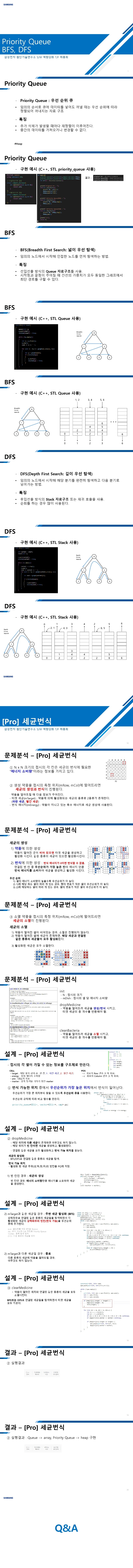
**목차   
1. 문제 풀이 강의 자료   
2. 소스 코드**  
  
  
**1. 문제 풀이 강의 자료**  
  
**2. 소스 코드**  
1) STL 활용

**#define MAX\_N 100**

**#include < queue >**

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

**struct** POS {

**int** r; *//y*

**int** c; *//x*

};

**struct** CELL {

**int** target;

**int** energy;

**int** marker;

POS pos;

};

**struct** cmp {

**bool** **operator**()(CELL\* a, CELL\* b) {

**if** (a->energy == b->energy) {

**if** (a->pos.r == b->pos.r) {

**return** a->pos.c > b->pos.c;

}

**else** {

**return** a->pos.r > b->pos.r;

}

}

**else** {

**return** a->energy < b->energy;

}

}

};

**int** N;

**int** marker;

**int** nCnt[3];

CELL map[MAX\_N+1][MAX\_N+1];

priority\_queue< CELL\*, vector< CELL\*>, cmp > pq;

**void** init(**int** N, **int** mDish[MAX\_N][MAX\_N])

{

::N = N;

**for** (**int** r = 1; r <= N; r++) {

**for** (**int** c = 1; c <= N; c++) {

map[r][c].energy = mDish[r-1][c-1];

map[r][c].target = 0;

map[r][c].pos = { r,c };

map[r][c].marker = 0;

}

}

marker = 0;

nCnt[1] = nCnt[2] = 0;

}

**const** **int** dr[] = { -1,0,0, 1 };

**const** **int** dc[] = { 0,-1,1,0 };

**void** update(**int** r, **int** c, **int** t) {

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

que.push(pair< **int**, **int** >(r, c));

**while** (!que.empty())

{

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

r = pos.first;

c = pos.second;

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

**int** nc = c + dc[dir];

**int** nr = r + dr[dir];

**if** (nc <= 0 || nr <= 0 || nc > N || nr > N) **continue**;

**if**(map[nr][nc].marker >= marker) **continue**;

**if**(map[nr][nc].target == (3-t)) **continue**;

map[nr][nc].marker = marker;

**if** (map[nr][nc].target == t){

que.push(pair< **int**, **int** >(nr,nc));

}

**else** **if** (map[nr][nc].target == 0) {

pq.push(&map[nr][nc]);

}

}

}

}

**int** dropMedicine(**int** mTarget, **int** mRow, **int** mCol, **int** mEnergy)

{

marker++;

**int** curType = map[mRow][mCol].target;

*// 다른 세균이 존재하면 바로 종료.*

**if** (curType != 0 && curType != mTarget) **return** nCnt[mTarget];

*// 바로 활성화*

CELL \*cell = &map[mRow][mCol];

**if**(cell->target == 0){

cell->target = mTarget;

nCnt[mTarget]++;

mEnergy -= cell->energy;

}

cell->marker = marker;

*// 연결된 세균 활성화 및 번식*

pq = {};

**while** (mEnergy > 0) {

update(cell->pos.r, cell->pos.c, mTarget);

**if**(pq.size() ==0) **break**;

cell = pq.top();pq.pop();

cell->target = mTarget;

nCnt[mTarget]++;

mEnergy -= cell->energy;

}

**return** nCnt[mTarget];

}

**int** cleanBacteria(**int** mRow, **int** mCol)

{

marker++;

**int** target = map[mRow][mCol].target;

*// 셀이 비어있으면 그냥 종료*

**if** (target == 0) **return** -1;

*// BFS로 연결된 모든 세균 제거*

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

que.push(pair< **int**, **int** >(mRow,mCol));

**while** (!que.empty())

{

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

**int** r = pos.first;

**int** c = pos.second;

map[r][c].target = 0;

nCnt[target]--;

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

**int** nc = c + dc[dir];

**int** nr = r + dr[dir];

**if** (nc <= 0 || nr <= 0 || nc > N || nr > N) **continue**;

**if** (map[nr][nc].marker >= marker) **continue**;

**if**(map[nr][nc].target != target) **continue**;

que.push(pair< **int**, **int** >(nr, nc));

map[nr][nc].marker = marker;

}

}

**return** nCnt[target];

}

2) heap, queue 구현

**#define MAX\_N 100**

**struct** POS {

**int** r; *//y*

**int** c; *//x*

};

**struct** CELL {

**int** target;

**int** energy;

**int** marker;

POS pos;

};

**bool** cmp(CELL& a, CELL& b) {

**if** (a.energy == b.energy) {

**if** (a.pos.r == b.pos.r) {

**return** a.pos.c < b.pos.c;

}

**else** {

**return** a.pos.r < b.pos.r;

}

}

**else** {

**return** a.energy > b.energy;

}

}

**#define MAX\_HEAP\_SIZE 20000**

CELL\* heap[MAX\_HEAP\_SIZE];

**int** heapSize;

**void** heapInit(**void**)

{

heapSize = 0;

}

**int** heapPush(CELL\* value)

{

**if** (heapSize + 1 > MAX\_HEAP\_SIZE)

**return** 0;

heap[heapSize] = value;

**int** current = heapSize;

**while** (current > 0 && cmp(\*heap[current], \*heap[(current - 1) / 2]))

{

CELL\* temp = heap[(current - 1) / 2];

heap[(current - 1) / 2] = heap[current];

heap[current] = temp;

current = (current - 1) / 2;

}

heapSize++;

**return** 1;

}

**int** heapPop(CELL\*\* value)

{

**if** (heapSize <= 0)

**return** -1;

\*value = heap[0];

heapSize = heapSize - 1;

heap[0] = heap[heapSize];

**int** current = 0;

**while** (current \* 2 + 1 < heapSize)

{

**int** child;

**if** (current \* 2 + 2 == heapSize)

child = current \* 2 + 1;

**else**

child = cmp(\*heap[current \* 2 + 1], \*heap[current \* 2 + 2]) ? current \* 2 + 1 : current \* 2 + 2;

**if** (cmp(\*heap[current], \*heap[child]))

**break**;

CELL\* temp = heap[current];

heap[current] = heap[child];

heap[child] = temp;

current = child;

}

**return** 1;

}

**int** N;

**int** marker;

**int** nCnt[3];

CELL map[MAX\_N+1][MAX\_N+1];

**void** init(**int** N, **int** mDish[MAX\_N][MAX\_N])

{

::N = N;

**for** (**int** r = 1; r <= N; r++) {

**for** (**int** c = 1; c <= N; c++) {

map[r][c].energy = mDish[r-1][c-1];

map[r][c].target = 0;

map[r][c].pos = { r,c };

map[r][c].marker = 0;

}

}

marker = 0;

nCnt[1] = nCnt[2] = 0;

}

**const** **int** dr[] = { 0,-1,1,0 };

**const** **int** dc[] = { -1,0,0, 1 };

**void** update(**int** r, **int** c, **int** t) {

*// 기존 mTarget 영역 탐색.*

POS que[1000];

**int** queIdx = 0;

**int** frontIdx =0;

que[queIdx++] = { r,c };

**while** (frontIdx< queIdx)

{

r = que[frontIdx].r;

c = que[frontIdx].c;

frontIdx++;

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

**int** nc = c + dc[dir];

**int** nr = r + dr[dir];

**if** (nc <= 0 || nr <= 0 || nc > N || nr > N) **continue**;

**if** (map[nr][nc].marker >= marker) **continue**;

**if**(map[nr][nc].target == (3-t)) **continue**;

map[nr][nc].marker = marker;

**if** (map[nr][nc].target == t)

que[queIdx++] = { nr,nc };

**else** **if** (map[nr][nc].target == 0)

heapPush(&map[nr][nc]);

}

}

}

**int** dropMedicine(**int** mTarget, **int** mRow, **int** mCol, **int** mEnergy)

{

marker++;

**int** curType = map[mRow][mCol].target;

**if** (curType != 0 && curType != mTarget) **return** nCnt[mTarget];

CELL\* cell = &map[mRow][mCol];

**if**(cell->target == 0){

cell->target = mTarget;

nCnt[mTarget]++;

mEnergy -= cell->energy;

}

cell->marker = marker;

heapInit();

**while**(mEnergy >0){

update(cell->pos.r,cell->pos.c,mTarget);

**if**(heapSize ==0) **break**;

heapPop(&cell);

cell->target = mTarget;

nCnt[mTarget]++;

mEnergy -= cell->energy;

}

**return** nCnt[mTarget];

}

**int** cleanBacteria(**int** mRow, **int** mCol)

{

marker++;

**int** target = map[mRow][mCol].target;

**if** (target == 0) **return** -1;

POS que[1000];

**int** queIdx = 0;

**int** frontIdx =0;

**int** r,c;

que[queIdx++] = { mRow,mCol };

**while** (frontIdx < queIdx)

{

r = que[frontIdx].r;

c = que[frontIdx].c;

frontIdx++;

map[r][c].target = 0;

nCnt[target]--;

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

**int** nc = c + dc[dir];

**int** nr = r + dr[dir];

**if** (nc <= 0 || nr <= 0 || nc > N || nr > N) **continue**;

**if** (map[nr][nc].marker >= marker) **continue**;

**if**(map[nr][nc].target !=target) **continue**;

que[queIdx++] = { nr,nc};

map[nr][nc].marker = marker;

}

}

**return** nCnt[target];

}