

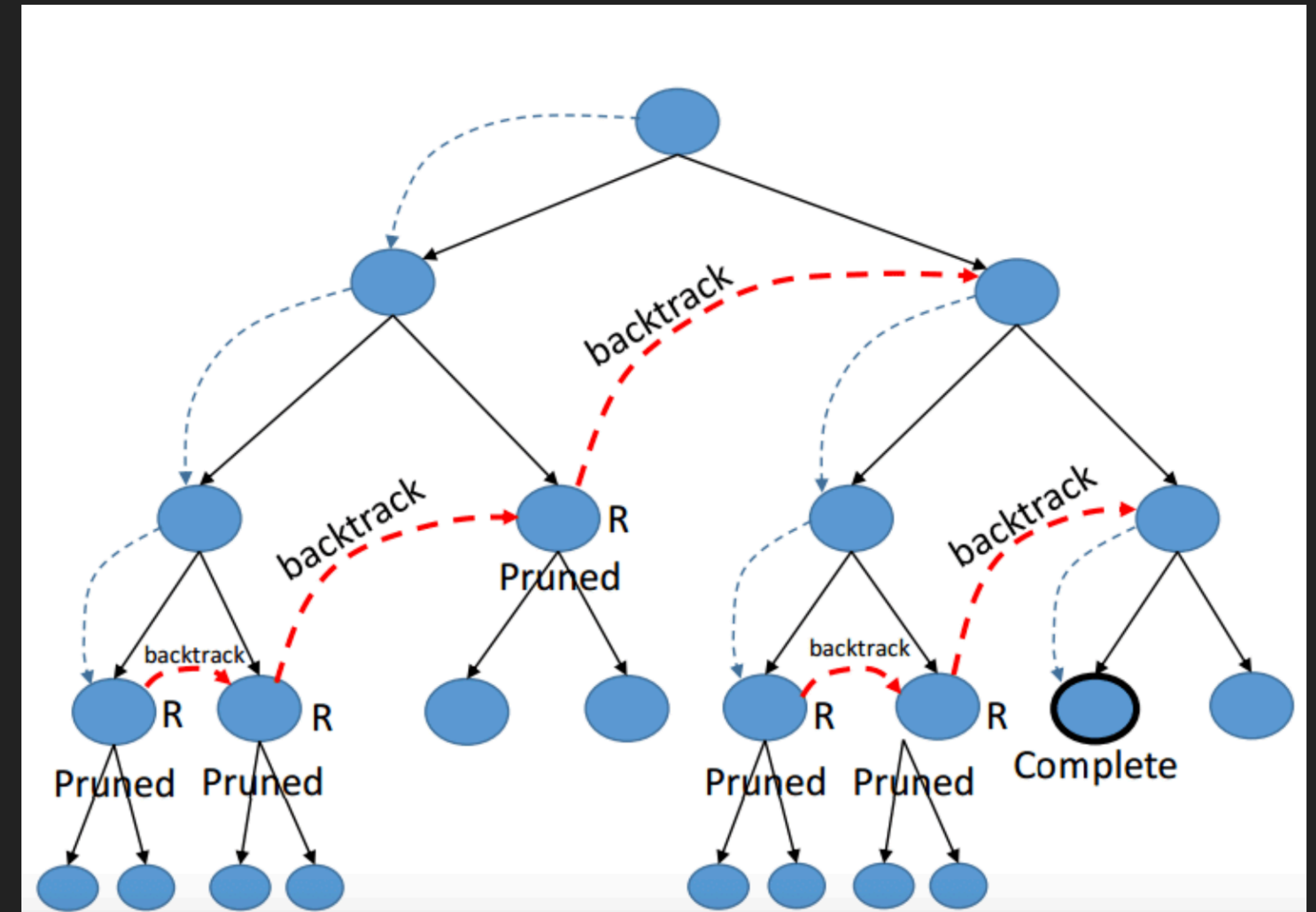
ALGOLIVE - CAU ALGORITHM STUDY

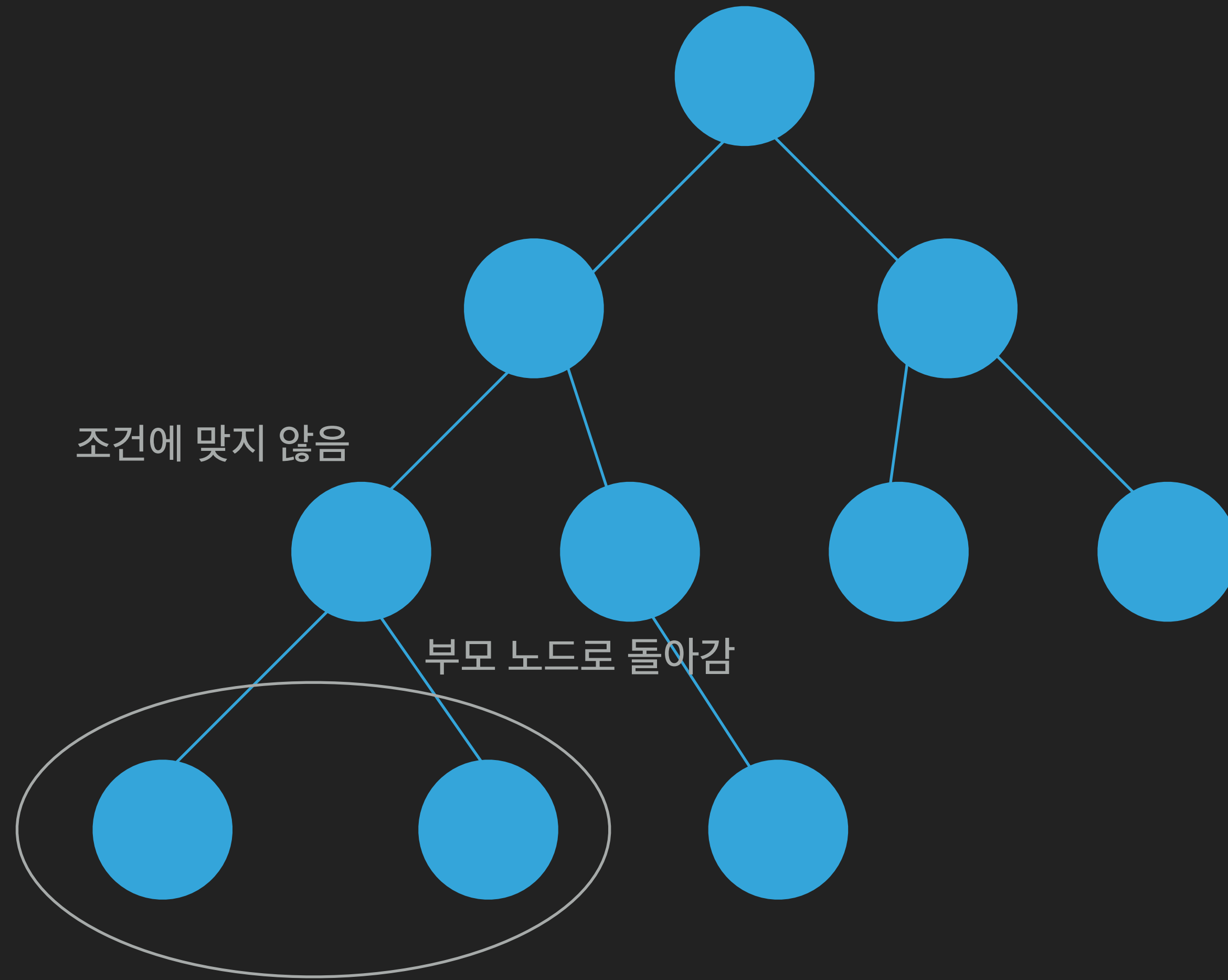
BACKTRACKING

**WHAT IS
BACKTRACKING?**

백트래킹 (BACKTRACKING)

- ▶ 모든 가능한 경우의 수 중에서 특정한 조건을 만족하는 경우만 살펴보는 것
- ▶ 지금 있는 경로가 답이 아닐 것 같으면 되돌아가는 과정
- ▶ “가지치기”
- ▶ 불필요한 부분을 쳐내고 최대한 올바른 쪽으로 가는 것
- ▶ 문제를 풀 때 트리를 직접 그리고 코드를 설계하면 편함





탐색 X

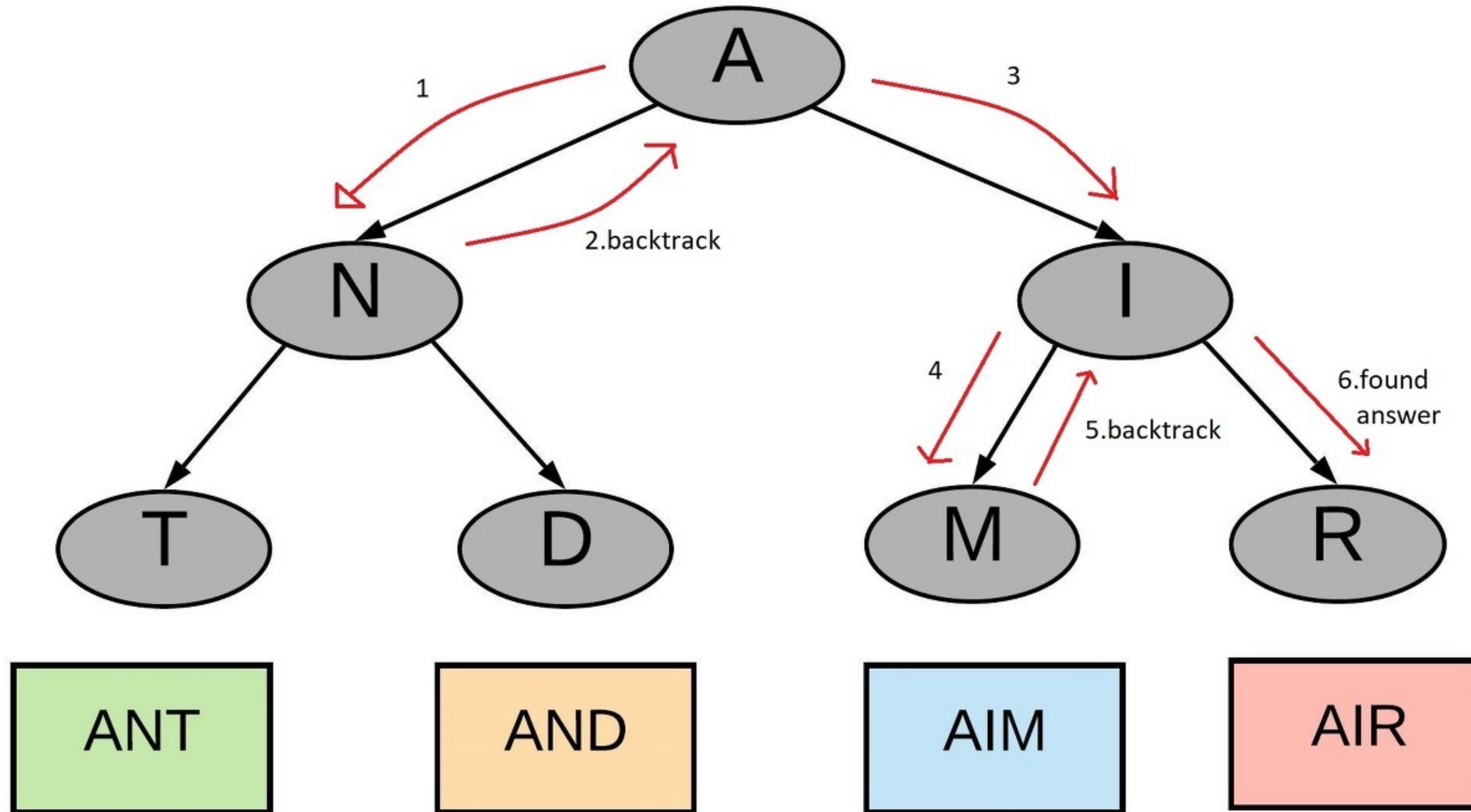
S	0	0 BackTrack	0
1	0	1 ↓	0 ↓
1	1	1	1 BackTrack
0	0	1	0
0	0	1	F

(2,2)

DFS

WITH CONDITION

MAKING WORD 'AIR'



DFS : 무한히 깊은 곳
탐색 불가

BACKTRACKING
: 무한히 탐색 전 미리 되돌아감

SILVER 3. #15649

N과 M (1)


```

#include <iostream>
using namespace std;

int N, M;
int arr[9];
bool visited[9];

void dfs(int depth){
    if(depth == M){
        for(int i= 0 ; i < M ; i++){
            cout << arr[i] << " ";
        }
        cout << "\n";
    }

    for(int i =1 ; i <= N; i++){
        if(!visited[i]){
            visited[i] = true;
            arr[depth] = i;
            dfs(depth+1);
            visited[i] = false;
        }
    }
}

int main(){
    cin >> N >> M;
    dfs(0);
}

```

#15649 - C++ (L)

```

#include <iostream>
#define MAX 9
using namespace std;

int n,m;
int arr[MAX] = {0,};
bool visited[MAX] = {0,};

void dfs(int cnt)
{
    if(cnt == m)
    {
        for(int i = 0; i < m; i++)
            cout << arr[i] << ' ';
        cout << '\n';
        return;
    }
    for(int i = 1; i <= n; i++)
    {
        if(!visited[i])
        {
            visited[i] = true;
            arr[cnt] = i;
            dfs(cnt+1);
            visited[i] = false;
        }
    }
}

int main() {
    cin >> n >> m;
    dfs(0);
}

```

#15649 - C++

```

1 import java.util.Scanner;
2
3 public class Main {
4
5     public static int[] arr;
6     public static boolean[] visit;
7
8     public static void main(String[] args) {
9
10         Scanner in = new Scanner(System.in);
11
12         int N = in.nextInt();
13         int M = in.nextInt();
14
15         arr = new int[M];
16         visit = new boolean[N];
17         dfs(N, M, 0);
18     }
19
20     public static void dfs(int N, int M, int depth) {
21         if (depth == M) {
22             for (int val : arr) {
23                 System.out.print(val + " ");
24             }
25             System.out.println();
26             return;
27         }
28
29         for (int i = 0; i < N; i++) {
30             if (!visit[i]) {
31                 visit[i] = true;
32                 arr[depth] = i + 1;
33                 dfs(N, M, depth + 1);
34                 visit[i] = false;
35             }
36         }
37     }
38 }

```

#15649 - JAVA

```
1 n,m = list(map(int,input().split()))
2
3 s = []
4
5 def dfs():
6     if len(s)==m:
7         print(' '.join(map(str,s)))
8         return
9
10    for i in range(1,n+1):
11        if i not in s:
12            s.append(i)
13            dfs()
14            s.pop()
15
16 dfs()
17
```

Colored by Color Scriptor

#15649 - Python

SILVER 2. #6603

로또


```

1  #include <iostream>
2  using namespace std;
3
4  int K=1;
5  int arr[13];
6  int temp[13];
7  bool visited[6];
8
9  void dfs(int depth, int prev){
10     if(depth == 6){
11         for(int i= 0 ; i < 6 ; i++){
12             cout << temp[i] << " ";
13         }
14         cout << "\n";
15         return;
16     }
17
18     for(int i =0 ; i < K; i++){
19         if(!visited[i] && prev < arr[i]){
20             visited[i] = true;
21             temp[depth] = arr[i];
22             prev = temp[depth];
23             dfs(depth+1, prev);
24             visited[i] = false;
25         }
26     }
27 }
28
29 int main(){
30     while(K != 0){
31         cin >> K;
32         for(int i = 0 ; i < K ; i++){
33             cin >> arr[i];
34         }
35         dfs(0,0);
36         cout << "\n";
37     }
38 }

```

#6603 - C++(L1)

```

1  #include<iostream>
2  #define MAX_SIZE 13
3  using namespace std;
4  int lotto[MAX_SIZE];
5  int combi[MAX_SIZE];
6  int k;
7
8  void dfs(int start, int depth) {
9
10     if(depth == 6) {                //탈출조건
11         for(int i=0; i<6; i++) {
12             cout << combi[i] << ' ';    //조합하나를 출력한 뒤 탈출
13         }
14         cout << '\n';
15         return;
16     }
17
18     for(int i=start; i<k; i++) {        //lotto배열 0부터 k-1까지 탐색함
19         combi[depth] = lotto[i];        //depth는 깊이 -> 0~5번째 깊이까지 재귀를통해 새로 탐색한 숫자를 넣음.
20         dfs(i+1, depth+1);              //재귀 들어가는 부분 , 하나의 깊이를 탐색 후 저장했으니 다음 함수호출할때.
21     }
22
23 }
24
25 int main() {
26
27
28     while(cin >> k && k) {              //0을 입력 받을 때 까지 무한루프
29         for(int i=0; i<k; i++) {
30             cin >> lotto[i];
31         }
32
33         dfs(0,0);
34         cout << '\n';
35
36     }
37     return 0;
38 }
39

```

#6603 - C++


```

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.util.*;

public class Main {
    static int k;
    static int [] s;
    static boolean [] chk;
    public static void main(String[] args) throws Exception {
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        while(true){
            String testCase=br.readLine();
            if(testCase.equals("0")) break;
            String [] input=testCase.split(" ");
            k=Integer.parseInt(input[0]);
            s=new int[k];
            chk=new boolean[k];
            for(int i=0;i<k;i++){
                s[i]=Integer.parseInt(input[i+1]);
            } //초기 값 세팅

            dfs(0,0);
            System.out.println();
        }
    }

    public static void dfs(int depth,int start){
        if(depth==6){
            for(int i=0;i<k;i++){
                if(chk[i]){
                    System.out.print(s[i]+" ");
                }
            }
            System.out.println();
        }
        for(int i=start;i<k;i++){
            chk[i]=true;
            dfs(depth+1,i+1);
            chk[i]=false;
        }
    }
}

```


#6603 - JAVA

```
def dfs(depth, idx):
    if depth == 6:
        print(*out)
        return

    for i in range(idx, k):
        out.append(S[i])
        dfs(depth + 1, i + 1)
        out.pop()

while True:
    array = list(map(int, input().split()))
    k = array[0]
    S = array[1:]
    out = []
    dfs(0, 0)
    if k == 0:
        exit()
    print()
```

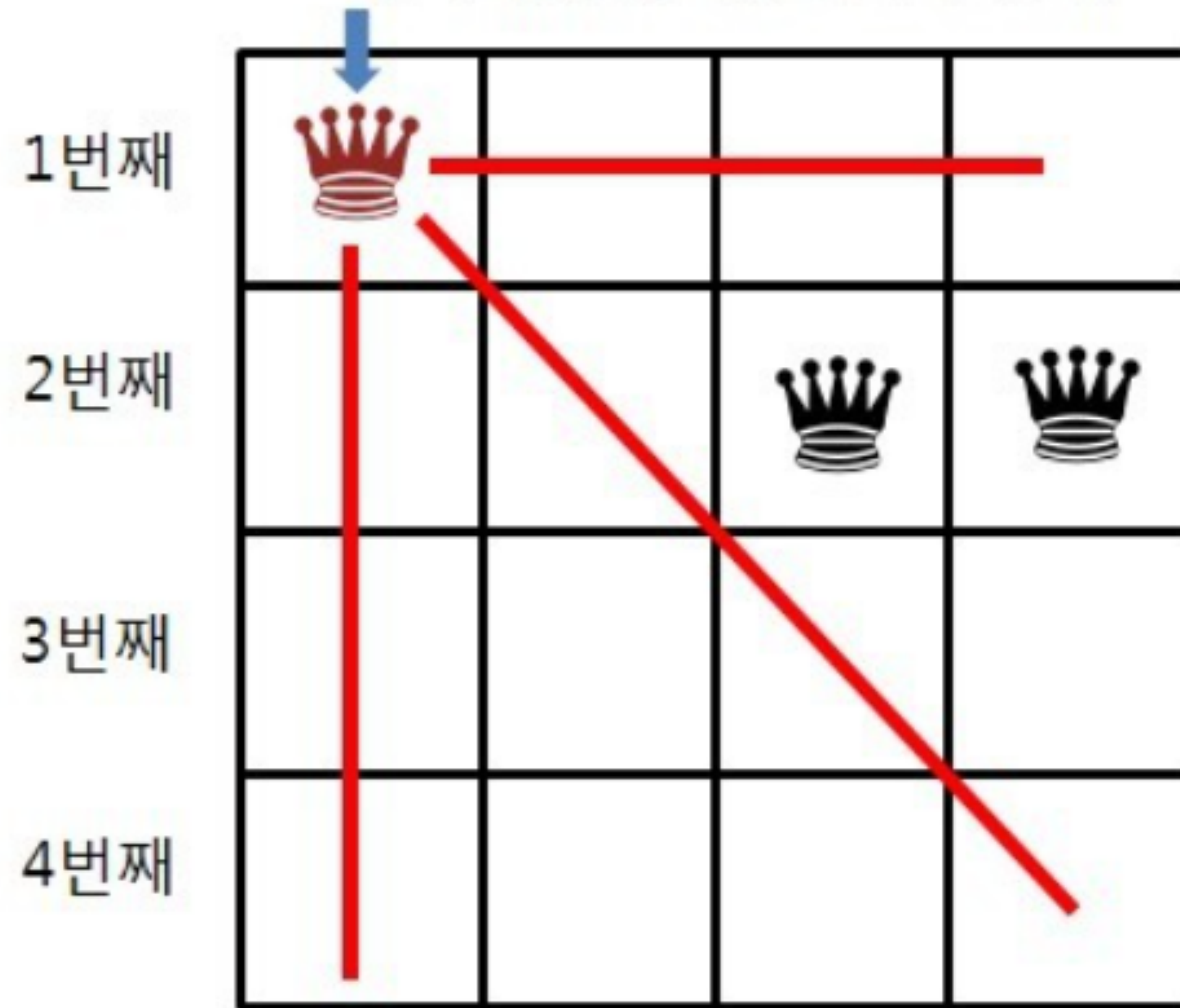
#6603 - Python

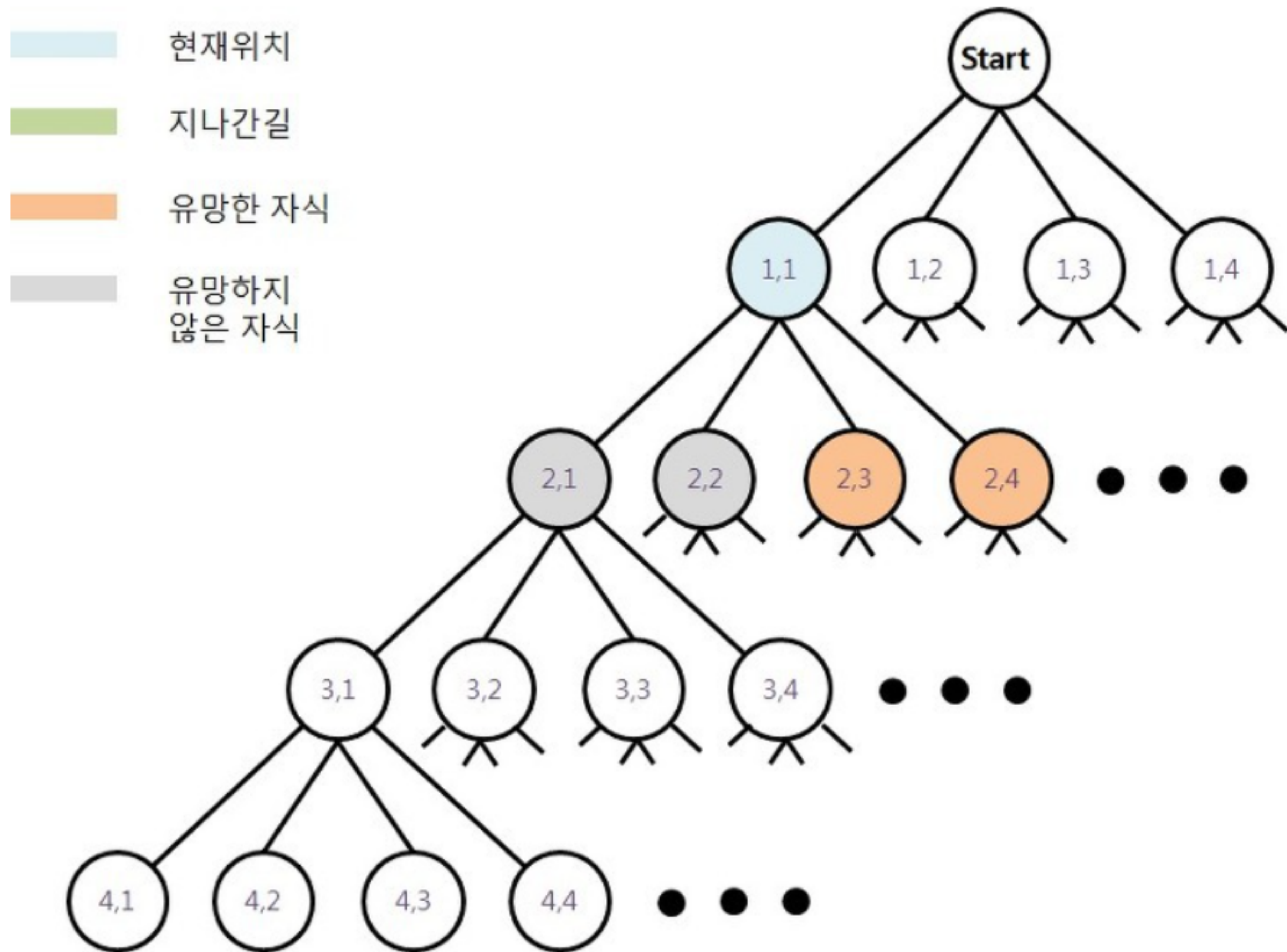


GOLD 4. #9663

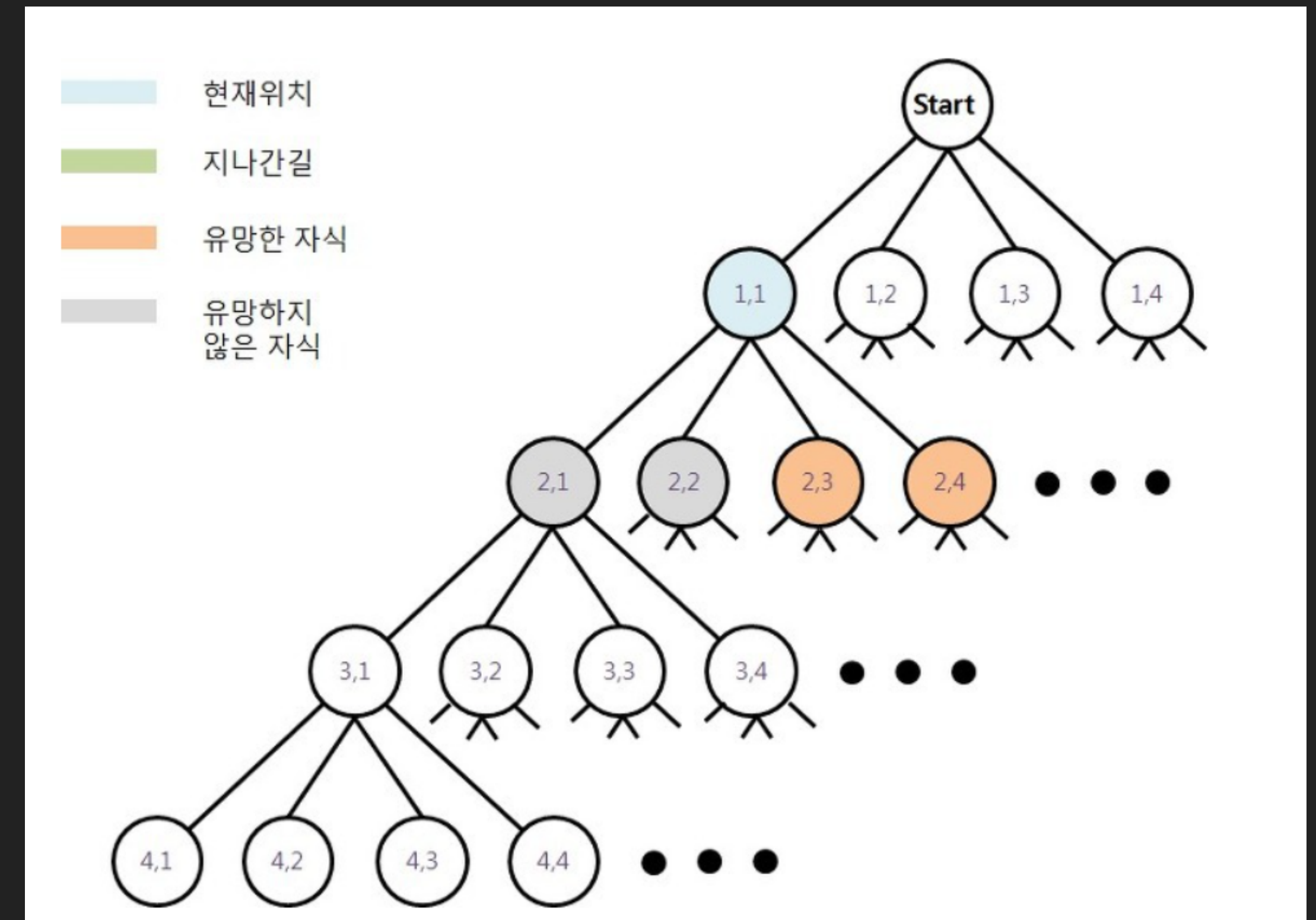
N-QUEEN

여기에 퀸이 있을 경우
2번째에 올수 있는 퀸의 위치





- ▶ 기존의 DFS는 모든 노드를 순회 -> (2,1), (2,2)도 탐색했을 것
- ▶ (2,1), (2,2)는 첫번째 퀸이 이동하는 경로에 중복되므로 절대로 답이 되지 않음
- ▶ '가지치기'를 통해 (2,1),(2,2)는 백트래킹 수행
- ▶ 필요한 함수 :
 - 특정 행, 열에 퀸을 놓을 수 있는 지 판단하는 함수
 - 현재 행에서 모든 열을 검사하여 퀸을 놓을 수 있다면, 다음 행 (자식 노드) 를 검사하는 DFS 함수
- ▶ => 오른쪽 트리를 DFS로 순회하면서 조건문이 추가되었다는 개념을 가지고 풀면 됨.



코드 분석 :

BACKTRACK을 하기 위한 조건문 :

```
bool isPossible(int row){
    for(int i =0 ; i < row ;i++){ //dfs를 통해 해당 행까지 탐색
        if((arr[i] == arr[row]) || row-i == abs(arr[i]-arr[row])){ //현재 위치의 행에
//놓여있는 퀸의 열과 같은 값의 열이 발견되거나, 대각선이라면
            return false; //탐색 종료 후 BackTrack
        } //같은 열 or 대각선
    }
    return true;
}
```

해당 위치에 Queen 체스말을 놓았을 때 조건이 안맞으면, 그 아래의 행에 Queen 체스말을 놓는 것은 무의미하므로 중단시켜버림

코드 분석 :

DFS + 조건문 => BACKTRACKING 가능한 DFS

```
void dfs(int row){
    if(row == N){
        ans++;
        return;
    }else{
        for(int i =0 ; i < N ; i++){ //행 row에서 0부터 N까지 열 탐색
            arr[row] = i; //해당 행,열에 Queen 체스말 우선 배치
            if(isPossible(row)){ //배치 후 조건이 안 맞으면 즉시 중단하고 다음 열 탐색
                dfs(row+1);
            }
        }
    }
}
```



```

#include <iostream>

using namespace std;

int N;
int arr[16]; //
int ans;

bool isPossible(int row){
    for(int i =0 ; i < row ;i++){ //dfs를 통해 해당 행까지 탐색
        if((arr[i] == arr[row]) || row-i == abs(arr[i]-arr[row])){ //현재 위치의 행에 놓여있는 퀸의 열과 같은 값의 열이 발견!
            //대각선이라면
            return false; //탐색 종료 후 BackTrack
        } //같은 열 or 대각선
    }
    return true;
}

void dfs(int row){
    if(row == N){
        ans++;
        return;
    }else{
        for(int i =0 ; i < N ; i++){ //행 row에서 0부터 N까지 열 탐색
            arr[row] = i; //해당 행,열에 Queen 체스말 우선 배치
            if(isPossible(row)){ //배치 후 조건이 안 맞으면 즉시 중단하고 다음 열 탐색
                dfs(row+1);
            }
        }
    }
}

int main(void){
    cin >> N;
    dfs(0);
    cout << ans;
}

```

#9663 - C++(나)

```

#include <iostream>
#define MAX 15
using namespace std;

int col[MAX];
int N, total = 0;

bool check(int level)
{
    for(int i = 0; i < level; i++)
        if(col[i] == col[level] || abs(col[level] - col[i]) == level - i) // 대각선이거나 같은 row
            return false;
    //col[i]가 의미하는 것이 X좌표, i가 의미하는것이 Y좌표이므로 차이가 일정하다면 대각선에 있다고 볼 수 있다.
    return true;
}

void nqueen(int x)
{
    if(x == N)
        total++;
    else
    {
        for(int i = 0; i < N; i++)
        {
            col[x] = i; // 해당 위치에 퀸을 배치
            if(check(x)) // 유효하다면 다음행의 퀸 배치, 유효하지않다면 다른 위치로 퀸 배치 변경
                nqueen(x+1);
        }
    }
}

int main() {
    cin >> N;
    nqueen(0);
    cout << total;
}

```

#9663 - C++

```

import java.util.Scanner;

public class Main {

    static int[] queen;
    static int n;
    static int ans;

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        n = sc.nextInt();
        queen = new int[n]; // 퀸을 배치할 행의 번호를 1~N번까지 사용 예정
        backtrack(0);
        System.out.println(ans);
    }

    static void backtrack(int row) {
        if (row == n) {
            ans++;
            return;
        }
        for (int j = 0; j < n; j++) {
            queen[row] = j;
            if (isOk(row)) { // 지금 row행의 퀸을 j번에 놓은게 괜찮
                backtrack(row + 1);
            } // 지금 봤던 퀸이 자리가 안좋으면 ? 그냥 다음 j로 넘어가겠
        }
    }

    static boolean isOk(int col) {
        // 지금 row행에 놓은 퀸이 이전 퀸들에 영향을 안받는 자리에 있는지 확인
        for (int i = 0; i < col; i++) {
            // 현재 row 위치에 퀸이 있음
            if (queen[col] == queen[i])
                return false;

            /*
             * 대각선상에 놓여있는 경우
             * (열의 차와 행의 차가 같을 경우가 대각선에 놓여있는 경우다)
             */
            if (Math.abs(col - i) == Math.abs(queen[i] -
queen[col])) {
                return false;
            }
        }
        return true;
    }
}

```

다면??

지.

#9663 - JAVA(나)

```

1 def adjacent(x): # x와 i 가 같으면 행이 같은거 근데 for문을 보면 x와 i가 같을 수가 없다.
2     for i in range(x): #인덱스가 행 row[n]값이 열
3         if row[x] == row[i] or abs(row[x] - row[i]) == x - i: # 열이 같거나 대각선이 같으면 false
4             return False # 대각선이 같은경우는 두 좌표에서 행 - 행 = 열 - 열 이 같으면 두개는 같은 대각선상에 있다.
5     return True
6
7 #한줄씩 재귀하며 dfs 실행
8
9 def dfs(x):
10     global result
11
12     if x == N:
13         result += 1
14     else:
15         # 각 행에 퀸 놓기
16         for i in range(N): # i 는 열번호 0부터 N 전까지 옮겨가면서 유망한곳 찾기
17             row[x] = i
18             if adjacent(x): # 행,열,대각선 체크함수 true이면 백트래킹 안하고 계속 진행
19                 dfs(x + 1)
20
21 # N = int(input())/
22 N = int(input())
23 row = [0] * N
24 result = 0
25 print(row)
26 dfs(0)
27 # print(row)
28 print(result)

```

Colored by Color Scripter

#9663 - Python