[1. SAMSUNG SW : Bids Escape 2]

- I spent 1 hour and half for this problem again!

- it was hard to match the restriction for me.

- see the code.

#include<iostream>

#include<vector>

#include<queue>

using namespace std;

int N, M;

vector<vector<char>> table;

bool visit[12][12][12][12];

int rx, ry, bx, by, ex, ey;

int dir[4][2] = { {-1,0},{0,-1},{1,0},{0,1} };

int BFS() {

queue<vector<int>> que;

que.push(vector<int>{rx, ry, bx, by,0});

visit[rx][ry][bx][by] = true;

while (!que.empty()) {

rx = que.front()[0]; ry = que.front()[1]; bx = que.front()[2]; by = que.front()[3]; int cnt = que.front()[4];

que.pop();

if (cnt > 10) continue;

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

int nrx = rx; int nry = ry; int nbx = bx; int nby = by;

bool flag = false;

bool flag2 = false;

while (nbx >= 0 && nby >= 0 && nbx < N && nby < M && table[nbx][nby] != '#') {

nbx += dir[i][0]; nby += dir[i][1];

if (nbx == ex && nby == ey) {

flag2 = true;

break;

}

}

if (flag2) continue;

if (table[nbx][nby] == '#') {

nbx -= dir[i][0]; nby -= dir[i][1];

}

while (nrx>=0 && nry>=0 && nrx<N && nry<M && table[nrx][nry] != '#') {

nrx += dir[i][0]; nry += dir[i][1];

if (nrx == ex && nry == ey) {

flag = true;

break;

}

}

if (flag && cnt < 10) return cnt + 1;

if (table[nrx][nry] == '#') {

nrx -= dir[i][0]; nry -= dir[i][1];

}

if (nrx == nbx && nry == nby) {

if (i == 0) {

if (rx < bx) {

nbx++;

}

else {

nrx++;

}

}

else if (i == 1) {

if (ry < by) {

nby++;

}

else {

nry++;

}

}

else if (i == 2) {

if (rx > bx) {

nbx--;

}

else {

nrx--;

}

}

else if (i == 3) {

if (ry > by) {

nby--;

}

else {

nry--;

}

}

}

if (visit[nrx][nry][nbx][nby] == false) {

visit[nrx][nry][nbx][nby] = true;

que.push(vector<int>{nrx, nry, nbx, nby, cnt + 1});

}

}

}

return -1;

}

int main() {

ios\_base::sync\_with\_stdio(false);

cin.tie(NULL); cout.tie(NULL);

//get input

cin >> N >> M;

table.assign(N, vector<char>(M, 0));

string input;

for (int i = 0; i < N; i++) {

cin >> input;

for (int j = 0; j < M; j++) {

table[i][j] = input[j];

if (input[j] == 'R') {

rx = i; ry = j;

}

if (input[j] == 'B') {

bx = i; by = j;

}

if (input[j] == 'O') {

ex = i; ey = j;

}

}

}

//algorithm part;

int answer = BFS();

cout << answer;

return 0;

}

[2. SAMSUNG SW : 2048]

- this was a simulation problem and it was like just previous problem : “Bids Escape 2”. Since we can just tilt the board and can not move each number of shell.

- I used DFS and I implemented every direction’s move.

- I spent 28 minutes and 26 seconds.

- see the code.

#include<iostream>

#include<vector>

#define max(a,b) a>b? a:b

using namespace std;

int N;

int answer = 0;

void maxValue(vector<vector<int>>& table) {

for (vector<int> row : table) {

for (int i : row) answer = max(answer, i);

}

}

vector<vector<int>> Move(vector<vector<int>> table,int dir) {

vector<vector<bool>> visit(N, vector<bool>(N, false));

switch (dir) {

case 0: //left

for (int i = 0; i < N; i++) {

for (int j = 1; j < N; j++) {

if (table[i][j] == 0) continue;

for (int k = j; k >= 1; k--) {

if (table[i][k - 1] == table[i][k] && visit[i][k - 1] == false) {

visit[i][k - 1] = true;

table[i][k - 1] \*= 2;

table[i][k] = 0;

break;

}

else if (table[i][k - 1] == 0) {

table[i][k - 1] = table[i][k];

table[i][k] = 0;

}

else break;

}

}

}

break;

case 1: // right

for (int i = 0; i < N; i++) {

for (int j = N - 2; j >= 0; j--) {

if (table[i][j] == 0) continue;

for (int k = j; k < N-1; k++) {

if (table[i][k + 1] == table[i][k] && visit[i][k + 1] == false) {

visit[i][k + 1] = true;

table[i][k + 1] \*= 2;

table[i][k] = 0;

break;

}

else if (table[i][k + 1] == 0) {

table[i][k + 1] = table[i][k];

table[i][k] = 0;

}

else break;

}

}

}

break;

case 2: //north

for (int j = 0; j < N; j++) {

for (int i = 1; i < N; i++) {

if (table[i][j] == 0) continue;

for (int k = i; k >= 1; k--) {

if (table[k - 1][j] == table[k][j] && visit[k - 1][j] == false) {

visit[k - 1][j] = true;

table[k - 1][j] \*= 2;

table[k][j] = 0;

break;

}

else if (table[k - 1][j] == 0) {

table[k - 1][j] = table[k][j];

table[k][j] = 0;

}

else break;

}

}

}

break;

case 3: //south

for (int j = 0; j < N; j++) {

for (int i = N - 2; i >= 0; i--) {

if (table[i][j] == 0) continue;

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

if (table[k + 1][j] == table[k][j] && visit[k + 1][j] == false) {

visit[k + 1][j] = true;

table[k + 1][j] \*= 2;

table[k][j] = 0;

break;

}

else if (table[k + 1][j] == 0) {

table[k + 1][j] = table[k][j];

table[k][j] = 0;

}

else break;

}

}

}

break;

}

return table;

}

void DFS(vector<vector<int>> table,int cnt) {

if (cnt == 5) {

maxValue(table);

return;

}

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

DFS(Move(table,i), cnt + 1);

}

int main() {

ios\_base::sync\_with\_stdio(false);

cin.tie(NULL); cout.tie(NULL);

//get input

cin >> N;

vector<vector<int>> table(N, vector<int>(N, 0));

for (int i = 0; i < N; i++) {

for (int j = 0; j < N; j++) {

cin >> table[i][j];

}

}

//algorithm part

DFS(table, 0);

cout << answer;

return 0;

}

[3. SAMSUNG SW : Snake]

- it was a simulation problem. there was a trick that the snake have to enlarge the head first and check whether the tail sholud be shrink or not.

- one more trivial trick is the given apple’s position start with (1,1) not (0,0).

- see the code.

#include<iostream>

#include<vector>

#include<unordered\_map>

using namespace std;

int N,K,L;

int cur\_dir = 3;

vector<vector<int>> table;

unordered\_map<int, char> curve;

int dir[4][2] = { {-1,0},{0,-1},{1,0},{0,1} };

void printTable() {

for (vector<int> row : table) {

for (int i : row) cout << i << " ";

cout << endl;

}

cout << endl;

}

int main() {

ios\_base::sync\_with\_stdio(false);

cin.tie(NULL); cout.tie(NULL);

//get input

cin >> N>>K;

table.assign(N, vector<int>(N, 0));

int x, y;

table[0][0] = 2;

for (int i = 0; i < K; i++) {

cin >> x >> y;

table[x-1][y-1] = 1;

}

cin >> L;

int l; char c;

for (int i = 0; i < L; i++) {

cin >> l >> c;

curve[l] = c;

}

//algorithm part

int sec = 0;

vector<pair<int, int>> snake;

snake.emplace\_back(make\_pair(0, 0));

while(true) {

sec += 1;

int hx = snake.back().first + dir[cur\_dir][0]; int hy = snake.back().second + dir[cur\_dir][1];

if (hx < 0 || hy < 0 || hx >= N || hy >= N || table[hx][hy] == 2) break;

else if (table[hx][hy] == 1) {

table[hx][hy] = 2;

snake.emplace\_back(make\_pair(hx, hy));

}

else if (table[hx][hy] == 0) {

table[hx][hy] = 2;

table[snake.front().first][snake.front().second] = 0;

snake.erase(snake.begin());

snake.emplace\_back(make\_pair(hx, hy));

}

if (curve.find(sec) != curve.end()) {

if (curve[sec] == 'L') cur\_dir = (cur\_dir + 1) % 4;

else {

cur\_dir -= 1;

if (cur\_dir < 0) cur\_dir = 3;

}

}

}

cout << sec;

return 0;

}

[4. SAMSUNG SW : Exam Invigilator]

- there was a trap that we have to consider data type. since there was no limit of sum of total invigilators.

- so I set answer’s data type as long long

- I spent 23 minutes.

- see the code.

#include<iostream>

#include<vector>

using namespace std;

int N, B, C;

vector<int> table;

long long answer = 0;

int main() {

ios\_base::sync\_with\_stdio(false);

cin.tie(NULL); cout.tie(NULL);

//get input

cin >> N;

table.assign(N, 0);

for (int i = 0; i < N; i++) cin >> table[i];

cin >> B >> C;

// algorithm part

answer += N;

for (int i = 0; i < N; i++) {

table[i] -= B;

if (table[i] > 0) {

long long sub;

if (table[i] % C >= -0.0001 && table[i] % C<=0.0001) sub = table[i] / C;

else sub = (long long)(table[i] / C)+1;

answer += sub;

}

}

cout << answer;

return 0;

}

[5. SAMSUNG SW : Roll a Dice]

- it was a simulation problem. implementing Dice was quite tricky. In my case, I made 4 x 3 array to represent the dice.

- they gave us number of order to move the dice, but the diretions started with 1 not 0, so there was a small confusing.

- I spent 22 minutes.

- see the code.

#include<iostream>

#include<vector>

using namespace std;

int N, M,x,y,K;

vector<vector<int>> table;

vector<vector<int>> dice = { {0,0,0},{0,0,0},{0,0,0},{0,0,0} };

vector<int> roll;

int dir[4][2] = { {0,1},{0,-1},{-1,0},{1,0} }; // d a w s

int main() {

ios\_base::sync\_with\_stdio(false);

cin.tie(NULL); cout.tie(NULL);

//get input

cin >> N >> M>>x>>y>>K;

table.assign(N, vector<int>(M, 0));

for (int i = 0; i < N; i++) {

for (int j = 0; j < M; j++) {

cin >> table[i][j];

}

}

roll.assign(K, 0);

for (int i = 0; i < K; i++) cin >> roll[i];

//algorithm part

for (int round = 0; round < K; round++) {

int cur\_dir = roll[round]-1;

x = x + dir[cur\_dir][0]; y = y + dir[cur\_dir][1];

if (x < 0 || y < 0 || x >= N || y >= M) {

x -= dir[cur\_dir][0]; y -= dir[cur\_dir][1];

continue;

}

//roll the dice

int temp;

switch (cur\_dir){

case 0://east

temp = dice[1][0];

dice[1][0] = dice[3][1];

dice[3][1] = dice[1][2];

dice[1][2] = dice[1][1];

dice[1][1] = temp;

break;

case 1://west

temp = dice[1][2];

dice[1][2] = dice[3][1];

dice[3][1] = dice[1][0];

dice[1][0] = dice[1][1];

dice[1][1] = temp;

break;

case 2://north

temp = dice[0][1];

dice[0][1] = dice[1][1];

dice[1][1] = dice[2][1];

dice[2][1] = dice[3][1];

dice[3][1] = temp;

break;

case 3://south

temp = dice[3][1];

dice[3][1] = dice[2][1];

dice[2][1] = dice[1][1];

dice[1][1] = dice[0][1];

dice[0][1] = temp;

break;

}

if (table[x][y] == 0) table[x][y] = dice[3][1];

else {

dice[3][1] = table[x][y];

table[x][y] = 0;

}

cout << dice[1][1]<<endl;

}

return 0;

}

[6. SAMSUNG SW : Tetromino]

- it was a simulation problem. but there was lots of manual labor.

- since I had to implement all the cases of tetromino shape.

- and for sliding each tetromino, I used quadruple for syntax.

- I spent 26 minute and 35 seconds.

- see the code.

#include<iostream>

#include<vector>

#define max(a,b) a>b?a:b

using namespace std;

vector<vector<int>> table;

int N, M;

vector<vector<vector<int>>> tetro = {

{{1,1,1,1}}, {{1},{1},{1},{1}}, //line

{{1,1},{1,1}},//square

{{1,0},{1,0},{1,1}},{{0,1},{0,1},{1,1} }, {{1,1},{0,1},{0,1}},{{1,1},{1,0},{1,0}},

{ {1,1,1},{1,0,0}}, {{1,1,1},{0,0,1}},{{1,0,0} ,{1,1,1}},{ {0,0,1} ,{1,1,1}},

{{1,0},{1,1},{0,1}},{{0,1},{1,1},{1,0}},{{1,1,0},{0,1,1}},{{0,1,1},{1,1,0}},

{{1,1,1},{0,1,0}},{{0,1,0},{1,1,1}},{{1,0},{1,1},{1,0}},{{0,1},{1,1},{0,1}}

};

int main() {

ios\_base::sync\_with\_stdio(false);

cin.tie(NULL); cout.tie(NULL);

//get input

cin >> N >> M;

table.assign(N, vector<int>(M, 0));

for (int i = 0; i < N; i++) {

for (int j = 0; j < M; j++) {

cin >> table[i][j];

}

}

//algorithm part

int answer = 0;

for (vector<vector<int>> piece : tetro) {

int row = piece.size(); int col = piece[0].size();

for (int x = 0; x + row <= N;x++) {

for (int y = 0; y + col <= M;y++) {

int temp = 0;

for (int i = x; i < x+row; i++) {

for (int j = y; j < y+col; j++) {

if (piece[i - x][j - y] == 1) temp += table[i][j];

}

}

answer = max(answer, temp);

}

}

}

cout << answer;

return 0;

}

[7. SAMSUNG SW : Quit The Job]

- it was a dp problem. the trick is that starting with the last day, get greater value between (acummulated valud of nextday of current day) and (current day + next possible counsel day).

- the logic is that we could take current maximum(dp[i+1]) value without counseling today or take next possible counseling day’s pay with counseling today(table[i]+dp[next\_possible\_day]).

- I spent 8 minutes and 25 seconds.

- see the code.

#include<iostream>

#include<vector>

#define max(a,b) a>b?a:b

using namespace std;

int N;

vector<pair<int, int>> table;

int main() {

ios\_base::sync\_with\_stdio(false);

cin.tie(NULL); cout.tie(NULL);

//get input

cin >> N;

table.assign(N+1, pair<int, int>());

for (int i = 0; i < N; i++) {

cin >> table[i].first >> table[i].second;

}

//algorithm part

vector<int> dp(N + 1, 0);

for (int i = N - 1; i >= 0; i--) {

if (table[i].first + i > N) dp[i] = dp[i + 1];

else dp[i] = max(dp[i + 1], table[i].second + dp[i + table[i].first]);

}

cout << dp[0];

return 0;

}

[8. SAMSUNG SW : Laboratory]

- it was a DFS with permutation problem.

- First, we had to choose 3 empty spaces to create walls to block the viruses. For this, I used permutation like DFS.

- Second, we had to make the viruses contagion their neighbor shells, I used DFS as well.

- Thrid, after contagion, find empty spaces in given 2D array.

- I spent 19 minute. actually, I missed to stop the time watch :)

- see the code.

#include<iostream>

#include<vector>

#define max(a,b) a>b?a:b

using namespace std;

int N, M;

vector<vector<int>> table;

vector<vector<int>> temp\_table;

vector<pair<int, int>> space;

vector<pair<int, int>> virus;

vector<vector<bool>> visit;

int num\_of\_virus;

int num\_of\_space;

int answer = 0;

int dir[4][2] = { {-1,0},{0,-1},{1,0},{0,1} };

void safetyArea() {

int res = 0;

for (vector<int> row : temp\_table) {

for (int i : row) if (i == 0) res++;

}

answer = max(answer, res);

}

void DFS(int x, int y) {

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

int nx = x + dir[i][0]; int ny = y + dir[i][1];

if (nx >= 0 && ny >= 0 && nx < N && ny < M && visit[nx][ny] == false && temp\_table[nx][ny] != 1) {

visit[nx][ny] = true;

temp\_table[nx][ny] = 2;

DFS(nx, ny);

}

}

}

void permutation(int start,int cnt) {

if (cnt == 3) {

temp\_table = table;

visit.assign(N, vector<bool>(M, false));

for (int i = 0; i < num\_of\_virus; i++)

DFS(virus[i].first,virus[i].second);

safetyArea();

return;

}

for (int i = start; i < num\_of\_space; i++) {

table[space[i].first][space[i].second] = 1;

permutation(i + 1, cnt + 1);

table[space[i].first][space[i].second] = 0;

}

}

int main() {

ios\_base::sync\_with\_stdio(false);

cin.tie(NULL); cout.tie(NULL);

//get input

cin >> N >> M;

table.assign(N, vector<int>(M, 0));

for (int i = 0; i < N; i++) {

for (int j = 0; j < M; j++) {

cin >> table[i][j];

if (table[i][j] == 0) space.emplace\_back(make\_pair(i, j));

else if (table[i][j] == 2) virus.emplace\_back(make\_pair(i, j));

}

}

num\_of\_virus = virus.size();

num\_of\_space = space.size();

//algorithm part

permutation(0, 0);

cout << answer;

return 0;

}

[9. SAMSUNG SW : Robot Vaccum]

- this was a simulation problem. I just had to follow given rule.

- there was no trick. so it was easy.

- I sepnt 20 minutes and 31 seconds.

- see the code.

#include<iostream>

#include<vector>

using namespace std;

int N, M,x,y,d;

vector<vector<int>> table;

int dir[4][2] = { {-1,0},{0,1},{1,0},{0,-1} };

int answer = 0;

int main() {

ios\_base::sync\_with\_stdio(false);

cin.tie(NULL); cout.tie(NULL);

//get input

cin >> N >> M;

cin >> x >> y >> d;

table.assign(N, vector<int>(M, 0));

for (int i = 0; i < N; i++) {

for (int j = 0; j < M; j++) {

cin >> table[i][j];

}

}

//algorithm part

while(true) {

if (table[x][y] == 0) {

answer += 1;

table[x][y] = 2;

}

bool flag = false;

for (int rotate = 0; rotate < 4; rotate++) {

d -= 1; if (d < 0) d = 3;

int nx = x + dir[d][0]; int ny = y + dir[d][1];

if (nx>=0 && ny>=0 && nx<N && ny<M && table[nx][ny] == 0) {

x = nx; y = ny;

flag = true;

break;

}

}

if (flag) continue;

else {

x -= dir[d][0]; y -= dir[d][1];

if (x < 0 || y < 0 || x >= N || y >= M || table[x][y]==1) break;

}

}

cout << answer;

return 0;

}

[10. SAMSUNG SW : Embed Operator]

- it was a permutation problem. I used DFS for permutation, but in this case, I could use next\_permutation.

- if I had used next\_permutation, I would have been faster than now. since visit vector and DFS stack would have not been needed.

- I spent 20 minutes.

- see the code.

#include<iostream>

#include<vector>

#define max(a,b) a>b? a:b

#define min(a,b) a>b? b:a

using namespace std;

int N;

vector<int> opers(4, 0);

vector<int> number;

vector<char> oper\_list;

vector<char> cur\_oper;

vector<bool> visit;

int maximum = -1000000001;

int minimum = 1000000001;

void initOperList(int i, char c) {

for (int j = 0; j < opers[i]; j++)

oper\_list.emplace\_back(c);

}

void Calculate() {

int temp = number[0];

for (int i = 1; i < N; i++) {

if (cur\_oper[i - 1] == '+') {

temp+= number[i];

}

else if (cur\_oper[i - 1] == '-') {

temp -= number[i];

}

else if (cur\_oper[i - 1] == '\*') {

temp \*= number[i];

}

else {

temp /= number[i];

}

}

maximum = max(maximum, temp);

minimum = min(minimum, temp);

}

void DFS(int cnt) {

if (cnt == N - 1) {

Calculate();

return;

}

for (int i = 0; i < oper\_list.size(); i++) {

if (visit[i] == false) {

visit[i] = true;

cur\_oper.emplace\_back(oper\_list[i]);

DFS(cnt + 1);

cur\_oper.pop\_back();

visit[i] = false;

}

}

}

int main() {

ios\_base::sync\_with\_stdio(false);

cin.tie(NULL); cout.tie(NULL);

//get input

cin >> N;

number.assign(N, 0);

visit.assign(N - 1, false);

for (int i = 0; i < N; i++) cin >> number[i];

for (int i = 0; i < 4; i++) cin >> opers[i];

//algorithm part

initOperList(0, '+'); initOperList(1, '-'); initOperList(2, '\*'); initOperList(3, '/');

DFS(0);

cout << maximum << endl;

cout << minimum << endl;

return 0;

}

[11. SAMSUNG SW : Start and Link]

- it was a combination problem. so when I used next\_permutation, time limit exceeded occurred.

- I implemented Combination using DFS.

- I spent 20 minutes.

- see the code.

#include<iostream>

#include<vector>

#include<algorithm>

#define min(a,b) a>b?b:a

using namespace std;

int N;

vector<vector<int>> table;

vector<int> players;

vector<int> team\_a;

vector<int> team\_b;

vector<bool> visit;

int answer = 987654321;

void Calculate() {

int score\_a = 0;

int score\_b = 0;

for (int i = 0; i < N / 2; i++) {

for (int j = i + 1; j < N / 2; j++) {

score\_a += table[team\_a[i]][team\_a[j]] + table[team\_a[j]][team\_a[i]];

score\_b += table[team\_b[i]][team\_b[j]] + table[team\_b[j]][team\_b[i]];

}

}

answer = min(answer, abs(score\_a - score\_b));

}

void DFS(int cnt,int start) {

if (cnt == (N / 2)) {

for (int i = 0; i < N; i++) if (visit[i] == false) team\_b.emplace\_back(i);

Calculate();

team\_b.clear();

return;

}

for (int i = start; i < N; i++) {

if (visit[i] == false) {

visit[i] = true;

team\_a.emplace\_back(i);

DFS(cnt + 1, i + 1);

team\_a.pop\_back();

visit[i] = false;

}

}

}

int main() {

ios\_base::sync\_with\_stdio(false);

cin.tie(NULL); cout.tie(NULL);

//get input

cin >> N;

table.assign(N, vector<int>(N, 0));

visit.assign(N, false);

for (int i = 0; i < N; i++) {

players.emplace\_back(i);

for (int j = 0; j < N; j++) {

cin >> table[i][j];

}

}

//algorithm part

DFS(0, 0);

cout << answer;

return 0;

}

[12. SAMSUNG SW : Ramp]

- it was a simulation problem. but there were lots of edge condition. so I had to be careful.

- when I met higher land than current, I should’ve checked previos lands I passed with how many sequencial plain lands had been.

- when I met lower land than current, I sholud’ve check post lands I would pass with how moany sequencial plain lands would be.

- I spent 30 minutes.

- see the code.

#include<iostream>

#include<vector>

using namespace std;

int N, L;

vector<vector<int>> table;

int main() {

ios\_base::sync\_with\_stdio(false);

cin.tie(NULL); cout.tie(NULL);

//get input

cin >> N >> L;

table.assign(N, vector<int>(N, 0));

for (int i = 0; i < N; i++) {

for (int j = 0; j < N; j++) {

cin >> table[i][j];

}

}

//algorithm part

int answer = 0;

//horizontal

for (int i = 0; i < N; i++) {

int prev = table[i][0];

int seq = 0;

bool flag = true;

for (int j = 0; j < N; j++) {

if (abs(prev - table[i][j]) >= 2) {

flag = false;

break; //when the height difference is greater than 1

}

else if (prev == table[i][j]) {

prev = table[i][j];

seq += 1;

}

else if (prev < table[i][j]) {

if (seq < L) {

flag = false;

break; // when we can't put a ramp

}

prev = table[i][j];

seq = 1;

continue;

}

else if (prev > table[i][j]) {

// in this case, we have to check futher shell whether we can put a ramp or not.

if (j + L > N) { // index out of range

flag = false;

break;

}

for (int k = j+1; k < j + L; k++) { //when we can't put a ramp since there is no enough sequence shell.

if (table[i][j] != table[i][k]) {

flag = false;

break;

}

}

if (!flag) break; //for just above "for" syntax

seq = 0;

j = j + L - 1;

prev = table[i][j];

}

}

if (flag) {

//cout << "i : "<< i << endl;

answer += 1;

}

}

//vertical

for (int j = 0; j < N; j++) {

int prev = table[0][j];

int seq = 0;

bool flag = true;

for (int i = 0; i < N; i++) {

if (abs(prev - table[i][j]) >= 2) {

flag = false;

break; //when the height difference is greater than 1

}

else if (prev == table[i][j]) {

prev = table[i][j];

seq += 1;

}

else if (prev < table[i][j]) {

if (seq < L) {

flag = false;

break; // when we can't put a ramp

}

prev = table[i][j];

seq = 1;

continue;

}

else if (prev > table[i][j]) {

// in this case, we have to check futher shell whether we can put a ramp or not.

if (i + L > N) { // index out of range

flag = false;

break;

}

for (int k = i + 1; k < i + L; k++) { //when we can't put a ramp since there is no enough sequence shell.

if (table[i][j] != table[k][j]) {

flag = false;

break;

}

}

if (!flag) break; //for just above "for" syntax

seq = 0;

i = i + L - 1;

prev = table[i][j];

}

}

if (flag) {

//cout << "j : " << j << endl;

answer += 1;

}

}

cout << answer;

return 0;

}

[13. SAMSUNG SW : Gear]

- it was a simulation problem. it was easy but I have been stucked due to direction.

- when a gear rotate, another gear has to rotate an opposite direction. I implement it with define command. but there was something wrong.

- afther I revise the define function. it worked well.

- I spent 33 minutes and 33 seconds.

- see the code.

#include<iostream>

#include<vector>

#define opposite(a) -a

using namespace std;

int K;

vector<vector<int>> table(4,vector<int>(8,0)); //2 and 6 is edge.

vector<pair<int, int>> order;

vector<bool> visit(4,false);

void Rotate(int i, int dir) {

int temp;

if (dir == 1) {

temp = table[i][7];

for (int j = 7; j >=1; j--) table[i][j] = table[i][j - 1];

table[i][0] = temp;

}

else {

temp = table[i][0];

for (int j = 0; j < 7; j++) table[i][j] = table[i][j + 1];

table[i][7] = temp;

}

}

void Play(int i, int dir) {

if (i == 0) {

if (table[0][2] != table[1][6] && visit[1] == false) {

visit[1] = true;

Play(1, opposite(dir));

visit[1] = false;

}

}

else if (i == 3) {

if (table[3][6] != table[2][2] && visit[2] == false) {

visit[2] = true;

Play(2, opposite(dir));

visit[2] = false;

}

}

else {

if (table[i][6] != table[i - 1][2] && visit[i - 1] == false) {

visit[i - 1] = true;

Play(i - 1, opposite(dir));

visit[i - 1] = false;

}

if (table[i][2] != table[i + 1][6] && visit[i + 1] == false) {

visit[i + 1] = true;

Play(i + 1, opposite(dir));

visit[i + 1] = false;

}

}

Rotate(i, dir);

}

int main() {

ios\_base::sync\_with\_stdio(false);

cin.tie(NULL); cout.tie(NULL);

//get input

string input;

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

cin >> input;

for (int j = 0; j < 8; j++) {

table[i][j]=input[j]-'0';

}

}

cin >> K;

order.assign(K, pair<int, int>());

for (int i = 0; i < K; i++) {

cin >> order[i].first >> order[i].second;

order[i].first -= 1; //gear number start at 0.

}

//algorithm part;

int answer = 0;

for (pair<int, int> gear : order) {

visit[gear.first] = true;

Play(gear.first, gear.second);

visit[gear.first] = false;

}

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

if (table[i][0] == 1) {

answer += 1 << i;

}

}

cout << answer;

return 0;

}

[14. SAMSUNG SW : Watch - CCTV]

- it was a combination problem. Implementing combination was not a big deal but removing blind spots using current cctvs’ direction is quite hard.

- so I made a function that remove blind spots of given direction with position (x,y)

- And I gave all the possible direction for each cctv type.

- I spent 34 minutes and 56 seconds.

- see the code.

#include<iostream>

#include<vector>

#define min(a,b) a>b? b:a

using namespace std;

int N, M;

vector<vector<int>> cctv;

int num\_of\_cctv;

int answer = 987654321;

int dir[4][2] = { {-1,0},{0,-1},{1,0},{0,1} };

void blindSpot(vector<vector<int>>& temp\_table) {

int res = 0;

for (vector<int> row : temp\_table) {

for (int i : row) if (i == 0) res += 1;

}

answer = min(answer, res);

}

vector<vector<int>> Watch(vector<vector<int>> temp\_table,int x,int y,vector<int> d) {

for (int i = 0; i < d.size(); i++) {

int nx = x+dir[d[i]][0]; int ny = y+dir[d[i]][1];

while (nx >= 0 && ny >= 0 && nx < N && ny < M && temp\_table[nx][ny] != 6) {

if(temp\_table[nx][ny]==0) temp\_table[nx][ny] = 7;

nx = nx + dir[d[i]][0]; ny = ny + dir[d[i]][1];

}

}

return temp\_table;

}

void DFS(vector<vector<int>> table,int cnt, int start) {

if (cnt == num\_of\_cctv) {

blindSpot(table);

return;

}

for (int i = start; i < num\_of\_cctv; i++) {

int x = cctv[i][0]; int y = cctv[i][1]; int num = cctv[i][2];

switch (num){

case 1:

for (int j = 0; j < 4; j++) DFS(Watch(table, x, y, vector<int>{j}),cnt+1,i+1);

break;

case 2 :

DFS(Watch(table, x, y, vector<int>{0, 2}), cnt + 1, i + 1);

DFS(Watch(table, x, y, vector<int>{1, 3}), cnt + 1, i + 1);

break;

case 3:

DFS(Watch(table, x, y, vector<int>{0, 1}), cnt + 1, i + 1);

DFS(Watch(table, x, y, vector<int>{1, 2}), cnt + 1, i + 1);

DFS(Watch(table, x, y, vector<int>{2, 3}), cnt + 1, i + 1);

DFS(Watch(table, x, y, vector<int>{3, 0}), cnt + 1, i + 1);

break;

case 4:

DFS(Watch(table, x, y, vector<int>{0,1,2}), cnt + 1, i + 1);

DFS(Watch(table, x, y, vector<int>{1,2,3}), cnt + 1, i + 1);

DFS(Watch(table, x, y, vector<int>{2,3,0}), cnt + 1, i + 1);

DFS(Watch(table, x, y, vector<int>{3,0,1}), cnt + 1, i + 1);

break;

case 5:

DFS(Watch(table, x, y, vector<int>{0,1,2,3}), cnt + 1, i + 1);

break;

}

}

}

int main() {

ios\_base::sync\_with\_stdio(false);

cin.tie(NULL); cout.tie(NULL);

//get input

cin >> N >> M;

vector<vector<int>> table(N, vector<int>(M, 0));

for (int i = 0; i < N; i++) {

for (int j = 0; j < M; j++) {

cin >> table[i][j];

if (table[i][j] != 0 && table[i][j] != 6) cctv.emplace\_back(vector<int>{i, j, table[i][j]});

}

}

//algorithm part

num\_of\_cctv = cctv.size();

DFS(table, 0, 0);

cout << answer;

return 0;

}

[15. SAMSUNG SW : Ladder Manipulation]

- it was a combination and simulation problem. combination was the main point.

- I used visit array first to determine a lane currently consider is a valid place to put a ladder. but time limit exceeded occurred.

- so I removed the visit array and just used given table.

- by the way, I implemented combination part using DFS. there was a restriction of maximum number we had to choose and when it is less than the maximum number, I also had to check simulation.

- see the code.

#include<iostream>

#include<vector>

#define min(a,b) a>b?b:a

using namespace std;

int N, M, H;

vector<vector<int>> table;

//vector<vector<vector<bool>>> visit;

vector<pair<int, int>> space;

int num\_of\_space;

int answer = 10;

bool Simulation() {

for (int j = 0; j < M; j++) {

int x = 0; int y = j; int px = 0; int py = j;

while (x < N) {

if (y - 1 >= 0 && table[x][y - 1] == 2) y -= 1;

else if (table[x][y] == 2) y += 1;

x += 1;

}

if (j != y) return false;

}

return true;

}

void Permutation(int cnt, int start) {

if (cnt == 4) return;

if (Simulation()) {

answer = min(answer, cnt);

return;

}

for (int i = 0; i < N; i++) {

for (int j = start; j < M-1; j++) {

if (table[i][j] == 2) continue;

if (j - 1 >= 0 && table[i][j - 1] == 2) continue;

if (j + 1 < M && table[i][j + 1] == 2) continue;

table[i][j] = 2;

Permutation(cnt + 1, j);

table[i][j] = 0;

}

}

}

int main() {

ios\_base::sync\_with\_stdio(false);

cin.tie(NULL); cout.tie(NULL);

//get input

cin >> M >> H >> N; // M is row, N is col, H is number of given horizon line.

table.assign(N, vector<int>(M, 0));

int x, y;

for (int i = 0; i < H; i++) {

cin >> x >> y;

x -= 1; y -= 1;

table[x][y] = 2;

}

//algorithm part

Permutation(0, 0);

if (answer == 10) cout << -1;

else cout << answer;

return 0;

}

[16. SAMSUNG SW : Dragon Curve]

- it was a simulation problem using stack.

- For each round, I had to record previous direction in the stack.

- By increasing generation, from stack.back, rotating the direction and drawing a line in the table is the way to simulate.

- I spent 43 minutes and 12 seconds.

- see the code.

#include<iostream>

#include<vector>

using namespace std;

int N;

vector<vector<int>> curve; //y x d g

vector<vector<int>> table(101, vector<int>(101, 0));

int dir[4][2] = { {0,1},{-1,0},{0,-1},{1,0} };

int main() {

ios\_base::sync\_with\_stdio(false);

cin.tie(NULL); cout.tie(NULL);

//get input

cin >> N;

curve.assign(N, vector<int>(4, 0));

for (int i = 0; i < N; i++) {

cin >> curve[i][0] >> curve[i][1] >> curve[i][2] >> curve[i][3];

}

//algorithm part

for (vector<int> cur\_curve : curve) {

int x = cur\_curve[1]; int y = cur\_curve[0]; int d = cur\_curve[2]; int g = cur\_curve[3];

table[x][y] = 1;

x = x + dir[d][0]; y = y + dir[d][1];

table[x][y] = 1;

if (g >= 1) {

vector<int> stk;

stk.emplace\_back(d);

for (int i = 1; i <= g; i++) {

int \_size = stk.size();

for (int j = \_size - 1; j >= 0; j--) {

int gen = stk[j];

gen = (gen + 1) % 4;

x = x + dir[gen][0]; y += dir[gen][1];

table[x][y] = 1;

stk.emplace\_back(gen);

}

}

}

}

int answer = 0;

for (int i = 1; i < 101; i++) {

for (int j = 1; j < 101; j++) {

if (table[i][j] == 1 && table[i][j - 1] == 1 && table[i - 1][j] == 1 && table[i - 1][j - 1] == 1) answer += 1;

}

}

cout << answer;

return 0;

}

[17. SAMSUNG SW : Chicken Delivery]

- this was a combination problem. but the key is the way to caclulate distance between each house and the closest chicken house.

- I implemented Combination using DFS.

- At first, I used BFS to find minimum distance to chicken house for each house. but time limit exceeded occurred.

- the key was that we know chicken houses’ positions and houses’ positions. so we don’t need to implement BFS but just calculate minimum distance among all the chicken houses with a house.

- I spent 42 minutes and 32 seconds.

- see the code.

#include<iostream>

#include<vector>

#include<queue>

#include<array>

#define min(a,b) a>b?b:a

using namespace std;

int N, M;

array<array<int, 50>, 50> table = {0,};

vector<pair<int, int>> chicken;

vector<pair<int, int>> house;

vector<pair<int, int>> chosen;

int dir[4][2] = { {-1,0},{0,-1},{1,0},{0,1} };

int answer = 987654321;

int num\_of\_chicken;

int BFS() {

int res = 0;

for (pair<int, int> h : house) {

int hx = h.first; int hy = h.second;

int minimum = 987654321;

for (pair<int, int> c : chosen) minimum = min(minimum, abs(hx - c.first) + abs(hy - c.second));

res += minimum;

}

return res;

}

void Combination(int cnt, int start) {

if (cnt == M) {

answer=min(answer,BFS());

return;

}

for (int i = start; i < num\_of\_chicken; i++) {

chosen.emplace\_back(make\_pair(chicken[i].first,chicken[i].second));

Combination(cnt + 1, i + 1);

chosen.pop\_back();

}

}

int main() {

ios\_base::sync\_with\_stdio(false);

cin.tie(NULL); cout.tie(NULL);

//get input

cin >> N >> M;

//table.assign(N, vector<int>(N, 0));

int input;

for (int i = 0; i < N; i++) {

for (int j = 0; j < N;j++) {

cin >> input;

if (input == 2) chicken.emplace\_back(make\_pair(i, j));

else if (input == 1) {

house.emplace\_back(make\_pair(i, j));

}

}

}

num\_of\_chicken = chicken.size();

//algorithm part

Combination(0, 0);

cout << answer;

return 0;

}

[18. SAMSUNG SW : Cubing]

- how hard to solve thist problem! there was so many hand-work.

- it was a simulation problem. but to simulate rotating a cube, I had to give all the index releated with current side of cube and direction.

- I spent 1 hour and 51 minutes and 39 seconds.

- see the code.

#include<iostream>

#include<vector>

using namespace std;

int T,N;

vector<pair<char, char>> rotation;

void Rotate(vector<vector<char>>& cur, char dir) {

char temp;

if (dir == '+') {

temp = cur[0][0];

cur[0][0] = cur[2][0]; cur[2][0] = cur[2][2]; cur[2][2] = cur[0][2]; cur[0][2] = temp;

temp = cur[0][1];

cur[0][1] = cur[1][0]; cur[1][0] = cur[2][1]; cur[2][1] = cur[1][2]; cur[1][2] = temp;

}

else {

temp = cur[0][0];

cur[0][0] = cur[0][2]; cur[0][2] = cur[2][2]; cur[2][2] = cur[2][0]; cur[2][0] = temp;

temp = cur[0][1];

cur[0][1] = cur[1][2]; cur[1][2] = cur[2][1]; cur[2][1] = cur[1][0]; cur[1][0] = temp;

}

}

void subRotate(vector<char\*> lst, char dir) {

char temp1,temp2,temp3;

if (dir == '+') {

temp1 = \*lst[0]; temp2 = \*lst[1]; temp3 = \*lst[2];

\*lst[2] = \*lst[11]; \*lst[1] = \*lst[10]; \*lst[0] = \*lst[9];

\*lst[11] = \*lst[8]; \*lst[10] = \*lst[7]; \*lst[9] = \*lst[6];

\*lst[8] = \*lst[5]; \*lst[7] = \*lst[4]; \*lst[6] = \*lst[3];

\*lst[5] = temp3; \*lst[4] = temp2; \*lst[3] = temp1;

}

else {

temp1 = \*lst[0]; temp2 = \*lst[1]; temp3 = \*lst[2];

\*lst[0] = \*lst[3]; \*lst[1] = \*lst[4]; \*lst[2] = \*lst[5];

\*lst[3] = \*lst[6]; \*lst[4] = \*lst[7]; \*lst[5] = \*lst[8];

\*lst[6] = \*lst[9]; \*lst[7] = \*lst[10]; \*lst[8] = \*lst[11];

\*lst[9] = temp1; \*lst[10] = temp2; \*lst[11] = temp3;

}

return;

}

void Play(vector<vector<char>> U, vector<vector<char>> F, vector<vector<char>> B, vector<vector<char>> D, vector<vector<char>> L, vector<vector<char>> R) {

for (pair<char, char> rot : rotation) {

switch (rot.first) {

case 'B' :

Rotate(B, rot.second);

subRotate(vector<char\*>{&U[0][2], &U[0][1], &U[0][0], &L[0][0], &L[1][0], &L[2][0], &D[2][0], &D[2][1], &D[2][2], &R[2][2], &R[1][2], &R[0][2]}, rot.second);

break;

case 'U':

Rotate(U, rot.second);

subRotate(vector<char\*>{&B[0][2], & B[0][1], & B[0][0], & R[0][2], & R[0][1], & R[0][0], & F[0][2], & F[0][1], & F[0][0], & L[0][2], & L[0][1], & L[0][0]}, rot.second);

break;

case 'F':

Rotate(F, rot.second);

subRotate(vector<char\*>{&U[2][0], & U[2][1], & U[2][2], & R[0][0], & R[1][0], & R[2][0], & D[0][2], & D[0][1], & D[0][0], & L[2][2], & L[1][2], & L[0][2]}, rot.second);

break;

case 'D':

Rotate(D, rot.second);

subRotate(vector<char\*>{&F[2][0], & F[2][1], & F[2][2], & R[2][0], & R[2][1], & R[2][2], & B[2][0], & B[2][1], & B[2][2], & L[2][0], & L[2][1], & L[2][2]}, rot.second);

break;

case 'L':

Rotate(L, rot.second);

subRotate(vector<char\*>{&U[0][0], & U[1][0], & U[2][0], & F[0][0], & F[1][0], & F[2][0], & D[0][0], & D[1][0], & D[2][0], & B[2][2], & B[1][2], & B[0][2]}, rot.second);

break;

case 'R':

Rotate(R, rot.second);

subRotate(vector<char\*>{&U[2][2], & U[1][2], & U[0][2], & B[0][0], & B[1][0], & B[2][0], & D[2][2], & D[1][2], & D[0][2], & F[2][2], & F[1][2], & F[0][2]}, rot.second);

break;

}

}

for (vector<char> row : U) {

for (char c : row) {

cout << c;

}

cout << endl;

}

}

int main() {

ios\_base::sync\_with\_stdio(false);

cin.tie(NULL); cout.tie(NULL);

// cube initialize.

vector<vector<char>> U = { {'w','w','w'},{'w','w','w'},{'w','w','w'} };

vector<vector<char>> F = { {'r','r','r'},{'r','r','r'},{'r','r','r'} };

vector<vector<char>> D = { {'y','y','y'},{'y','y','y'},{'y','y','y'} };

vector<vector<char>> B = { {'o','o','o'},{'o','o','o'},{'o','o','o'} };

vector<vector<char>> L = { {'g','g','g'},{'g','g','g'},{'g','g','g'} };

vector<vector<char>> R = { {'b','b','b'},{'b','b','b'},{'b','b','b'} };

//get number of testcase;

cin >> T;

for (int q = 0; q < T; q++) {

//get input

cin >> N;

rotation.assign(N, pair<char, char>());

for (int i = 0; i < N; i++) cin >> rotation[i].first >> rotation[i].second;

Play(U, F, B, D, L, R);

}

return 0;

}

[19. SAMSUNG SW : Population Movement]

- it was a BFS problem. To determine unions, I used BFS, and record each country in a vector.

- After determining a union, using vecter that has countries, I allocated population to all the country in the union evenly.

- And if there was a change, I set bool flag == false so as to continue.

- I spent 20 minutes.

- see the code.

#include<iostream>

#include<vector>

#include<queue>

using namespace std;

int N,L,R;

vector<vector<int>> table;

vector<vector<bool>> visit;

int dir[4][2] = { {-1,0},{0,-1},{1,0},{0,1} };

bool flag;

void BFS(int x, int y) {

queue<vector<int>> que;

int sum = table[x][y];

int cnt = 1;

vector<pair<int, int>> uni;

que.push(vector<int>{x, y}); //x y cnt

visit[x][y] = true;

uni.emplace\_back(make\_pair(x, y));

while (!que.empty()) {

x = que.front()[0]; y = que.front()[1]; que.pop();

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

int nx = x + dir[i][0]; int ny = y + dir[i][1];

if (nx >= 0 && ny >= 0 && nx < N && ny < N && visit[nx][ny] == false && L<=abs(table[x][y] - table[nx][ny]) && abs(table[x][y] - table[nx][ny]) <= R) {

visit[nx][ny] = true;

que.push(vector<int>{nx,ny});

uni.emplace\_back(make\_pair(nx, ny));

cnt += 1;

sum += table[nx][ny];

}

}

}

int res = sum / cnt;

if (cnt >= 2) flag = false;

for (pair<int, int> shell : uni) {

table[shell.first][shell.second] = res;

}

}

void printTable() {

cout << endl;

for (vector<int> row : table) {

for (int i : row) cout << i << " ";

cout << endl;

}

cout << endl;

}

int main() {

ios\_base::sync\_with\_stdio(false);

cin.tie(NULL); cout.tie(NULL);

//get input

cin >> N>>L>>R;

table.assign(N, vector<int>(N, 0));

for (int i = 0; i < N; i++) {

for (int j = 0; j < N; j++) {

cin >> table[i][j];

}

}

//algorithm part

int answer = 0;

while (true) {

flag = true;

visit.assign(N, vector<bool>(N, 0));

for (int i = 0; i < N; i++) {

for (int j = 0; j < N; j++) {

if (visit[i][j] == false) {

BFS(i,j);

}

}

}

//printTable();

if (flag) break;

answer += 1;

}

cout << answer;

return 0;

}

[20. SAMSUNG SW : Tree Jaetech]

- it was a simulation problem. there were 4 given rules to simulate : Sping, Summer, Autumn, Winter.

- the time limit was 0.3 seconds. So I thought that it would be hard to reach the time limit. But it was not that matter.

- I used sort() in algorithm header to sort trees with respect to age in a shell.

- In Spring and Summer, if I find a tree that can not eat enough energy, From the tree to end of the list in a shell, I record the trees’ age and remove the trees.

- Autumn was not a big deal, I just had to check index range, and add a tree around a shell.

- I spent 33 minutes.

- see the code.

#include<iostream>

#include<vector>

#include<algorithm>

using namespace std;

int N, M, K;

vector<vector<int>> energy;

vector<vector<int>> table;

vector<vector<vector<int>>> trees;

int dir[8][2] = { {-1,-1},{-1,0},{-1,1},{0,1},{1,1},{1,0},{1,-1},{0,-1} }; //starts at north west, goes clock-wisely.

void printTable() {

cout << endl;

for (vector<vector<int>> row : trees) {

for (vector<int> i : row) cout << i.size() << " ";

cout << endl;

}

cout << endl;

}

void SpringAndSummer() {

for (int i = 0; i < N; i++) {

for (int j = 0; j < N; j++) {

if (trees[i][j].size() >= 1) {

int \_size = trees[i][j].size();

int t;

for (t = 0; t < \_size; t++) {

if (table[i][j] - trees[i][j][t] < 0) break;

else {

table[i][j] -= trees[i][j][t];

trees[i][j][t] += 1;

}

}

for (int e=\_size-1; t<=e; e--) {

table[i][j] += trees[i][j][e] / 2;

trees[i][j].pop\_back();

}

}

}

}

}

void Autumn() {

for (int i = 0; i < N; i++) {

for (int j = 0; j < N; j++) {

if (trees[i][j].size() >= 1) {

int \_size = trees[i][j].size();

for (int t = 0; t < \_size; t++) {

if (trees[i][j][t] % 5 == 0) {

for (int d = 0; d < 8; d++) {

int nx = i + dir[d][0]; int ny = j + dir[d][1];

if (nx >= 0 && ny >= 0 && nx < N && ny < N) {

trees[nx][ny].emplace\_back(1);

}

}

}

}

}

}

}

}

int main() {

ios\_base::sync\_with\_stdio(false);

cin.tie(NULL); cout.tie(NULL);

//get input

cin >> N >> M >> K;

table.assign(N, vector<int>(N, 5));

energy.assign(N, vector<int>(N, 0));

for (int i = 0; i < N; i++) {

for (int j = 0; j < N; j++) {

cin >> energy[i][j];

}

}

trees.assign(N, vector<vector<int>>(N, vector<int>{}));

int x, y, z;

for (int i = 0; i < M; i++) {

cin >> x >> y >> z;

trees[x - 1][y - 1].emplace\_back(z);

}

//algorithm part

for (int year = 1; year <= K; year++) {

SpringAndSummer(); Autumn();

if (year != K) {

for (int i = 0; i < N; i++) {

for (int j = 0; j < N; j++) {

sort(trees[i][j].begin(), trees[i][j].end());//sort

table[i][j] += energy[i][j]; //winter

}

}

}

}

int answer = 0;

for (int i = 0; i < N; i++) {

for (int j = 0; j < N; j++) {

answer += trees[i][j].size();

}

}

cout << answer;

return 0;

}

[21. SAMSUNG SW : Baby Shark]

- it was a simulation problem. but a trick was the way the shark moves.

- since if there were several prey that the shark can reach in same second, shark eats the highest prey and if their height is same, then eats the leftmost prey.

- To solve this trick, I had to record preys that the shark can reach in same seconds.

- I used BFS, and set to get highest with leftmost prey with O(1) time. Notice that insert a value into a set take O(log n) time.

- I spent 1 hour and 11 minutes and 3 seconds.

- see the code.

#include<iostream>

#include<vector>

#include<queue>

#include<set>

using namespace std;

int N;

vector<vector<int>> table;

vector<vector<bool>> visit;

int bx, by;

int dir[4][2] = { {-1,0},{0,-1},{0,1},{1,0} };

void printTable() {

cout << endl;

for (vector<int> row : table) {

for (int i : row) cout << i << " ";

cout << endl;

}

}

/\*

printTable();

cout << "x : " << nx << " y : " << ny << endl;

cout << "size : " << \_size + 1 << " sec : " << sec + 1 << endl;

\*/

int BFS() {

queue<vector<int>> que;

que.push(vector<int>{bx, by, 2,0}); // x y size sec

visit[bx][by] = true;

int eat = 0;

int last\_sec = 0;

while (!que.empty()) {

int sec = que.front()[3];

set<pair<int, int>> fish = {};

int \_size = que.front()[2];

while (!que.empty()) {

int x = que.front()[0]; int y = que.front()[1]; int prev\_sec = que.front()[3];

if (sec != prev\_sec) break;

que.pop();

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

int nx = x + dir[i][0]; int ny = y + dir[i][1];

if (nx >= 0 && ny >= 0 && nx < N && ny < N && visit[nx][ny] == false && table[nx][ny] <= \_size) {

if (table[nx][ny] != 0 && table[nx][ny] < \_size) {

fish.insert(make\_pair(nx, ny));

}

else {

visit[nx][ny] = true;

que.push(vector<int>{nx, ny, \_size, sec + 1});

}

}

}

}

if (fish.size() > 0) {

last\_sec = sec+1;

set<pair<int, int>>::iterator iter = fish.begin();

int nx = iter->first; int ny = iter->second;

table[nx][ny] = 0;

eat += 1;

visit.assign(N, vector<bool>(N, false));

table[nx][ny] = 0;

visit[nx][ny] = true;

que = queue<vector<int>>{};

if (eat == \_size) {

que.push(vector<int>{nx, ny, \_size + 1, sec + 1});

eat = 0;

}

else {

que.push(vector<int>{nx, ny, \_size, sec + 1});

}

}

}

return last\_sec;

}

int main() {

ios\_base::sync\_with\_stdio(false);

cin.tie(NULL); cout.tie(NULL);

//get input

cin >> N;

table.assign(N, vector<int>(N, 0));

visit.assign(N, vector<bool>(N, 0));

for (int i = 0; i < N; i++) {

for (int j = 0; j < N; j++) {

cin >> table[i][j];

if (table[i][j] == 9) {

bx = i; by = j;

table[i][j] = 0;

}

}

}

//algorithm part

int answer = BFS();

cout << answer;

return 0;

}

[22. SAMSUNG SW : Goodbye Finedust!]

- it was a weird problem. I will tell you later.

- it was a simulation problem.

- there was an airconditioner and I had to implement the way that the airconditioner clean air.

- it was not that hard. but the weird thing occurred.

- as you can see below code. there is a comment block that is an original function Contation I maded.

- and there is anoter Contagion function that has a code of resetting 2D array and reallocate dust value.

- we usually can think that the comment block’s Contagion function would be faster than the other.

- since there is no resetting of 2D array.

- but when I submitted the code. the former one was stucked by time limit exceeded.

- Surpisingly, the latter one passed. what the heck?

- Even now, my brain doesn’t understand why those things happened.

- Because of above reason, I spent quite long time. I spent 53 minutes and 44 seconds.

- see the code.

#include<iostream>

#include<vector>

using namespace std;

int N, M, T;

int tx, bx;

vector<vector<int>> table;

int dir[4][2] = { {-1,0},{0,-1},{1,0},{0,1} };

/\*

void Contagion() {

vector<vector<int>> stk;

for (int i = 0; i < N; i++) {

for (int j = 0; j < M; j++) {

if (table[i][j] > 0) {

int res5 = table[i][j] / 5;

int num\_of\_contagion = 0;

for (int d = 0; d < 4; d++) {

int x = i + dir[d][0]; int y = j + dir[d][1];

if (x >= 0 && y >= 0 && x < N && y < M && table[x][y] != -1) {

stk.emplace\_back(vector<int>{x, y, res5});

num\_of\_contagion += 1;

}

}

table[i][j] -= (res5 \* num\_of\_contagion);

}

}

}

for (vector<int> row : stk) table[row[0]][row[1]] += row[2];

}

\*/

void Contagion() {

vector<vector<int>> new\_dust;

for (int i = 0; i < N; i++) {

for (int j = 0; j < M; j++) {

int x = i; int y = j;

if (table[x][y] == -1) continue;

int count = 0;

if (table[x][y] > 0) {

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

int tx = x + dir[i][0]; int ty = y + dir[i][1];

if (tx >= 0 && ty >= 0 && tx < N && ty < M && table[tx][ty] != -1) {

new\_dust.push\_back(vector<int>{tx, ty, table[x][y] / 5});

count++;

}

}

}

table[x][y] -= (table[x][y] / 5) \* count;

if (table[x][y] != 0) new\_dust.push\_back(vector<int>{x, y, table[x][y]});

}

}

table = vector<vector<int>>(N, vector<int>(M));

table[tx][0] = -1; table[bx][0] = -1;

for (vector<int> pos : new\_dust) table[pos[0]][pos[1]] += pos[2];

}

void Cleaning() {

//top - AntiClockWise

int x = tx; int y = 0;

for (int i = x-1; i >= 1; i--) table[i][0] = table[i - 1][0];

for (int j = 0; j < M - 1; j++) table[0][j] = table[0][j + 1];

for (int i = 0; i <= x - 1; i++) table[i][M - 1] = table[i + 1][M - 1];

for (int j = M - 1; j >= y + 1; j--) {

table[x][j] = table[x][j - 1];

}

table[x][y + 1] = 0;

//bottom - ClockWise

x = bx;

for (int i = x+1; i < N - 1; i++) table[i][0] = table[i + 1][0];

for (int j = 0; j < M - 1; j++) table[N - 1][j] = table[N - 1][j + 1];

for (int i = N - 1; i >= x + 1; i--) table[i][M - 1] = table[i - 1][M - 1];

for (int j = M - 1; j >= y + 1; j--) table[x][j] = table[x][j - 1];

table[x][y + 1] = 0;

}

int main() {

ios\_base::sync\_with\_stdio(false);

cin.tie(NULL); cout.tie(NULL);

//get input

cin >> N >> M >> T;

table.assign(N, vector<int>(M, 0));

bool flag = true;

for (int i = 0; i < N; i++) {

for (int j = 0; j < M; j++) {

cin >> table[i][j];

}

}

for (int i = 0; i < N; i++)

if (table[i][0] == -1) {

tx = i; bx = i + 1;

break;

}

//algorithm part

for (int t = 0; t < T; t++) {

Contagion();

Cleaning();

}

int answer = 2; //Airconditioner's position.

for (vector<int> row : table) {

for (int i : row) answer += i;

}

cout << answer;

return 0;

}

[23. SAMSUNG SW : Fishing King]

- It was a simulation problem. To reach the time limit, I had to use a trick.

- Each shark has their own direcition and speed. Futhermore, the given area is restricted.

- so we can calcluate a shark’s future position int a certain extent.

- like if a shark’s direction is north and speed is “s” then, the shark will be at start position with

opposite direction after (x \* 2) move, x is the shark’s row position.

- I spent 1 hour and 5 minutes.

- see the code.

#include<iostream>

#include<vector>

#include<map>

using namespace std;

int N, M, K; // row col num of shark

vector<vector<vector<int>>> table;

int answer = 0;

int dir[4][2] = { {-1,0},{1,0},{0,1},{0,-1} };

void printTable() {

cout << endl;

for (vector<vector<int>> row : table) {

for (vector<int> i : row) {

if (i.empty()) cout << "0" << " ";

else cout << i[2] << " ";

}

cout << endl;

}

cout << endl;

}

void GetShark(int y) {

for (int i = 0; i < N; i++) {

if (table[i][y].size() != 0) {

answer += table[i][y][2]; //size

table[i][y].clear();

break;

}

}

}

void Move(map<pair<int, int>, vector<int>>& stk, int x, int y, int s, int d, int z) {

int os = s;

while ((d==0&&x\*2<s) ||(d==1 &&((N-1-x)\*2<s))) {

if (d == 0) {

s -= (x\*2);

d = 1;

}

else {

s -= (N - 1 - x) \* 2;

d = 0;

}

}

while ((d == 2 && ((M - 1 - y) \* 2) < s) || (d == 3 && y \* 2 < s) ) {

if (d == 2) {

s -= (M - 1 - y) \* 2;

d = 3;

}

else {

s -= y \* 2;

d = 2;

}

}

for (int i = 0; i < s; i++) {

x = x + dir[d][0]; y = y + dir[d][1];

if (x < 0) {

d = 1;x = 1;

}

else if (y < 0) {

d = 2; y = 1;

}

else if (x >= N) {

d = 0; x = N - 2;

}

else if (y >= M) {

d = 3; y = M - 2;

}

}

if (stk.find(make\_pair(x, y)) == stk.end()) stk[make\_pair(x, y)] = vector<int>{os,d,z};

else if (stk[make\_pair(x, y)][2] >= z) return;

else stk[make\_pair(x, y)] = vector<int>{ os,d,z };

}

void MoveShark() {

map<pair<int, int>, vector<int>> stk = {};

for (int i = 0; i < N; i++) {

for (int j = 0; j < M; j++) {

if (!table[i][j].empty()) {

Move(stk, i, j, table[i][j][0], table[i][j][1], table[i][j][2]);

table[i][j] = vector<int>{};

}

}

}

for (map<pair<int, int>, vector<int>>::iterator iter = stk.begin(); iter != stk.end(); iter++)

table[iter->first.first][iter->first.second] = vector<int>{ iter->second[0],iter->second[1],iter->second[2] };

}

int main() {

ios\_base::sync\_with\_stdio(false);

cin.tie(NULL); cout.tie(NULL);

//get input

cin >> N >> M >> K;

table.assign(N, vector<vector<int>>(M, vector<int>()));

int a, b, c, d, e;

for (int i = 0; i < K; i++) {

cin >> a >> b >> c >> d >> e;

table[a - 1][b - 1] = vector<int>{ c,d-1,e }; //speed direction size

}

//algorithm part

for (int j = 0; j < M; j++) {

GetShark(j);

//printTable();

if (j == M - 1) break;

MoveShark();

//printTable();

}

cout << answer;

return 0;

}

[24. SAMSUNG SW : 2D Array and Calculation]

- it was a sorting problem. but it was quite hard.

- since I had to re allocate 2D array’s size by increasing row or column’s size.

- in terms of vector, columns size increasing is easy due to emplace or push\_back, but row size increasing is not that easy.

- I used stable\_sort in algorhtm header.

- I spent 57 minutes and 9 seconds.

- see the code.

#include<iostream>

#include<vector>

#include<map>

#include<algorithm>

#define max(a,b) a>b?a:b

using namespace std;

int r, c, k;

vector<vector<int>> table(3, vector<int>(3, 0));

bool row\_turn = true;

void PrintTable() {

cout << endl;

for (vector<int> row : table) {

for (int i : row) cout << i << " ";

cout << endl;

}

cout << endl;

}

int main() {

ios\_base::sync\_with\_stdio(false);

cin.tie(NULL); cout.tie(NULL);

//get input

cin >> r >> c >> k;

r -= 1; c -= 1;

for (int i = 0; i < 3; i++) {

for (int j = 0; j < 3; j++) {

cin >> table[i][j];

}

}

//algorithm part

int answer = 0;

int row\_size = 3;

int col\_size = 3;

while (answer <= 101) {

if (answer > 100) {

answer = -1;

break;

}

if (row\_size>r && col\_size >c &&table[r][c] == k) {

break;

}

answer += 1;

vector<vector<int>> n\_table;

int pre\_row\_size = row\_size;

int pre\_col\_size = col\_size;

if (row\_turn) {

vector<vector<pair<int, int>>> order(pre\_row\_size, vector<pair<int, int>>());

for (int i = 0; i < pre\_row\_size; i++) {

map<int, int> stk;

for (int j = 0; j < pre\_col\_size; j++) {

if (table[i][j] == 0) continue;

stk[table[i][j]] += 1;

}

for (map<int, int>::iterator iter = stk.begin(); iter != stk.end(); iter++)

order[i].emplace\_back(make\_pair(iter->first, iter->second));

stable\_sort(order[i].begin(), order[i].end(), [](pair<int, int> a, pair<int, int> b) {return a.second < b.second; });

col\_size = max(col\_size, order[i].size() \* 2);

}

if (col\_size > 100) col\_size = 100;

n\_table.assign(row\_size, vector<int>(col\_size, 0));

for (int i = 0; i < row\_size && i<100; i++) {

for (int j = 0; j < order[i].size() && j<100; j++) {

n\_table[i][j\*2] = order[i][j].first;

n\_table[i][(j \* 2) + 1] = order[i][j].second;

}

}

}

else {

vector<vector<pair<int, int>>> order(pre\_col\_size, vector<pair<int, int>>());;

for (int j = 0; j < pre\_col\_size; j++) {

map<int, int> stk;

for (int i = 0; i < pre\_row\_size; i++) {

if (table[i][j] == 0) continue;

stk[table[i][j]] += 1;

}

for (map<int, int>::iterator iter = stk.begin(); iter != stk.end(); iter++)

order[j].emplace\_back(make\_pair(iter->first, iter->second));

stable\_sort(order[j].begin(), order[j].end(), [](pair<int, int> a, pair<int, int> b) {return a.second < b.second; });

row\_size = max(row\_size, order[j].size() \* 2);

}

if (row\_size > 100) row\_size = 100;

n\_table.assign(row\_size, vector<int>(col\_size, 0));

for (int j = 0; j < col\_size && j<100; j++) {

for (int i = 0; i < order[j].size() && i<100; i++) {

n\_table[i\*2][j] = order[j][i].first;

n\_table[(i \* 2) + 1][j] = order[j][i].second;

}

}

}

table = n\_table;

if (row\_size >= col\_size) row\_turn = true;

else row\_turn = false;

}

cout << answer;

return 0;

}

[25. SAMSUNG SW : Laboratory 3]

- It was a simulation problem. I chose viruses using DFS-Combination and simulated it using BFS.

- but there was an edge condition that what we should do if we met an unactivated virus.

- In the case, we had to check whether all the shell has a virus even if we don’t pass the virus.

- I spent 41 minutes and 55 seconds.

- see the code.

#include<iostream>

#include<vector>

#include<queue>

#define min(a,b) a>b?b:a

#define max(a,b) a>b?a:b

using namespace std;

int N, M;

vector<vector<int>> table;

vector<vector<bool>> visit;

vector<pair<int, int>> virus;

vector<pair<int, int>> chosen;

int dir[4][2] = { {-1,0},{0,-1},{1,0},{0,1} };

int num\_of\_virus = 0;

int answer = 987654321;

bool isValid(vector<vector<int>>& temp\_table) {

for (int i = 0; i < N; i++) {

for (int j = 0; j < N; j++) {

if (temp\_table[i][j] == 0) return false;

}

}

return true;

}

int BFS() {

vector<vector<bool>> temp\_visit = visit;

vector<vector<int>> temp\_table = table;

queue<vector<int>> que;

for (pair<int, int> cur : chosen) que.push(vector<int>{cur.first, cur.second, 0});

int res = 0;

while (!que.empty()) {

int x = que.front()[0]; int y = que.front()[1]; int sec = que.front()[2]; //int index = que.front()[3];

que.pop();

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

int nx = x + dir[i][0]; int ny = y + dir[i][1];

if (nx >= 0 && ny >= 0 && nx < N && ny < N && temp\_table[nx][ny] !=1 && temp\_visit[nx][ny] == false) {

temp\_visit[nx][ny] = true;

if (temp\_table[nx][ny] == 1000 && isValid(temp\_table))

return res;

temp\_table[nx][ny] = 2;

que.push(vector<int>{nx, ny, sec + 1});

res = max(res, sec + 1);

}

}

}

if (isValid(temp\_table)) return res;

return 987654321;

}

void Combination(int cnt, int start) {

if (cnt == M) {

int res = BFS();

answer=min(res,answer);

return;

}

for (int i = start; i < num\_of\_virus; i++) {

int x = virus[i].first; int y = virus[i].second;

chosen.emplace\_back(make\_pair(x, y));

visit[x][y] = true;

table[x][y] = 2;

Combination(cnt + 1, i + 1);

table[x][y] = 1000;

visit[x][y] = false;

chosen.pop\_back();

}

}

int main() {

ios\_base::sync\_with\_stdio(false);

cin.tie(NULL); cout.tie(NULL);

//get input

cin >> N >> M;

table.assign(N, vector<int>(N, 0));

visit.assign(N, vector<bool>(N, false));

for (int i = 0; i < N; i++) {

for (int j = 0; j < N; j++) {

cin >> table[i][j];

if (table[i][j] == 2) {

table[i][j] = 1000;

virus.emplace\_back(make\_pair(i, j));

}

}

}

num\_of\_virus = virus.size();

//algorithm part

Combination(0,0);

if (answer == 987654321) cout << -1;

else cout << answer;

return 0;

}

[26. SAMSUNG SW : Gerrymandering 3]

- It was a simulation problem. Actuaaly it was a classification problem.

- I can have divided the given area to 5 district and find minimum difference between a maximum population district and a minimum population district

- it was quite hard to find the way to divide the area. I tried lots of things. Finally, I found an easy way.

- First, drawing boundaries for 4 diagonal shape of district 5.

- Second, for each district, if we meet district 5, stop allocating number to a shell.

- I spent 1 hour and 27 minutes and 47 seconds.

- see the code.

#include<iostream>

#include<vector>

#include<algorithm>

#define max(a,b) a>b?a:b

#define min(a,b) a>b?b:a

using namespace std;

int N;

vector<vector<int>> table;

vector<vector<int>> dis;

int answer = 987654321;

void printTable() {

cout << endl;

for (int i = 0; i < N; i++) {

for (int j = 0; j < N; j++) {

cout << dis[i][j] << " ";

}

cout << endl;

}

}

void Gerrymandering(int x, int y, int d1, int d2) {

if (!(d1 >= 1 && d2 >= 1 && 1 <= x + 1 && x + 1 < x + 1 + d1 + d2 && x + 1 + d1 + d2 <= N && 1 <= y + 1 - d1 && y + 1 - d1 < y + 1 && y + 1 < y + 1 + d2 && y + 1 + d2 <= N)) return;

for (int i = x, j = y; i <= x + d1 && j >= y - d1; i++,j--) {

dis[i][j] = 5;

}

for (int i = x, j = y; i <= x + d2 && j <= y + d2; i++,j++) {

dis[i][j] = 5;

}

for (int i = x + d1, j = y - d1; i <= x + d1 + d2 && j <= y - d1 + d2; i++,j++) {

dis[i][j] = 5;

}

for (int i = x + d2, j = y + d2; i <= x + d1 + d2 && j >= y + d2 - d1; i++, j--) {

dis[i][j] = 5;

}

for (int i = 0; i < x + d1; i++) {

for (int j = 0; j < y+1; j++) {

if (dis[i][j] == 5) break;

dis[i][j] = 1;

}

}

for (int i = 0; i < x + d2+1; i++) {

for (int j = N - 1; j >= y+1; j--) {

if (dis[i][j] == 5) break;

dis[i][j] = 2;

}

}

for (int i = x + d1; i < N; i++) {

for (int j = 0; j < y - d1 + d2; j++) {

if (dis[i][j] == 5) break;

dis[i][j] = 3;

}

}

for (int i = x + d2+1; i < N; i++) {

for (int j = N - 1; j >= y - d1 + d2; j--) {

if (dis[i][j] == 5) break;

dis[i][j] = 4;

}

}

vector<int> sum(6,0);

for (int i = 0; i < N; i++) {

for (int j = 0; j < N; j++) {

sum[dis[i][j]] += table[i][j];

}

}

sum[0] += sum[5];

sum.pop\_back();

sort(sum.begin(), sum.end());

answer = min(answer, sum[4] - sum[0]);

}

int main() {

ios\_base::sync\_with\_stdio(false);

cin.tie(NULL); cout.tie(NULL);

//get input

cin >> N;

table.assign(N, vector<int>(N, 0));

for (int i = 0; i < N; i++) {

for (int j = 0; j < N; j++) {

cin >> table[i][j];

}

}

//algorithm part

for (int x = 0; x <= N - 3; x++) {

for (int y = 1; y <= N - 2; y++) {

for (int d1 = 1; x+d1<=N-2 && y-d1>=0; d1++) {

for (int d2 = 1; x + d2 <= N - 2 && y + d2 <= N - 1; d2++) {

dis.assign(N, vector<int>(N, 0));

Gerrymandering(x, y, d1, d2);

}

}

}

}

cout << answer;

return 0;

}

[27. SAMSUNG SW : New Game 2]

- It was a simulation problem. there were 3 colored shell : white, blue, red.

- Basic rule is if a horse move and a next shell has a horse, then the moving horse will be placed above the next shell’s horse.

- In case of white, a horse move to the shell with horses above itself.

- In case of blue or chess board, a horse move to opposite direction. if the opposite shell is also blue or out of chess board, stop.

- In case of red, a horse move to the shell with horses above itself with reversed order.

- see the code.

#include<iostream>

#include<vector>

#include<algorithm>

using namespace std;

int N, K;

vector<vector<int>> table;

vector<vector<vector<int>>> stk;

vector<vector<int>> horse; // order, x,y,dir

int dir[4][2] = { {0,1},{0,-1},{-1,0},{1,0} }; // r l n s

int main() {

ios\_base::sync\_with\_stdio(false);

cin.tie(NULL); cout.tie(NULL);

//get input

cin >> N >> K;

table.assign(N, vector<int>(N, 0));

stk.assign(N, vector<vector<int>>(N, vector<int>()));

for (int i = 0; i < N; i++) {

for (int j = 0; j < N; j++) {

cin >> table[i][j];

}

}

int q, w, e;

horse.assign(K, vector<int>(3, 0));

for (int i = 0; i < K; i++) {

cin >>q>>w>>e;

horse[i][0] = q-1; horse[i][1] = w-1; horse[i][2] = e - 1;

stk[q-1][w-1].emplace\_back(i);

}

//algorithm part

int answer = 0;

bool flag = false;

while (!flag) {

if (answer > 1000) {

answer = -1;

break;

}

answer += 1;

for (int i = 0; i < K; i++) {

int x = horse[i][0]; int y = horse[i][1]; int d = horse[i][2];

int nx = x + dir[d][0]; int ny = y + dir[d][1];

if (nx < 0 || ny < 0 || nx >= N || ny >= N || table[nx][ny] == 2) { //blue or out of range

if (d == 0) d = 1;

else if (d == 1) d = 0;

else if (d == 2) d = 3;

else if (d == 3) d = 2;

nx = x + dir[d][0]; ny = y + dir[d][1];

horse[i][2] = d;

if (nx < 0 || ny < 0 || nx >= N || ny >= N || table[nx][ny] == 2) {

continue;

}

}

if (table[nx][ny] == 0) {

vector<int> temp;

for (int j = stk[x][y].size() - 1; j >= 0; j--) {

horse[stk[x][y][j]][0] = nx; horse[stk[x][y][j]][1] = ny;

temp.emplace\_back(stk[x][y][j]);

if (stk[x][y][j] == i) break;

}

for (int j = temp.size() - 1; j >= 0; j--) {

stk[nx][ny].emplace\_back(temp[j]);

stk[x][y].pop\_back();

}

}

else if (table[nx][ny] == 1) {

for (int j = stk[x][y].size() - 1; j >= 0; j--) {

horse[stk[x][y][j]][0] = nx; horse[stk[x][y][j]][1] = ny;

stk[nx][ny].emplace\_back(stk[x][y][j]);

if (stk[x][y][j] == i) break;

stk[x][y].pop\_back();

}

stk[x][y].pop\_back();

}

if (stk[nx][ny].size() >= 4) {

flag = true;

break;

}

}

}

cout << answer;

return 0;

}

[28. SAMSUNG SW : Rotating Circle Board]

- it was a simulation problem. even thought they gave us a circle board. we can represent it as a 2D array if we check y index’s range.

- To rotate the circle board, I calculated after rotating index first and rearragne the values following the indice.

- To remove adjacent values, I used BFS and visit 2D array.

- I Spent 35 minutes and 19 seconds.

- see the code.

#include<iostream>

#include<vector>

#include<queue>

using namespace std;

int N, M, T;

vector<vector<int>> table;

vector<vector<int>> order;

bool flag = true;

int dir[4][2] = { {-1,0},{0,-1},{1,0},{0,1} };

void PrintTable() {

cout << endl;

for (vector<int> row : table) {

for (int i : row) cout << i << " ";

cout << endl;

}

cout << endl;

}

void Rotate(int i, int d, int k) {

k = k % M;

vector<int> temp(M,0);

vector<int> temp\_table = table[i];

if (d == 0) { //clockwise

for (int i = 0; i < M; i++) temp[i] = (i+k)%M;

}

else { //anti clock-wise

int t;

for (int i = 0; i < M; i++) {

t = i - k;

if (t < 0) t = M + t;

temp[i] = t;

}

}

for (int j = 0; j < M; j++) table[i][temp[j]] = temp\_table[j];

}

void Remove() {

vector<vector<bool>> visit(N,vector<bool>(M,false));

for (int i = 0; i < N; i++) {

for (int j = 0; j < M; j++) {

if (table[i][j] != 0 && visit[i][j] == false) {

int cur = table[i][j];

queue<vector<int>> que;

que.push(vector<int>{i, j});

visit[i][j] = true;

while (!que.empty()) {

int x = que.front()[0]; int y = que.front()[1]; que.pop();

for (int d = 0; d < 4; d++) {

int nx = x + dir[d][0]; int ny = y + dir[d][1];

if (ny >= M) ny = 0; else if (ny < 0) ny = M - 1;

if (nx >= 0 && nx < N && visit[nx][ny]==false && table[nx][ny]==cur) {

table[x][y] = 0;

visit[nx][ny] = true;

table[nx][ny] = 0;

flag = false;

que.push(vector<int>{nx, ny});

}

}

}

}

}

}

}

void Average() {

int cnt = 0;

double sum = 0;

for (int i = 0; i < N; i++) {

for (int j = 0; j < M; j++) {

if (table[i][j] == 0) continue;

cnt += 1;

sum += table[i][j];

}

}

double average = sum / cnt;

for (int i = 0; i < N; i++) {

for (int j = 0; j < M; j++) {

if (table[i][j] == 0) continue;

else if (table[i][j] > average) table[i][j] -= 1;

else if (table[i][j] < average) table[i][j] += 1;

}

}

}

bool isEmpty() {

for (vector<int> row : table) {

for (int i : row) if (i != 0) return false;

}

return true;

}

int main() {

ios\_base::sync\_with\_stdio(false);

cin.tie(NULL); cout.tie(NULL);

//get input

cin >> N >> M >> T;

table.assign(N, vector<int>(M, 0));

for (int i = 0; i < N; i++) {

for (int j = 0; j < M; j++) {

cin >> table[i][j];

}

}

order.assign(T, vector<int>(3, 0));

for (int i = 0; i < T; i++) {

cin >> order[i][0] >> order[i][1] >> order[i][2];

}

//algorithm part

for (vector<int> row : order) {

int x = row[0]; int d = row[1]; int k = row[2];

for (int i = x-1; i < N; i += x) Rotate(i, d, k);

flag = true;

if (!isEmpty()) {

Remove();

if (flag) Average();

}

}

int answer = 0;

for (int i = 0; i < N; i++) {

for (int j = 0; j < M; j++) {

answer += table[i][j];

}

}

cout << answer;

return 0;

}

[29. SAMSUNG SW : Dice Yut Game]

- It was a permutation problem with simulation.

- For Yut Game, I had to implement look-up table by myself. It was a manual labour.

- To implement permutation, I used DFS. I hesitated since permutation usually consume lots of computational time. but there was no time limet exceeded.

- In terms of look-up table, I used map<int,vector<int>>, the key is an index of a shell on the board and the value[0] is a score a shell has, value[1:5] is next shell after moving 1~5 shell.

- I spent 45 minutes and 35 seconds.

- see the code.

#include<iostream>

#include<vector>

#include<map>

#define max(a,b) a>b?a:b

using namespace std;

vector<int> order(10,0);

vector<int> horse(4,0);

map<int, vector<int>> table; // index : value , after 1,2,3,4,5

int answer = 0;

vector<bool> visit(33, false);

void InitBoard() {

//making Yut board

table[0] = vector<int>{ 0,1,2,3,4,5 }; // start position

table[1] = vector<int>{ 2,2,3,4,5,6 };

table[2] = vector<int>{ 4,3,4,5,6,7 };

table[3] = vector<int>{ 6,4,5,6,7,8 };

table[4] = vector<int>{ 8,5,6,7,8,9 };

table[5] = vector<int>{ 10,22,23,24,30,31 }; //first blue circle

table[6] = vector<int>{ 12,7,8,9,10,11 };

table[7] = vector<int>{ 14,8,9,10,11,12 };

table[8] = vector<int>{ 16,9,10,11,12,13 };

table[9] = vector<int>{ 18,10,11,12,13,14 };

table[10] = vector<int>{20,25,26,30,31,32};// second blue circle, 28 is just before 40

table[11] = vector<int>{ 22,12,13,14,15,16 };

table[12] = vector<int>{ 24,13,14,15,16,17 };

table[13] = vector<int>{ 26,14,15,16,17,18 };

table[14] = vector<int>{ 28,15,16,17,18,19 };

table[15] = vector<int>{30,27,28,29,30,31};// third blue circle

table[16] = vector<int>{ 32,17,18,19,20,21 };

table[17] = vector<int>{ 34,18,19,20,21,21 };

table[18] = vector<int>{ 36,19,20,21,21,21 };

table[19] = vector<int>{ 38,20,21,21,21,21 };

table[20] = vector<int>{ 40,21,21,21,21,21 };

table[21] = vector<int>{ 0,21,21,21,21,21 }; //end position

table[22] = vector<int>{ 13,23,24,30,31,32 };

table[23] = vector<int>{ 16,24,30,31,32,20 };

table[24] = vector<int>{ 19,30,31,32,20,21 };

table[25] = vector<int>{ 22,26,30,31,32,20 };

table[26] = vector<int>{ 24,30,31,32,20,21 };

table[27] = vector<int>{ 28,28,29,30,31,32 };

table[28] = vector<int>{ 27,29,30,31,32,20 };

table[29] = vector<int>{ 26,30,31,32,20,21 };

table[30] = vector<int>{ 25,31,32,20,21,21 };

table[31] = vector<int>{ 30,32,20,21,21,21 };

table[32] = vector<int>{ 35,20,21,21,21,21 };

}

void DFS(int cnt, int res) {

if (cnt == 10 ) {

answer = max(answer, res);

return;

}

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

if (horse[i] == 21) continue;

int next\_shell= table[horse[i]][order[cnt]];

if (next\_shell != 21 && visit[next\_shell] == true) continue;

int temp = horse[i];

horse[i] = next\_shell;

visit[temp] = false;

visit[horse[i]] = true;

DFS(cnt + 1,res+table[next\_shell][0]);

visit[horse[i]] = false;

visit[temp] = true;

horse[i] = temp;

}

}

int main() {

ios\_base::sync\_with\_stdio(false);

cin.tie(NULL); cout.tie(NULL);

//get input

for (int i = 0; i < 10; i++) cin >> order[i];

InitBoard();

//algorithm part

DFS(0,0);

cout << answer;

return 0;

}