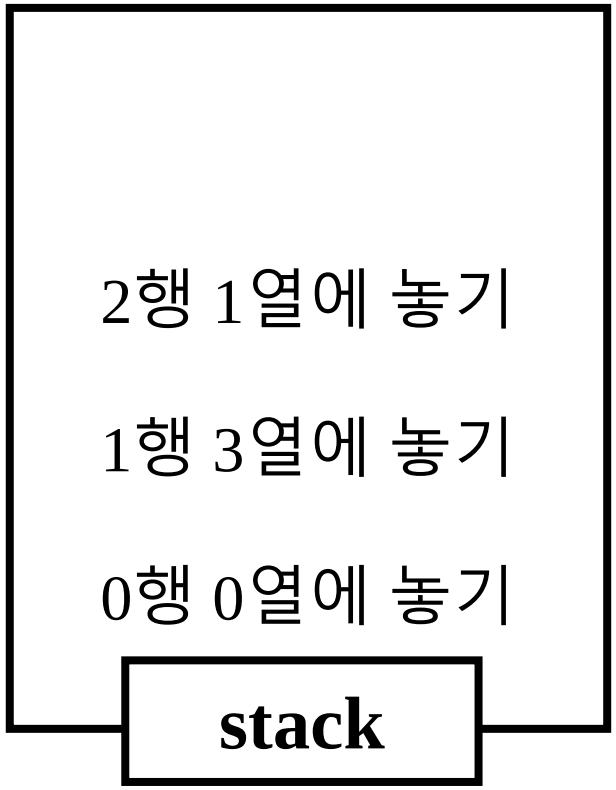


n = 4 체스판(X는 퀸)



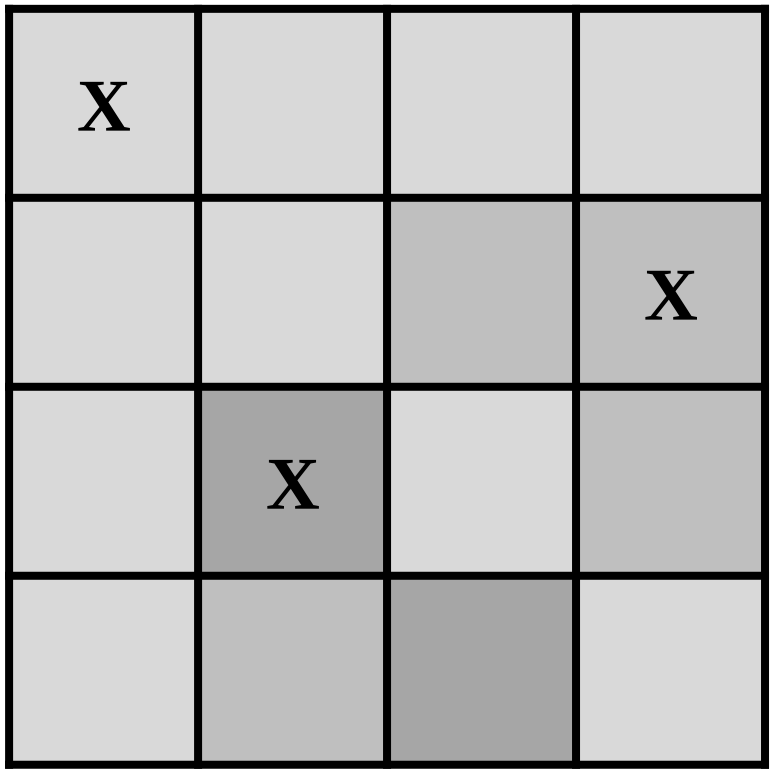
answer: **0**

9663 N-Queen



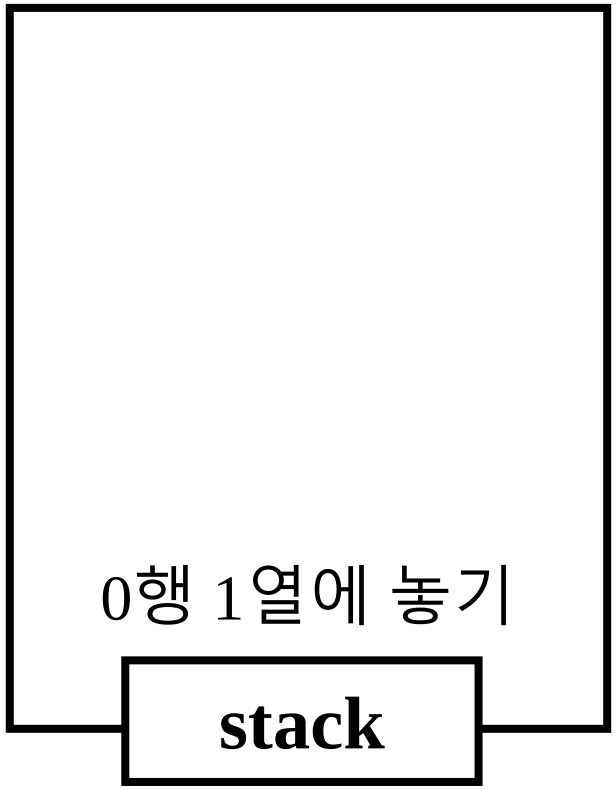
answer: **0**

n = 4 체스판(X는 퀸)

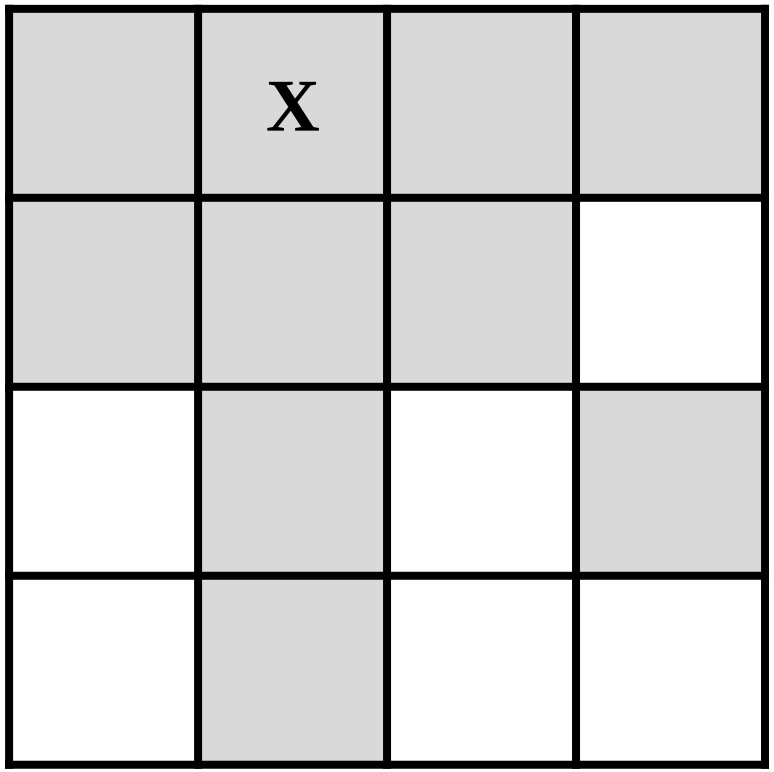


3행에 놓을 칸이 없으므로 **return** → 0행

9663 N-Queen

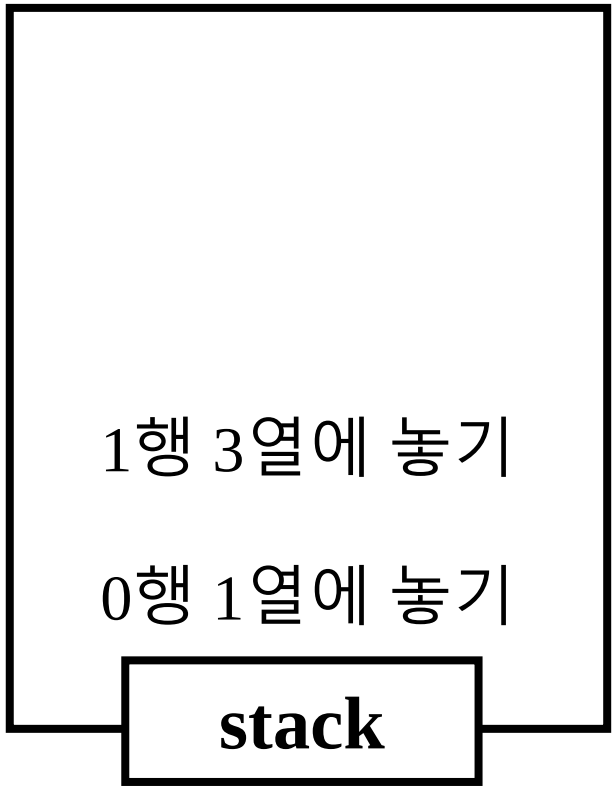


n = 4 체스판 (X는 퀸)

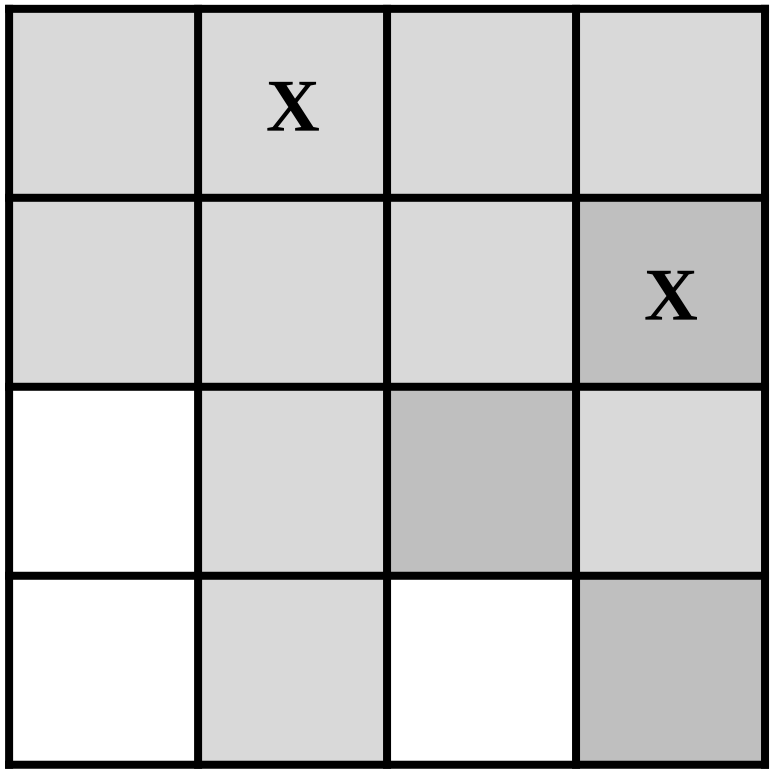


answer: **0**

9663 N-Queen

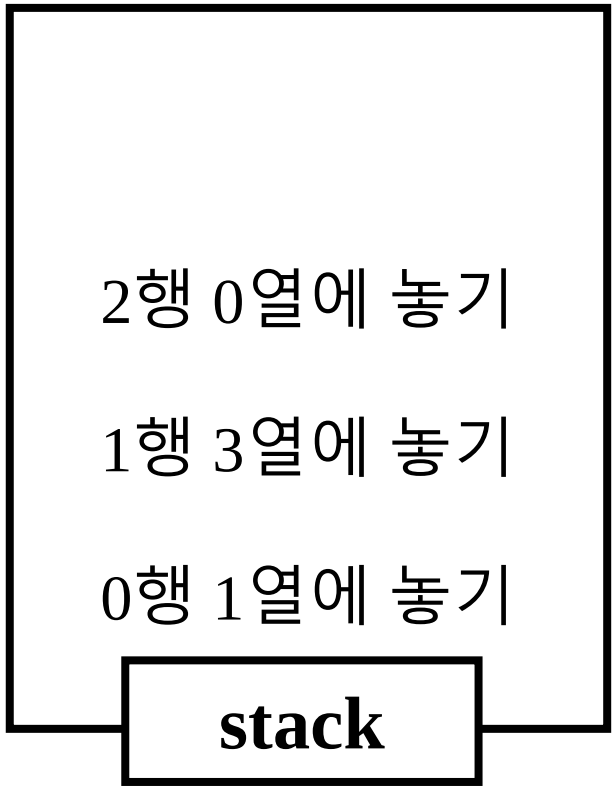


n = 4 체스판 (X는 퀸)



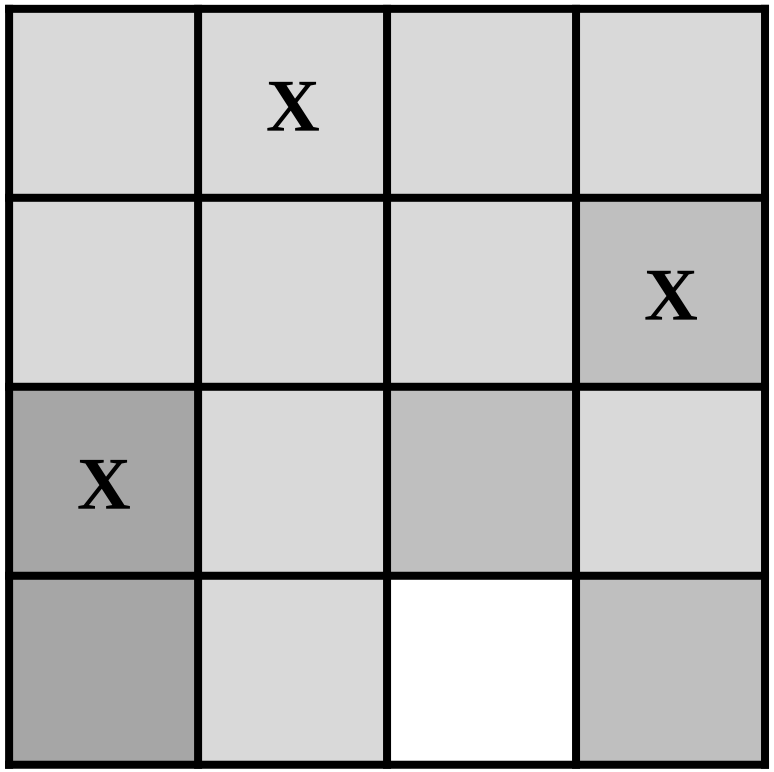
answer: **0**

9663 N-Queen



answer: **0**

n = 4 체스판 (X는 퀸)



9663 N-Queen

3행 2열에 놓기

2행 0열에 놓기

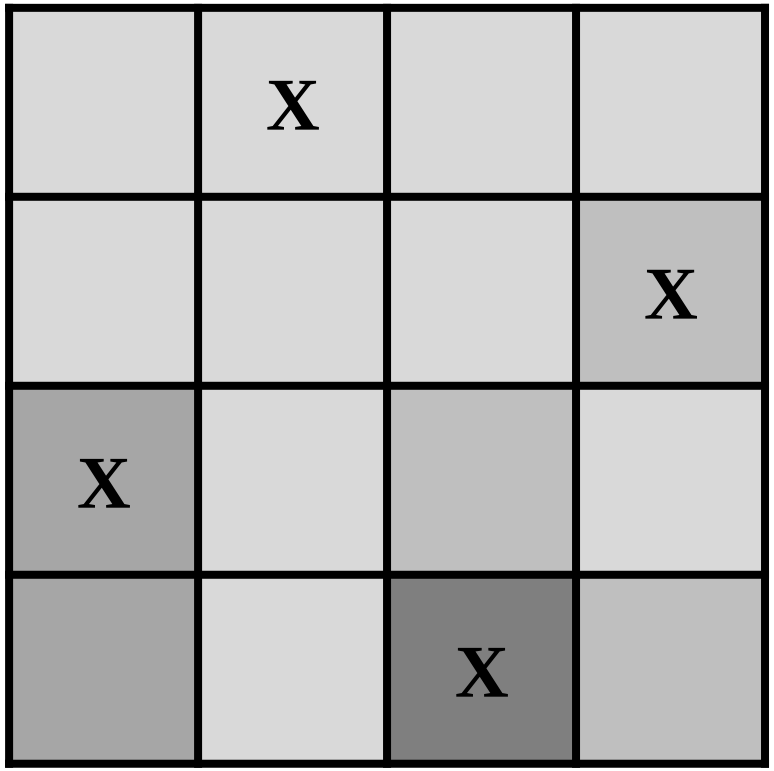
1행 3열에 놓기

0행 1열에 놓기

stack

answer: **1**

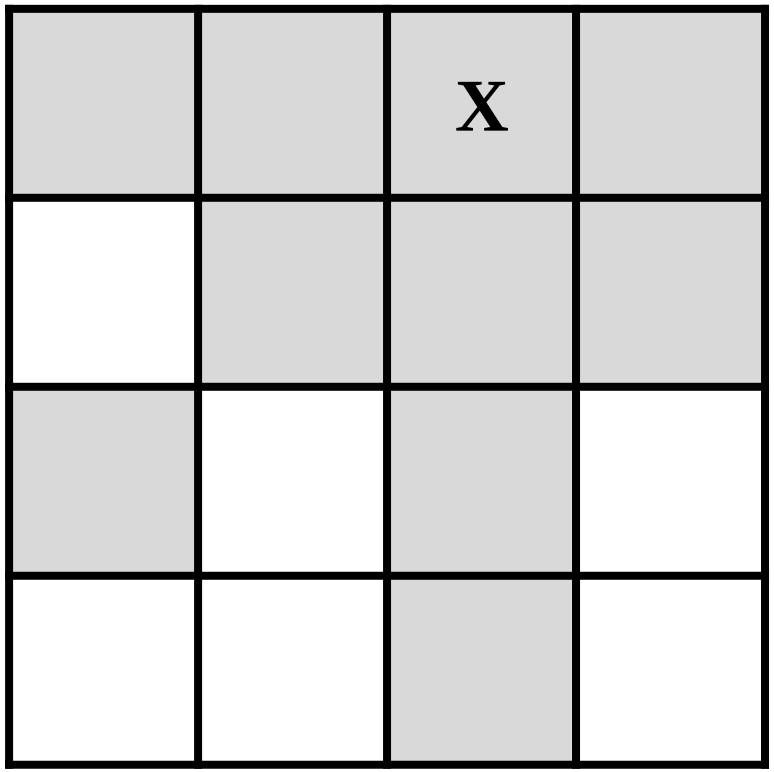
n = 4 체스판 (X는 퀸)



가능한 해이므로 **answer** 증가

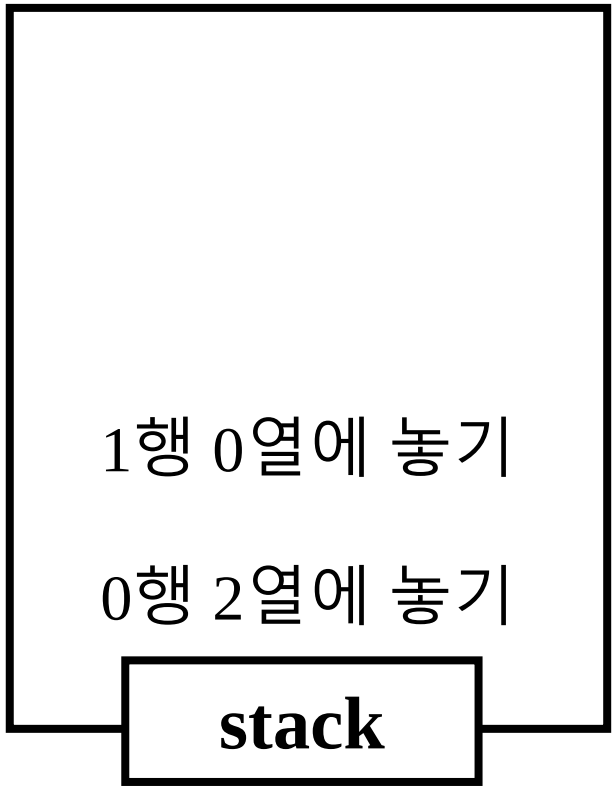
9663 N-Queen

n = 4 체스판 (X는 퀸)

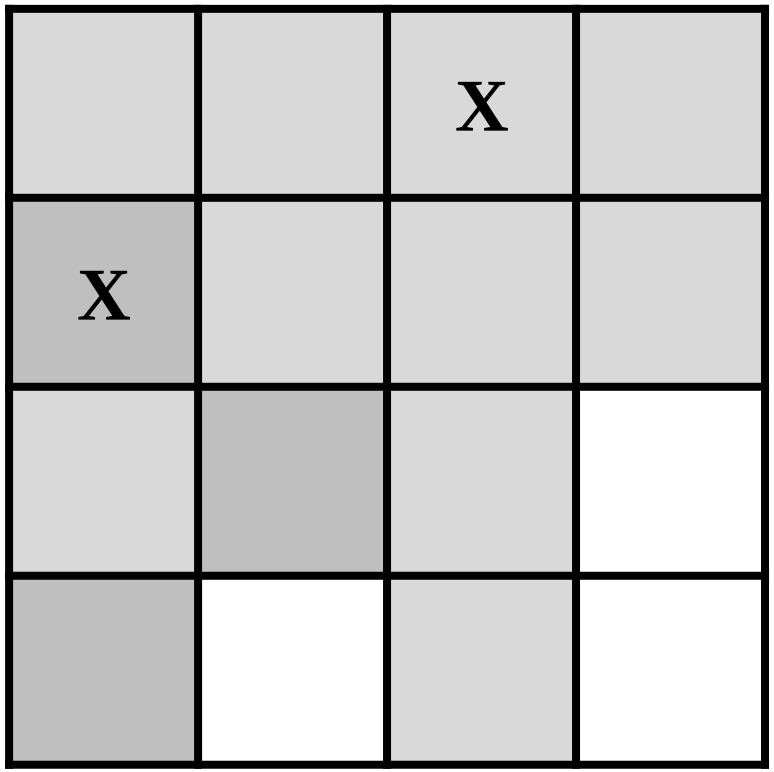


answer: **1**

9663 N-Queen

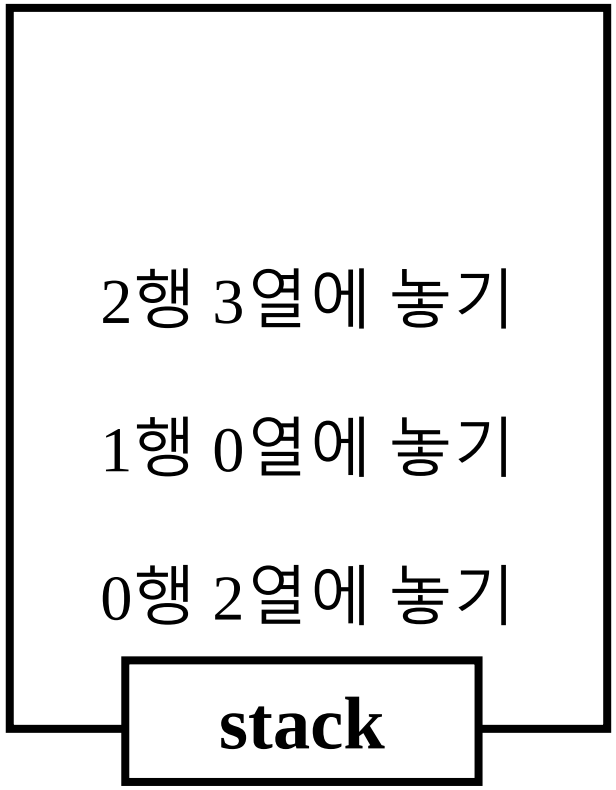


n = 4 체스판 (X는 퀸)

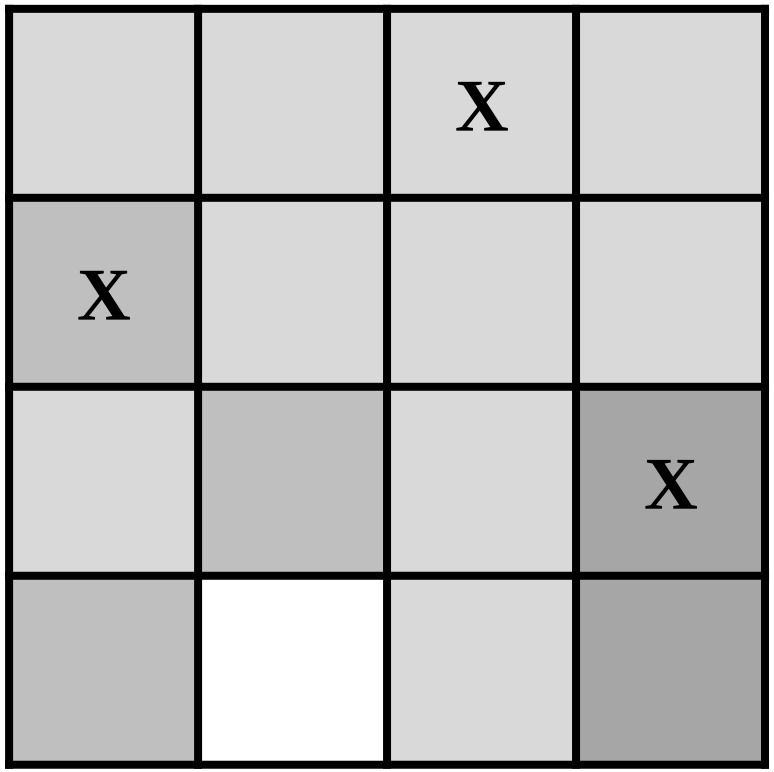


answer: **1**

9663 N-Queen

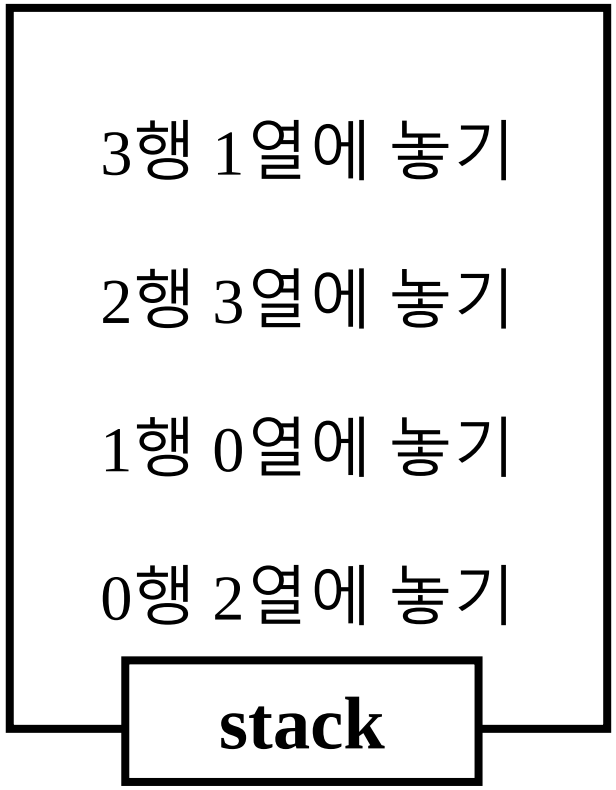


n = 4 체스판 (X는 퀸)



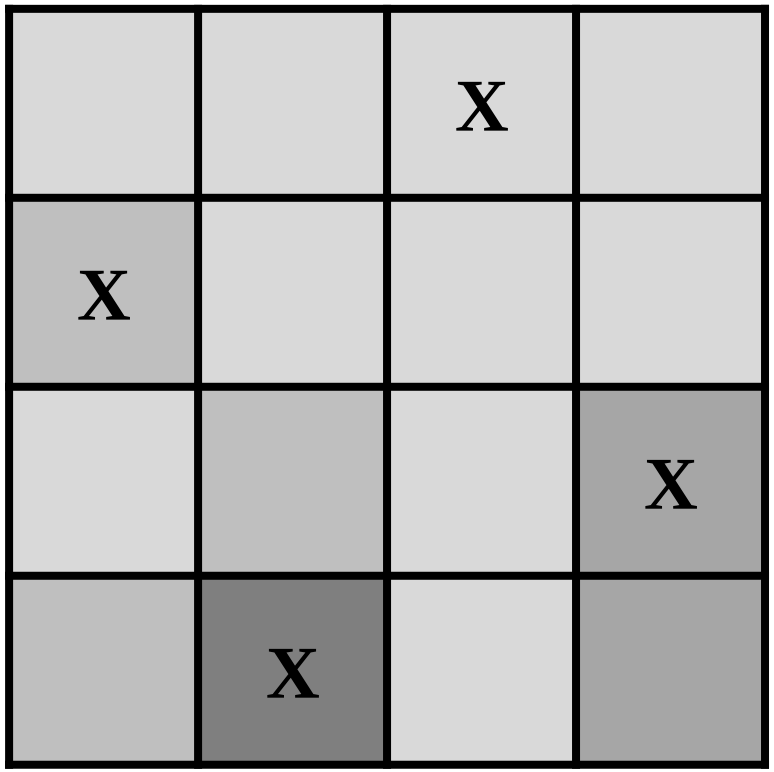
answer: **1**

9663 N-Queen



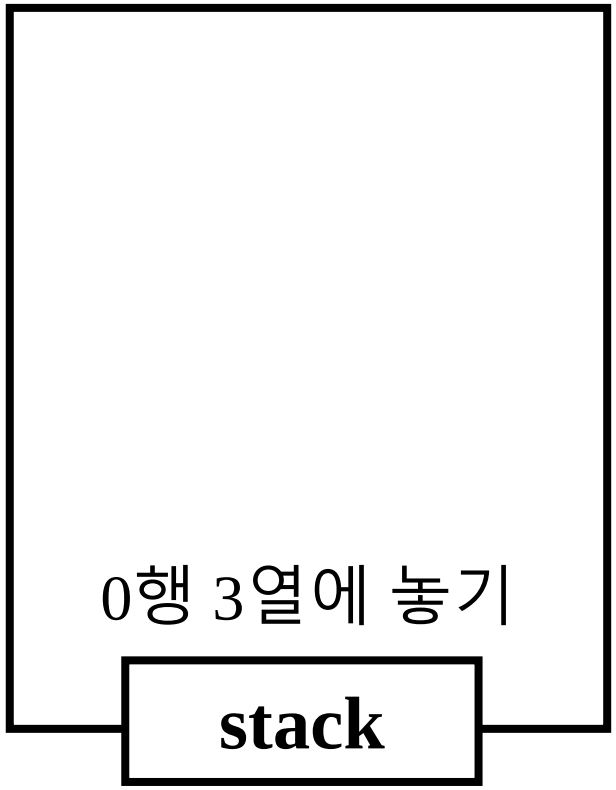
answer: **2**

n = 4 체스판 (X는 퀸)

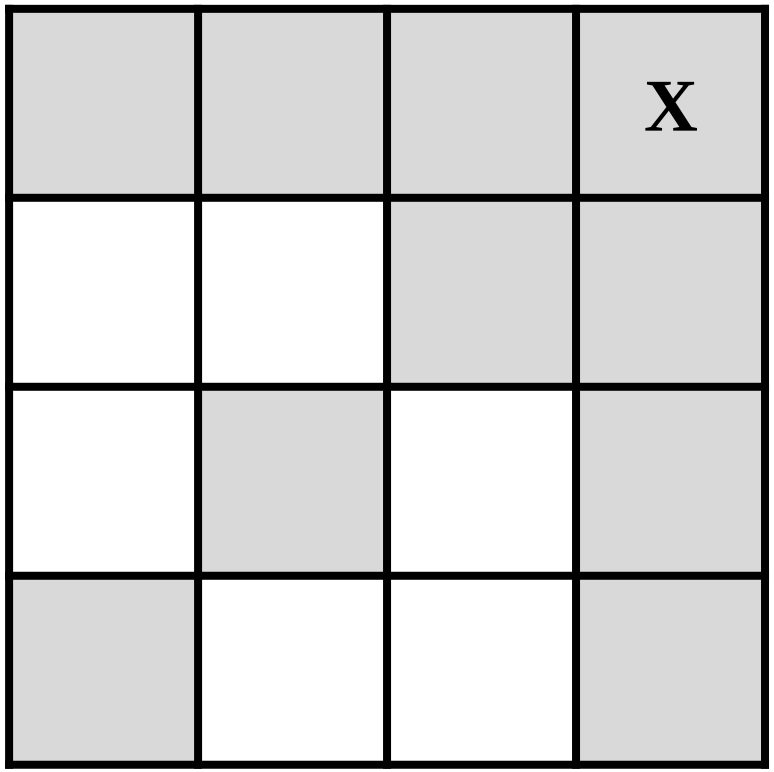


가능한 해이므로 answer 증가

9663 N-Queen

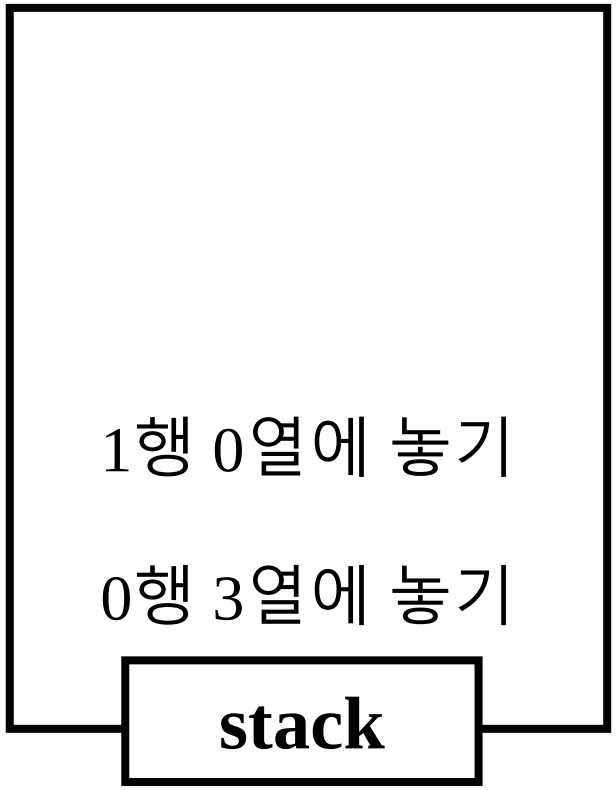


n = 4 체스판 (X는 퀸)

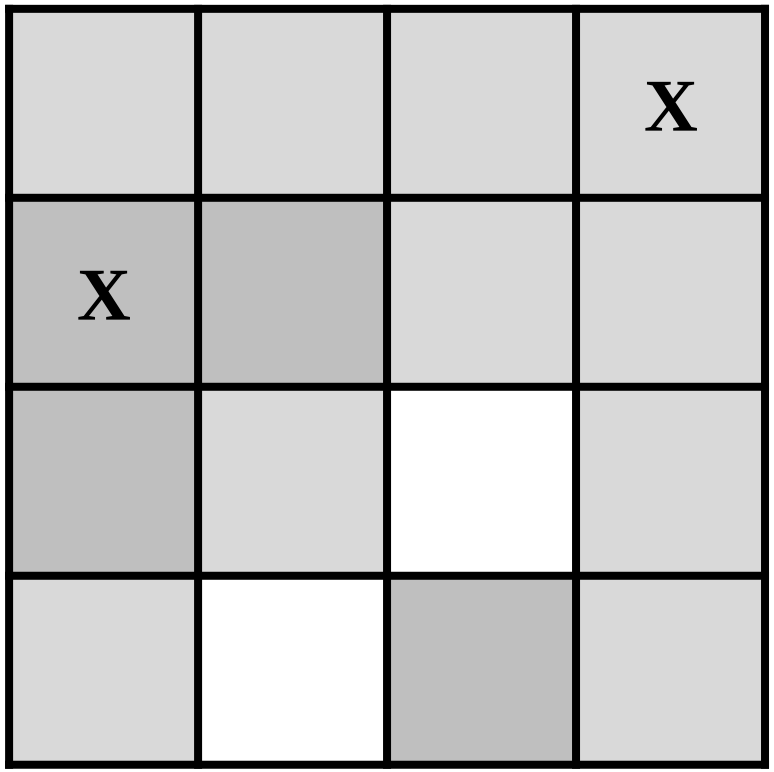


answer: **2**

9663 N-Queen

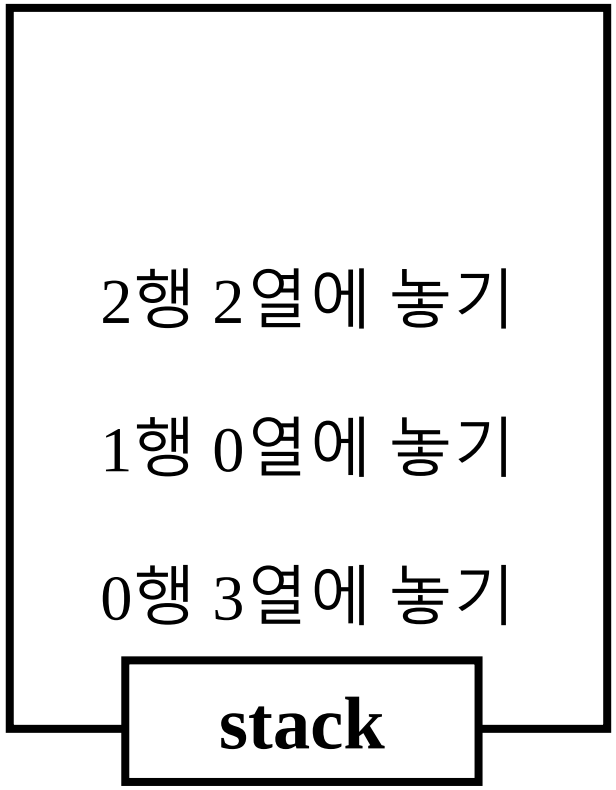


n = 4 체스판 (X는 퀸)



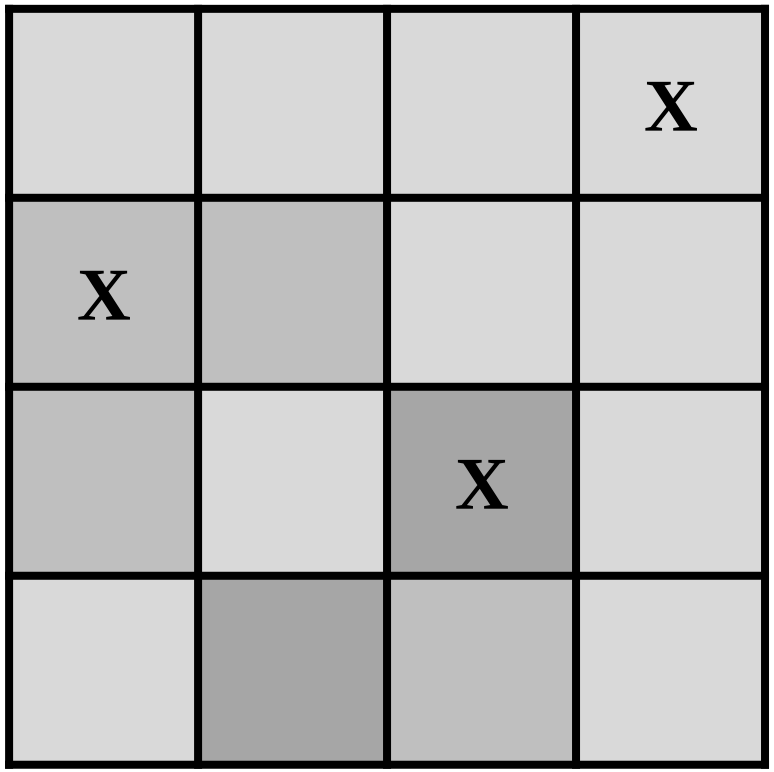
answer: **2**

9663 N-Queen



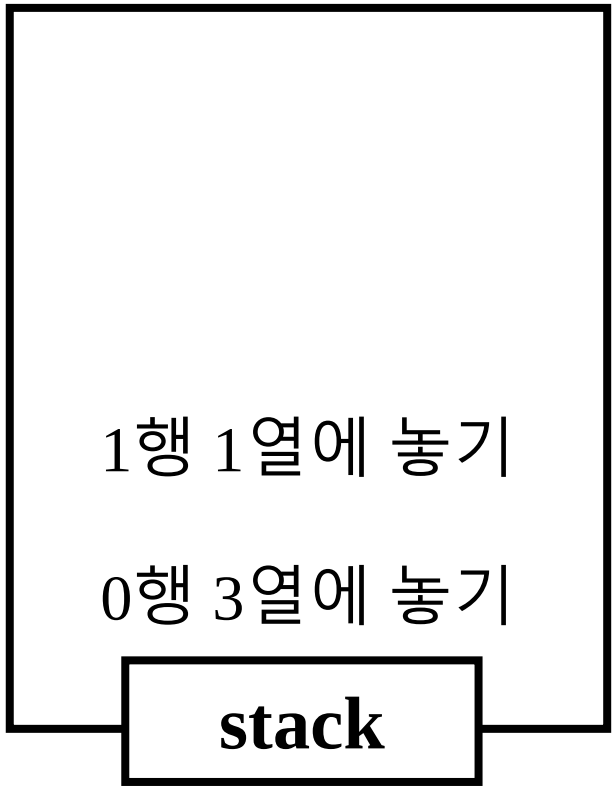
answer: **2**

n = 4 체스판 (X는 퀸)



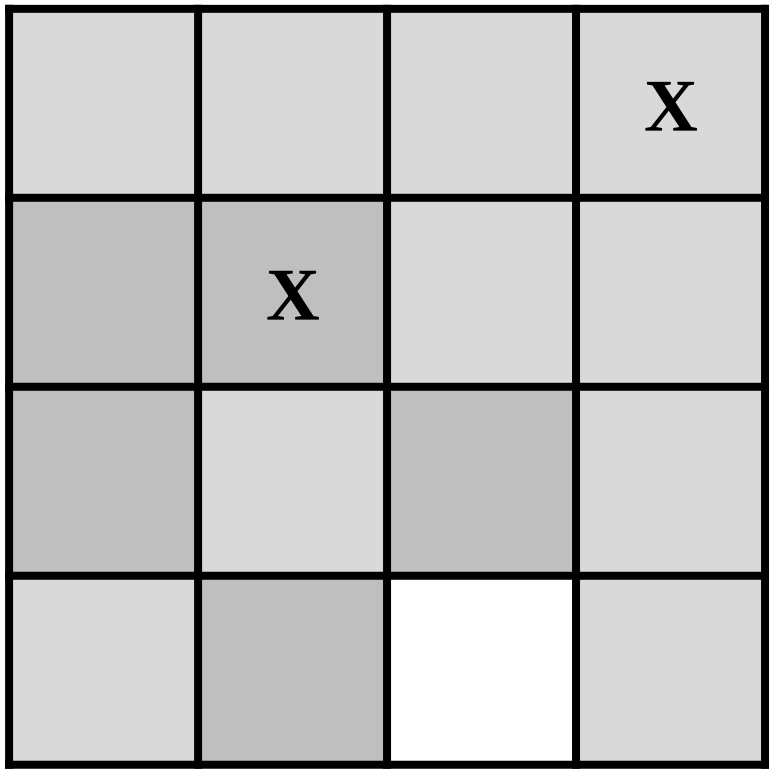
3행에 놓을 칸이 없으므로 **return** → 1행

9663 N-Queen



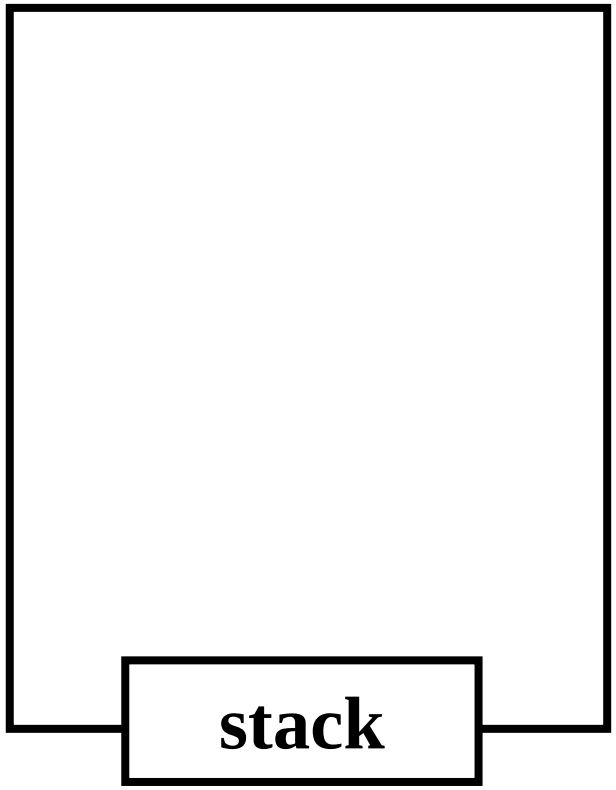
answer: **2**

n = 4 체스판 (X는 퀸)



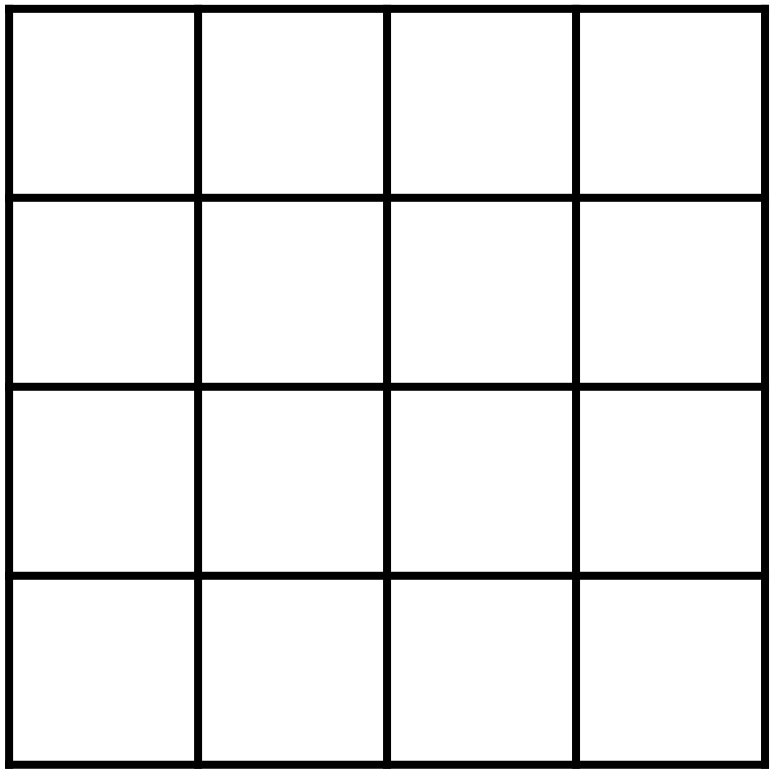
2행에 놓을 칸이 없으므로 **return** → 종료

9663 N-Queen



answer: **2**

n = 4 체스판 (X는 퀸)



탐색 종료!

N-Queen 코드

```
4  int n;
5  int pos[20];
6  int an = 0;
7
8  void back(int depth) {
9      if (depth == n) { // n번 탐색이 끝나면
10         an++;
11         return;
12     }
13     for (int i = 0; i < n; i++) { // (depth, i)는 퀸의 위치
14         int check = 0;
15         for (int j = 0; j < depth; j++) {
16             if (pos[j] == i || j + pos[j] == i + depth || j - pos[j] == depth - i) {
17                 check = 1; // (depth, i)에 놓는게 불가능하면
18                 break;
19             }
20         }
21         if (check == 0) { //가능하면
22             pos[depth] = i;
23             back(depth + 1);
24         }
25     }
26 }
```

```
29 int main() {
30     ios_base::sync_with_stdio(false);
31     cin.tie(NULL);
32     cin >> n;
33     back(0);
34     cout << an;
35     return 0;
36 }
```

Problem Set – DFS, BFS

2178 미로 탐색

6576 토마토

10026 적록색약

14502 연구소

Problem Set – Backtracking

2529 부등호

1987 알파벳

2661 좋은수열

2580 스도쿠

Problem Set – 도전!

1194 달이 차오른다, 가자.

1799 비숍