

BFS

BFS(Breadth First Search)

: 다차원 배열에서 각 칸을 방문할 때 너비를 우선으로 방문하는 알고리즘

logic

1. 시작하는 칸을 큐에 넣고 방문했다는 표시를 남김
2. 큐에서 원소를 꺼내어 그 칸에 상하좌우로 인접한 칸에 대해 3번을 진행
3. 해당 칸을 이전에 방문했다면 아무 것도 하지 않고, 처음으로 방문했다면 방문했다는 표시를 남기고 해당 칸을 큐에 삽입
4. 큐의 빌 때까지 2번을 반복

모든 칸이 큐에 1번씩 들어가므로 시간복잡도는 칸이 N개일 때 $O(N)$.

행이 R, 열이 C라면 $O(RC)$

STL pair

```
#include <bits/stdc++.h>
using namespace std;

int main(void) {
    pair<int,int> t1 = make_pair(10, 13);
    pair<int,int> t2 = {4, 6}; // C++11
    cout << t2.first << ' ' << t2.second << '\n'; // 4 6
    if(t2 < t1) cout << "t2 < t1"; // t2 < t1
}
```

first 값 비교 후 second 값 비교

자주 하는 실수

1. 시작점에 방문했다는 표시를 남기지 않는다. 두 번 방문할 수 있음
2. 큐에 넣을 때 방문했다는 표시를 하는 대신 큐에서 빼낼 때 방문했다는 표시를 남겼다.
여러번 들어갈 수 있음
3. 이웃한 원소가 범위를 벗어났는지에 대한 체크를 잘못했다. 창조 오류

연습문제 1926 ... BFS - Flood Fill

1. 상하좌우로 연결된 그림의 크기를 알아내기

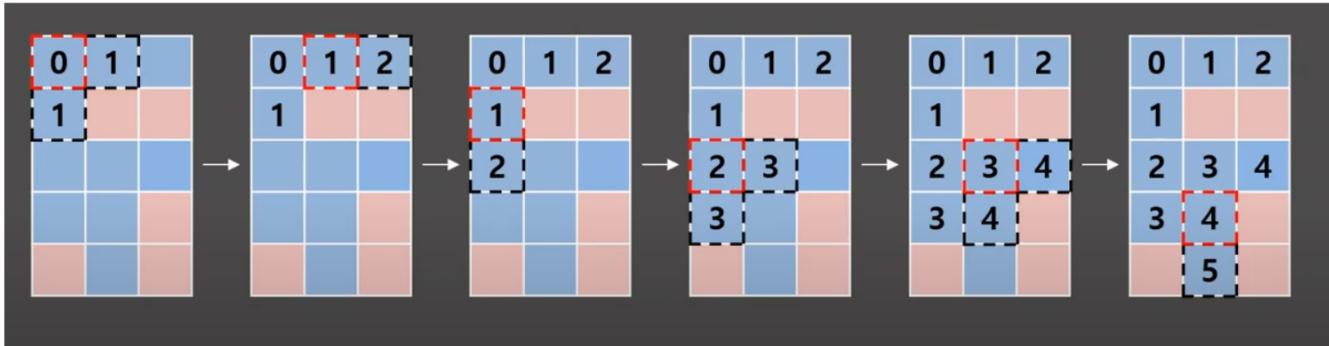
→ 큐에서 pop되는 것의 개수를 세면 됨

2. 도화지에 있는 모든 그림을 찾아내기

→ $v[i][j] == 0$ 이고 $board[i][j] == 1$ 인
것을 $BFS(,)$ 돌리면 됨

각 파란 칸은 큐에 딱 한 번씩만 들어가서
시간복잡도는 칸의 개수만큼 $O(mn)$

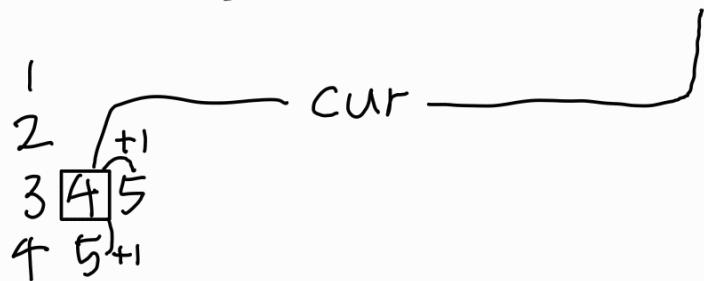
연습문제 2178 ... 다차원 배열에서의 거리 측정



시작점과의 거리를 전부 계산할 수 있음

거리를 저장할 dist 배열을 -1로 초기화하면 visited 배열을 또 만들지 않아도 된다

$dist[nx][ny] = dist[curr.first][curr.second] + 1$
변수를 따로 저장하는 게 아니라 기준에서 +1 해야 함

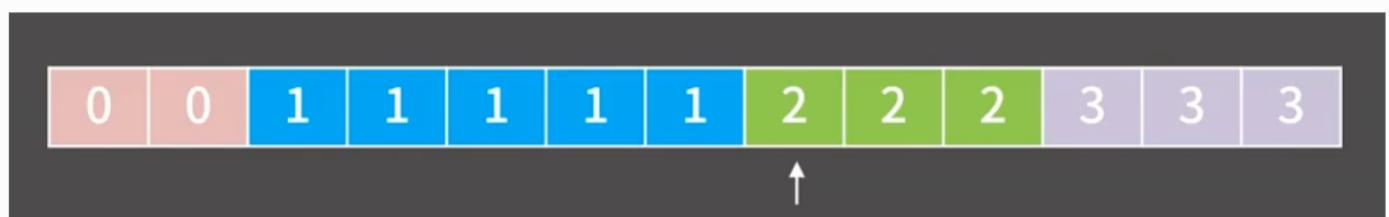
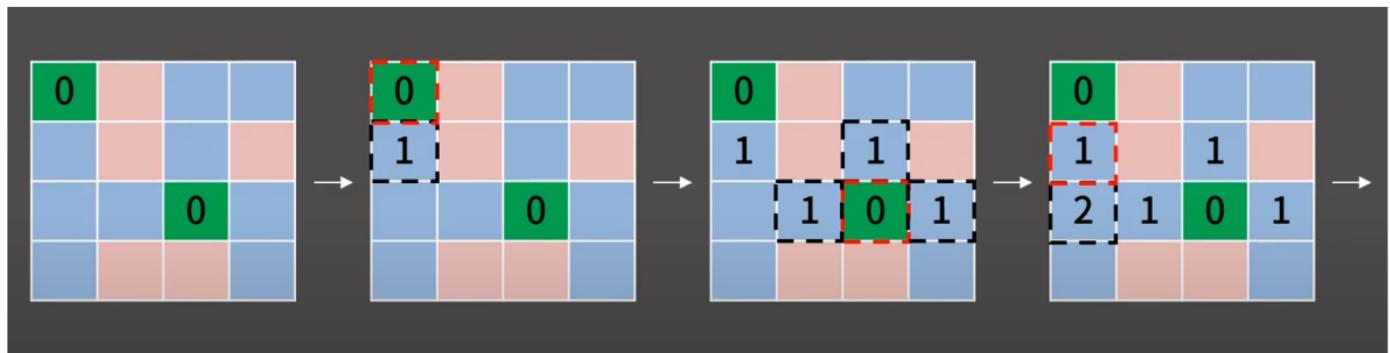


연습문제 7576 ... 시작점이 여러개일 때

익은 토마토가 한 개라면 위의 미로 문제와 똑같이
풀면 된다

익은 토마토가 한 개가 아닐 때 각 익은 토마토에 대해
BFS를 돌리면 BFS $O(NM)$ 익은 토마토는 최대 NM 개
 $\rightarrow O(N^2M^2) \dots \text{TLE}$

그냥 익은 토마토를 다 큐에 넣고 시작하면 된다



큐에 쌓이는 순서는 반드시 거리순

연습문제 7569 ... 3차원 토마토

STL tuple

```
#include<bits/stdc++.h>
using namespace std;
int main() {
    tuple<int, int, int, int> t1=make_tuple(1,2,3,4);
    tuple<int, int, int> t2 = {8,6,4};

    cout << get<0>(t1) << '\n'; // 1
    cout << get<1>(t1) << '\n'; // 2

    cout << get<2>(t2) << '\n'; // 4
    get<2>(t2) = 3; // 4 -> 3
    cout << get<2>(t2) << '\n'; // 3
}
return 0;
```

연습문제 4/79 ... 시작점이 두 종류

내 풀이 - 앞에서 본 큐에 쌓이는 순서를 활용함

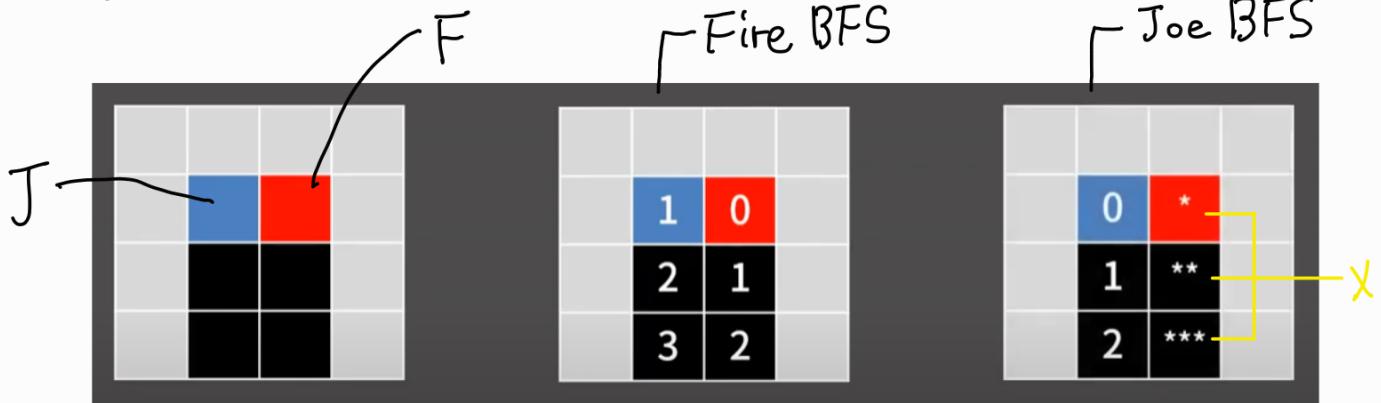
```
// 알고리즘 구상
// 처음에 dist를 -1로 초기화 (visited를 설정하지 않아도 됨)
// board에서는 초기 Joe, Fire 위치와 변하지 않는 벽의 위치만 참조
// 이후 변하는 Joe, Fire는 dist에서 관리
// Joe와 Fire는 같이 움직이지만 큐에 들어가는 순서는 Joe 다음 Fire이기 때문에
// 처음에 모든 Joe와 Fire를 넣고 다음 Joe가 오면 1분이 지난 것을 이용
// Joe는 -1(not visited and not fire)이고 벽이 아닌 칸만 갈 수 있고
// Fire는 not fire인 칸과 벽이 아닌 칸을 갈 수 있다
// 그래서 dist[cur.first][cur.second] 이게 큐에 들어온 칸인데 이게 0이상이면
// Joe Fire 다음 Joe가 들어온 것이라고 1분이 지난 것
// 그래서 여기서 board의 끝자락에 있는지 조사하면 된다
//

// 오답 1 - 무한 루프 - 메모리 초과
// 이미 Fire인 칸을 배제하지 않거나 Joe의 visited를 체크하지 않아서 메모리 초과가 났다
// 256MB를 넘어간 줄 알고 이상한 짓 많이 했는데 알고보니 무한루프였다....
//
// 오답 2 - nx, ny, r, c 헷갈림 - 런타임 에러
// 그냥 row와 column 받고
// board[row][column] 자리로 기억하자
// board[nx][ny]
// 그냥 row nx, column ny 대응
//

// 질문
// 이게 최소시간을 보장할 수 있는지 고민해 봤는데
// bfs를 돌리고 그 큐에 들어간 것들 중 벽에 바로 도달하자마자(Joe Fire Joe에서)
// return 하니까 최소시간이지 않을까 ...?

// 큐에 원소들은 거리순으로 들어가니 최초로 탈출한 시간이 최소시간이다
```

모범 풀이 - Fire와 Joe에 대한 BFS를 각각 돌려 처리



일단 Fire BFS 돌려서 각 칸에 불이 전파되는 시간을
구한다

Joe BFS 돌려서 이동시키는데, 갈 수 있는 시간이
그 칸 이상이면 이동 불가

☆ 단, Fire와 Joe 같이 Fire만 Joe에게 영향을
줄 수 있는 경우에만 이 방식이 유효하다
서로에게 영향을 주는 경우에는 따로 BFS 하면 안 됨

연습문제 1697 - 차원에서의 BFS

0	1	2	3	4	5	6	7	8	9	10
		2	1	0	1		2		1	

이런식으로 생각하면 쉽다

* BFS 돌리는 범위를 생각해볼 필요가 있는데,

단순히 $0 \leq \leq 100,000$ 이 아니라

더나갈 수도 있다고 생각해야 한다

일단 음수는 아니겠고 100,000이 넘어가면

계속 -1을 해야 한다는 것을 고려해보면

$\times 2 -1 -1 -1 \dots$ 하는 것 보다 $-1 -1 -1 \dots \times 2$

하는 게 빠르다

그래서 $0 \leq \leq 100,000$ 으로 잡는 게 맞다