

JooHyun – Lee (comkiwer)

# TS좌석예약

Hancom Education Co. Ltd.

# Problem

영화제가 열리는 지역에 영화관이  $N$  개 있다.

각 영화관은 1 번부터  $N$  번까지 고유 번호를 가진다

$N$  개 영화관의 좌석을 예약하는 서비스를 시뮬레이션 하려고 한다.

예약을 같이한 친구들은 같은 영화관에서 상하좌우 중 한 방향 이상 연결된 좌석에 자리를 붙여서 예약을 하고 싶어한다.

좌석 예약할 때 이를 고려해야 한다.

한 개의 영화관에는  $10 * 10$  개의 좌석이 있다. 좌석 번호는 왼쪽에서 오른쪽 방향으로,

앞에서 뒤쪽 방향으로 순서대로 번호가 붙어있다.

각 좌석의 번호는 [Fig. 1] 과 같다.

1	2	3	4	5	6	7	8	9	10
11	.	.	.	.	.	.	.	.	20
.									.
.									.
.									.
.									.
.									.
.									.
81	.	.	.	.	.	.	.	.	90
91	92	93	94	95	96	97	98	99	100

[Fig. 1]

영화관의 좌석 예약은 예약 ID 와 예약하는 좌석 개수  $K$  로 주어지고, 예약 방법은 다음과 같다.

1. 1 번 영화관부터  $N$  번 영화관까지 차례대로 아래 내용을 확인하여 예약을 한다.

( 한 개의 영화관이 선택되어 자리 예약이 된다. )

2. 예약하는 좌석들은 모두 상하좌우 중 한 방향 이상으로 서로 연결되어 있어야 한다.

만약, 영화관의 상하좌우 연결되는 좌석이  $K$  개 이상이 없어 좌석 예약을 할 수 없을 경우,

예약은 다음 영화관으로 넘어간다. ( 예제 Order 19. 참고 )

3. 상하좌우 연결되고,  $K$  개 이상 비어 있는 좌석들을 하나로 묶어 빈좌석 묶음이라 한다.
4. 빈좌석 묶음이 여러 개 일 경우, 좌석 번호가 가장 작은 좌석이 포함된 빈좌석 묶음을 선택한다.
5. 첫번째 예약 좌석은 선택한 빈좌석 묶음에서 번호가 가장 작은 좌석이다.
6. 다음 예약 좌석은 새롭게 예약된 좌석들과 상하좌우 중 한 방향 이상으로 연결되어야 하며,  
그 중 번호가 가장 작은 비어 있는 좌석이다. ( [Fig. 2] 번호 순서 참고 )
7.  $K$  개의 좌석을 모두 예약할 때까지 6번을 반복한다.

							1	2	
						4	3	5	
	15					6	7	8	
	14	13	12	11	10	9			

[Fig. 2]

[Fig. 2] 와 같이 한 개의 영화관은  $10 * 10$  좌석으로 되어 있다.

(각각의 좌석 번호는 [Fig. 1] 참고)

회색은 이미 예약이 되어 예약 불가능한 자리이고, 흰색은 예약 가능한 자리이다.

15개의 좌석을 예약할 경우 아래와 같은 순서대로 15개의 좌석이 예약된다.

(첫번째 예약 좌석의 번호는 18 이다.)

좌석 번호가 1 ~ 4 인 좌석은 상하좌우 연결된 예약 가능한 좌석이 4 개만 있기 때문에 제외한다.

							1	2	
						4	3	5	
	15					6	7	8	
	14	13	12	11	10	9			

[Fig. 2]

위와 같은 좌석 예약 서비스 시뮬레이션 프로그램을 구현하라.

※ 아래는 User Code 부분에 작성해야 하는 API 의 설명이다.

`void init(int N)`

각 테스트 케이스의 처음에 호출된다.

1 번부터 N 번까지 N 개의 영화관이 있다.

(한 개의 영화관은 10 \* 10 개의 좌석이 있다.)

N 개의 영화관은 모두 비어 있어 예약 가능하다.

*Parameters*

N : 영화관의 개수 (  $1 \leq N \leq 2,000$  )

### Result reserveSeats(int mID, int K)

mID 예약 번호로 K 개의 좌석을 예약하고, 아래의 값을 반환한다.

Result 구조 내의 id = 예약된 영화관 번호,  
num = 예약 좌석 중에서 가장 작은 좌석 번호를 반환한다.

모든 영화관이 K 개 이상의 상하좌우 연결되는 좌석이 없어 예약이 실패하는 경우,  
Result 구조 내의 id = 0, num = 0 을 반환한다.

좌석 예약 방법은 본문 설명과 같다.

mID 예약 번호는 이전에 해당 함수에서 호출된 적이 없다.

### Parameters

mID : 예약 번호 (  $1 \leq \text{mID} \leq 50,000$  )

K : 좌석 예약 개수 (  $1 \leq K \leq 50$  )

### Returns

Result 구조 내의 id = 예약된 영화관 번호, num = 예약 좌석 중에서 가장 작은 좌석 번호.  
예약 실패할 경우 Result 구조 내의 id = 0, num = 0



### Result cancelReservation(int mID)

mID 예약 번호로 예약된 좌석을 모두 취소하고, 아래의 값을 반환한다.

Result 구조 내의 id = 취소된 영화관 번호,

num = 취소된 좌석 번호를 모두 더한 값을 반환한다.

mID 예약 번호로 예약된 좌석이 있음을 보장한다.

### Parameters

mID : 취소하려는 예약 번호 (  $1 \leq mID \leq 50,000$  )

### Returns

Result 구조 내의 id = 취소된 영화관 번호, num = 취소된 좌석 번호를 모두 더한 값

### [제약사항]

1. 각 테스트 케이스 시작 시 `init()` 함수가 호출된다.
2. `reserveSeats()` 에서 좌석 예약은 한 개의 영화관에서 예약 되어야 한다.
3. 각 테스트 케이스에서 `reserveSeats()` 함수의 호출 횟수는 20,000 이하이다.
4. 각 테스트 케이스에서 `cancelReservation()` 함수의 호출 횟수는 10,000 이하이다.

# Problem analysis

## Problem analysis : 예제

## TS좌석예약

## 초기에 3개의 극장이 있다.

Order	Function	Return
1	init(3)	

Theater1

[illegible]

Theater1

[illegible]

Theater3

[illegible]

## Problem analysis : 예제

## TS좌석예약

순서2 ~ 순서6까지 예약 상황이다.

Order	Function	Return
2	reserveSeats(1, 17)	id = 1, num = 1
3	reserveSeats(21, 2)	id = 1, num = 18
4	reserveSeats(33, 2)	id = 1, num = 20
5	reserveSeats(14, 2)	id = 1, num = 21
6	reserveSeats(15, 4)	id = 1, num = 23

Theater1

[illegible]

Theater1

[illegible]

Theater3

[illegible]

## Problem analysis : 예제

## TS좌석예약

순서7 ~ 순서10까지 예약 상황이다.

Order	Function	Return
7	reserveSeats(65, 3)	id = 1, num = 27
8	reserveSeats(27, 2)	id = 1, num = 31
9	reserveSeats(58, 2)	id = 1, num = 33
10	reserveSeats(39, 4)	id = 1, num = 35

Theater1

[illegible]

Theater1

[illegible]

Theater3

[illegible]

## TS좌석예약

순서11 ~ 순서14까지 예약 취소 상황이다.

Order	Function	Return
11	cancelReservation(21)	id = 1, num = 37
12	cancelReservation(14)	id = 1, num = 43
13	cancelReservation(65)	id = 1, num = 84
14	cancelReservation(58)	id = 1, num = 67

[illegible][illegible][illegible]

## TS좌석예약

(붉은색 숫자는 15개 좌석 예약할 때 예약 순서이다.)

Order	Function	Return
15	reserveSeats(22, 15)	id = 1, num = 18

검은색 숫자가 채워진 자리는 이미 예약된 자리이므로 예약이 불가능한 자리이고  
그렇지 않은 자리는 예약 가능한 자리이다.

Theater1[illegible]Theater1[illegible]Theater3[illegible]



## Problem analysis : 예제

## TS좌석예약

순서16 까지 예약 상황이다.

Order	Function	Return
16	reserveSeats(111, 13)	id = 1, num = 41

Theater1

[illegible]

Theater1

[illegible]

Theater3

[illegible]

## TS좌석예약

순서17, 18 까지 취소 상황이다.

Order	Function	Return
17	cancelReservation(15)	id = 1, num = 98
18	cancelReservation(39)	id = 1, num = 146

[illegible][illegible][illegible]

## Problem analysis : 예제

## TS좌석예약

순서19, 20 까지 예약 상황이다.

Order	Function	Return
19	reserveSeats(100, 48)	id = 2, num = 1
20	reserveSeats(51, 2)	id = 1, num = 21

Theater1

[illegible]

Theater1

[illegible]

Theater3

[illegible]

## Problem analysis : 예제

## TS좌석예약

순서21 ~ 24 까지 예약 상황이다.

Order	Function	Return
21	reserveSeats(110, 18)	id = 1, num = 61
22	reserveSeats(511, 50)	id = 2, num = 49
23	reserveSeats(32, 50)	id = 3, num = 1
24	reserveSeats(1133, 1)	id = 1, num = 23

Theater1

[illegible]

Theater1

[illegible]

Theater3

[illegible]

## Problem analysis : 예제

## TS좌석예약

순서25 ~ 27 까지 예약 및 취소 상황이다.

Order	Function	Return
25	reserveSeats(447, 37)	id = 3, num = 51
26	cancelReservation(100)	id = 2, num = 1176
27	reserveSeats(69, 50)	id = 0, num = 0

Theater1

[illegible]

Theater1

[illegible]

Theater3

[illegible]

## TS좌석예약

순서28 까지 예약 상황이다.

Order	Function	Return
28	reserveSeats(59, 19)	id = 1, num = 79

[illegible][illegible][illegible]

- 한개의 영화관에는 100개의 좌석이 있다.
- 영화관의 수는 최대 2,000개 이다.
- 한번의 예약으로 채워지는 좌석수는 최대 50개이다.
- 하나의 TC에서 예약은 최대 20,000 번 이루어진다. => reserveSeats(int mID, int K)
- 하나의 TC에서 취소는 최대 10,000 번 이루어진다. => cancelReservation(int mID)
- 예약 취소는 예약 id로 관리한다면 어렵지 않게 처리될 것으로 보인다.
- 그런데 예약을 처리하는 문제는 어떨까?
- 매 예약 쿼리에 대하여 순차 탐색을 실행하는 경우 시간복잡도는  $100(\text{좌석수}) * 2,000(\text{영화관 수}) * 20,000(\text{함수 호출 수})$  이므로 시간에 맞추기 힘들다.  
어떻게 검색 효율을 높일 수 있을까?

- (1순위. 연결된 빈 좌석의 수 asc,  
2순위. 영화관 번호 asc  
3순위. 좌석 번호 asc) 로 데이터를 정렬된 상태로 관리하는 것은 어떨까?

: reserveSeats(int mID, int K) 호출시 lower\_bound()를 이용하여  
K이상을 만족하는 첫 번째 자료를 찾을 수 있다.

그런데 이 자료가 문제에서 요구한 자료라고 할 수 있을까?

문제에서 요구하는 자료는 좌석이 K이상 확보된 경우 중에서  
영화관 번호가 가장 작은 것이다.

따라서 위와 같은 방법으로 자료를 관리하는 것이 크게 도움이 되지 않는다.  
오히려 자료 관리를 위한 추가적인 어려움이 클 수 있다.

그렇다면 어떻게 할 것인가?



# Solution sketch

- 예약 ID는 1~50,000이므로 해시가 필요 없다.  
예약 ID를 바로 배열의 인덱스로 사용할 수 있다.
- 또한 한 예약에서 최대 좌석수는 50을 넘지 않으므로  
예약 시 좌석 정보를 저장해 둔다면 좀 더 효율적으로  
예약 취소를 수행할 수 있다.
- 따라서 예약 시에 적절한 처리가 이루어진다면  
예약 취소는 어렵지 않게 처리할 수 있다.

```
struct Reserve {  
    int hid;  
    vector<pii> reservedSeats;  
}reserveArr[50001];
```

- 예약은 어떻게 처리할 수 있을까?
- 매 예약 쿼리에 대하여 단순한 순차 탐색을 실행하는 경우 시간복잡도는  $100(\text{좌석수}) * 2,000(\text{영화관 수}) * 20,000(\text{함수 호출수})$  이지만 각 영화관마다 빈 자리 수를 관리한다면 검색 효율을 좀더 높일 수 있다.

$1,000(\text{사용중인 영화관 평균 수}) * 20,000(\text{함수 호출 수})$  가 되어 문제를 해결할 수 있게 된다.

- 필요한 각 자료를 정의해 보면 다음과 같다.
- 예약 정보를 관리하기 위하여 다음 클래스를 정의 할 수 있다.

```
struct Reserve {  
    int hid;                // 예약한 영화관 번호          : 취소시 사용  
    vector<pii> reservedSeats; // hid극장에서 예약한 좌석 목록 : 취소시 사용  
}reserveArr[50001];
```

- 영화관 별 좌석 상태를 관리하기 위하여 다음 클래스를 정의 할 수 있다.

```
struct Hall {  
    int hid;                // 영화관 번호  
    int vacant;             // 현재 영화관의 빈 좌석 수  
    int seats[10][10]; // 좌석 상태(사용중이라면 예약 ID를, 빈 곳이라면 0으로 표시)  
}halls[2001];
```

- 이제 각 API함수 별 할 일을 정의 해 보자.

`void init(int N)`

영화관 수 N을 전역변수 `::N`에 저장한다.

각 영화관을 초기화 한다.

```
void init(int N) {  
    ::N = N;  
    for (int i = 1; i <= N; ++i)  
        halls[i] = { i, 100 };  
}
```

Result reserveSeats(int mID, int K)

1번부터 N번 영화관까지 진행하며 다음 작업을 수행한다.

- K개 이상의 연속한 빈 공간이 있는지 알아본다.
- 없다면 다음 영화관으로 넘어간다. 이때, **BFS탐색을 이용할 수 있다.**
- 있다면 예약을 할당한다. 이때, **우선순위가 있는 BFS탐색을 이용할 수 있다.**

```
Result reserveSeats(int mID, int mNum) {  
    ret = { 0, 0 };  
    reserveID = mID;           // 예약 ID  
    targetVacant = mNum;       // 예약하고자 하는 좌석수  
    for (int i = 1; i <= N; ++i) { // 가능한 자리 찾아보기  
        if (halls[i].findSeats()) // BFS탐색, 우선순위BFS탐색  
            break;               // 가능한 자리 찾은 경우  
    }  
    return ret;  
}
```

# Solution sketch

## TS좌석예약

```
struct Hall {
    int hid, vacant, seats[10][10];

    // (r, c)로 시작하는 연결된 구간의 크기가 targetVacant 이상인지 알아보기
    bool isAble(int r, int c) { // BFS
        fr = re = 0;
        que[re++] = { r, c };
        visited[r][c] = ++visCnt; // 출발 위치 방문 체크
        while (fr < re) {
            r = que[fr].first, c = que[fr++].second;
            for (int i = 0; i < 4; ++i) {
                int nr = r + dr[i], nc = c + dc[i];
                if (nr < 0 || nr > 9 || nc < 0 || nc > 9)
                    continue;
                if (seats[nr][nc] || visited[nr][nc] == visCnt)
                    continue;
                que[re++] = { nr, nc };
                visited[nr][nc] = visCnt;
            }
            if (re >= targetVacant) return 1;
        }
        return 0;
    }
};
```

```
void alloc(int r, int c) { // 우선순위 큐를 이용한 BFS탐색
    pq = {};
    pq.push(r * 10 + c); // 1.행 asc, 2.열 asc
    visited[r][c] = ++visCnt;
    reserveArr[reserveID].hid = hid;
    vector<pii>&seatsList = reserveArr[reserveID].seatsList;
    seatsList.clear(); // 예약목록 초기화

    for (int i = 0; i < targetVacant; ++i) {
        r = pq.top() / 10, c = pq.top() % 10;
        seats[r][c] = reserveID; // 예약된 좌석은 예약ID로 채우기
        seatsList.push_back({ r, c }); // 예약좌석목록에 추가
        pq.pop();
        for (int j = 0; j < 4; ++j) { // 인접한 좌석 알아보기
            int nr = r + dr[j], nc = c + dc[j];
            if (nr < 0 || nr > 9 || nc < 0 || nc > 9) continue;
            if (seats[nr][nc] || visited[nr][nc] == visCnt) continue;
            pq.push(nr * 10 + nc);
            visited[nr][nc] = visCnt;
        }
    }
    vacant -= targetVacant;
}
```

```
bool findSeats() {
    if (vacant < targetVacant) return false;
    for (int i = 0; i < 10; ++i) {
        for (int j = 0; j < 10; ++j) {           // 빈좌석 찾아
            if (seats[i][j] == 0 && isAble(i, j)) { // 가능한지 알아보고 : BFS탐색
                alloc(i, j);                       // 예약하기 : 우선순위 BFS탐색
                ret = { hid, i * 10 + j + 1 };      // 결과값 구하기 : 2차원 좌표로 좌석 번호 산출
                return true;
            }
        }
    }
    return false;
}

}halls[2001];
```

열 크기 : 5

	0	1	2	3	4
0	1	2	3	4	5
1	6	7	8	9	10
2	11	12	13	14	15

[좌석 번호]

행 좌표 \* 열 크기 + 열 좌표 + 1



### Result cancelReservation(int mID)

단순히 mID 예약 정보를 찾아 취소시킨다.

```
Result cancelReservation(int mID) {
    int hid = reserveArr[mID].hid;
    vector<pii>&reservedSeats = reserveArr[mID].reservedSeats;
    ret = { hid, (int)reservedSeats.size() };           // 결과 초기값
    halls[hid].vacant += (int)reservedSeats.size();    // 빈 자리 수 업데이트
    for (auto&v : reservedSeats) {                    // 예약 좌석을 순회하면서
        ret.num += (v.first)*10 + v.second;           // 결과 업데이트
        halls[hid].seats[v.first][v.second] = 0;     // 빈 자리로 표시
    }
    return ret;
}
```

```
struct Reserve {
    int hid;
    vector<pii> reservedSeats;
}reserveArr[50001];

struct Hall {
    int hid;
    int vacant;
    int seats[10][10];
}halls[2001];
```

### [Summary]

- 문제를 분석하여 시간복잡도를 계산하고 적절한 문제 해결 전략을 세울 수 있다.
- 문제를 그래프 문제로 변환하여 생각할 수 있다.
- BFS를 문제 해결에 적절하게 사용할 수 있다.
- 우선순위 자료구조 PQ 등을 문제 해결에 적절하게 사용할 수 있다.

# Code example

# Code example

## TS좌석예약

```
struct Result {    int id, num;};
////////////////////////////////////

#include <vector>
#include <queue>
#include <algorithm>
using namespace std;

using pii = pair<int, int>;
const int LM = 50005;
Result ret;                // 반환할 결과값
int N;                     // 극장수
int reserveID;             // 현재 예약 ID
int targetVacant;          // 현재 예약할 좌석 수

int visited[11][11], visCnt; // 방문체크배열, 방문체크레벨
pii que[100];               // 큐
int fr, re;                 // 큐의 front, rear
int dr[] = { -1, 1, 0, 0 }, dc[] = { 0, 0, -1, 1 };

priority_queue<int, vector<int>, greater<int>> pq; // (r * 10 + c) 의 오름차순 우선순위
struct Reserve {
    int hid;                // 예약한 극장 번호
    vector<pii> seatsList; // hid극장에서 예약한 좌석 목록
}reserveArr[LM];
```

```
struct Hall {
    int hid, vacant, seats[10][10];
    bool isAble(int r, int c) { // (r, c)로 시작하는 연결된 구간의 크기가 targetVacant 이상인지 알아보기
        fr = re = 0;
        que[re++] = { r, c };
        visited[r][c] = ++visCnt;
        while (fr < re) {
            r = que[fr].first, c = que[fr++].second;
            for (int i = 0; i < 4; ++i) {
                int nr = r + dr[i], nc = c + dc[i];
                if (nr < 0 || nr > 9 || nc < 0 || nc > 9)
                    continue;
                if (seats[nr][nc] || visited[nr][nc] == visCnt)
                    continue;
                que[re++] = { nr, nc };
                visited[nr][nc] = visCnt;
            }
            if (re >= targetVacant) return 1;
        }
        return 0;
    }
}
```

# Code example

## TS좌석예약

```
void alloc(int r, int c) {
    pq = {};
    pq.push(r * 10 + c);           // 1.행우선asc, 2.열차선asc
    visited[r][c] = ++visCnt;
    reserveArr[reserveID].hid = hid;
    vector<pii>&seatsList = reserveArr[reserveID].seatsList;
    seatsList.clear();             // 예약목록 초기화

    for (int i = 0; i < targetVacant; ++i) {
        r = pq.top() / 10, c = pq.top() % 10;
        seats[r][c] = reserveID;   // 예약된 좌석은 예약ID로 채우기
        seatsList.push_back({ r, c }); // 예약좌석목록에 추가
        pq.pop();
        for (int j = 0; j < 4; ++j) { // 인접한 좌석 알아보기
            int nr = r + dr[j], nc = c + dc[j];
            if (nr < 0 || nr > 9 || nc < 0 || nc > 9) continue;
            if (seats[nr][nc] || visited[nr][nc] == visCnt) continue;
            pq.push(nr * 10 + nc);
            visited[nr][nc] = visCnt;
        }
    }
    vacant -= targetVacant;
}
```

# Code example

## TS좌석예약

```
bool findSeats() {
    if (vacant < targetVacant) return false;
    for (int i = 0; i < 10; ++i) {
        for (int j = 0; j < 10; ++j) {
            if (seats[i][j] == 0 && isAble(i, j)) { // 빈좌석 찾아
                alloc(i, j); // 가능한지 알아보고
                ret = { hid, i * 10 + j + 1 }; // 예약하기
                return true; // 결과값 구하기
            }
        }
    }
    return false;
}

}halls[2001];

void init(int N) {
    ::N = N;
    for (int i = 1; i <= N; ++i) halls[i] = { i, 100 };
}
```

# Code example

## TS좌석예약

```
Result reserveSeats(int mID, int mNum) {
    ret = { 0, 0 };
    reserveID = mID;
    targetVacant = mNum;
    for (int i = 1; i <= N; ++i) {
        if (halls[i].findSeats())
            break;
    }
    return ret;
}

Result cancelReservation(int mID) {
    int hid = reserveArr[mID].hid;
    vector<pii>&seatsList = reserveArr[mID].seatsList;
    ret = { hid, (int)seatsList.size() };
    halls[hid].vacant += (int)seatsList.size();
    for (auto&v : seatsList) {
        ret.num += (v.first)*10 + v.second;
        halls[hid].seats[v.first][v.second] = 0;
    }
    return ret;
}
```

// 예약 ID  
// 예약하고자 하는 좌석수  
// 가능한 자리 찾아보기  
  
// 가능한 자리 찾은 경우

// 결과 초기값  
// 빈 자리 확보  
// 예약좌석을 순회하면서  
// 결과 업데이트  
// 빈 자리 표시



**Thank you.**