Chapter. 02

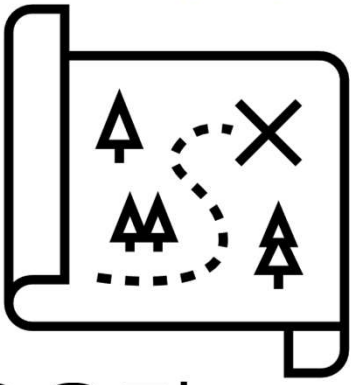
알고리즘

그래프와 탐색
Graph & Search – (응용편)

FAST CAMPUS
ONLINE
알고리즘 공채 대비반I

강사. 류호석

Life Changing Education



Chapter. 02

알고리즘
그래프와 탐색(Graph & Search) - 응용편

FAST CAMPUS
ONLINE
류호석 강사.

BOJ 2667 – 단지번호 붙이기

난이도: 2

$5 \leq$ 지도의 크기, $N < 25$

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

<그림 1>

0	1	1	0	2	0	0
0	1	1	0	2	0	2
1	1	1	0	2	0	2
0	0	0	0	2	2	2
0	3	0	0	0	0	0
0	3	3	3	3	3	0
0	3	3	3	0	0	0

<그림 2>

1: 7개
2: 8개
3: 9개

I 문제 파악하기 – 정답의 최대치

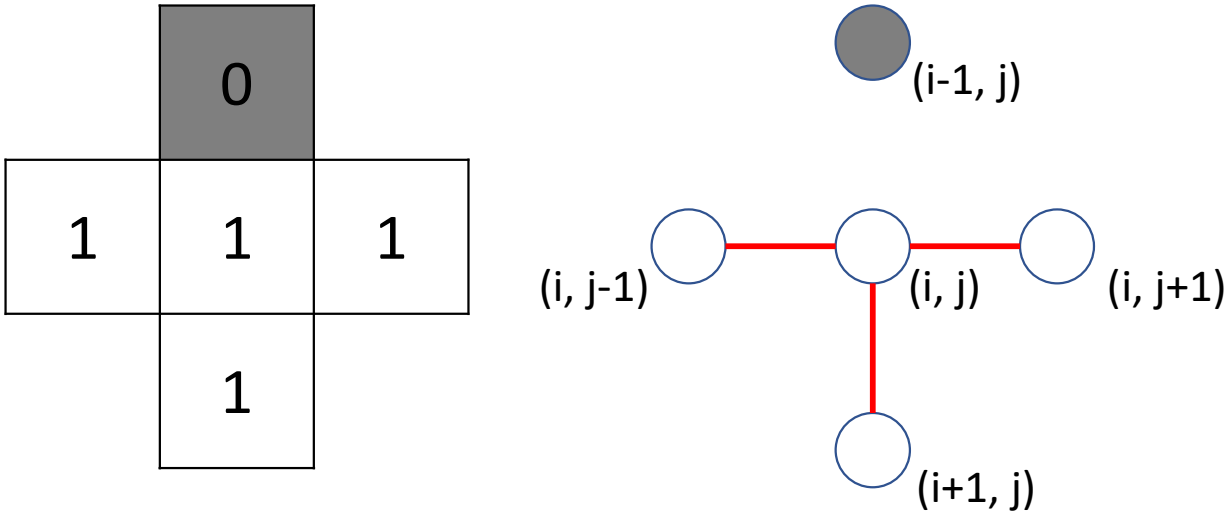
1	1	...	1
1	1	...	1
...	...	...	1
1	1	1	1

집의 개수 최댓값: $O(N)$

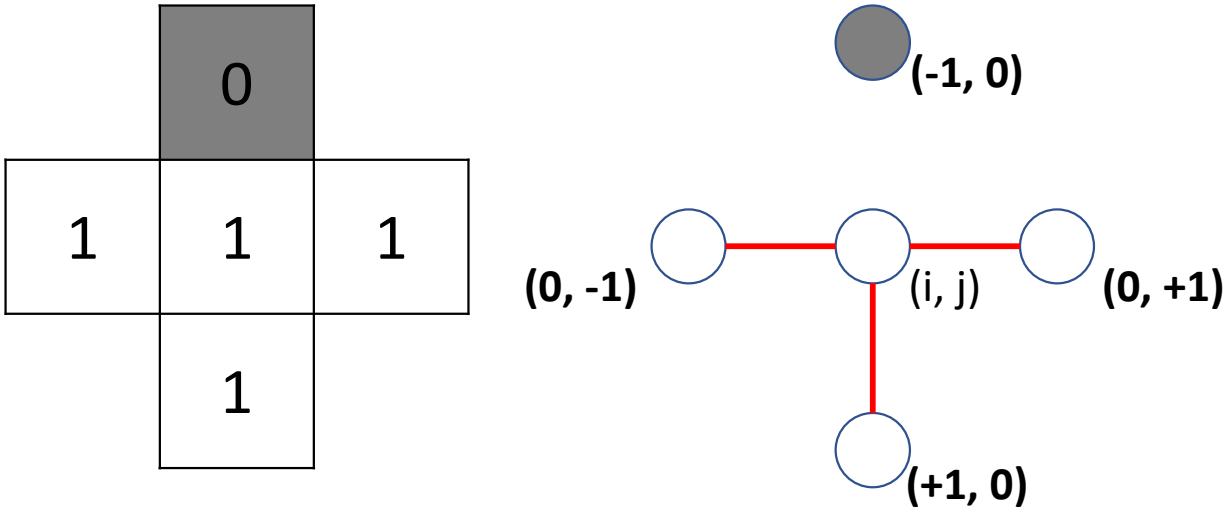
1	0	1	...	1
0	1	0	...	0
1	0	1	...	1
...	...	...	...	0
1	0	1	0	1

단지 개수 최댓값: $O(N\text{^}2)$

I 접근 – 격자형 그래프 구성

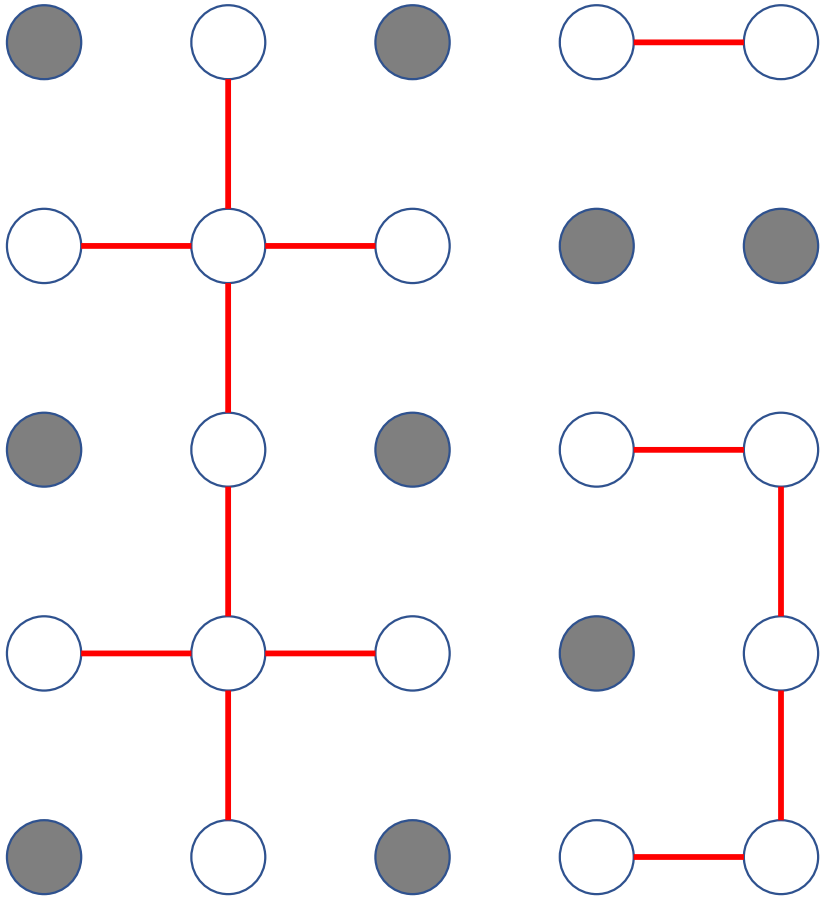


I 접근 – 격자형 그래프 구성

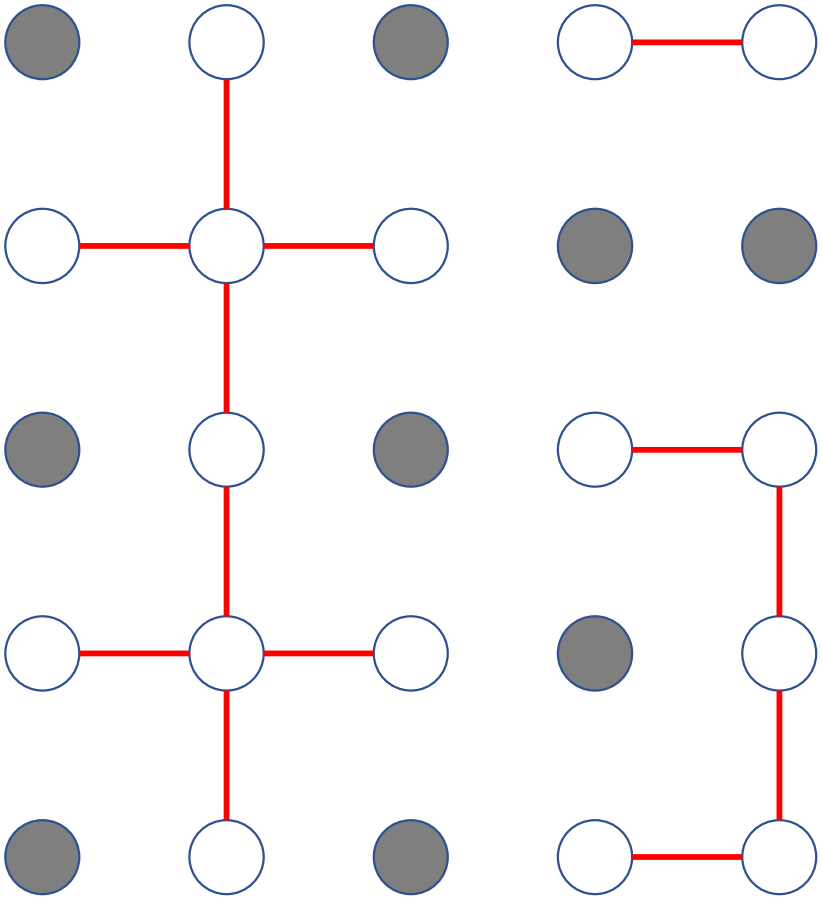


I 접근 – 격자형 그래프 구성

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



I 접근 – 단지 수 세기



새로운 단지를 찾을 때마다,

해당 단지에 속해 있는 집들은

처리되었다는 표시를 해주자!

I 시간, 공간 복잡도 계산하기

<격자형 그래프>

정점: $O(N^2)$

간선: $O(N^2 \times 4)$

BFS 이든 DFS 이든, 시간 복잡도는 모두 $O(N^2)$

I 구현

```
// x, y 를 갈 수 있다는 걸 알고 방문한 상태
static void dfs(int x, int y) {
    // 단지에 속한 집의 개수 증가, visit 체크 하기
    /* TODO */

    // 인접한 집으로 새로운 방문하기
    /* TODO */
}

static void pro() {
    group = new ArrayList<>();
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            if (!visit[i][j] && a[i].charAt(j) == '1') {
                // 갈 수 있는 칸인데, 이미 방문처리 된, 즉 새롭게 만난 단지인 경우!
                /* TODO */
            }
        }
    }

    // 찾은 단지의 정보를 출력하기
    /* TODO */
}
```

I 연습 문제

<격자형 그래프 연습>

- BOJ 1012 – 유기농 배추
- BOJ 11724 – 연결 요소의 개수
- BOJ 4963 – 섬의 개수
- BOJ 3184 – 양

<일반 그래프 연습>

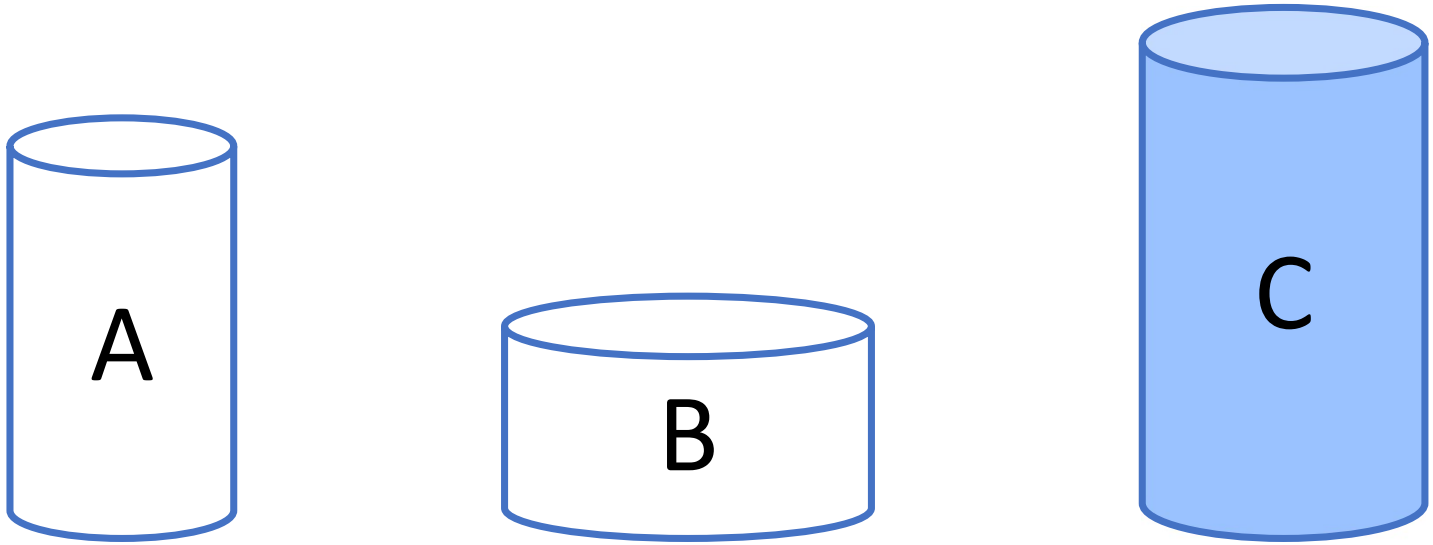
- BOJ 2606 – 바이러스
- BOJ 11403 – 경로 찾기
- BOJ 11725 – 트리의 부모 찾기

이외의 추천 문제가 추가되면 Github 자료에 코드 업로드

I BOJ 2251 – 물통

난이도: 4

$1 \leq$ 물통의 크기, $A, B, C \leq 200$

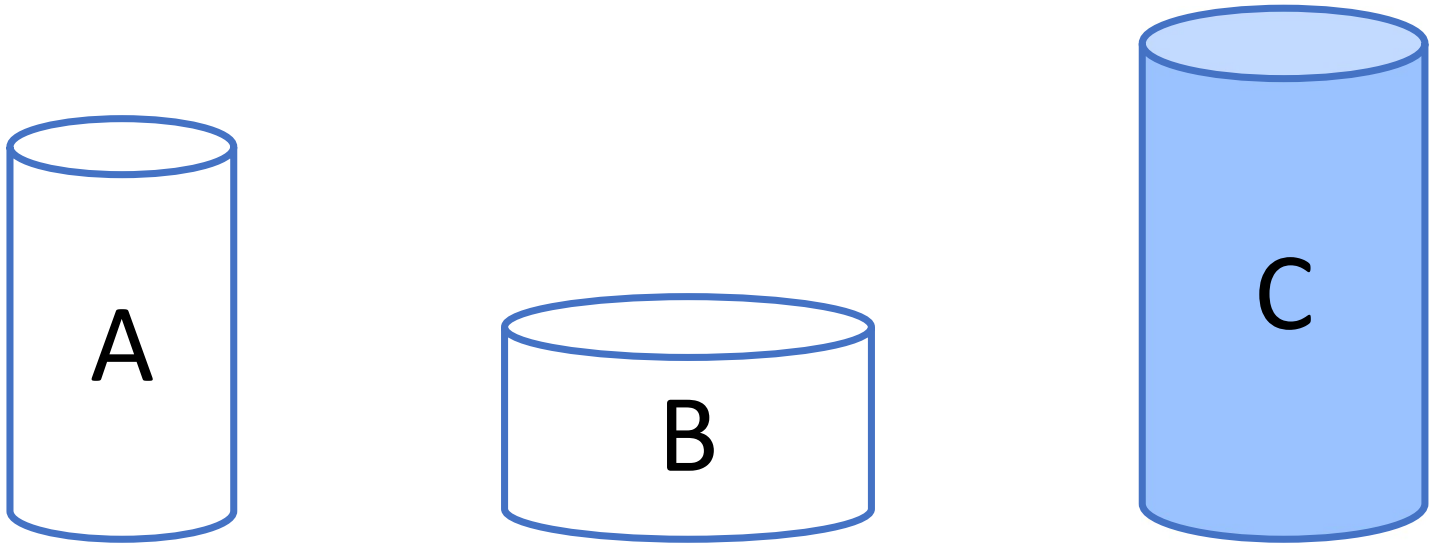


물을 옮겨 넣을 수 있을 때, C 에 남아있는 물의 가능한 경우
(단, A는 비어 있어야 한다.)

I 접근 – 문제의 탐색 영역

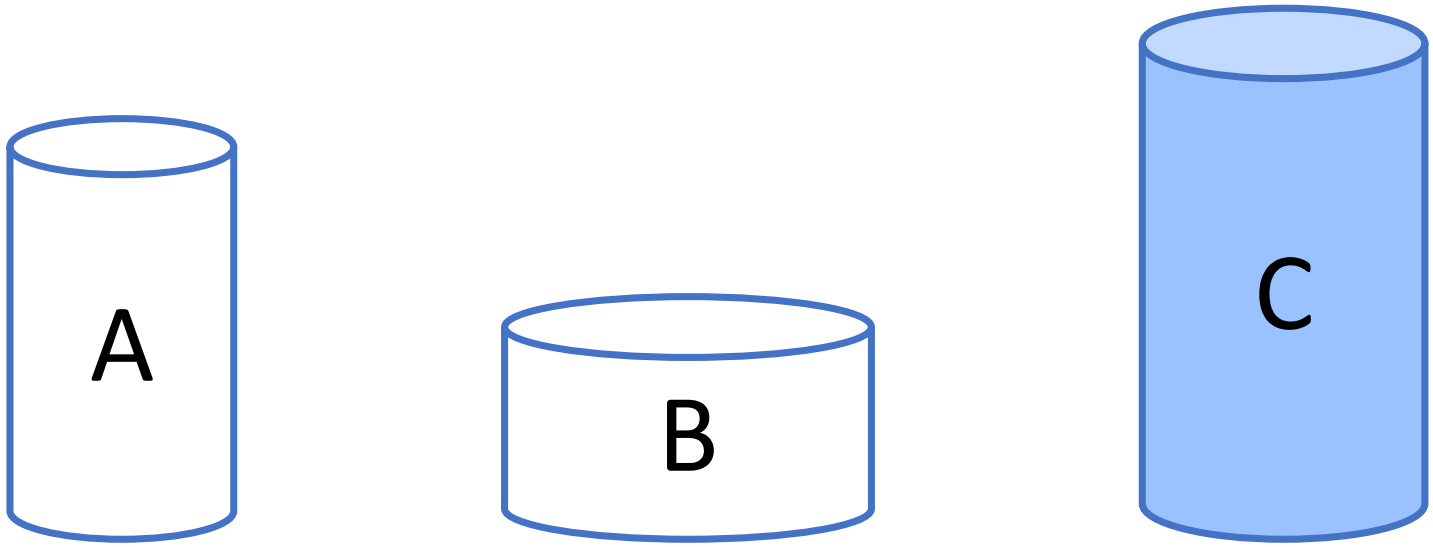
a, b, c를 각각 A, B, C 물통에 남아있는 물의 양이라고 하자.

한 가지 상태 ➔ 하나의 (a, b, c) 페어로 표현이 된다.



I 접근 – 문제의 탐색 영역

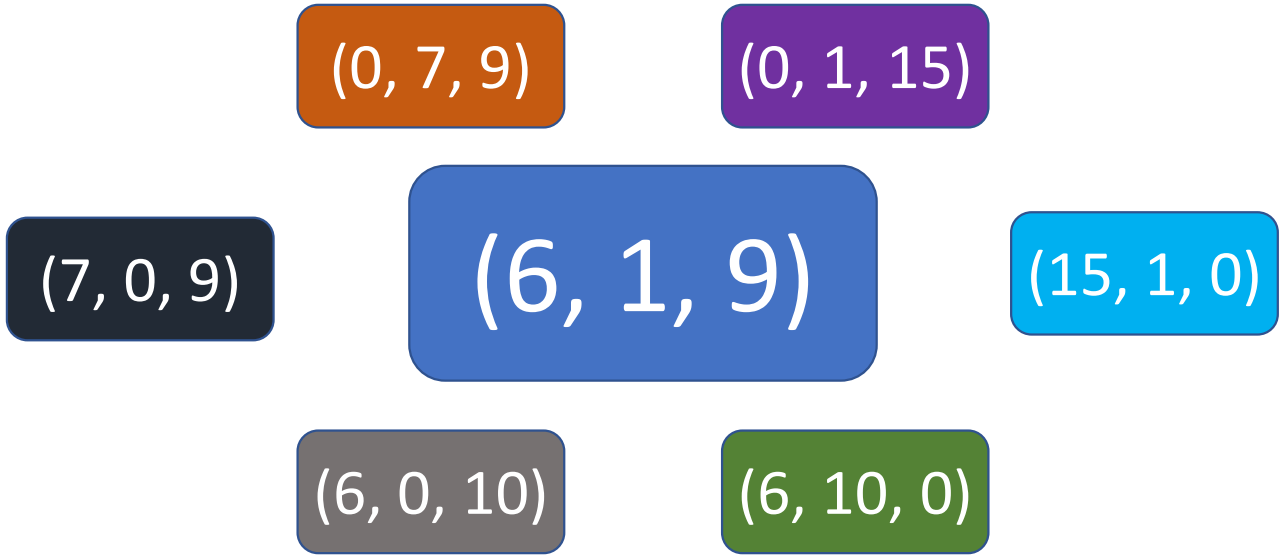
최대 $200 * 200 * 200 = 8 * 10^6$ 가지 상태가 존재한다.



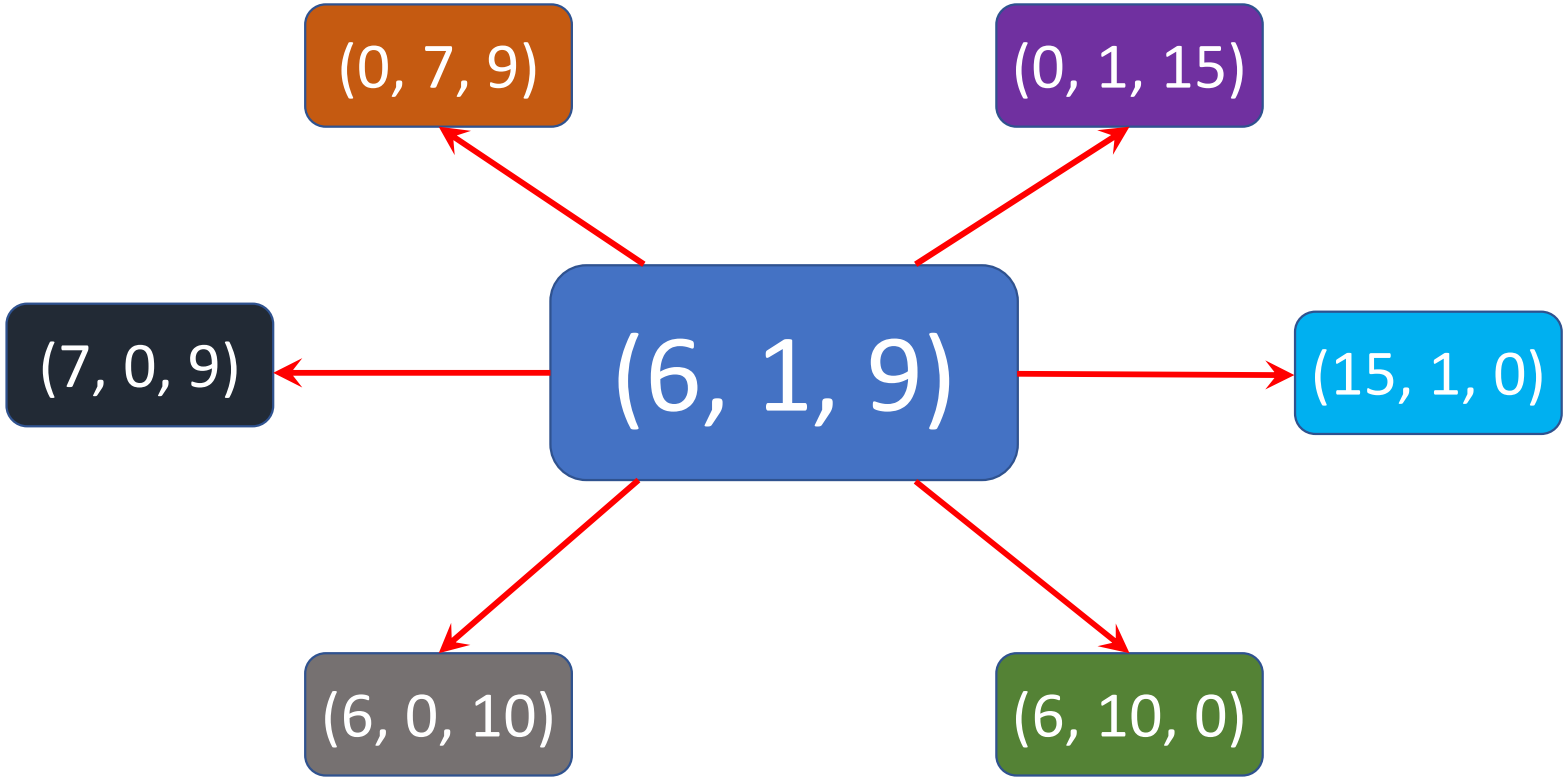
I 접근 – 문제의 탐색 영역

물을 붓는 행위 ➔ (a, b, c) 상태가 (a', b', c')로 바뀐다.

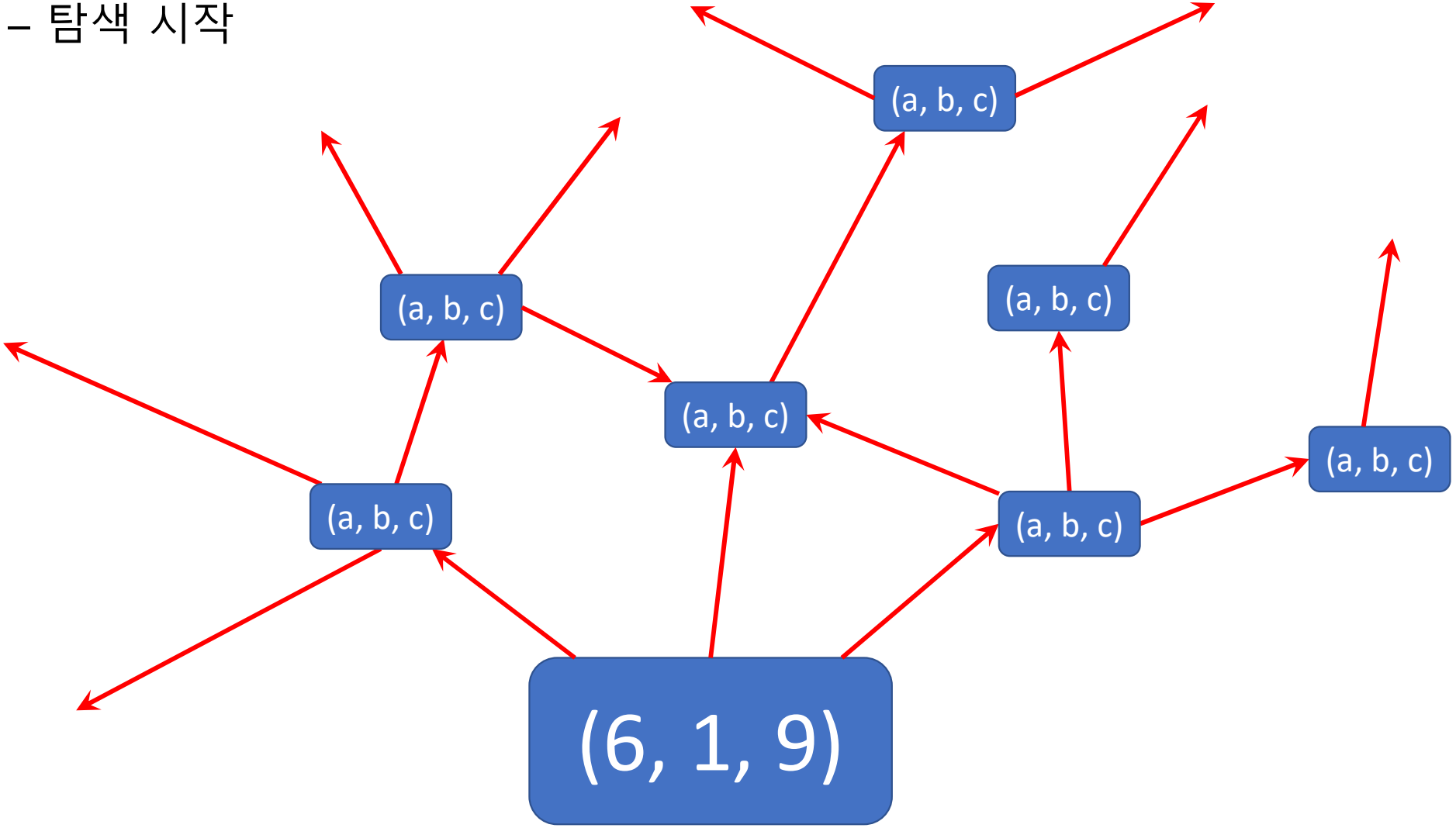
그렇다면, 한 가지 상태가 "정점"이고 물을 부어서 다른 상태로 이동하는 것을 "간선"으로 해보자.



I 접근 – 새로운 그래프 정의



I 접근 – 탐색 시작



I 시간, 공간 복잡도 계산하기

<격자형 그래프>

정점: $O(8 * 10^6)$

간선: $O(8 * 10^6 * 6)$

BFS 이든 DFS 이든, 시간 복잡도는 모두 $O(8 * 10^6)$

I 구현

```java
// 물통의 현재 상태와 물을 붓는 행위를 관리하는 구조체
class State{
    int[] X;
    State(int[] _X){
        X = new int[3];
        for (int i=0;i<3;i++) X[i] = _X[i];
    }

    State move(int from,int to,int[] Limit){
        // from 물통에서 to 물통으로 물을 옮긴다.
        int[] nX = new int[]{X[0], X[1], X[2]};
        /* TODO */
        return new State(nX);
    }
};
```

```java
// 물통 탐색 시작~
static void bfs(int x1, int x2, int x3) {
    Queue<State> Q = new LinkedList<>();

    // BFS 과정 시작
    /* TODO */
}

static void pro() {
    bfs(0, 0, Limit[2]);
    // 정답 계산하기
    /* TODO */
}
```

BOJ 14502 – 연구소

난이도: 4

$3 \leq$ 지도의 크기 $N, M \leq 8$
$2 \leq$ 바이러스의 개수 ≤ 10

2	0	0	0	1	1	0
0	0	1	0	1	2	0
0	1	1	0	1	0	0
0	1	0	0	0	0	0
0	0	0	0	0	1	1
0	1	0	0	0	0	0
0	1	0	0	0	0	0

2	1	0	0	1	1	0
1	0	1	0	1	2	0
0	1	1	0	1	0	0
0	1	0	0	0	1	0
0	0	0	0	0	1	1
0	1	0	0	0	0	0
0	1	0	0	0	0	0

정답: 27

I 문제 파악하기 – 정답의 최대치

2	2	2	1	0	0	0
1	1	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

단지 개수 최댓값: $O(N\text{^}2)$

I 접근 – 1. 바이러스에 노출된 지역 확인

먼저, 바이러스가 얼마나 퍼질 수 있는 지 확인하는 방법은?

“바이러스”에서 “갈 수 있는” 칸들을 찾는다? ➔ **탐색**

2	0	0	0	1	1	0
0	0	1	0	1	2	0
0	1	1	0	1	0	0
0	1	0	0	0	0	0
0	0	0	0	0	1	1
0	1	0	0	0	0	0
0	1	0	0	0	0	0

I 접근 – 2. 완전 탐색을 제일 먼저 확인해보기!

세 칸을 막을 수 있는 모든 경우의 수는 얼마나 될까?

2	0	0	0	1	1	0
0	0	1	0	1	2	0
0	1	1	0	1	0	0
0	1	0	0	0	0	0
0	0	0	0	0	1	1
0	1	0	0	0	0	0
0	1	0	0	0	0	0

I 접근 – 2. 완전 탐색을 제일 먼저 확인해보기!

빈 공간: $B \leq 64$

막을 수 있는 방법: $\binom{B}{3} = {}_{3}^{B}\mathrm{C} \leq 41K$

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	$O({}_{M}^{N}\mathrm{C}) = O\left(\frac{N!}{M!\,(N-M)!}\right)$	$O(M)$

Remind. 완전 탐색

I 접근 – 정리

1. $\binom{B}{3}$ 가지 경우만큼 직접 벽을 세운다. (완전 탐색) ➔ 약 $41K$ 번
2. 매번 직접 “탐색”을 통해서 바이러스로 부터 안전한 구역 확인하기 ➔ $O(T)$
3. 탐색 방법에 따라서 $O(41K * T)$ 만큼의 시간이 걸릴 것이다.

I 접근 – 시작점이 여러 개인 탐색 (Multisource BFS)

2. 매번 직접 “탐색”을 통해서 바이러스로 부터 안전한 구역 확인하기 ➔ $O(T)$

2	0	0	0	1	1	0
0	0	1	0	1	2	0
0	1	1	0	1	0	0
0	1	0	0	0	0	0
0	2	0	0	0	1	1
0	1	0	0	0	2	0
0	1	0	0	0	0	0

Queue

?	?	?	?

I 접근 – 시작점이 여러 개인 탐색 (Multisource BFS)

<Multisource BFS>

2	0	0	0	1	1	0
0	0	1	0	1	2	0
0	1	1	0	1	0	0
0	1	0	0	0	0	0
0	2	0	0	0	1	1
0	1	0	0	0	2	0
0	1	0	0	0	0	0

Queue | (1,1) | (2,6) | (5,2) | (6,6) |

모든 시작점을 **전부** Queue에 넣은 상태로 BFS를 시작하면 된다.

시간 복잡도는, $O(V+E)$가 유지된다.

I 시간, 공간 복잡도 계산하기

1. $\binom{B}{3}$ 가지 경우만큼 직접 벽을 세운다. (완전 탐색) ➔ 약 $41K$ 번
2. 매번 직접 "탐색"을 통해서 바이러스로 부터 안전한 구역 확인하기 ➔ $\boldsymbol{O(N^2)}$
3. 총 시간 복잡도는 $O(41K * N^2 \cong 41K * 64 = 260만)$ 이다.

I 구현

```
// 바이러스 퍼뜨리기!!
static void bfs() {
    Queue<Integer> Q = new LinkedList<>();

    // 모든 바이러스가 시작점으로 가능하니까, 전부 큐에 넣어준다.
    /* TODO */

    // BFS 과정
    /* TODO */

    // 탐색이 종료된 시점이니, 안전 영역의 넓이를 계산하고, 정답을 갱신한다.
    /* TODO */
}

// idx 번째 빈 칸에 벽을 세울 지 말 지 결정해야 하고, 이 전까지 selected_cnt 개의 벽을 세웠다.
static void dfs(int idx, int selected_cnt) {
    if (selected_cnt == 3) {  // 3 개의 벽을 모두 세운 상태
        /* TODO */
        return;
    }
    if (idx > B) return;  // 더 이상 세울 수 있는 벽이 없는 상태

    /* TODO */
}
```