#18428 감시피하기

문제

문제

*N*x*N* 크기의 복도가 있다. 복도는 1x1 크기의 칸으로 나누어지며, 특정한 위치에는 선생님, 학생, 혹은 장애물이 위치할 수 있다. 현재 몇 명의 학생들은 수업시간에 몰래 복도로 빠져나왔는데, 복도로 빠져나온 학생들은 선생님의 감시에 들키지 않는 것이 목표이다.

각 선생님들은 자신의 위치에서 상, 하, 좌, 우 4가지 방향으로 감시를 진행한다. 단, 복도에 장애물이 위치한 경우, 선생님은 장애물 뒤편에 숨어 있는 학생들은 볼 수 없다. 또한 선생님은 상, 하, 좌, 우 4가지 방향에 대하여, 아무리 멀리 있더라도 장애물로 막히기 전까지의 학생들은 모두 볼 수 있다고 가정하자.

다음과 같이 3x3 크기의 복도의 정보가 주어진 상황을 확인해보자. 본 문제에서 위치 값을 나타낼 때는 (행,열)의 형태로 표현한다. 선생님이 존재하는 칸은 T, 학생이 존재하는 칸은 S, 장애물이 존재하는 칸은 O로 표시하였다. 아래 그림과 같이 (3,1)의 위치에는 선생님이 존재하며 (1,1), (2,1), (3,3)의 위치에는 학생이 존재한다. 그리고 (1,2), (2,2), (3,2)의 위치에는 장애물이 존재한다.

S	O	
S	O	
T	O	S

이 때 (3,3)의 위치에 존재하는 학생은 장애물 뒤편에 숨어 있기 때문에 감시를 피할 수 있다. 하지만 (1,1)과 (2,1)의 위치에 존재하는 학생은 선생님에게 들키게 된다.

학생들은 복도의 빈 칸 중에서 장애물을 설치할 위치를 골라, 정확히 3개의 장애물을 설치해야 한다. 결과적으로 3개의 장애물을 설치하여 모든 학생들을 감시로부터 피하도록 할 수 있는지 계산하고자 한다. *N*x*N* 크기의 복도에서 학생 및 선생님의 위치 정보가 주어졌을 때, 장애물을 정확히 3개 설치하여 모든 학생들이 선생님들의 감시를 피하도록 할 수 있는지 출력하는 프로그램을 작성하시오.

예를 들어 *N*=5일 때, 다음과 같이 선생님 및 학생의 위치 정보가 주어졌다고 가정하자.

	S			T
T		S		
	T			
		T		

이 때 다음과 같이 3개의 장애물을 설치하면, 모든 학생들을 선생님의 감시로부터 피하도록 만들 수 있다.

	S		O	T
T	O	S		
		O		
	T			
		T		

N*N 의 복도에 학생들과 선생님들이 배치되어있음. 선생님들은 상하좌우 방향으로 감시를 할 수 있고, 학생들은 장애물 뒤에 숨어서 감시를 피할 수 있다.

장애물 3개를 설치하여 학생들 모두가 선생님의 감시를 피할 수 있는지를 판단하라.

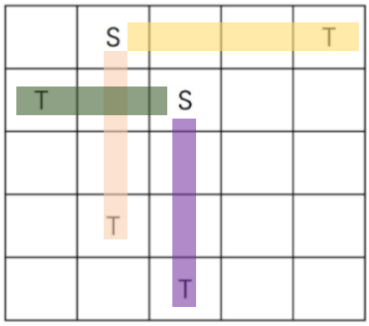
(3≤N≤6)

실패 사례

풀이시도 - 실패

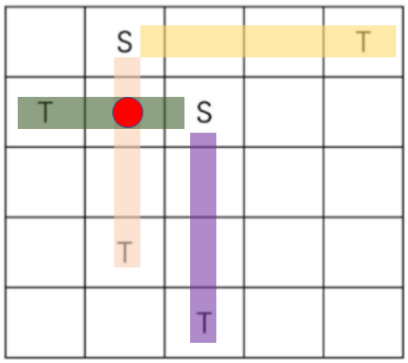
규칙을 찾는 문제로 파악함

- T를 기준으로 상하좌우 탐색 + 학생 찾기

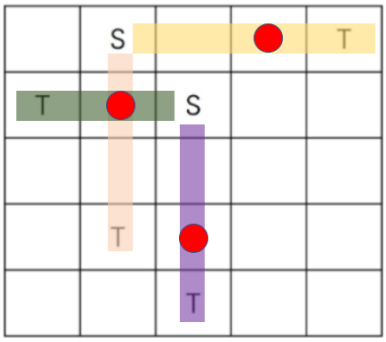


- 사용할 수 있는 장애물이 제한적이므로, 최대한 효율적으로 장애물을 배치시켜야한다고 판단.

  ⇒ 장애물 하나로 여러 학생을 막을 수 있는 위치에 장애물 우선 배치하기



- 모든 T를 탐색한 후, 학생들이 가려지지 않은 위치에 장애물 추가 배치



- 총 배치한 장애물이 3개 이하 인지 판단.
  - 이미 위의 로직으로 모든 학생들을 가린 상태.
  - 따라서, 3개보다 작은 경우 빈 공간에 장애물을 배치하여 장애물이 3개가 되도록 만들어 주기만 하면 된다.

구현

- T의 상하좌우를 DFS로 탐색 → 만약 방문한 위치에 S가 있다면 탐색 중지 + 되돌아 오면서 마킹
- 되돌아올 때, 이미 마킹되어있는 칸을 만났다면 장애물 우선 배치

▼ 코드

```
#include <iostream>
#include <vector>
#include <queue>

using namespace std;

int N;
int i,j;
int checkedSNum = 0;
int haveToCheck=0;
int obstacle= 0;
int sNum=0,tNum=0;

vector<pair<int,int>> surround = {make_pair(1,0),make_pair(-1,0),make_pair(0,1),make_pair(0,-1)};
vector<vector<int>>hall;
int dfs(int x,int y,pair<int,int>direction,int idx);

int main()
{
    char c;

    cin >>N;
    queue<pair<int,int>>teachers;

    for(i=0;i<N;i++){
         hall.push_back(vector<int>(N));
    }

    // init
    for(i=0;i<N;i++){
        for(j=0;j<N;j++){
```

```
            cin>>c;
            if(c=='X')
                hall[i][j] =0;
            else if (c=='T'){
                hall[i][j]=-1;
                tNum++;
                teachers.push(make_pair(i,j));
            }
            else{
                hall[i][j]=-2;
                sNum++;
            }
        }
    }


    int idx=1;
    while(!teachers.empty()){
        pair<int,int> teacher = teachers.front();
        int x = teacher.first;
        int y = teacher.second;

        // 상하좌우 조회
        for(i=0;i<4;i++){
            x+=surround[i].first;
            y+=surround[i].second;

            if (x < 0 || x>=N || y<0 || y>=N)
                continue;
            else if(hall[x][y]==-2){
                cout <<"NO";
                return 0;
            }else
                dfs(x,y,surround[i],idx);
        }
        idx++;
        teachers.pop();
    }

    for(i=0;i<N;i++){
        for(j=0;j<N;j++){
            if(hall[i][j] == -1){
                cout << "T";
            }
            else if(hall[i][j] == -2){
                cout << "S";
            } else if(hall[i][j] == -3){
                cout << "X";
            }
            else
                cout<<hall[i][j];
        }
        cout<<endl;
    }

    cout <<haveToCheck + obstacle <<"개" <<'\n';
    if(haveToCheck + obstacle > 3)
        cout<<"NO";
    else
        cout << "YES";
    return 0;
}

int dfs(int x,int y,pair<int,int>direction, int idx){
    if (x < 0 || x>=N || y<0 || y>=N)
        return 0;
```

```
    int result;

    if(hall[x][y]==-2){ // 학생
        haveToCheck++;
        return -2;
    }
    else{
        result = dfs(x+direction.first,y+direction.second,direction,idx);
    }

    if(result != -2)
        return 0;

    if(hall[x][y]!=0 && hall[x][y]!=-1 ){
        hall[x][y]=-3;
        haveToCheck-=2;
        obstacle++;
        return 0;
    }else if(hall[x][y]==-3){
        haveToCheck--;
        return 0;
    }else if( hall[x][y] ==0){
        hall[x][y] = idx;
        return -2;
    }

}
```

실패

코드는 의도한 바대로 구현되었으나, 실패

찾지 못한 반례가 존재하는 것으로 추측..

성공 사례

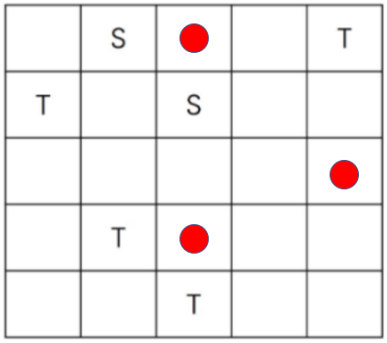
검색 후 재시도 - 완전탐색 사용(Brute Force)

- 주어진 이차원 배열의 크기가 매우 작음 (N: 3~6)
- 문제에서 장애물을 3개'만' 사용해야 한다고 명시

⇒ 빈칸 중 3개를 골라 장애물을 배치한 후, 모든 학생들이 가려지는지 판단을 한다고 가정해도, 빈칸을 고르는 경우의 수가 7000개에 못미침. ($_{35}C_3 = 6545$)

⇒ 경우의 수가 적은 축에 속하므로 **완전탐색(Brute Force)** 적용가능

완전탐색을 이용한 문제해결 방법



- 존재하는 빈칸 중 3개를 골라 장애물 배치
- 배치가 완료되면 T를 기준으로 상하 좌우 탐색 ⇒ 장애물을 만나면 탐색 종료
- 탐색 중 S가 발견되면, 해당 경우는 “NO”에 해당하므로 탐색 중지 → 다른 빈칸 3개를 다시 골라 처음부터 다시 탐색
- S가 탐색되지 않은 경우가 존재하면 “YES” 출력, 모든 경우를 탐색했음에도 해당 경우가 발생하지 않으면 “NO” 출력.

빈칸 3개 고르기 - 조합

존재하는 빈칸 중 3개를 골라야함.

이때, 빈칸을 고르는 순서는 중요하지 않으며, 중복해서 선택하지 않을 것임.

⇒ 조합을 사용하여 선택해야 함.

조합

재귀를 이용해 조합 구현하기

아래의 공식을 활용.

$$ {}_nC_r = {}_{n-1}C_r + {}_{n-1}C_{r-1} $$

《의미》

n개 중 r개 고르기 = 특정 아이템을 포함하지 않고 나머지 중에서 r개 고르기 + 특정 아이템을 포함시킨 뒤 나머지 아이템 중 r-1개 고르기

위 공식 자체에서 재귀의 형태를 찾을 수 있음.

구현 코드

```
vector<int> arr={1,2,3,4,5}; // 전체 집합
vector<int> comb(3); // 조합으로 뽑은 것들을 보관하는 배열

// now: 전체 배열에서 현재 뽑을까 말까 고민하는 인덱스
// idx: 조합을 담는 배열의 인덱스
// r : 선택해야할 숫자들의 갯수
void combination(int now,int idx, int r){
    if(r==0){// 재귀 끝남
        for(int i=0;i<3;i++)
          cout<< comb[i]; // 한개의 조합 세트 나옴
    }
    else if( now==arr.size())
        return ; // 하나도 안뽑고 끝까지 옴
    else{
        // 뽑음
        comb[idx] = arr[now];
        combination(now+1,idx+1,r-1);

        // 안뽑음
        combination(now+1,idx,r);
    }
}

combination(0,0,3);
```

Solution Code

- 존재하는 빈칸 중 3개를 골라 장애물 배치 - 조합 알고리즘
- 배치가 완료되면 T를 기준으로 상하 좌우 탐색 ⇒ 장애물을 만나면 탐색 종료 - DFS
- 탐색 중 S가 발견되면, 해당 경우는 “NO”에 해당하므로 탐색 중지 → 다른 빈칸 3개를 다시 골라 처음부터 다시 탐색
- S가 탐색되지 않은 경우가 존재하면 “YES” 출력, 모든 경우를 탐색했음에도 해당 경우가 발생하지 않으면 “NO” 출력.

```
#include <iostream>
#include <vector>

using namespace std;

int N;
int i,j;
int sNum=0,tNum=0;

vector<pair<int,int>> surround = {make_pair(1,0),make_pair(-1,0),make_pair(0,1),make_pair(0,-1)};
vector<vector<char>>hall;
vector<pair<int,int>>teachers;
vector<pair<int,int>> empties;
vector <pair<int,int>> obstacles(3,make_pair(0,0));

int dfs(int x,int y,pair<int,int>direction);
void combination(int now,int oIdx, int r);

int main()
{
```

```
    cin >>N;
    for(i=0;i<N;i++){
         hall.push_back(vector<char>(N));
    }

    // init
    for(i=0;i<N;i++){
        for(j=0;j<N;j++){
            cin>> hall[i][j];
            if(hall[i][j]=='T'){
                tNum++;
                teachers.push_back(make_pair(i,j));
            }
            else if (hall[i][j]=='X'){
                empties.push_back(make_pair(i,j));
            }
        }
    }

    combination(0,0,3);
    cout<<"NO";

    return 0;

}

void combination(int now,int oIdx, int r){
    if(r==0){// 재귀 끝남 (3개 다 뽑음)
        int k,l;

        for( k=0; k<3;k++){
            int x=obstacles[k].first;
            int y = obstacles[k].second;
            hall[x][y] = 'O';
        }

        bool canFindStudent= false;
        for(k=0;k<tNum;k++){
            int x=teachers[k].first;
            int y = teachers[k].second;

            for(l=0;l<4;l++){
                int result;
                result=dfs(x,y,surround[l]);
                if(result == -1){ // 학생이 발견됨.
                    canFindStudent=true;
                    break;
                }

            }
        }
        if(!canFindStudent) {
            cout<<"YES";
            exit(0);
        }

         for(k=0;k<3;k++){
             hall[obstacles[k].first][obstacles[k].second] ='X';
        }

    }
    else if( now==empties.size())
        return ; // 하나도 안뽑고 끝까지 옴
    else{
        // 뽑음
        obstacles[oIdx] = empties[now];
```

```
        combination(now+1,oIdx+1,r-1);

        // 안뽑음
        combination(now+1,oIdx,r);
    }

}

int dfs(int x,int y,pair<int,int>direction){
    if (x < 0 || x>=N || y<0 || y>=N)
        return 0;

    int result;

    if(hall[x][y]=='S'){ // 학생
        return -1;
    }
    else if(hall[x][y]=='O')
        return 0;
    else{
        result=dfs(x+direction.first,y+direction.second,direction);
        if(result ==-1)
            return -1;
        else
            return 0;
    }


}
```

깨달은 점📝

1. 경우의 수가 적을 때는 완전탐색(Brute Force)도 고려해보기
2. 조합 알고리즘 구현방법
3. 문제를 잘 읽자!