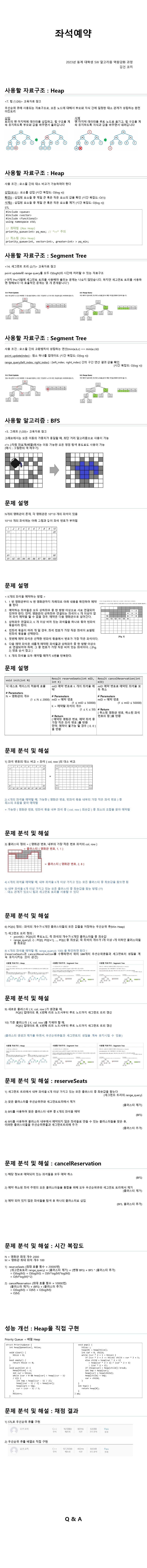
**목차   
1. 문제 풀이 강의 자료   
2. 소스 코드**  
  
  
**1. 문제 풀이 강의 자료   
**  
  
**2. 소스 코드**

**#include <algorithm>**

**#include <queue>**

**#include <functional>**

**#include <vector>**

**#define MAX\_N 2048**

**#define MAX\_MID 50001**

**#define NN 128**

**#define INF 999999999**

**#define INF\_PIP { INF, { INF, INF } }**

**using** **namespace** std;

**typedef** pair<**int**, pair<**int**,**int**> > pip;

**struct** Result { **int** id, num; }; *// API return 형식*

**int** n; *// 영화관 개수*

**int** seat[MAX\_N][11][11]; *// 클러스터 크기(클러스터 min 좌석: 클러스터 크기, else: 0)*

**int** reserved[MAX\_N][11][11]; *// 영화관 예약 여부(예약 없음: 0, else: 예약 번호)*

pip reserve\_info[MAX\_MID]; *// 예약 정보가 저장되는 배열*

*// leaf 노드가 힙인 세그먼트 트리*

priority\_queue<pip, vector<pip>, greater<pip> > leaf[NN + 1];

pip tree[NN \* 2];

*// BFS를 위한 큐와 이동 가능 범위 배열*

**int** dx[4] = { 1, -1, 0, 0 }, dy[4] = { 0, 0, 1, -1 };

priority\_queue<pair<**int**, **int**>, vector<pair<**int**, **int**> >, greater<pair<**int**, **int**> > > PQ;

queue<pair<**int**, **int**> > Q;

**int** check\_cnt, check[11][11];

*// 영화관 좌석 정보(x,y)를 좌석 번호로 인코딩*

**int** encoding(pair<**int**, **int**> pos) {

**return** (pos.first - 1) \* 10 + pos.second;

}

*// 세그먼트 트리 쿼리 검색*

pip treeFind(**int** lev, **int** l, **int** r, **int** x, **int** y) {

**int** mid = (l + r) / 2;

**if** (x <= l && r <= y) **return** tree[lev];

**if** (r < x || y < l) **return** INF\_PIP;

**return** min(treeFind(lev \* 2, l, mid, x, y), treeFind(lev \* 2 + 1, mid + 1, r, x, y));

}

*// 세그먼트 트리 리프에서 루트 노드까지 재귀로 업데이트*

**void** treeUpdate(**int** lev) {

tree[lev] = min(tree[lev \* 2], tree[lev \* 2 + 1]);

**if** (lev > 1) treeUpdate(lev / 2);

}

*// 세그먼트 트리 리프 노드(힙) 정보 업데이트*

**void** treeUpdateLeaf(**int** mSeat) {

pip oriValue = tree[NN + mSeat - 1];

tree[NN + mSeat - 1] = INF\_PIP;

**while** (!leaf[mSeat].empty()) {

pip top = leaf[mSeat].top();

**if** (seat[top.first][top.second.first][top.second.second] == mSeat) {

tree[NN + mSeat - 1] = top;

**break**;

}

leaf[mSeat].pop();

}

**if** (oriValue != tree[NN + mSeat - 1]) treeUpdate((NN + mSeat - 1) / 2);

}

*// 새로운 클러스터 추가*

**void** addCluster(pip clusterInfo, **int** mSeat) {

seat[clusterInfo.first][clusterInfo.second.first][clusterInfo.second.second] = mSeat;

leaf[mSeat].push(clusterInfo);

treeUpdateLeaf(mSeat);

}

*// 기존 클러스터 제거(무효화)*

**void** removeCluster(pip clusterInfo) {

**int** mSeat = seat[clusterInfo.first][clusterInfo.second.first][clusterInfo.second.second];

seat[clusterInfo.first][clusterInfo.second.first][clusterInfo.second.second] = 0;

treeUpdateLeaf(mSeat);

}

**void** init(**int** \_n) {

n = \_n;

**for** (**int** i = 1; i <= NN; i++) {

**while** (!leaf[i].empty()) leaf[i].pop();

}

**for** (**int** i = 1; i < NN \* 2; i++) {

tree[i] = INF\_PIP;

}

**for** (**int** i = 1; i <= n; i++) {

**for** (**int** x = 1; x <= 10; x++) {

**for** (**int** y = 1; y <= 10; y++) {

reserved[i][x][y] = seat[i][x][y] = 0;

}

}

addCluster({ i, { 1, 1 } }, 100);

}

}

Result reserveSeats(**int** mID, **int** k) {

*// 세그먼트 트리에서 가장 작은 번호의 영화관, 좌석을 찾음*

pip res = treeFind(1, 1, NN, k, NN);

**if** (res.first > n) **return** { 0, 0 };

reserve\_info[mID] = res;

*// 기존 클러스터 제거*

removeCluster(res);

*// 변형 BFS로 k개의 상하좌우 연결된 좌석 찾기*

check\_cnt += 1;

PQ.push(res.second);

**for** (**int** i = 1; i <= k; i++) {

pair<**int**, **int**> pos = PQ.top(); PQ.pop();

reserved[res.first][pos.first][pos.second] = mID;

**for** (**int** j = 0; j < 4; j++) {

**int** tx = pos.first + dx[j];

**int** ty = pos.second + dy[j];

**if** (1 <= tx && tx <= 10 && 1 <= ty && ty <= 10) {

**if** (reserved[res.first][tx][ty] == 0 && check[tx][ty] != check\_cnt) {

check[tx][ty] = check\_cnt;

PQ.push({ tx, ty });

}

}

}

}

*// 좌석 예약으로 인해 절단되어 새롭게 생긴 클러스터 찾기*

check\_cnt += 1;

**while** (!PQ.empty()) {

pair<**int**, **int**> top = PQ.top(); PQ.pop();

**if** (check[top.first][top.second] != check\_cnt) {

*// 새로운 클러스터를 찾으면 BFS로 클러스터의 전체 좌석 개수와 가장 작은 좌석 번호 찾기*

pair<**int**, **int**> min\_seat = { INF, INF };

**int** cnt = 0;

check[top.first][top.second] = check\_cnt;

Q.push(top);

**while** (!Q.empty()) {

pair<**int**, **int**> pos = Q.front(); Q.pop();

min\_seat = min(min\_seat, pos);

cnt += 1;

**for** (**int** i = 0; i < 4; i++) {

**int** tx = pos.first + dx[i];

**int** ty = pos.second + dy[i];

**if** (1 <= tx && tx <= 10 && 1 <= ty && ty <= 10

&& check[tx][ty] != check\_cnt && reserved[res.first][tx][ty] == 0) {

check[tx][ty] = check\_cnt;

Q.push({ tx, ty });

}

}

}

*// 찾은 정보를 바탕으로 클러스터 추가*

addCluster({ res.first, min\_seat }, cnt);

}

}

**return** { res.first, encoding(res.second) };

}

Result cancelReservation(**int** mID) {

*// 좌석 예약이 되어 있는 클러스터 제거*

removeCluster(reserve\_info[mID]);

*// BFS로 주변 좌석 예약이 없는 클러스터 탐색*

**int** mTheater = reserve\_info[mID].first;

pair<**int**, **int**> pos\_init = reserve\_info[mID].second;

pair<**int**, **int**> min\_seat = { INF, INF };

**int** cnt = 0, sum\_seat = encoding(pos\_init);

check[pos\_init.first][pos\_init.second] = ++check\_cnt;

reserved[mTheater][pos\_init.first][pos\_init.second] = 0;

Q.push(pos\_init);

**while** (!Q.empty()) {

pair<**int**, **int**> pos = Q.front(); Q.pop();

min\_seat = min(min\_seat, pos);

cnt += 1;

**for** (**int** i = 0; i < 4; i++) {

**int** tx = pos.first + dx[i];

**int** ty = pos.second + dy[i];

**if** (1 <= tx && tx <= 10 && 1 <= ty && ty <= 10 && check[tx][ty] != check\_cnt) {

check[tx][ty] = check\_cnt;

**if** (reserved[mTheater][tx][ty] == mID) {

sum\_seat += encoding({ tx, ty });

reserved[mTheater][tx][ty] = 0;

Q.push({ tx, ty });

}

**else** **if** (reserved[mTheater][tx][ty] == 0) {

**if** (seat[mTheater][tx][ty] > 0) {

removeCluster({ mTheater, { tx, ty } });

}

Q.push({ tx, ty });

}

}

}

}

*// 새로운 하나의 큰 클러스터 추가*

addCluster({ mTheater, min\_seat }, cnt);

**return** { mTheater, sum\_seat };

}