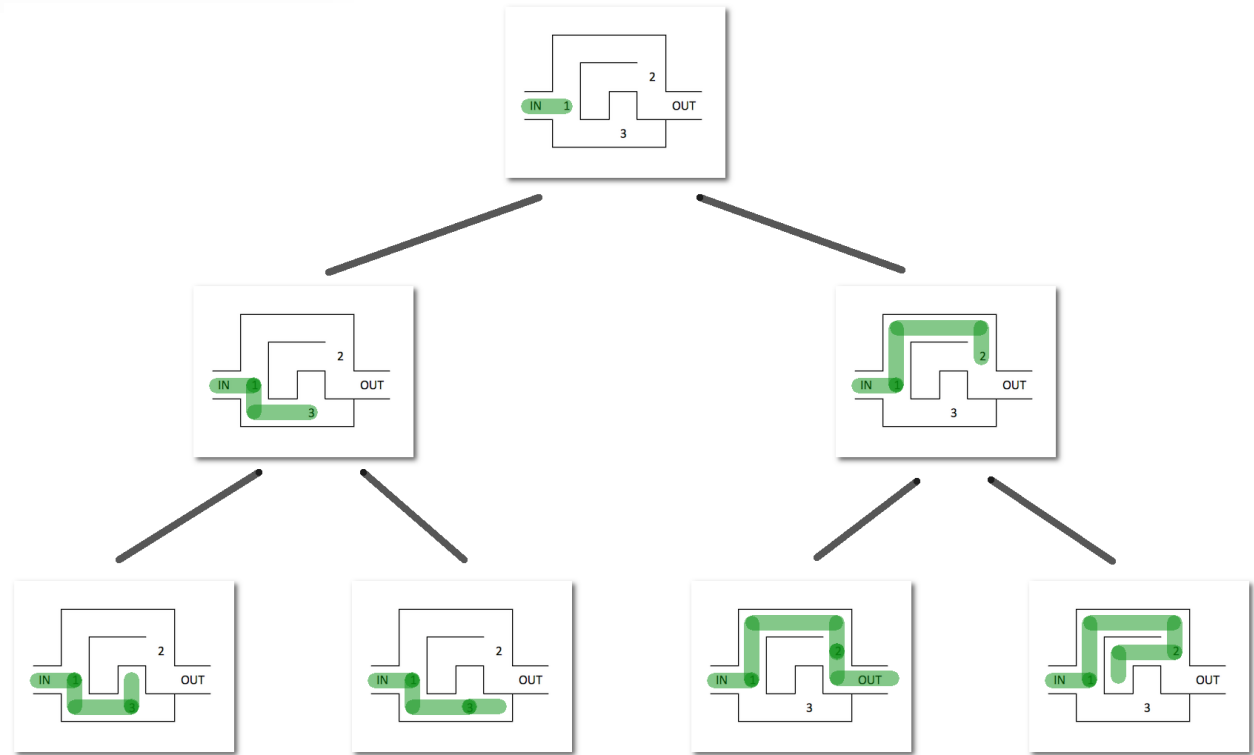


# 2021 여름학기 동국대학교 SW역량강화캠프

## 11일차. 백트래킹

## ● 백트래킹 (Backtracking)

- ▶ 해를 찾는 도중 해가 아니어서 막히면, 전 상태로 되돌아가며 찾는 해결기법
- ▶ DFS에서 전역변수를 이용하여 구현할 때 백트래킹을 이용할 수 있다.
- ▶ 가지치기로 최적화 가능



### ● N과 M 1(3016)

#### | 문제

1부터 N까지 자연수 중 중복없이 M개를 고른 수열을 사전순으로 출력하는 프로그램을 만드세요.

#### | 입력

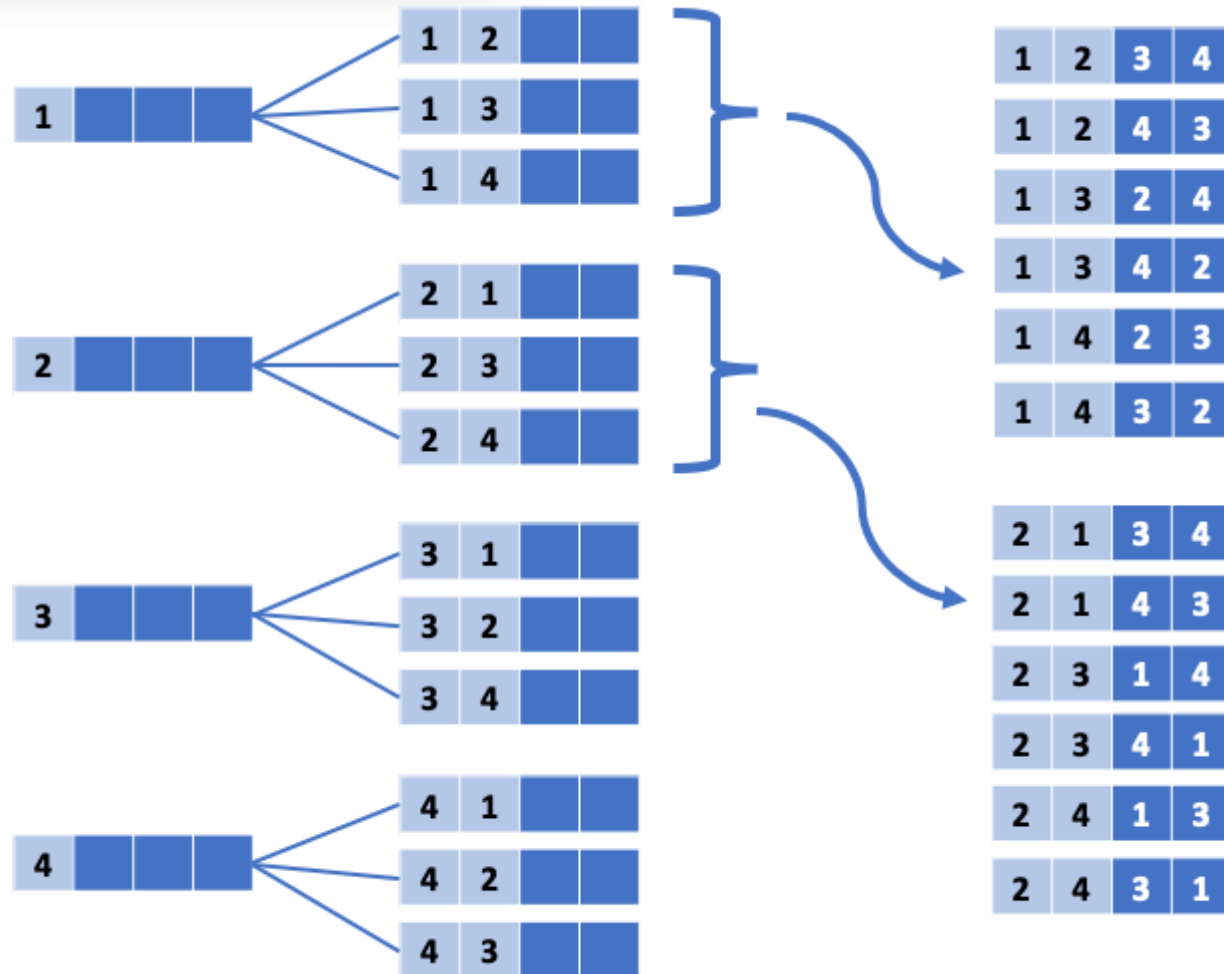
첫째 줄에 자연수 N과 M이 주어진다. ( $1 \leq M \leq N \leq 8$ )

#### | 출력

한 줄에 하나씩 문제의 조건을 만족하는 수열을 출력한다. 중복되는 수열을 여러 번 출력하면 안되며, 각 수열은 공백으로 구분해서 출력해야 한다.

수열은 사전 순으로 증가하는 순서로 출력해야 한다.

● N과 M 1(4313)



- ▶ 백트래킹을 이용해서 출력할 배열을 DFS로 관리하자
- ▶ `void dfs(int x)` : 출력할  $x$ 번째 수를 `arr[x]`에 채우고 `dfs(x+1)`을 호출하는 함수
- ▶ 맨 처음 `dfs(1)`을 호출
- ▶ 1~ $M$ 번째 수를 모두 채우고 `dfs(M+1)`이 호출되면, 지금까지 채운  $M$ 개의 수를 출력한다.

```
public static void dfs(int x) { // arr[x]의 값을 할당할 차례, arr[1] ~ arr[M]
    if (x == M + 1) {
        for (int i = 1; i <= M; i++) {
            sb.append(arr[i] + " ");
        }
        sb.append("\n");
        return;
    }
    for (int i = 1; i <= N; i++) {
        if (!visit[i]) {
            arr[x] = i;
            visit[i] = true;
            dfs(x + 1);
            arr[x] = 0; // backtrack
            visit[i] = false; // backtrack
        }
    }
}
```

### ● N과 M 2(3017)

#### | 문제

1부터 N까지 자연수 중 중복없이 M개를 고른 오름차순인 수열을 사전순으로 출력하는 프로그램을 만드세요.

#### | 입력

첫째 줄에 자연수 N과 M이 주어진다. ( $1 \leq M \leq N \leq 8$ )

#### | 출력

한 줄에 하나씩 문제의 조건을 만족하는 수열을 출력한다. 중복되는 수열을 여러 번 출력하면 안되며, 각 수열은 공백으로 구분해서 출력해야 한다.

수열은 사전 순으로 증가하는 순서로 출력해야 한다.

▶ 이번에는 수를 오름차순으로 골라야 한다.

▶ 완전탐색에서는 오름차순으로 값을 만들기 위해  $j$ 를  $i+1$ 부터,  $k$ 를  $j+1$ 부터 탐색

▶ `void dfs(int x, int p)`

$x$ 번째 수를  $p$  이상으로 탐색하고,  $x$ 번째 수를  $q$ 로 결정했다면 `dfs(x+1, q+1)`을 호출하는 함수

▶ 맨 처음 `dfs(1,1)`을 호출

▶ 1~ $M$ 번째 수를 모두 결정하고 `dfs(M+1, *)`가 호출되면 지금까지 채운  $M$ 개의 수를 출력한다.





```
public static void dfs(int x, int p) { // arr[x]의 값을 p 이상으로 할당할 차례, arr[1] ~ arr[M]
    if (x == M + 1) {
        for (int i = 1; i <= M; i++) {
            sb.append(arr[i] + " ");
        }
        sb.append("\n");
        return;
    }
    for (int i = p; i <= N; i++) {
        arr[x] = i;
        dfs(x + 1, i + 1);
        arr[x] = 0; // backtrack
    }
}
```

```
public static void dfs(int x, int p) { // arr[x]의 값을 p 이상으로 할당할 차례, arr[1] ~ arr[M]
    if (x == M + 1) {
        for (int i = 1; i <= M; i++) {
            sb.append(arr[i] + " ");
        }
        sb.append("\n");
        return;
    }
    for (int i = p; i + (M - x) <= N; i++) {
        arr[x] = i;
        dfs(x + 1, i + 1);
        arr[x] = 0; // backtrack
    }
}
```

### ● N-Queen (2861)

#### | 문제

체스라는 게임에서 퀸은 가장 강력한 말로, 가로, 세로, 대각선을 모두 이동할 수 있다.

$N * N$  크기에 체스 판에 서로를 잡을 수 없도록 퀸을 배치할 때, 최대  $N$ 개의 퀸을 배치할 수 있다.

그런데,  $N$ 이 6보다 큰 경우에는  $N$ 개의 퀸을 배치하는 방법이 유일하지는 않다.

$N$ 이 주어질 때,  $N$ 개의 퀸을 배치할 수 있는 방법의 개수를 출력해보자.

#### | 입력

체스판의 크기  $N$ 이 주어진다.(단,  $N$ 은 6이상 13이하이다)

#### | 출력

퀸을 배치하는 경우의 수를 출력합니다.

- ▶ 백트래킹에서의 정석과도 같은 Well-Known 문제
- ▶ 같은 줄, 같은 칸, 같은 대각선에 배치되어서는 안됨
- ▶ 한 줄에 하나씩 퀸을 설치해보자
- ▶ `void dfs(int x):` x번 줄에 퀸을 놓을 자리를 결정하고 `dfs(x+1)`을 호출하는 함수  $\rightarrow arr[x] = i$
- ▶ x번 줄의 i번 칸에 퀸을 놓을 수 있을지 판단할 수 있어야 함.

(1,1)	(1,2)	(1,3)	(1,4)	(1,5)
(2,1)	(2,2)	(2,3)	(2,4)	(2,5)
(3,1)	(3,2)	(3,3)	(3,4)	(3,5)
(4,1)	(4,2)	(4,3)	(4,4)	(4,5)
(5,1)	(5,2)	(5,3)	(5,4)	(5,5)

두번째 좌표가 같으면 서로 공격할 수 있다

$(j, arr[j])$ 와  $(x, i)$ 가 공격 가능한지 체크는

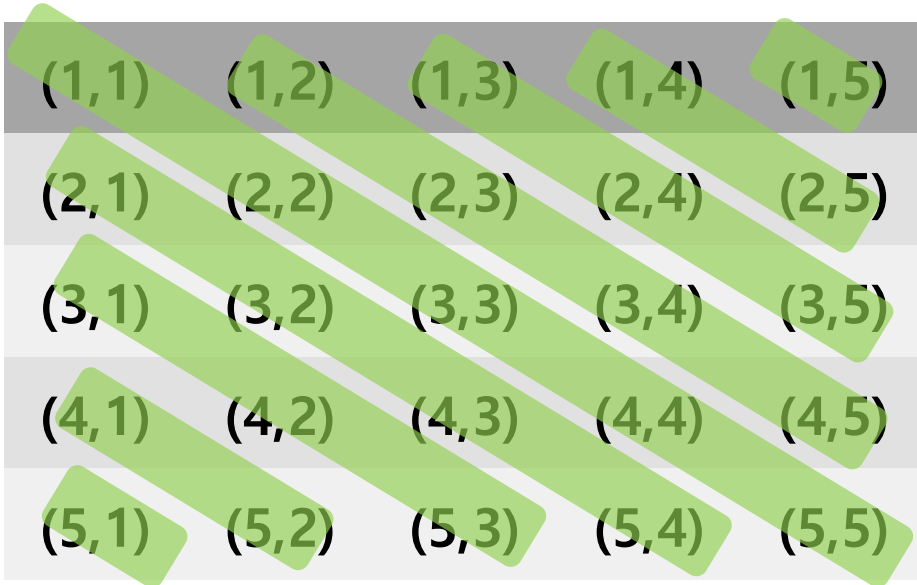
`if(arr[j] == i)` 로 가능하다.

(1,1)	(1,2)	(1,3)	(1,4)	(1,5)
(2,1)	(2,2)	(2,3)	(2,4)	(2,5)
(3,1)	(3,2)	(3,3)	(3,4)	(3,5)
(4,1)	(4,2)	(4,3)	(4,4)	(4,5)
(5,1)	(5,2)	(5,3)	(5,4)	(5,5)

첫번째 좌표와 두번째 좌표의 합이 같으면 서로 공격할 수 있다

$(j, arr[j])$ 와  $(x, i)$ 가 공격 가능한지 체크는

$if(j + arr[j] == x + i)$  로 가능하다.



(1,1)	(1,2)	(1,3)	(1,4)	(1,5)
(2,1)	(2,2)	(2,3)	(2,4)	(2,5)
(3,1)	(3,2)	(3,3)	(3,4)	(3,5)
(4,1)	(4,2)	(4,3)	(4,4)	(4,5)
(5,1)	(5,2)	(5,3)	(5,4)	(5,5)

첫번째 좌표와 두번째 좌표의 차가 같으면 서로 공격할 수 있다

$(j, arr[j])$ 와  $(x, i)$ 가 공격 가능한지 체크는

$if(j - arr[j] == x - i)$  로 가능하다.



```
public static void dfs(int x) { // 위에서 x번째 줄의 몇 번째 칸에 넣을지 판단, 처음은 dfs(1)
    if (x == N + 1) {
        ans++;
        return;
    }
    for (int i = 1; i <= N; i++) { // x번째 줄에 i를 놓을 수 있는지 판단
        boolean pos = true;
        for (int j = 1; j < x; j++) { // 위쪽 줄과 서로 공격할 수 있는지 판단
            // (j, arr[j])와 (x, i)
            if (i == arr[j] || j - arr[j] == x - i || j + arr[j] == x + i)
                pos = false;
        }

        if (pos) {
            arr[x] = i;
            dfs(x + 1);
            arr[x] = 0; // backtrack
        }
    }
}
```

- ▶ 반복문으로 공격가능한 퀸이 있는지 확인하는 방법
- ▶  $ver[x] = x$ 번 칸에 퀸이 있으면 true, 없으면 false
- ▶ 대각선은 이런 배열을 만들 수 없을까?
- ▶  $diag1[x] = (p + q == x)$ 인  $(p, q)$ 에 퀸이 있는가
- ▶  $diag2[x] = (p - q + N == x)$ 인  $(p, q)$ 에 퀸이 있는가

```
static boolean[] ver, diag1, diag2;
// ver[x] == true -> 이미 찢힌 (p,q)들 중 q == x 인 칸이 있다.
// diag1[x] == true -> 이미 찢힌 (p,q)들 중 (p+q) == x 인 칸이 있다.
// diag2[x] == true -> 이미 찢힌 (p,q)들 중 (p-q+N) == x 인 칸이 있다.

public static void main(String[] args) throws IOException {
    BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
    N = Integer.parseInt(br.readLine());

    arr = new int[N + 1];
    ver = new boolean[N+1];
    diag1 = new boolean[2*N+1];
    diag2 = new boolean[2*N+1];
}
```

```
public static void dfs(int x) { // 위에서 x번째 줄의 몇 번째 칸에 넣을지 판단, 처음은 dfs(1)
    if (x == N + 1) {
        ans++;
        return;
    }
    for (int i = 1; i <= N; i++) { // x번째 줄에 i를 놓을 수 있는지 판단
        if(ver[i] || diag1[x+i] || diag2[x-i+N]) // (x, i)가 위쪽 줄과 서로 공격할 수 있는지 판단
            continue;

        ver[i] = diag1[x+i] = diag2[x-i+N] = true;
        dfs(x+1);
        ver[i] = diag1[x+i] = diag2[x-i+N] = false; // backtrack
    }
}
```