[1. n! 끝자리 0의 개수]

- n!을 할 때 인수 5가 몇 개 들어가는지 세면됨

ex) n=30 : 5, 10, 15 20, 25=(5x5), 30 이므로 6개.

[2. 인접하지 않은 인덱스만 골라서 원소 합 최대 구하기 – House robber]

- DP로 푸는 거임. 배열하나 설정하고, 해당 배열에 max(현재인덱스 + 2번째 전 인덱스, 1번째 전인덱스)를 하면서, 계속 더해나가면 됨.

int rob(vector<int>& nums) {

if(nums.empty() || nums.size()==0) return 0;

if(nums.size()==1) return nums[0];

int prev\_1=nums[1];

int prev\_2=nums[0];

int cur;

for(int i=2;i<nums.size();i++){

if(nums[i]+prev\_2>prev\_1){

cur=prev\_1;

prev\_1=nums[i]+prev\_2;

prev\_2=max(cur,prev\_2);

}

else

prev\_2=prev\_1;

}

return max(prev\_1,prev\_2);

}

[3. 이진수 스트링의 합]

- 스트링을 인트로 변환하려면 atoi(s.c\_str); 을 사용하면 됨.

- 인트를 바이너리 스트링으로 출력하려면 biset 라이브러리를 이용하면 됨.

ex) biset<셋의 크기>(인트 변수).to\_string();

- 또는 answer = sum%2; sum/=2 를 한후 얻은 문자열을 reverse하면 됨.

- 이 문제는 각 스트링의 오른쪽 끝부터 시작하여 carry값을 유지하며 진행함.

즉, 둘다 1이었을 경우 합은 2 인데, 2%2==0 이므로, 해당 위치에는 0을 대입하고 캐리 값은 2 /1 = 1을 가지고 올라감. 그럼으로써, 캐리가 될 수 있도록 함.

string addBinary(string a, string b) {

int i=a.size()-1;

int j=b.size()-1;

int cur=0;

string answer;

while(i>=0 || j>=0|| cur!=0){

if(i>=0){

cur+=a[i]=='0'?0:1;

i--;

}

if(j>=0){

cur+=b[j]=='0'?0:1;

j--;

}

answer+=((cur%2)==0?'0':'1');

cur/=2;

}

reverse(answer.begin(),answer.end()); //이렇게 해주는 것이 O(n)으로 풀수있는 방법임.

return answer;

}

[4. 배열의 각 원소들을 연이어서 수를 만들 때, 그 수의 +1을 리턴하는 문제 – Plus One]

- 즉, [4,3,2,1] 이면 [4,3,2,2]를 리턴하는 문제, 왜냐하면 4321+1=4322

수의 범위가 크지 않다면 아래 방법처럼 간단한 덧셈과 인트 -> 스트링 -> 벡터변환으로 풀 수 있음.

vector<int> plusOne(vector<int>& digits) {

int answer=digits[0];

for(int i=1;i<digits.size();i++){

answer=answer\*10+digits[i];

}

answer++;

string temp=to\_string(answer);

vector<int> ret;

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

ret.push\_back(temp[i]-48);

}

return ret;

}

그러나 수의 범위가 크다면, 덧셈을 이용할 수 없음. 아래 방법을 이용하면 됨.

vector<int> plusOne(vector<int>& digits) {

vector<int>::reverse\_iterator iter=digits.rbegin();

//rbegin 하려면 reverse\_iterator 라고 명시해줘야한다.

vector<int> answer;

int carry=1;

for(;iter!=digits.rend();iter++){

if(carry+\*iter>=10){

carry=1;

answer.push\_back(0);

}

else{

answer.push\_back(\*iter+carry);

carry=0;

}

}

// 인풋이 9 였을 때, answer에 10 이 들어가야되는데 0 만들어감.

// 이런 경우른 대비해서 끝났을 때 carry가 1이면 1을 한번더 push 해줌

if(carry==1)

answer.push\_back(1);

reverse(answer.begin(),answer.end());

return answer;

}

[4. 해당 수가 power of prime인지 확인하는 문제]

- 사실 power of three 문제였는데, 3도 소수이므로 동일하게 풀 수 있다.

- 3의 제곱들 중 최댓 값을 안다면(1162261467, Math.log(0x7fffffff)) 해당 수가 3으로 나누어지면 power of three이다.

- 마찬가지로 다른 소수의 제곱들 중 최댓 값을 알면 풀 수 있다.

[5. 옆 사람과 가장 멀리 떨어져 앉는 문제]

- 모든 인덱스에 대해서 나의 오른쪽 사람과의 거리를 right 배열에 저장하고, 왼쪽 사람과의 거리를 left에 저장한다. 그 후 두 배열의 같은 인덱스에 대해서 min값을 구하고, 최종적으로 구해진 배열의 max값을 리턴하면 된다.

- 이때 양쪽 끝에는 사람이 없는 것으로 간주한다. 이것 때문에 나의 알고리즘이 속도가 많이 느려졌다. 이것을 처리하려고 count를 사용했는데, 이때 마지막 자리와 끝 자리와의 거리가 구해진다면, 그 사이의 거리는 min처리에서 제대로 처리되므로 상관 없다.

-90.24%의 속도가 나왔다.

int alloc\_right(int\* cur, vector<int>& seats,int index,int count){

if(index>=seats.size()) return count+1;

if(seats[index]==1) return 1;

cur[index]=alloc\_right(cur,seats,index+1,count+1);

return cur[index]+1;

}

int alloc\_left(int\* cur, vector<int>& seats,int index,int count){

if(index<0) return count+1;

if(seats[index]==1) return 1;

cur[index]=alloc\_left(cur,seats,index-1,count+1);

return cur[index]+1;

}

int maxDistToClosest(vector<int>& seats) {

int\* left=new int[seats.size()]{0,};

int\* right=new int[seats.size()]{0,};

for(int i=0, j=0;i<seats.size();i++){

if(seats[i]==1) continue;

j=i;

i=i+alloc\_right(right,seats,j,0)-1;

}

for(int i=seats.size()-1,j=i;i>=0;i--){

if(seats[i]==1) continue;

j=i;

i=i-alloc\_left(left,seats,j,0)+1;

}

vector<int> answer;

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

answer.push\_back(min(left[i],right[i]));

}

int maximum=INT\_MIN;

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

maximum=max(maximum,answer[i]);

return maximum;

}

[6. 큐로 스택 구현하기]

- 큐는 항상 front에서 pop하기 때문에, stack처럼 LIFO로 pop 하려면, pop할 때 queue.size()-1만큼 큐 앞에 원소를 지워서 뒤에 붙여준 후, 마지막에 pop을 해야 됨. 이때 C++ 에서는 pop을 하면 원소가 지워지기만 하고 void 가 return 되므로, queue.front() 로 먼저 해당 원소를 저장해 주어야 함.

queue<int> Que;

/\*\* Push element x onto stack. \*/

void push(int x) {

Que.push(x);

}

/\*\* Removes the element on top of the stack and returns that element. \*/

int pop() {

int a;

for(int i=0;i<Que.size()-1;i++){

Que.push(Que.front());

Que.pop();

}

a=Que.front();

Que.pop();

return a;

}

/\*\* Get the top element. \*/

int top() {

return Que.back();

}

/\*\* Returns whether the stack is empty. \*/

bool empty() {

return Que.empty();

}

[7. 균형트리인지 확인하기]

-postOrder로 트리를 탐색하는 데, 이때 재귀로 각 트리의 왼쪽 서브트리와 오른쪽 서브트리의 높이 중 큰 값을 취하여 리턴해주고, 만약 왼쪽, 오른쪽 서브트리의 높이차가 1이상이면 flag=false로 하고, 아니면 falg=true인 상태로 놔둠.

bool flag=true;

int postOrder(TreeNode\* root){

if(!root) return 0;

int left=0,right=0;

left+=postOrder(root->left);

right+=postOrder(root->right);

if(abs(left-right)>1)

flag=false;

return max(left,right)+1;

}

bool isBalanced(TreeNode\* root) {

postOrder(root);

return flag;

}

[8. License key Formatting – 주어진 K에 대해서 dash 사이에 K개의 문자를 배치하는 문제]

- 문자열 전체를 대문자로 바꾸기 위해서는 std::transform(S.begin(),S.end(),S.begin(), ::toupper)를 하면 된다.

- 문자열 끝에서부터 시작하여 dash를 제외한 alphanumeric의 개수를 count로 세주고, given K만큼 세어졌을 때, dash를 추가한다.

- 또한 문자열 맨 처음의 연속된 dash를 방지하기 위하여, dash가 아닌 제일 첫번째 문자의 인덱스를 문자열 시작점으로 한다.

string licenseKeyFormatting(string S, int K) {

string answer;

int count=1;

int begin=0;

while(S[begin]=='-') begin++; // 맨 앞에 연이은 dash를 방지함.

for(int i=S.size()-1;i>=begin;i--){

if(count%(K+1)==0){

answer+='-';

count=1;

i++; // K개 만큼 센후 dash추가하려는 데, 그 원소가 dash가 아닐 경우를 대비.

continue;

}

if(S[i]!='-'){

answer+=S[i];

count++;

}

}

std::transform(answer.begin(),answer.end(),answer.begin(),::toupper);

reverse(answer.begin(),answer.end());

return answer;

}

[9. Find Pivot Index – 인덱스를 기준으로 왼쪽과 오른쪽 원소들의 합이 같은 인덱스(pivot)을 찾는문제]

int pivotIndex(vector<int>& nums) {

nums.push\_back(0);

int pivot=0;

int left=0;

int right=0;

for(int i=1;i<nums.size();i++)

right+=nums[i];

while(pivot<nums.size()-1){

if(left==right)

break;

left+=nums[pivot];

right-=nums[pivot+1];

pivot++;

}

return pivot==nums.size()-1? -1 : pivot;

}

[10. power of two – 주어진 수가 2의 제곱 꼴인지 확인하는 문제]

- 비트연산을 이용하여 푼다. 2의 제곱들은 1000, 0100, 0010, 0001 등 비트형태에서 1을 하나 가지고 있다. 이때 2의 제곱들 -1은 0111, 0011, 0001, 0000 등 2의 제곱들보다 한자리 낮은 수부터 1로 가득 차있다.

-그러므로 n&(n-1)==0 이면 2의 제곱이다.

bool isPowerOfTwo(int n) {

return n>0 && (n&(n-1))==0 ;

}

[11. Reverse Vowels of a String – 모음 위치를 반대 순서로 바꾸는 문제]

- 한번 훑으면서 모음을 벡터에 저장하고, 그 모음의 인덱스를 다른 벡터에 저장한 후, 인덱스의 역순에 따라서 모음을 넣어주면 됨.

-주의할 점은 vector가 비었을 때, vec.back()을 호출하면 heap overflow 에러가 발생한다는 것임.

if(s.size()<2)

return s;

vector<char> table;

vector<int> index;

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

if(s[i]=='a'||s[i]=='e'||s[i]=='i'||s[i]=='o'||s[i]=='u'||

s[i]=='A'||s[i]=='E'||s[i]=='I'||s[i]=='O'||s[i]=='U'){

table.push\_back(s[i]);

index.push\_back(i);

}

}

reverse(index.begin(),index.end());

string answer;

int i=0;

while(i<s.size()){

if(!index.empty()&&i==index.back()){

answer+=table.back();

table.pop\_back();

index.pop\_back();

}

else{

answer+=s[i];

}

i++;

}

return answer;

[12. **Second Minimum Node In a Binary Tree – 두개의 자식 중 최소 값이 부모가 되는 이진트리에서 두번째 최소값 찾는 문제]**

- 전부 순회를 하되, 두번째 값을 LONG\_MAX로 초기화 해놓고, 순회할때마다 해당 노드의 value가 최소값(루트값) 보다 크고, 현재의 second값보다 작은지 확인하여, 조건이 만족하면 second=value를 하면 됨.

int first;

long second;

void search(TreeNode\* root){

if(!root) return;

if(first<root->val && root->val<second)

second=root->val;

search(root->left);

search(root->right);

}

int findSecondMinimumValue(TreeNode\* root) {

first=root->val;

second=LONG\_MAX;

search(root);

return second==LONG\_MAX? -1: second;

}

[13. **Convert a Number to Hexadecimal – 십진수를 16진수 스트링으로 바꾸는 문제]**

- 스트링하나에 0부터 f까지 지정해 준 후, 주어진 십진수를 16으로 % 연산을 하여 해당 인덱스의 값을 결과 값에 더해나간다.

- 마지막에 해당 값을 reverse해주면 된다.

- 이때 -를 다루기 위해서는 unsigned int 로 인풋값을 재 지정해준다. 그럴경우 -1이 unsigned int의 최대값이 된다.

string toHex(int num) {

unsigned int n=num;

string table="0123456789abcdef";

string answer;

do{

answer+=table[n%16];

n/=16;

}while(n);

return {answer.rbegin(),answer.rend()};

}

[14 **Quad Tree Intersection - 4개의 자식이 있는 두개의 트리에 대하여 각 자식끼리의 OR연산을 하는 문제]**

**-** 난 아래처럼 최종 모습이 4개의 자식을 가진 노드를 만드는 건줄 알았는 데, 아닌 듯?

bool search(Node\* root){

if(root->isLeaf) return root->val;

return search(root->topLeft)||search(root->topRight)||search(root->bottomLeft)||search(root->bottomRight);

}

Node\* intersect(Node\* quadTree1, Node\* quadTree2) {

bool one=search(quadTree1->topLeft)||search(quadTree2->topLeft);

bool two=search(quadTree1->topRight)||search(quadTree2->topRight);

bool three=search(quadTree1->bottomLeft)||search(quadTree2->bottomLeft);

bool four=search(quadTree1->bottomRight)||search(quadTree2->bottomRight);

Node\* answer=new Node();

answer->val = true;

answer-> isLeaf = false;

answer->topLeft = new Node(one,true,nullptr,nullptr,nullptr,nullptr);

answer->topRight = new Node(two,true,nullptr,nullptr,nullptr,nullptr);

answer->bottomLeft = new Node(three,true,nullptr,nullptr,nullptr,nullptr);

answer->bottomRight = new Node(four,true,nullptr,nullptr,nullptr,nullptr);

return answer;

}

- 마지막에 4개의 자식이 모두 같은 boolean 값을 가지면 하나로 합쳐줘야함.

- 아래 2개의 방법은 다른 사람들이 푼 것으로, 읽으니까 이해할 수 있었음.

Node\* intersect(Node\* quadTree1, Node\* quadTree2) {

if(quadTree1->isLeaf && quadTree1->val) return quadTree1;

if(quadTree2->isLeaf && quadTree2->val) return quadTree2;

if(quadTree1->isLeaf && !quadTree1->val) return quadTree2;

if(quadTree2->isLeaf && !quadTree2->val) return quadTree1;

auto tl = intersect(quadTree1->topLeft, quadTree2->topLeft);

auto tr = intersect(quadTree1->topRight, quadTree2->topRight);

auto bl = intersect(quadTree1->bottomLeft, quadTree2->bottomLeft);

auto br = intersect(quadTree1->bottomRight, quadTree2->bottomRight);

if(tl->val == tr->val && tl->val == bl->val && tl->val == br->val && tl->isLeaf && tr->isLeaf && bl->isLeaf && br->isLeaf)

return new Node(tl->val, true, nullptr, nullptr, nullptr, nullptr);

else

return new Node(false, false, tl, tr, bl, br);

}

Node\* intersect(Node\* quadTree1, Node\* quadTree2)

{

// if a is leaf+true, choose a.

// if b is leaf+false, choose a.

if (quadTree1->isLeaf && quadTree1->val == true ||

quadTree2->isLeaf && quadTree2->val == false)

return quadTree1;

// if b is leaf+true, choose b.

// if a is leaf+false, choose b;

if (quadTree2->isLeaf && quadTree2->val == true ||

quadTree1->isLeaf && quadTree1->val == false)

return quadTree2;

// intersect.

Node\* ret = new Node (false, false,

intersect (quadTree1->topLeft, quadTree2->topLeft),

intersect (quadTree1->topRight, quadTree2->topRight),

intersect (quadTree1->bottomLeft, quadTree2->bottomLeft),

intersect (quadTree1->bottomRight, quadTree2->bottomRight));

// merge if all children are leaves and have the same value.

if (ret->topLeft->val == ret->topRight->val && ret->topLeft->val == ret->bottomLeft->val &&

ret->topLeft->val == ret->bottomRight->val &&

ret->topLeft->isLeaf && ret->topRight->isLeaf && ret->bottomLeft->isLeaf && ret->bottomRight->isLeaf)

{

ret->val = ret->topLeft->val;

ret->isLeaf = true;

ret->topLeft = ret->topRight = ret->bottomLeft = ret->bottomRight = NULL;

}

// done.

return ret;

}

[15. **Most Common Word – 문자열에서 금지된 단어를 제외하고 가장 많이 등장하는 단어를 찾아내는 문제]**

- map을 이용해서 각 단어마다 등장횟수를 센 후, map을 순회하면서 해당 단어가 banned되어 있지 않고, 현재의 단어보다 많이 등장하였다면 answer를 바꾸어 줌.

- 문자열을 소문자로 바꾸기 위해서는 transfrom(s.begin(),s.end(),s.begin(),::tolower)를 하면 됨.

- vector안에서 해당 아이템, 또는 문자열을 찾기 위해서는 find(vec.begin(),vec.end(),item)을 하면 해당 item의 iterator가 반환됨. 그러므로 !=vec.end() 를 조건으로 주면 해당 아이템이 없으면 false, 있으면 true가 됨.

std::transform(paragraph.begin(),paragraph.end(),paragraph.begin(),::tolower);

unordered\_map<string,int> table;

string temp;

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

if('a'<=paragraph[i] && paragraph[i]<='z')

temp+=paragraph[i];

else{

table[temp]++;

temp="";

}

}

table[temp]++; //마지막에 고려되지 못한 단어를 위해서

unordered\_map<string,int>::iterator iter=table.begin();

string answer="";

int count=0;

for(;iter!=table.end();iter++){

if(std::find(banned.begin(),banned.end(),iter->first)==banned.end()&&iter->second>count

&& 'a'<=iter->first[0] && iter->first[0]<='z'){// 해당 문자열이 banned이고, iter가 구두점이나 공백이 아니라면

answer=iter->first;

count=iter->second;

}

}

return answer;

}

[16. **Subtree of Another Tree – 트리 s가 트리 t를 서브트리로 가지고 있는지 확인하는 문제]**

**-** 이렇게 푸는게 맞는지 모르겠는데, 일단 오래걸렸음.

- 먼저 트리 t의 shape을 vector에 담고, 트리 s에서 트리 t의 root를 찾은 순간, 그때부터 트리 t와 같은 subtree구조를 가졌는지 확인하는 방식임.

bool answer=false;

vector<int> t\_table;

int i;

int first;

void t\_search(TreeNode\* root){ //4, 1, min, min, 2, min, min

if(!root){

t\_table.push\_back(INT\_MIN);

return;

}

t\_table.push\_back(root->val);

t\_search(root->left);

t\_search(root->right);

}

void s\_search(TreeNode\* root){

if(!root) return;

if(root->val==first){ //if tree t's root is found, then check whether the subtree is same with t

check(root);

}

if(answer==true)

return;

s\_search(root->left);

s\_search(root->right);

}

void check(TreeNode\* root){

if(!root && t\_table[i]==INT\_MIN){

if(i==t\_table.size()-1){

answer=true;

return;

}

i++;

return;

}

if((!root && t\_table[i]!=INT\_MIN) || ( root &&root->val !=t\_table[i])){

i=0;

return;

}

i++;

check(root->left);

check(root->right);

}

bool isSubtree(TreeNode\* s, TreeNode\* t) {

t\_search(t);

first=t->val;

i=0;

s\_search(s);

return answer;

}

- 그리고 훨씬 좋은 solution을 봐버렸다.

class Solution {

public:

bool isSubtree(TreeNode\* s, TreeNode\* t) {

if(!s) return false;

if (isSame(s,t)) return true;

return isSubtree(s->left,t) || isSubtree(s->right,t);

}

bool isSame(TreeNode \*s, TreeNode \*t)

{

if (!s && !t) return true;

if (!s || !t) return false;

if (s->val != t->val) return false;

return isSame(s->left, t->left) && isSame(s->right, t->right);

}

};

[17. **Remove Duplicates from Sorted List – 링크드리스트에서 중복 원소 제거하는 문제]**

- Discussion 찾아보니까 new로 생성한거 아니면 dlete할 필요 없다는 거 같은데 잘 모르겠음.

- 일단 같은 원소가 아닐때까지 loop돌고 다른원소 발견하면 이전 원소의 맨 앞에거랑 연결해주는 방식으로 했음. 이것이 속도가 좀 빠름.

ListNode\* deleteDuplicates(ListNode\* head) {

ListNode\* answer= head;

ListNode\* temp;

int value;

while(head){

value=head->val;

temp=head;

while(temp->next && value==temp->next->val) temp=temp->next;

if(!temp->next){

head->next=nullptr;

head=head->next;

break;

}

else{

head->next=temp->next;

head=head->next;

}

}

return answer;

}

[18. **Add to Array-Form of Intege – 벡터로 정수가 주어지고, 또 다른 정수 K가 주어져서 두개를 더한 후 벡터모양을 리턴하는 문제]**

- 벡터를 일단 정수로 바꿔서 더할려고 했는데, 정수 값을 엄청 크게 줘서 안됨.

- 그래서 carry를 사용해서 풀었는 데, 내가봤는데도 너무 느린 효율을 가짐

- 참고로 K%10 은 마지막 숫자를 리턴해줌. K/10은 마지막 숫자를 제외한 숫자를 리턴함.

vector<int> addToArrayForm(vector<int>& A, int K) {

int cur=0;

int carry=0;

vector<int> answer;

for(int i=A.size()-1;i>=0;i--){

cur=A[i]+K%10+carry;

if(cur>=10) {

carry=1;

cur%=10;

}

else carry=0;

K/=10;

answer.push\_back(cur);

}

while(K){ //A의 자릿수가 K보다 작은 경우를 cover함.

cur=K%10+carry;

if(cur>=10){

carry=1;

cur%=10;

}

else carry=0;

answer.push\_back(cur);

K/=10;

}

if(carry==1) answer.push\_back(1); //답이 1082 막 이런데 벡터에 082 이런거 까지만 들어가는 경우 cover함

return {answer.rbegin(),answer.rend()};

}

- 이거는 나랑 같은 방식인데 설명을 더 잘해놓음.

/\* An important observation ---

1) num%10 gives us the last digit of a number

2) num = num/10 cuts off the last digit of the number

3) numVector.back() gives us the last digit of the number in vector form

4) numVector.pop\_back() cuts off the last digit of the number in vector form

5) The extra space required can be reduced by overwriting the first vector.

\*/

class Solution

{

public:

vector<int> addToArrayForm(vector<int>& a, int k);

};

/\* Returns the sum of 2 numbers in vector form \*/

vector<int> Solution :: addToArrayForm(vector<int>& a, int k)

{

// Get the length of the first number

int n = a.size();

// Vector to store the answer

vector<int> answer;

/\* Start adding both the numbers from the end \*/

int carry = 0;

// As long as one of the number exists, keep adding them

while(!a.empty() || k!=0)

{

// Get the last digits of both the numbers. If a vector has finished off, the last digit is zero

int lastDigit\_1 = a.empty() ? 0 : a.back();

int lastDigit\_2 = k%10;

// Sum up the digits and add the carry

int sum = lastDigit\_1 + lastDigit\_2 + carry;

answer.push\_back(sum%10);

carry = sum/10;

// Remove the last digits of both the numbers

if(!a.empty()) a.pop\_back();

k = k/10;

}

// If the carry is remaining, add it

if(carry!=0) answer.push\_back(carry);

// Reverse the answer, since we were summing up from the end

reverse(answer.begin(), answer.end());

// Return the answer in vector format

return answer;

}

[19. **Long Pressed Name – 중복적으로 눌린 키보드 문자로 원래의 이름을 만들 수 있는지 확인하는 문제.]**

- 문제 설명하기가 힘들어서 원본을 붙여 넣음.

Your friend is typing his name into a keyboard.  Sometimes, when typing a character c, the key might get long pressed, and the character will be typed 1 or more times.

You examine the typed characters of the keyboard.  Return True if it is possible that it was your friends name, with some characters (possibly none) being long pressed.

- 두 문자열이 match 되면 pass, 아니면 long pressed인지 확인하여 그만큼 string typed를 진행시키고, 아니면 return false.

- while 문이 종료되었을 때, i가 name.size()보다 작으면, string typed의 character의 숫자가 모자른 것이므로 false.

bool isLongPressedName(string name, string typed) {

int i=0,j=0;

char prev=name[0];

while(j<typed.size()){

// if the two strings is matched.

if(name[i]==typed[j]){

prev=name[i];

i++;j++;

continue;

}

//if typed has been long pressed.

if(prev!=name[i] && prev==typed[j]){

while(prev==typed[j]) j++;

continue;

}

return false;

}

//if the type has not enough character.

return i>=name.size()? true : false;

}

[20. **Path Sum III – 주어진 트리의 노드들을 더하여 주어진 sum을 만들 수 있는 path가 몇 개인지 구하는 문제]**

- 두개의 재귀를 생각해야해서 어려운 문제임. 덕분에 좋은 공부가 되었음.

- 전수조사 밖에 답이없는 것 같음.

int search(TreeNode\* root, int sum, int pre){

if(!root) return 0;

return (root->val+pre == sum) + search(root->left,sum,pre+root->val) + search(root->right,sum,pre+root->val);

}

int pathSum(TreeNode\* root, int sum) {

if(!root) return 0;

return search(root,sum,0) + pathSum(root->left,sum)+pathSum(root->right,sum);

}

[21. **Find Smallest Letter Greater Than Target – 주어진 target보다 큰 가장 작은 character를 반환하는 문제]**

- 그냥 간단하게 target의 ascii code값 보다 큰 값이 나오면 return 해주는 것으로 했음, 끝날때까지 찾지 못하면, 첫번째 원소. 이거는 O(n)임.

char nextGreatestLetter(vector<char>& letters, char target) {

if(letters[letters.size()-1]<=target) return letters[0];

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

if(target<letters[i])

return letters[i];

}

return letters[0];

}

-이것을 이진탐색으로도 풀 수 있음.

char nextGreatestLetter(vector<char>& letters, char target) {

int left=0;

int right=letters.size()-1;

int mid;

while(left<=right){

mid=(right-left)/2+left; // to handle INT overflow.

if(mid==0){

if(letters[mid]>target) return letters[0];

else{

left=mid+1;

continue;

}

}

if(letters[mid-1]<=target && letters[mid]>target) return letters[mid];

else if(letters[mid]<=target)

left=mid+1;

else right=mid-1;

}

return letters[0];

}

- upper\_bound를 사용한 풀이 방법도 있음. 이거 알아두자.

- upper\_bound는 이진탐색 기반의 탐색 법임. 사용전 list가 정렬되어 있어야함.

- key값을 초과하는 가장 첫번째 원소의 반복자를 리턴함.

- upper\_bound(vec.begin(),vec.end(),value)의 형식으로 사용함.

- lower\_bound는 해당 value를 포함하여 이상인 값을 리턴함. 즉, lower는 이상, upper는 초과!

char nextGreatestLetter(vector<char>& letters, char target) {

vector<char>::iterator iter=upper\_bound(letters.begin(),letters.end(),target);

return iter==letters.end() ? letters[0] : \*iter;

}

p.s) 세상 많이 편해졌다.

[22. **Longest Harmonious Subsequence – max와 min의 차이가 정확히 1인 subsequence 찾는 문제]**

- 여기서 subsequence는 substring이 아니라, 조합될수 있는 모든 가지수임.

ex) apple의 subsequence는 a,ap,al,ae,app,apl,ape … 이런 식임.

- 이 문제는 map을 활용하여 2번의 for문을 쓰는게 관건이었음.

- 생각하기 쉽지 않아서 Discussion의 도움을 받음.

int findLHS(vector<int>& nums) {

unordered\_map<int,int> table;

for(int i : nums){

table[i]++;

}

int answer=0;

unordered\_map<int,int>::iterator iter=table.begin();

for(;iter!=table.end();iter++){

if(table.count(iter->first-1)>0)

answer=max(answer,iter->second+table[iter->first-1]);

}

return answer;

}

[23. **Longest Continuous Increasing Subsequence – 벡터 내 오름차순 부분배열 중 가장 긴 길이를 구하는 문제] 2020-01-26 7:30.**

- empty인 vector에 대해서 vec.empty()로 체크하는 것 보다 vec.size()==0 으로 체크하는 것이 훨씬 빠르다.

int findLengthOfLCIS(vector<int>& nums) {

if(nums.size()==0) return 0;

int count=1;

int answer=1;

int i=0;

while(i<nums.size()-1){

while(i+1<nums.size() && nums[i]<nums[i+1]){

count++;

i++;

}

answer=max(answer,count);

count=1;

i++;

}

return answer;

}

[24. **Symmetric Tree – 트리가 대칭 모양인지 구하는 문제]**

- 나는 left subtree와 right subtree를 preorder search를 통해서 vector에 각 value값을 저장하고, 마지막에 두개의 벡터가 같은 값을 가지는지 확인하였음.

- 이때 null node에 대해서는 INT\_MIN을 대입함으로써, 모양을 기억함.

vector<int> left\_tree;

vector<int> right\_tree;

void leftSearch(TreeNode\* root){

if(!root){

left\_tree.push\_back(INT\_MIN);

return;

}

left\_tree.push\_back(root->val);

leftSearch(root->left);

leftSearch(root->right);

}

void rightSearch(TreeNode\* root){

if(!root){

right\_tree.push\_back(INT\_MIN);

return;

}

right\_tree.push\_back(root->val);

rightSearch(root->right);

rightSearch(root->left);

}

bool isSymmetric(TreeNode\* root) {

if(!root) return true;

leftSearch(root->left);

rightSearch(root->right);

if(left\_tree.size()!=right\_tree.size())

return false;

for(int i=0;i<left\_tree.size();i++)

if(left\_tree[i]!=right\_tree[i])

return false;

return true;

}

- Discussion의 방법인데, 좀 더 빠르고 간단함.

bool isSymmetric(TreeNode \*root) {

if (!root) return true;

return helper(root->left, root->right);

}

bool helper(TreeNode\* p, TreeNode\* q) {

if (!p && !q) {

return true;

} else if (!p || !q) {

return false;

}

if (p->val != q->val) {

return false;

}

return helper(p->left,q->right) && helper(p->right, q->left);

}

[25. **Maximum Subarray – 최대부분배열문제] 2020-01-27 4:30 PM.**

- dp를 이용해서 푸는건데, dp 벡터에 현재 인덱스의 값과 바로 이전까지 구한 최대부분배열을 더하여 현재 인덱스 값보다 크면 그 값을 그 인덱스의 dp값으로 하는 방식임.

- O(n) solution인데, 떠올리기 쉽지 않음.  
int maxSubArray(vector<int>& nums) {

int maximum=nums[0];

vector<int> dp;

dp.push\_back(nums[0]);

for(int i=1;i<nums.size();i++){

dp.push\_back(max(nums[i],nums[i]+dp[i-1]));

maximum=max(maximum,dp[i]);

}

return maximum;

}

[26. **Base 7 – 주어진 수를 7진수 string으로 표현하는 문제]**

- 이게 인풋이 그냥 7이면 곱하기 10을 해도 7%7=0 이므로 자릿수가 올라가지 않아서, 처음부터 string으로 다루는 것이 포인트임.

string convertToBase7(int num) {

if(num==0)

return "0";

string base7;

int temp=num;

num=abs(num);

while(num){

base7+=to\_string(num%7);

num/=7;

}

reverse(base7.begin(),base7.end());

return temp<0? '-'+base7 : base7;

}

[27. **Add Strings – 두개의 string으로 주어진 정수를 더하는 문제] 2020-01-27 5:45 PM**

- string 최대길이가 5100이므로 정수로 바꿔서 더할 수 없고, carry를 활용해야함.

- 더 짧게 짤 수 있을 것 같긴한데, 알고리즘은 대체로 나와 같을 것이라 예상됨.

string addStrings(string num1, string num2) {

int carry=0;

string answer;

int temp;

int i,j;

for(i=num1.size()-1,j=num2.size()-1;i>=0 && j>=0 ; i--,j--){

temp=(num1[i]-48)+(num2[j]-48)+carry;

answer+=to\_string(temp%10);

if(temp>=10) carry=1;

else carry=0;

}

while(i>=0){

temp=(num1[i]-48+carry);

answer+=to\_string(temp%10);

i--;

if(temp>=10) carry=1;

else carry=0;

}

while(j>=0){

temp=(num2[j]-48+carry);

answer+=to\_string(temp%10);

j--;

if(temp>=10) carry=1;

else carry=0;

}

if(carry)

answer+=to\_string(carry);

return {answer.rbegin(),answer.rend()};

}

[28. **Climbing Stairs – 계단을 한번에 1개나 2개씩 오를 수 있을 때, 특정 높이의 계단까지 가는 방법의 수를 구하는 문제] 2020-01-27 6:20**

**-** 간만에 DP를 풀어서 그런지, 피보나치임을 눈으로 보고도 알아채지 못 했음.

int climbStairs(int n) {

if(n<=2) return n;

int temp;

int prev=1;

int cur=2;

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

temp=cur;

cur=prev+cur;

prev=temp;

}

return cur;

}

[29. **Binary Watch – 손목시계에 들어온 LED의 개수로 표현할 수 있는 모든 시간을 구하는 문제] 2020-01-27 7:00**

- 이건 내가 못푸는 문제임. Discussion 보고도 한참 걸렸음.

- bitset을 이용하는 문제임. 어렵다.

- bitset<10> a으로 선언할 수 있는데, a에 자리수가 10개인 비트를 저장할 수 있다는 것이다.

이때 bitset<10>(3) a 로 선언하면, 10개의 비트에 대해서 십진수 3을 이진수로 표현하여 a에 저장해준다. 즉 0000 00011 이 된다.

- a.set() 을 하면 모든 bit이 1이되고, a.reset() 하면 0이 된다. a.set(3,true)로 하면 3+1인 4번 비트를 1로 설정한다.

- a.flip()을 하면 모든 비트를 반전하고, a.flip(3)을 하면 3+1인 4번 비트를 반전한다.

- a.to\_string() 하면 전체 비트를 string화 시킨다.

- a[4], a.test[4] 처럼 배열처럼 접근할 수 있다.

- a.any()는 비트 중 하나라도 1이면 1을 반환하고, a.none()은 모두 0이어야만 1을 반환한다.

- a.test(n)은 n+1번째 비트가 1인지 0인지 검사한다.

vector<string> readBinaryWatch(int num) {

vector<string> answer;

for(int h=0;h<12;h++){

for(int m=0;m<60;m++){

if(bitset<10>(h).count()+bitset<10>(m).count()==num)

answer.push\_back(to\_string(h)+":"+(m<10? "0":"")+to\_string(m));

}

}

return answer;

}

-이런걸 생각해 내는 사람들도 대단함.

[30. **Student Attendance Record I – 학생이 상을 받을 수 있는 출석률을 가졌는지 아닌지 판단하는 문제]**

- 간단한 문제임. if문을 여러 개 사용하면 됨.

- 결석이 2회이상이면 탈락이고, 3연속 지각을 하면 탈락임.

bool checkRecord(string s) {

bool absent=false;

int late=0;

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

if(absent && s[i]=='A')

return false;

else if(s[i]=='A'){

absent=true;

late=0;

continue;

}

else if(late==2&&s[i]=='L')

return false;

else if(s[i]=='L'){

late++;

continue;

}

late=0;

}

return true;

}

[31. **Number of 1 Bits – uint\_8 로 주어진 bit에 1이 몇 개인지 세는 문제]**

**-** bit연산을 통해서 풀었음. 주어진 n과 1을 &연산을 하여 true면 count++를 함. 그 이후 n=n>>1을 하여 n==0일 때까지 loop를 돌림.

int hammingWeight(uint32\_t n) {

int count=0;

while(n){

n&1? count++:count;

n=n>>1;

}

return count;

}

[32. **Pascal's Triangle II – 파스칼 트라이앵글의 특정 row의 원소를 구하는 문제]**



- 벡터 메모리 해제를 위해 vector<int>().swap(answer)를 이용하였다.

vector<int> getRow(int rowIndex) {

if(rowIndex==0) return {1};

vector<int> prev(2,1);

vector<int> answer;

vector<int> res=prev;

for(int i=0;i<rowIndex-1;i++){

answer.push\_back(1);

for(int j=0;j<prev.size()-1;j++){

answer.push\_back(prev[j]+prev[j+1]);

}

answer.push\_back(1);

prev=answer;

res=answer;

answer.clear();

}

vector<int>().swap(prev); // deallocate prev

vector<int>().swap(answer);

return res;

}

- 다음 문장에 의거하여 더 빠르게 풀 수 있다.

*Based on math, the kth element for nth row is C(n, k) = n! / (k!\*(n-k)!), then res[k] = res[n - k]*

*so the relationship between res[i] and res[i-1] is n! / (k!(n-k)!) / n!****/****((k-1)!(n-k + 1)!) = (n - k + 1) / k;*

*Note that this solution is math derived from number of Combinations.*

*Each line of Pascal's Triangle is a full set of Combination number based on k .*

*comb(k,p) = k! /( p! \*(k-p)!) = comb(k,k-p)*

*if p < k-p*

*comb(k,p) = comb(k,p-1) \* (k-p+1) / p*

*Because :*

*comb(k,p) = [ k \* (k-1) \* (k-2) \*... (k-p+1)] / [1 \* 2 \* 3 \*...(p)]*

vector<int> getRow(int rowIndex) {

vector<int> answer(rowIndex+1,1);

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

answer[i]=answer[rowIndex-i]=(long)answer[i-1]\*(long)(rowIndex-i+1)/i;

}

return answer;

}

[33. **Implement Queue using Stacks – 스택을 이용해 큐를 구현하는 문제] 2020-01-28 5:00**

- 스택을 2개 사용하여, push는 스택1에 push하고, pop이나 peek이 호출된 순간, 스택2가 비어있으면 스택1의 모든 원소를 pop하여 스택 2에 넣고, 스택 2의 top을 리턴한다. 스택2가 비어있지 않을 경우 그냥 스택 2의 탑을 리턴한다.

- 이때 pop 호출시 해당 top을 삭제하여 준다.

stack<int> stk1;

stack<int> stk2;

/\*\* Push element x to the back of queue. \*/

void push(int x) {

stk1.push(x);

}

/\*\* Removes the element from in front of queue and returns that element. \*/

int pop() {

if(!stk2.empty()){

int temp=stk2.top();

stk2.pop();

return temp;

}

while(!stk1.empty()){

stk2.push(stk1.top());

stk1.pop();

}

int temp=stk2.top();

stk2.pop();

return temp;

}

/\*\* Get the front element. \*/

int peek() {

if(!stk2.empty())

return stk2.top();

while(!stk1.empty()){

stk2.push(stk1.top());

stk1.pop();

}

return stk2.top();

}

/\*\* Returns whether the queue is empty. \*/

bool empty() {

if(stk1.empty() && stk2.empty())

return true;

return false;

}

[34. **Number of Equivalent Domino Pairs - 2차원 벡터에서 순서에 상관없이 동일한 원소를 가진 행벡터를 구하는 문제] 2020-01-28 5:20**

- 나는 일단 잘 안떠올라서 sort시키고 시작했음. sort 후 map의 키로서 pair<int,int>를 활용함.

- map의 키로서 pair를 사용할 경우 unordered\_map은 사용 불가능함. 그래서 더 느린듯.

int numEquivDominoPairs(vector<vector<int>>& dominoes) {

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

sort(dominoes[i].begin(),dominoes[i].end());

map<pair<int,int>,int> table;

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

table[make\_pair(dominoes[i][0],dominoes[i][1])]++;

}

map<pair<int,int>,int>::iterator iter=table.begin();

int answer=0;

for(;iter!=table.end();iter++)

answer+=(iter->second)\*(iter->second -1)/2;

return answer;

}

- 다른 방법을 찾아봤는데, Discussion 에서 이런 신박한 방법이 있었음.

**Explanation**

You need to distinguish the different dominoes and count the same.

I did it in this way:  
f(domino) = min(d[0], d[1]) \* 10 + max(d[0], d[1])  
For each domino d, calculate min(d[0], d[1]) \* 10 + max(d[0], d[1])  
This will put the smaller number on the left and bigger one on the right (in decimal).  
So same number same domino, different number different domino.

Take the example from the problem:  
dominoes = [[1,2],[2,1],[3,4],[5,6]]  
now we transform it into [12,12,34,56].

int numEquivDominoPairs(vector<vector<int>>& dominoes) {

unordered\_map<int, int> count;

int res = 0;

for (auto& d : dominoes) {

res += count[min(d[0], d[1]) \* 10 + max(d[0], d[1])]++;

}

return res;

}

- 천재 xx들…

[35. **Maximum Product of Three Numbers – 벡터의 원소 중 3개를 골라 곱하였을 때의 최댓값을 구하는 문제] 2020-01-28 5:48**

- 최댓값은 벡터에 음수가 존재할 경우, 가장 작은 음수 2개와 가장 큰 양수 하나를 곱하거나, 가장 큰 양수 3개를 차례로 곱하는 2가지 방법만이 존재한다.

- 그러므로 가장 큰 양수 3개와, 가장 작은 음수 2개를 찾은 후 두개를 비교하면 된다.

int maximumProduct(vector<int>& nums) {

int max1=INT\_MIN+2,max2=INT\_MIN+1, max3=INT\_MIN;

int min1=INT\_MAX-1,min2=INT\_MAX;

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

if(nums[i]>max1){

max3=max2;

max2=max1;

max1=nums[i];

}

else if(nums[i]>max2){

max3=max2;

max2=nums[i];

}

else if(nums[i]>max3) max3=nums[i];

if(nums[i]<min1){

min2=min1;

min1=nums[i];

}

else if(nums[i]<min2) min2=nums[i];

}

return max(max1\*max2\*max3,min1\*min2\*max1);

}

[36. **Pairs of Songs With Total Durations Divisible by 60 – 주어진 벡터에서 두 원소의 합이 60으로 나누어 떨어지는 원소 쌍의 개수를 구하는 문제] 2020-01-28 6:14**

- 일단 map을 이용하여, 60-원소값 즉, 해당 원소가 60으로 나누어떨어지기 위해서 더해져야 하는 값을 확인하고, 해당 값이 존재하면 answer에 더해준다.

- 이때 i<j인 쌍에서만 확인한다. 즉 중복된 두개의 원소쌍에 대해서는 한번만 연산한다는 것이 중요하다.

- table에는 key값으로 60을 가질 수 없기 때문에, time[i]==60일 경우 %60을 한번 더해줌으로써 table[0]을 검색하게 해준다.

int numPairsDivisibleBy60(vector<int>& time) {

unordered\_map<int,int> table;

int answer=0;

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

answer+=table[(60-(time[i]%60))%60]; // to handle when a time[i]==60, since there will be no value 60 in the table.

table[time[i]%60]++;

}

return answer;

}

[37. **Sum of Nodes with Even-Valued Grandparent – 현재 노드의 grandparent가 짝수인 모든 노드의 합을 구하는 문제] 2020-01-29 6:37**

**-** helper함수 하나와 재귀를 사용하여 풀었다.

int sum=0;

int search(TreeNode\* root){

int temp=0;

if(!root) return 0;

if(root->left)

temp+=root->left->val;

if(root->right)

temp+=root->right->val;

return temp;

}

int sumEvenGrandparent(TreeNode\* root) {

if(!root) return 0;

if(root->val%2==0){

sum+=search(root->left)+search(root->right);

}

sumEvenGrandparent(root->left);

sumEvenGrandparent(root->right);

return sum;

}

[38. **Deepest Leaves Sum – 가장 깊은 노드들의 value의 합을 구하는 문제] 2020-01-29 6:47**

- 첫번째로 max\_depth를 구하기 위해 트리를 한번 돌고, 두번째로 max\_depth인 노드들의 합을 구하기 위해 한번 돌고, 트리를 총 2번 도는 알고리즘임.

int max\_depth=INT\_MIN;

void findMaxDepth(TreeNode\* root,int depth){

if(!root) return;

max\_depth=max(max\_depth,depth);

findMaxDepth(root->left,depth+1);

findMaxDepth(root->right,depth+1);

}

int sumOfDeepest(TreeNode\* root,int depth){

if(!root) return 0;

if(depth==max\_depth)

return root->val;

return sumOfDeepest(root->left,depth+1)+sumOfDeepest(root->right,depth+1);

}

int deepestLeavesSum(TreeNode\* root) {

findMaxDepth(root,0);

return sumOfDeepest(root,0);

}

[39. **Group the People Given the Group Size They Belong To – 주어진 groupsize의 각 인덱스의 위치한 숫자가, 해당 인덱스가 포함될 그룹의 사이즈임]**

**-** 문제를 이해하는 것 자체가 힘듬. 일단 0번째 인덱스의 원소가 3이면, 0번 ID를 가진 사람의 그룹 사이즈가 3이어서, 사이즈가 3인 그룹에 해당 ID를 할당해야하는 것임.

- map에 벡터를 value로 사용하는 방법을 배웠음. 해당 key가 empty일때는 table[key]=vector<int>{value} 로 맨처음에 값을 넣어주고, 만약에 값이 존재하면 즉, table[key].count()>0 이 true이면 그냥 바로 table[key].push\_back(value); 해주면 됨.  
vector<vector<int>> groupThePeople(vector<int>& groupSizes) {

unordered\_map<int,vector<int>> table;

vector<vector<int>> answer;

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

int key=groupSizes[i];

if(table.count(key)>0)

table[key].push\_back(i);

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

if(table.find(key)->second.size()>=key){

answer.push\_back(table[key]);

table.erase(key);

}

}

return answer;

}

[40. **Max Increase to Keep City Skyline – 각 row, col에 대하여 해당 원소가 증가할 수 있는 최대치의 합을 구하는 문제]**

- 각 row와 column의 최대 값을 구한 벡터를 2개를 가진 후, 각 원소가 증가할 수 있는 최대치를 구하여 더해줌.

int maxIncreaseKeepingSkyline(vector<vector<int>>& grid) {

vector<int> row\_height;

vector<int> col\_height;

int temp\_max=INT\_MIN;

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

for(int j=0;j<grid[i].size();j++)

temp\_max=max(temp\_max,grid[i][j]);

row\_height.push\_back(temp\_max);

temp\_max=INT\_MIN;

}

for(int i=0;i<grid[0].size();i++){

for(int j=0;j<grid.size();j++)

temp\_max=max(temp\_max,grid[j][i]);

col\_height.push\_back(temp\_max);

temp\_max=INT\_MIN;

}

int sum=0;

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

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

sum+=min(row\_height[i],col\_height[j])-grid[i][j];

}

}

return sum;

}

[41. **Binary Search Tree to Greater Sum Tree – 이진탐색트리에서 자신보다 큰 노드의 값을 모두 합친 값이 자신의 값이 되도록 수정하는 문제]**

- 전역 변수를 사용하는게 포인트임. 사용안하고 풀어보려다가 시간 개오래 걸림.

int current\_sum=0;

void modify(TreeNode\* root){

if(!root) return;

modify(root->right);

root->val+=current\_sum;

current\_sum=root->val;

modify(root->left);

return;

}

TreeNode\* bstToGst(TreeNode\* root) {

modify(root);

return root;

}

[42. **Encode and Decode TinyURL – 내 마음대로 URL을 encode한 후 decode하는 문제]**

**-** 그냥 간단하게 각 문자에 3을 더한 ascii 코드값을 string에 저장하고, decode할 때 3을 빼줌.

string enc;

string dec;

// Encodes a URL to a shortened URL.

string encode(string longUrl) {

enc.clear();

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

enc+=longUrl[i]+3;

enc+='\0';

return enc;

}

// Decodes a shortened URL to its original URL.

string decode(string shortUrl) {

dec.clear();

for(int i=0;shortUrl[i]!='\0';i++)

dec+=shortUrl[i]-3;

return dec;

}

[43. **Insert into a Binary Search Tree – BST에 삽입하는 문제]**

- 자신 보다 큰 노드를 발견할때까지 오른쪽으로 가고, 자신 보다 큰 노드를 만났을 때, 왼쪽 자식이 비어있으면 거기에 넣고, 아니면 왼쪽으로 한칸가서 다시 자신보다 큰 노드를 발견할 때 까지 반복.

void search(TreeNode\* root, int val){

if(root->val<val){

if(!root->right){

root->right=new TreeNode(val);

return;

}

search(root->right,val);

}

else if(!root->left){

root->left= new TreeNode(val);

return;

}

else search(root->left,val);

}

TreeNode\* insertIntoBST(TreeNode\* root, int val) {

search(root,val);

return root;

}

- 근데 이문제 iterative로 풀면 O(1) space만 쓰면 된다. recursive로 풀면 stack에 함수가 쌓여서 O(n) space가 필요하다.

[44. **Maximum Binary Tree – 주어진 벡터에 대해서 맥스 값을 root로 왼쪽 오른쪽을 나누고, 같은 방식으로 트리를 만드는 문제] 2020-01-30 4:20**

**-** 처음에는 iterator로 풀려고 했는데, 자꾸 bad\_allocation 이라는 runtime error가 발생해서, 그냥 인덱스 가지고 for문 돌리는 것으로 바꿈

- 내생각에 이거는 O(n^2)에 푸는 건데, Discussion을 확인해 봐야겠음.

TreeNode\* constructTree(vector<int>& nums){

int maximum=INT\_MIN;

int max\_index=0;

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

if(maximum<nums[i]){

maximum=nums[i];

max\_index=i;

}

vector<int> left;

vector<int> right;

for(int i=0;i<max\_index;i++)

left.push\_back(nums[i]);

for(int i=max\_index+1;i<nums.size();i++){

right.push\_back(nums[i]);

}

TreeNode\* root=new TreeNode(maximum);

if(max\_index!=0) root->left=constructTree(left);

if(max\_index!=nums.size()-1) root->right=constructTree(right);

return root;

}

TreeNode\* constructMaximumBinaryTree(vector<int>& nums) {

return constructTree(nums);

}

- 다음 방법은 stack을 이용한 건데, 이해는 했는데, 이게 O(n)인지는 좀 확실치 않음, 스택에 다 넣었다가 빼는 과정도 있어서.. 그래도 O(2n)정도인듯.

TreeNode\* constructMaximumBinaryTree(vector<int>& nums) {

vector<TreeNode\*> stk;

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

{

TreeNode\* cur = new TreeNode(nums[i]);

while (!stk.empty() && stk.back()->val < nums[i])

{

cur->left = stk.back();

stk.pop\_back();

}

if (!stk.empty())

stk.back()->right = cur;

stk.push\_back(cur);

}

return stk.front();

}

[45. **Sort the Matrix Diagonally – 주어진 2차원 벡터에 대해서 모든 대각선을 오름차순으로 sort하는 문제] 2020-01-30 4:40**

**-** 쉽게 생각해서 O(n^2)에 풀었는데, 각 원소의 오른쪽 아래원소가 자신보다 작으면 두개를 스왑하는 방식으로, 그것을 주어진 2차원 벡터의 row수만큼 반복함.

vector<vector<int>> diagonalSort(vector<vector<int>>& mat) {

for(int k=0;k<mat.size();k++)

for(int i=0;i<mat.size()-1;i++)

for(int j=0;j<mat[i].size()-1;j++)

if(mat[i][j]>mat[i+1][j+1])

swap(mat[i][j],mat[i+1][j+1]);

return mat;

}

[46. **All Elements in Two Binary Search Trees – 주어진 두개의 BST의 원소들을 오름차순으로 벡터에 집어넣는 문제] 2020-01-30 5:03**

- BST이므로 중위순회를 하면 sort가 됨. 이것을 각각 벡터로 하나씩 가지고 있고, 두개의 벡터의 원소들을 비교해가면서 answer 벡터에 넣어줌.

-마지막에 빈 벡터와 위의 sort 벡터 2개를 swap함으로써 메모리 해제를 해줌.

- 이때 벡터 2개를 사용안하고 벡터 1개에 집어넣고 sort하는 방식도 있긴함. 근데 이러면 O(nlogn)이 걸림. 내 방법은 해봐야 O(4n)임.

void inOrder(TreeNode\* root,vector<int>& bst){

if(!root) return;

inOrder(root->left,bst);

bst.push\_back(root->val);

inOrder(root->right,bst);

}

vector<int> getAllElements(TreeNode\* root1, TreeNode\* root2) {

vector<int> bst1;

vector<int> bst2;

vector<int> answer;

inOrder(root1,bst1);

inOrder(root2,bst2);

vector<int>::iterator left=bst1.begin();

vector<int>::iterator right=bst2.begin();

while(left!=bst1.end()|right!=bst2.end()){

if(left==bst1.end()){

while(right!=bst2.end()){

answer.push\_back(\*right);

right++;

}

break;

}

if(right==bst2.end()){

while(left!=bst1.end()){

answer.push\_back(\*left);

left++;

}

break;

}

if(\*left>\*right){

answer.push\_back(\*right);

right++;

}

else{

answer.push\_back(\*left);

left++;

}

}

vector<int>().swap(bst1);

vector<int>().swap(bst2);

return answer;

}

[47. **Construct Binary Search Tree from Preorder Traversal – preorder로 주어진 벡터로 트리를 만드는 문제] 2020-01-30 5:34**

**-** 아 이거 학교에서 했던건데 알고리즘이 떠오르지 않아서 되게 과격한 방식으로 풀었음. Discussion을 좀 확인해봐야겠음.

- 일단 나의 알고리즘은, 벡터의 첫번째 원소가 항상 root이고 이제 벡터를 왼쪽 서브트리와 오른쪽 서브트리로 나눠야 하는데, 첫번째 원소보다 큰 값을 가진 첫번째 원소가 오른쪽 서브트리의 루트고, 왼쪽 서브트리의 루트는 두번째 원소임.

- 위와 같은 방식으로 벡터 2개를 구해서 각각 재귀해줌. 그런데 이거 iterator로 풀수 있을 것 같은 데 왜 안풀리냐 ㅡㅡ, 그래서 그냥 index로 품. 속도 보장 못함.

TreeNode\* helper(vector<int> preorder){

if(preorder.size()==0) return nullptr;

TreeNode\* root=new TreeNode(preorder[0]);

int next\_index=preorder.size();

for(int i=1;i<preorder.size();i++)

if(preorder[i]>preorder[0]){

next\_index=i;

break;

}

vector<int> left;

vector<int> right;

for(int i=1;i<next\_index;i++)

left.push\_back(preorder[i]);

for(int i=next\_index;i<preorder.size();i++)

right.push\_back(preorder[i]);

if(right.size()!=0)

root->right=helper(right);

if(left.size()!=0)

root->left=helper(left);

return root;

}

TreeNode\* bstFromPreorder(vector<int>& preorder) {

return helper(preorder);

}

- 이런식으로 하면 iterator 쓸수 있을 듯.

TreeNode\* bstFromPreorder(vector<int>& preorder) {

return helper(preorder.begin(), preorder.end());

}

TreeNode \* helper(vector<int>::iterator begin, vector<int>::iterator end) {

if (begin == end) {

return nullptr;

}

auto node = new TreeNode(\*begin);

auto right = upper\_bound(begin + 1, end, \*begin);

node->left = helper(begin + 1, right);

node->right = helper(right, end);

return node;

}

[48. **Letter Tile Possibilities – 주어진 string으로 만들 수 있는 모든 sequence의 개수를 출력하는 문제]**

**-** 이 문제 어렵다 ㄷㄷ.

- Discussion 보고 배껴서 풀긴 했는데, 아직도 이해가 잘 안감.

unordered\_set<string> answer;

unordered\_map<char,int> table;

void helper(string current,int length){

if(current!="")

answer.insert(current);

if(current.size()>=length)

return;

for(unordered\_map<char,int>::iterator iter=table.begin();iter!=table.end();iter++){

if(iter->second>0){

iter->second--;

current+=iter->first;

helper(current,length);

current.pop\_back();

iter->second++;

}

}

}

int numTilePossibilities(string tiles) {

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

table[tiles[i]]++;

helper("",tiles.size());

return answer.size();

}

[49. **Delete Leaves With a Given Value – 주어진 target value와 동일한 값을 가지고 있는 leaf node를 삭제하는 문제] 2020-01-31 8:50**

- 리프노드를 삭제했을 때, 그 부모가 리프노드가 되고 또 리프노드가 된 그 노드가 target value를 가지고 있으면 또 지워줘야함.

- 또한 트리 전체의 root 까지 지워야 하는 경우를 대비하여, root를 가리키는 임의의 노드를 생성하여, 그것을 가상의 루트로 함. 여기서는 answer node임.

bool end=false;

void remove(TreeNode\* root, int target){

if(!root) return;

if(root->left){

if(root->left->val==target && !root->left->left && !root->left->right){

root->left=nullptr;

end=false;

}

}

if(root->right){

if(root->right->val==target && !root->right->left && !root->right->right){

root->right=nullptr;

end=false;

}

}

remove(root->left,target);

remove(root->right,target);

}

TreeNode\* removeLeafNodes(TreeNode\* root, int target) {

TreeNode\* answer=new TreeNode(target-1);

answer->left=root;

while(!end){

end=true;

remove(answer,target);

}

return answer->left;

}

[50. **Partition Labels – 주어진 string을 분할하는 문제인데, 각 문자는 하나의 partition에만 속할 수 있음. 이렇게 나눌 수 있는 최대의 파티션을 구하고, 각 파티션의 길이를 리턴하는 문제] 2020-01-31 9:20**

- 나는 brute force로 푼거 같음. 현재 고려하는 문자를 cur라고 하고 그 위치를 left로 함. 그 후 가장 오른쪽에 위치한 cur를 찾고 그위치를 right로 하고, left와 right 사이의 문자 중 right 보다 더 오른쪽에 존재하는 문자를 찾아봄. 있으면 그 문자가 right가 됨. 그후 right-left+1이 해당 partition의 길이임.

-위 과정을 주어진 string이 끝날때까지 하면됨.

- O(n^2) solution 인듯.

int rightEnd(string& S,char cur,int left){

int right=0;

for(int i=left;i<S.size();i++)

if(S[i]==cur) right=i;

return right;

}

vector<int> partitionLabels(string S) {

vector<int> answer;

char cur=S[0];

int left=0,right=0;

int i=0;

while(i<S.size()){

cur=S[i];

left=i;

right=rightEnd(S,cur,left);

for(int j=left+1;j<right;j++){

int temp\_right=rightEnd(S,S[j],j);

if(right<temp\_right){

right=temp\_right;

}

}

answer.push\_back(right-left+1);

i=right+1;

}

return answer;

}

-생각해보니 다르게 풀 수 있을 것 같음. map 써가지고 O(n)시간안에 푸는건데, O(n)이긴 한데 좀 오래걸리는 O(n)임.

vector<int> partitionLabels(string S) {

unordered\_map<char,vector<int>> table;

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

if(table.count(S[i])>0) table[S[i]][1]=i;

else table[S[i]]=vector<int>{i,i};

}

int i=0;

int left,right;

char cur;

vector<int> answer;

while(i<S.size()){

cur=S[i];

left=table.find(S[i])->second[0];

right=table.find(S[i])->second[1];

for(int j=left+1;j<right;j++){

if(table.find(S[j])->second[1]>right){

right=table.find(S[j])->second[1];

}

}

answer.push\_back(right-left+1);

i=right+1;

}

return answer;

}

- void **swap**(a, b) : a와 b를 교환

- T **exchange**(T t, U u) : t에 u를 할당, 원래 t를 반환

- 이거는 O(n) solution 임. Discussion에서 봄. 확실히 간단함.

vector<int> partitionLabels(string S) {

vector<int> res, pos(26, 0);

for (auto i = 0; i < S.size(); ++i) pos[S[i] - 'a'] = i;

for (auto i = 0, idx = INT\_MIN, last\_i = 0; i < S.size(); ++i) {

idx = max(idx, pos[S[i] - 'a']);

if (idx == i) res.push\_back(i - exchange(last\_i, i + 1) + 1);

}

return res;

}

[51. **Reveal Cards In Increasing Order – 주어진 단계에 따랐을 때, 오름차순으로 카드가 정렬될 수 있도록 하는 문제. 설명을 봐야됨] 2020-01-31 10:09**

In a deck of cards, every card has a unique integer.  You can order the deck in any order you want.

Initially, all the cards start face down (unrevealed) in one deck.

Now, you do the following steps repeatedly, until all cards are revealed:

1. Take the top card of the deck, reveal it, and take it out of the deck.
2. If there are still cards in the deck, put the next top card of the deck at the bottom of the deck.
3. If there are still unrevealed cards, go back to step 1.  Otherwise, stop.

Return an ordering of the deck that would reveal the cards in **increasing order.**

The first entry in the answer is considered to be the top of the deck.

- 이 문제 처음에는 가늠도 안갔음. 어떻게 풀어야 하는지.

- Discussion을 참고했더니, 하는 단계를 역으로 하는 해법이 있었음. 세상의 천재는 많음. ㄹㅇ…

vector<int> deckRevealedIncreasing(vector<int>& deck) {

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

vector<int> answer;

while(deck.size()){

// second step

if(answer.size()){

answer.insert(answer.begin(),answer.back());

answer.pop\_back();

}

//first step

answer.insert(answer.begin(),deck.back());

deck.pop\_back();

}

return answer;

}

[52. **All Paths From Source to Target – 그래프의 모든 paths를 반환하는 문제] 2020-02-04 11:43**

**-** “from node 0 to node N-1”이 말이 키포인트 임. 문제를 잘 읽어야 함.

- dfs로 푸는건데 생각해내기 쉽지 않음.

- 와 코딩 열심히 해야겠따 ㄹㅇ...

void dfs(vector<vector<int>>& graph,vector<vector<int>>& answer,vector<int> table,int cur){

table.push\_back(cur);

if(cur==graph.size()-1){

answer.push\_back(table);

return;

}

for(int i=0;i<graph[cur].size();i++)

dfs(graph,answer,table,graph[cur][i]);

}

vector<vector<int>> allPathsSourceTarget(vector<vector<int>>& graph) {

vector<vector<int>> answer;

vector<int> table;

dfs(graph,answer,table,0);

return answer;

}

[53. **Binary Tree Pruning – 1을 포함하지 않는 모든 subtree를 제거하는 문제] 2020-02-04 11:51**

**-** 이 문제는 리프노드이면서 value가 0인 노드를 리프에서부터 제거해서 루트로 올라오면 된다.

- 조건에 부합하여 제거된 노드의 부모가 다시 조건에 부합하게 되면, 그역시 삭제해야 한다.

void helper(TreeNode\* root){

if(!root) return;

helper(root->left);

helper(root->right);

if(root->left && !root->left->left && !root->left->right && root->left->val==0)

root->left=nullptr;

if(root->right&& !root->right->left && !root->right->right && root->right->val==0)

root->right=nullptr;

}

TreeNode\* pruneTree(TreeNode\* root) {

helper(root);

return root;

}

[54. **All Possible Full Binary Trees – 어려워서 못풀었음.]**

**-**  DFS 랑 BFS로 풀라그랬는데 실패함. discussion봤는데 divide and conquer 길래 읽어봐도 뭔소린지 모르겠음. 그래서 youtube를 참고하려고 함.

- youtube에도 중국인 설명밖에 없음. 모르겠다 이문제 ㄷㄷ;

- 이해했다. 1개로 만들수 있는 트리, 3개로 만들 수 있는 트리를 계속저장해가면서 불러오는거 같다.

- 이게 정답인데, 이해할려면 시간 좀 걸릴 듯.

unordered\_map<int, vector<TreeNode\*>> cache;

vector<TreeNode\*> allPossibleFBT(int N) {

vector<TreeNode\*> res;

if(cache[N].size() != 0) return cache[N];

if(N == 1) {

res.push\_back(new TreeNode(0));

} else {

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

int l = i, r = N - i - 1;

for (TreeNode\* left : allPossibleFBT(l)) {

for (TreeNode\* right : allPossibleFBT(r)) {

TreeNode \* root = new TreeNode(0);

root->left = left;

root->right = right;

res.push\_back(root);

}

}

}

}

cache[N] = res;

return res;

}

[55. **Find and Replace Pattern – 주어진 pattern과 같은 순열을 가지는 word들을 반환하는 문제]**

- 나는 unordered\_map을 이용해서, 주어진 string의 pattern을 숫자로 나타내었음. 그리고 word들을 숫자로 나타낸 것과 비교함.

vector<string> findAndReplacePattern(vector<string>& words, string pattern) {

vector<string> answer;

unordered\_map<char,int> p\_table;

int count=1;

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

if(!p\_table[pattern[i]]) p\_table[pattern[i]]=count++;

}

string answer\_pattern;

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

answer\_pattern+=p\_table[pattern[i]]+48;

}

unordered\_map<char,int> match;

count=1;

string temp;

for(string word:words){

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

if(!match[word[i]]) match[word[i]]=count++;

}

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

temp+=match[word[i]]+48;

}

if(temp.compare(answer\_pattern)==0)

answer.push\_back(word);

count=1;

temp="";

match.clear();

}

return answer;

}

[56. **Find Elements in a Contaminated Binary Tree – 오염된 트리를 재구성하고 target value를 찾는 문제]**

1. root.val == 0
2. If treeNode.val == x and treeNode.left != null, then treeNode.left.val == 2 \* x + 1
3. If treeNode.val == x and treeNode.right != null, then treeNode.right.val == 2 \* x + 2

- 위 공식대로 트리를 재구성한 후 target을 찾으면 된다.

- set을 이용하여 모든 노드 값을 저장하는 것이 target을 더 빨리 찾는 방법이다. 왜냐하면 순회를 하지 않아도 되기 때문이다.

TreeNode\* answer;

unordered\_set<int> value\_set;

void recover(TreeNode\* root){

if(!root) return;

if(root->left)

root->left->val=2\*root->val+1;

if(root->right)

root->right->val=2\*root->val+2;

value\_set.insert(root->val);

recover(root->left);

recover(root->right);

}

FindElements(TreeNode\* root) {

root->val=0;

answer=root;

recover(root);

}

bool helper(TreeNode\* root,bool flag,int target){

if(!root) return flag;

if(root->val==target) return true;

flag=helper(root->left,flag,target);

flag=helper(root->right,flag,target);

return flag;

}

bool find(int target) {

//return helper(answer,false,target);

return value\_set.count(target);

}

[57. **Matrix Block Sum – 주어진 조건에 맞는 모든 원소를 더해 벡터로 만드는 문제]**

**-** 일단 문제 이해하는데 좀 걸림.

- for문을 4개나 썼는데, 더 빨리 풀 수 있는방법이 있을 것 같음. 전에 합해놓은 값을 그대로 이용한다던가 하는…

vector<vector<int>> matrixBlockSum(vector<vector<int>>& mat, int k) {

vector<vector<int>> res;

vector<int> temp\_answer;

int temp\_sum=0;

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

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

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

if(r<0 || r>=mat.size())

continue;

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

if(c<0||c>=mat[i].size())

continue;

temp\_sum+=mat[r][c];

}

}

temp\_answer.push\_back(temp\_sum);

temp\_sum=0;

}

res.push\_back(temp\_answer);

temp\_answer.clear();

}

return res;

}

[58. **Minimum Add to Make Parentheses Valid – 주어진 string에서 괄호가 valid하게 되기위해 필요한 괄호의 개수를 반환하는 문제]**

**-** 그냥 각 케이스의 경우를 생각해서 풀었음.

int minAddToMakeValid(string S) {

int res=0;

int left=0,right=0;

bool left\_first=false;

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

if(S[i]=='('){

left++;

res++;

continue;

}

if(S[i]==')'){

if(left){

left--;

res--;

continue;

}

res++;

continue;

}

}

return res;

}

[59. **Score After Flipping Matrix – 주어진 matrix를 자유롭게 toggle 즉, 0과 1을 뒤바꾸어 이진수로 취급할 때의 최대 합을 구하는 문제]**

**-** 일단 맨 첫 비트가 1이되도록 각 row를 toggle해주고, 이후 모든 columns을 탐색하면서 0의 개수가 1의 개수보다 많으면 toggle한다.

int sumUp(vector<vector<int>> temp){

int sum=0;

int power=0;

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

for(int j=temp[i].size()-1;j>=0;j--,power++){

if(temp[i][j])

sum+=pow(2,power);

}

power=0;

}

return sum;

}

int matrixScore(vector<vector<int>>& A) {

//row toggle

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

if(A[i][0]==0){// if leading bit is 1, then toggle it.

for(int j=0;j<A[i].size();j++)

A[i][j]^=1;

}

}

//col toggle when 0s are more than 1s in the current column.

int zeros=0;

int ones=0;

for(int i=0;i<A[0].size();i++){

for(int j=0;j<A.size();j++){

if(A[j][i]==0) zeros++;

else ones++;

}

if(zeros>ones){ // toggle the column.

for(int j=0;j<A.size();j++)

A[j][i]^=1;

}

ones=0;

zeros=0;

}

return sumUp(A);

}

[60. **Maximum Level Sum of a Binary Tree – 노드들의 총합이 가장 큰 minimum level을 찾는 문제]**

- 나는 pair와 map을 이용하여 bfs로 풀었다. pair 접근할 때, -> 로 접근하는 것이 아니라 ‘.’ 마침표로 접근한다. root.first 이렇게

- 생각해보니 dfs로 풀면 훨씬 빠를 것 같다. que도 pair도 필요없이 재귀로만 풀 수 있을 듯.

void bfs(queue<pair<TreeNode\*,int>>& que,unordered\_map<int,int>& level\_sum){

if(que.empty()) return;

pair<TreeNode\*,int> root=que.front();

que.pop();

level\_sum[root.second]+=root.first->val;

if(root.first->left) que.push(make\_pair(root.first->left,root.second+1));

if(root.first->right) que.push(make\_pair(root.first->right,root.second+1));

bfs(que,level\_sum);

}

int maxLevelSum(TreeNode\* root) {

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

int sum=root->val;

que.push(make\_pair(root,1));

unordered\_map<int,int> level\_sum;

bfs(que,level\_sum);

int max\_level=1;

for(unordered\_map<int,int>::iterator iter=level\_sum.begin();iter!=level\_sum.end();iter++){

if(iter->second>sum){

sum=iter->second;

max\_level=iter->first;

}

}

return max\_level;

}

[61. **Path In Zigzag Labelled Binary Tree – 레벨마다 내림차순, 오름차순이 바뀌는 포화 이진트리에서 주어진 label까지의 경로를 출력하는 문제]**

- 주어진 트리에 숫자가 순서대로는 들어가 있으므로, 부모노드는 자신의 value/2라는 것을 이용함.

-이때 홀수 레벨은 오름차순이고, 짝수 레벨은 내림차순이므로 계산을 따로 해줌.

vector<int> pathInZigZagTree(int label) {

//leftmost node of level n is 2^(n-1) when root's level is 1.

//rightmost node of level n is 2^n -1.

//find rightmost node greater than label. Then we can find the level of label.

//if n is odd, then the nodes of the level have normal sequence, otherwise reverse sequence.

//In normal case, parents node's value is label / 2, we can use it.

vector<int> answer;

answer.push\_back(label);

int level=1;

while(pow(2,level)-1<label) level++; //find the label's level.

int temp=label;

int position=0;

while(level>1){

if(level%2==0) position=(pow(2,level)-1-temp)/2; //find next postion.

else position=(temp-pow(2,level-1))/2;

level--; // we find the postion of the upper level. so we don't need current level anymore.

if(level%2==0) temp=pow(2,level)-1-position;

else temp=pow(2,level-1)+position;

answer.push\_back(temp);

}

return {answer.rbegin(),answer.rend()};

}

[62. **Maximum Nesting Depth of Two Valid Parentheses Strings – 최소 깊이를 가지도록 괄호들을 두개의 그룹으로 나누는 문제]**

- 문제 설명부터 이해하기 힘듬. 주어진 스트링에서 나열된 순서와 상관 없이, 왼쪽 오른쪽 짝만 맞으면 한 그룹에 들어갈 수 있음. 이렇게 짝을 맞추면서 두개의 그룹으로 나눌 때, 최소의 깊이를 가지게 하면 됨.

- discussion을 보고서 문제를 이해했고, 답도 거기서 힌트를 얻었음.

- 일단 각 괄호의 깊이를 모두 구함. 그리고 홀수는 A에 짝수는 B에 넣음. 즉, 깊이가 하나 깊어질 때마다 다른 그룹에 넣어주면 깊이가 최소가 됨.

- 이런 문제는 누가 생각해내는 거야…

vector<int> maxDepthAfterSplit(string seq) {

vector<int> depth;

int cur\_depth=0;

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

if(seq[i]=='(') depth.push\_back(++cur\_depth);

else depth.push\_back(cur\_depth--);

vector<int> answer;

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

depth[i]%2? answer.push\_back(0):answer.push\_back(1);

return answer;

}

- for문을 두개 안쓰고, 하나만 쓰면서 깊이를 구하자마자 바로 answer에 넣을 수도 있음.

vector<int> maxDepthAfterSplit(string seq) {

vector<int> depth;

vector<int> answer;

int cur\_depth=0;

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

if(seq[i]=='(') depth.push\_back(++cur\_depth);

else depth.push\_back(cur\_depth--);

depth[i]%2? answer.push\_back(0):answer.push\_back(1);

}

return answer;

}

[63. **Distribute Coins in Binary Tree – 모든 노드가 1개의 coin을 갖도록 분배할 때, 코인이 움직이는 개수를 구하는 문제]**

**-** 한번에 한칸 움직일 수 있고, 그때 움직인 coin의 개수를 총합한 것이 답이다.

- bottom up으로 풀려고 했고, 실제로 discussion의 방법들과 동일했는데, 나는 중간에 막혔다.

- discussion을 참고하여 풀었다. abs로 move를 더해주는 것이 key였다.

int getCoin(TreeNode\* root,int& move){

if(!root) return 0;

root->val+=getCoin(root->left,move);

root->val+=getCoin(root->right,move);

move+=abs(root->val-1);

return root->val-1;

}

int distributeCoins(TreeNode\* root) {

int move=0;

getCoin(root,move);

return move;

}

[64. **Queens That Can Attack the King – king과 같은 row,column,diagnal 에 있는 가장 가까운 queen의 좌표를 찾는 문제]**

- 처음에는 BFS를 사용할까 생각했는데, 그냥 8x8 배열 만들어 퀸 좌표를 모두 넣은 다음, 킹에서부터 가장 가까운 것 찾는 for문8번돌리는게 시간상 빠를 것 같아서 그렇게 함.

vector<vector<int>> queensAttacktheKing(vector<vector<int>>& queens, vector<int>& king) {

vector<vector<int>> answer;

vector<vector<bool>> chess;

chess.assign(8,vector<bool>(8)); //2차원 벡터 메모리 초기화

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

chess[queens[i][0]][queens[i][1]]=true;

int x=king[0];

int y=king[1];

//left

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

if(chess[i][y]){

answer.push\_back(vector<int>{i,y});

break;

}

}

//right

for(int i=x+1;i<chess.size();i++){

if(chess[i][y]){

answer.push\_back(vector<int>{i,y});

break;

}

}

//up

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

if(chess[x][i]){

answer.push\_back(vector<int>{x,i});

break;

}

}

//down

for(int i=y+1;i<chess.size();i++){

if(chess[x][i]){

answer.push\_back(vector<int>{x,i});

break;

}

}

//up-left

for(int i=x-1,j=y-1;i>=0 && j>=0;i--,j--){

if(chess[i][j]){

answer.push\_back(vector<int>{i,j});

break;

}

}

//up-right

for(int i=x+1,j=y-1;i<chess.size() && j>=0;i++,j--){

if(chess[i][j]){

answer.push\_back(vector<int>{i,j});

break;

}

}

//down-left

for(int i=x-1,j=y+1;i>=0&&j<chess.size();i--,j++){

if(chess[i][j]){

answer.push\_back(vector<int>{i,j});

break;

}

}

//down-right

for(int i=x+1,j=y+1;i<chess.size()&&j<chess.size();i++,j++){

if(chess[i][j]){

answer.push\_back(vector<int>{i,j});

break;

}

}

return answer;

}

[65. **Battleships in a Board – 인접하지 않은 the number of rows of Xs or cols of Xs 를 찾는 문제]**

- 푸는 건 어렵지 않았음. 그냥 top-left 부터 시작해서 X만날 때 연이은 X를 모두 ‘.’으로 바꾸어주면 됨.

int countBattleships(vector<vector<char>>& board) {

int res=0;

int temp;

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

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

if(board[i][j]=='X'){

res++;

board[i][j]='.';

temp=j+1;

while(temp<board[i].size()&&board[i][temp]=='X') board[i][temp++]='.';

temp=i+1;

while(temp<board.size()&&board[temp][j]=='X') board[temp++][j]='.';

}

}

}

return res;

}

- follow up에 O(1) memory without modifying the value of board가 있어서 discussion을 들어가 봄. head of ships 를 찾는 문제로 변형 됨.

int countBattleships(vector<vector<char>>& board) {

if(board.empty() || board.front().empty()) return 0;

int res=0;

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

for(int j=0;j<board[i].size();j++) //just find ship's head.

res+= board[i][j]=='X' && (i==0|| board[i-1][j]!='X') && (j==0||board[i][j-1]!='X');

return res;

}

[66. **Count Square Submatrices with All Ones – 사각형이 몇 개 존재하는 지 구하는 문제]**

- 처음에는 top-left를 기준으로 만들어지는 square의 개수를 만드려고 다음과 같은 코드를 짰음.

- 그런데 time limit exceeded 함.

- 짱구를 더 굴려 보자구.

int isSquare(int x, int y, int size,vector<vector<int>> matrix){

int res=0; //how many square current coordinates can make.

int temp=1; // current size of square +1

bool flag=true; // whether current considering square is valid square or not.

while(temp<size){

cout<<"x : "<<x<<" y : "<<y<<" temp : "<<temp<<endl;

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

if(x+temp>=matrix.size()||j>=matrix[0].size()||!matrix[x+temp][j]){

flag=false;

break;

}

}

cout<<"pass1"<<endl;

// thr bottom-right value doesn't need to be considered twice. so i<x+temp

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

if(y+temp>=matrix[0].size()||i>matrix.size()||!matrix[i][y+temp]){

flag=false;

break;

}

}

if(!flag) break;

cout<<"pass2"<<endl;

res++;

temp++;

}

return res;

}

int countSquares(vector<vector<int>>& matrix) {

int res=0;

int minSize=min(matrix.size(),matrix[0].size());

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

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

if(matrix[i][j]){

res+=isSquare(i,j,minSize,matrix) + 1; // the rightmost 1 is for square of side 1

}

}

}

return res;

}

- 특정되는 사각형 수가 크기가 N X N 일때, 총 사각형의 개수는1^2 + 2^2 + … + N^2 임.

을 참고하면 될 듯.

- 위에 것도 접근이 너무 어려움. DP로 푸는 거였음. 하 나는 DP가 너무 약해…

- 근데 이 DP는 진짜 생각해내기 너무 어려운 것 같은데?

- 내 생각에는 이건 그냥 암기다.

int countSquares(vector<vector<int>>& matrix) {

int res=0;

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

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

if(matrix[i][j] && i && j) matrix[i][j]+=min(matrix[i][j-1],min(matrix[i-1][j-1],matrix[i-1][j]));

res+=matrix[i][j];

}

}

return res;

}

1. if matrix[i][j] == 0, skip
2. use bottom right to count the square, if a 2 \* 2 grid like  
   " A B "  
   " C X "  
   every character means the nums of the square that position can represent. For example A = 2 represents 2 squares in the "A" position, one is itself and the other is a 2 \* 2 square , a 2 \* 2 square means A is in the position like  
   " 1 1 "  
   " 1 A "  
   so X should be Math.min(A+1, B+1, C+1) which ensures X is in the postion like  
   " 1 ... 1 " //k \* k, k = Math.min(A+1, B+1, C+1)  
   " 1 .... . "  
   " 1 ... X "  
   for example  
     
   in the yellow grid, we know (D4=3) (E4=2) (D5=2) and then how to calculate the (E5).  
   D4 = 3 means the largest sqaure it can represent is the black one(3\* 3) === we can find 3 sqaures which its bottom right is D4 (1\* 1, 2\* 2, 3\* 3 squares).  
   Then the same as E4(red square) and D5(green one). When it comse to E5, (E5) = Math.min(D4+1, E4+1, D5+1) means the largest square E5 can represent which is (C3 ~ E5)
3. because we are using the bottom right to count the square, the "A B C" will always be calculated before "X". So just go through the grid then and add all nums in the grid

[67. **Spiral Matrix III – 나선모양으로 걸으면서 거친 좌표를 차례대로 벡터에 넣는 문제]**

- 이 문제 왜캐 어려웠지… Discussion을 안보고 풀라고 애썼는 데, 실패했다.

- 이 문제의 포인트는 boundary 바깥으로 나가는 것을 고려하지 않아도 된다는 것이었다. 나선형만 제대로 짜 놓으면 어차피 다시 안으로 들어올 것이고, 주어진 boundary안에 좌표가 있을 때만 answer에 더해주면 되는 것이었다.

vector<vector<int>> spiralMatrixIII(int R, int C, int r0, int c0){

vector<vector<int>> res={{r0,c0}};

int max\_step=R\*C;

vector<int> cur\_pos={r0,c0};

vector<vector<int>> dir={{0,1},{1,0},{0,-1},{-1,0}};

int head=0;

int cur\_step=1;

int how\_many\_step=0;

while(cur\_step<max\_step){

if(head==0 || head==2) how\_many\_step++; // when east or west, the step is added one.

for(int i=0;i<how\_many\_step;i++){ //how many step we have to go before turning right.

//by using the limit R and C, we don't need to consider a walk around outside.

//because it will be returned into the boundary. we just need to check the walk occurs in the boundary or not.

cur\_pos[0]+=dir[head][0];

cur\_pos[1]+=dir[head][1];

if(0<=cur\_pos[0] && cur\_pos[0]<R && 0<=cur\_pos[1] && cur\_pos[1]<C){

cur\_step++;

res.push\_back(cur\_pos);

}

}

head=(head+1)%4;

}

return res;

}

[68. **Reduce Array Size to The Half – 배열의 길이를 반 이하로 만들기 위하여 삭제해야하는 원소의 최소개수를 구하는 문제. 이때 각 원소는 중복될 수 있음]**

- 내가 생각한 방법의 point는 각 원소의 개수대로 sort를 하는 것이었다. 그래서 pair를 이용했다.

- map 만들때 O(n) sort할 때 O(nlogn)이 걸렸고, 마지막에 length 찾을때 O(n)이 걸렸으므로 O(nlogn)알고리즘 인 것 같다.

- 주의할 점음 pair는 ->이 아니라 . 으로 first, second에 접근한다.

- iterator의 위치를 변경할때는 advance(iter,3)의 방식으로 3번째 다음 인덱스로 iter를 옮길 수 있고, iter간의 거리도 distance(iter1, iter2) 의 형식으로 구할 수 있다.

- 추가로 sort할 때 sort(iter.begin(),iter.end(),greater<int>()); 의 형식으로 하면 내림차순 정렬이 가능하다.

int minSetSize(vector<int>& arr) {

unordered\_map<int,int> table;

vector<int> answer\_array;

for(int ele : arr){

table[ele]++;

}

vector<pair<int,int>> sorted\_pair; //(count,number)

for(unordered\_map<int,int>::iterator iter=table.begin();iter!=table.end();iter++)

sorted\_pair.push\_back(make\_pair(iter->second,iter->first));

sort(sorted\_pair.begin(),sorted\_pair.end());

int half=arr.size()/2; // size is always even.

int current\_length=0;

for(int i=sorted\_pair.size()-1;i>=0;i--){

current\_length+=sorted\_pair[i].first;

answer\_array.push\_back(sorted\_pair[i].second);

if(current\_length>=half) break;

}

return answer\_array.size();

}

[69. **Counting Bits - 0부터 주어진 숫자까지의 수들의 bit 표현에서 1의 개수를 세어 vector에 넣는 문제]**

**-** 나는 그냥 bitset을 이용했다 bitset<32>(i).count()로.

vector<int> countBits(int num) {

vector<int> answer;

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

answer.push\_back(bitset<32>(i).count());

return answer;

}

- 다음과 같은 logic으로도 풀 수 있다.

There is one imporant observation we can make about the number of bits in each number.

1. Each Power of 2 has exactly only 1 bit. (2 : 0010 , 4: 0100, 8:1000, 16:10000)
2. Each number after the power of 2 follows a pecular pattern :  
   0 → 0  
   1 → 0  
   2 → 1 + dp[0] Nearest Power of 2  
   3 → 1 + dp[1] 1 greater than nearest  
   4 → 1 + dp[0] Nearest  
   5 → 1+ dp[1] 1 greater than nearest  
   6 → 1+ dp[2] 2 greater than nearest  
   7 → 1+ dp[3] 3 greater than nearest  
   8 → 1+ dp[0] Nearest  
   9 → 1+ dp[1]  
   10 → 1+ dp[2]  
   11 → 1+ dp[3]  
   12 → 1+ dp[4]

You can easily see the pattern here.

[70. **Complex Number Multiplication – 복소수 2개를 주고 곱하여 string으로 출력하는 문제]**

**-** 그냥 각 복소수의 실수와 허수부분을 나누어 곱해주는 방식으로 했는데, 부호 구분하는 데서 오래 걸렸다.

string complexNumberMultiply(string a, string b) {

string answer;

vector<int> a\_sub;

vector<int> b\_sub;

string temp="";

int i=1;

if(a[0]=='-'){

while('0'<=a[i] && a[i]<='9') temp+=a[i++];

a\_sub.push\_back(std::stoi(temp)\*-1);

}

else{

i=0;

while('0'<=a[i] && a[i]<='9') temp+=a[i++];

a\_sub.push\_back(std::stoi(temp));

}

temp.clear();

if(b[0]=='-'){

i=1;

while('0'<=b[i] && b[i]<='9') temp+=b[i++];

b\_sub.push\_back(std::stoi(temp)\*-1);

}

else{

i=0;

while('0'<=b[i] && b[i]<='9') temp+=b[i++];

b\_sub.push\_back(std::stoi(temp));

}

i=a.size()-2;

temp.clear();

while('0'<=a[i] && a[i]<='9') i--;

if(a[i]=='-'){

i++;

while('0'<=a[i] && a[i]<='9') temp+=a[i++];

a\_sub.push\_back(std::stoi(temp)\*-1);

}

else{

i++;

while('0'<=a[i] && a[i]<='9') temp+=a[i++];

a\_sub.push\_back(std::stod(temp));

}

i=b.size()-2;

temp.clear();

while('0'<=b[i] && b[i]<='9') i--;

if(b[i]=='-'){

i++;

while('0'<=b[i] && b[i]<='9') temp+=b[i++];

b\_sub.push\_back(std::stoi(temp)\*-1);

}

else{

i++;

while('0'<=b[i] && b[i]<='9') temp+=b[i++];

b\_sub.push\_back(std::stoi(temp));

}

answer+=std::to\_string(a\_sub[0]\*b\_sub[0]+a\_sub[1]\*b\_sub[1]\*-1);

answer+='+';

answer+=std::to\_string(a\_sub[1]\*b\_sub[0]+a\_sub[0]\*b\_sub[1]);

answer+='i';

return answer;

}

- find 함수 이용 간단하게 풀 수 있는 discussion이 있었다.

public:

string complexNumberMultiply(string a, string b) {

pair<int, int> av = parse(a);

pair<int, int> bv = parse(b);

int ra = av.first \* bv.first - av.second \* bv.second;

int rb = av.first \* bv.second + av.second \* bv.first;

return to\_string(ra) + "+" + to\_string(rb) + "i";

}

pair<int, int> parse(const string& a) {

int plus = find(a.begin(), a.end(), '+') - a.begin();

int i = find(a.begin(), a.end(), 'i') - a.begin();

int ra = stoi(a.substr(0, plus));

int rb = stoi(a.substr(plus + 1, i - plus));

return {ra, rb};

}

[71. **Fizz Buzz Multithreaded -Thread 사용하는 문제]**

- 덕분에 thread 공부했다. 하지만 나는 풀 수 없었기에, discussion 보고 thread 어떻게 쓰는지 배웠다.

private:

int n;

int count;

mutex m;

condition\_variable cv;

public:

FizzBuzz(int n) {

this->n = n;

this->count = 1;

}

void fizz(function<void()> printFizz) {

while (true) {

unique\_lock<mutex> lock(m);

while (count <= n && (count % 3 != 0 || count % 5 == 0))

cv.wait(lock);

if (count > n) return;

printFizz();

++count;

cv.notify\_all();

}

}

void buzz(function<void()> printBuzz) {

while (true) {

unique\_lock<mutex> lock(m);

while (count <= n && (count % 5 != 0 || count % 3 == 0))

cv.wait(lock);

if (count > n) return;

printBuzz();

++count;

cv.notify\_all();

}

}

void fizzbuzz(function<void()> printFizzBuzz) {

while (true) {

unique\_lock<mutex> lock(m);

while (count <= n && (count % 5 != 0 || count % 3 != 0))

cv.wait(lock);

if (count > n) return;

printFizzBuzz();

++count;

cv.notify\_all();

}

}

void number(function<void(int)> printNumber) {

while (true) {

unique\_lock<mutex> lock(m);

while (count <= n && (count % 5 == 0 || count % 3 == 0))

cv.wait(lock);

if (count > n) return;

printNumber(count++);

cv.notify\_all();

}

}

[72. **[951] Flip Equivalent Binary Trees – when we choose any node and swap left and right subtrees, we call it a flip operation. Write a function whether two given trees are flip equivalent.]**

- no matter a tree is filped or not, its children have to be same or just changed the order.

- So we can just compare every possible situation – Greedy using DFS.

bool flag=true;

void helper(TreeNode\* root1,TreeNode\* root2){

if(!root1->left && !root1->right && !root2->left && !root2->right)

return;

if(root1->left && root2->left && root1->left->val == root2->left->val)

helper(root1->left,root2->left);

else if(root1->left && root2->right && root1->left->val==root2->right->val)

helper(root1->left,root2->right);

else if(root1->left){

flag=false;

return;

}

if(!flag) return;

else if(root1->right && root2->left && root1->right->val==root2->left->val)

helper(root1->right,root2->left);

else if(root1->right && root2->right && root1->right->val==root2->right->val)

helper(root1->right,root2->right);

else if(root1->right){

cout<<"flag"<<endl;

flag=false;

}

}

bool flipEquiv(TreeNode\* root1, TreeNode\* root2) {

if(!root1 && !root2) return true;

if((root1 && !root2) ||(!root1 && root2)) return false;

helper(root1,root2);

return flag;

}

[73. [**1123] Lowest Common Ancestor of Deepest Leave]**

**-** find deepest level from root. If the deepest depth is same, then the root is the answer.

- if the depth is different. determine which one has deeper depth between left and right child.

- and doing recursive with the child who has deeper detph.

int helper(TreeNode\* root){

if(!root) return 0;

return 1+max(helper(root->left),helper(root->right));

}

TreeNode\* lcaDeepestLeaves(TreeNode\* root) {

int left=helper(root->left);

int right=helper(root->right);

if(left==right) return root;

else if(left>right) return lcaDeepestLeaves(root->left);

else return lcaDeepestLeaves(root->right);

}

[74. **[986]** **Interval List Intersections – Given two sorted closed interval, return the intersection of these two lists.]**

- At first try, I use map to record interval of two lists into one store with using vector in the map for checking every interval’s end.

- I think I just need O(total interval of A and B + map.size()). But time limit exceeded occurs.

class Solution {

public:

vector<vector<int>> intervalIntersection(vector<vector<int>>& A, vector<vector<int>>& B) {

map<int,vector<int>> inter;

vector<vector<int>> answer;

int i;

for(vector<int> ele : A){

for(i=ele[0];i<ele[1];i++){

inter[i]={1,0,0};

}

inter[i]={1,1,0};

}

for(vector<int> ele : B){

for(i=ele[0];i<=ele[1];i++){

if(inter[i].empty()) inter[i]={1,0,0};

else inter[i][0]++;

}

inter[i-1][2]=1;

}

map<int,vector<int>>::iterator iter=inter.begin();

int left=0;

for(;iter!=inter.end();iter++){

if(iter->second[0]==2){

left=iter->first;

while(iter->second[0]==2){

if(iter->second[1]==1 || iter->second[2]==1)

break;

else iter++;

}

answer.push\_back({left,iter->first});

}

}

return answer;

}

};

- I guess the reason why the time limit exceed occurs is because I search all the element in each intervals. So I should figure out how to solve this problem by just using left and right limit of each intervals only.

- the final answer is that if A[i][1] is less than B[j][0], i++, vice versa j++.

- after that we can find overlapping range by two lists. From there, we just determine which one is left and right limit.

- see the code.

class Solution {

public:

vector<vector<int>> intervalIntersection(vector<vector<int>>& A, vector<vector<int>>& B) {

vector<vector<int>> answer;

for(int i=0, j=0;i<A.size() && j<B.size();){

if(A[i][1]<B[j][0]) i++;

else if(A[i][0]>B[j][1]) j++;

else{

answer.push\_back({max(A[i][0],B[j][0]),min(A[i][1],B[j][1])});

if(A[i][1]<B[j][1]) i++;

else j++;

}

}

return answer;

}

};

[75. [**1110] Delete Nodes And Return Forest – delete given node, and return roots of every subtrees.]**

- I solved this problme using bottom-up approach.

- First at all, using postorder search and if a current node has to be deleted, insert their children into answer vector, and come back to parents node, finally delete the node.

- but the point is how I can handle the original root node, value 1, because of that, I have to use another find function whether the to\_delete vector has 1 or not. so it cunsume pretty much time I think.

- Tommrow I will revise this algorithm.

public:

TreeNode\* helper(TreeNode\* root,vector<TreeNode\*>& answer, vector<int>& to\_delete){

if(!root) return nullptr;

TreeNode\* left=nullptr;

TreeNode\* right=nullptr;

left=helper(root->left,answer,to\_delete);

right=helper(root->right,answer,to\_delete);

if(left) root->left=nullptr;

if(right) root->right=nullptr;

for(int i=0;i<to\_delete.size();i++){

if(to\_delete[i]==root->val){

if(root->left) answer.push\_back(root->left);

if(root->right) answer.push\_back(root->right);

//to\_delete.erase(to\_delete.begin()+i);

return root;

}

}

return nullptr;

}

vector<TreeNode\*> delNodes(TreeNode\* root, vector<int>& to\_delete) {

vector<TreeNode\*> answer;

helper(root,answer,to\_delete);

if(find(to\_delete.begin(),to\_delete.end(),1)==to\_delete.end()) answer.push\_back(root);

return answer;

}

};

[76. [**969] Pancake Sorting – find the order of indexs to reverse the given vector from vector.begin() so that the vector is sorted]**

- the flip begin from A[0]. so we have to sort from A.end() descendantly.

- at first, we have to find the largest elements that can be found using A.length.

- then flip from start to the largest elements index so that the largest elements come to first index.

- and reverse all A so as to put the largest elements to last index. and so on.

class Solution {

public:

vector<int> pancakeSort(vector<int>& A) {

int largest=A.size();

vector<int>::iterator last=A.end();

vector<int>::iterator cur;

vector<int> answer;

for(int i=0; i<A.size()-1; i++){

cur=find(A.begin(),last,largest);

cur++;

answer.push\_back(distance(A.begin(),cur));

answer.push\_back(distance(A.begin(),last));

reverse(A.begin(),cur);

reverse(A.begin(),last);

largest--;

advance(last,-1);

}

return answer;

}

};

[77. [**959] Regions Cut By Slashes – return how many distinct regions appear after dividing whole square by /, \]**

**-** I solved this problem using BFS. But the result was time limit exceeded.

- 1. make the given array 3 time widen. Because to recognize distinct region, we need at least 3 time wider array.

- 2. draw given line to divide region.

- 3. using BFS, paint regions and count hom many regions are.

class Solution {

public:

void helper(vector<vector<int>>& table,queue<pair<int,int>> que){

while(!que.empty()){

int x=que.front().first;

int y=que.front().second;

que.pop();

table[x][y]=1;

if(x && table[x-1][y]==0) que.push(make\_pair(x-1,y));

if(y && table[x][y-1]==0) que.push(make\_pair(x,y-1));

if(x!=table.size()-1 && table[x+1][y]==0) que.push(make\_pair(x+1,y));

if(y!=table[0].size()-1 && table[x][y+1]==0) que.push(make\_pair(x,y+1));

}

}

void paintRegion(vector<vector<int>>& table,int& answer){

queue<pair<int,int>> que;

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

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

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

que.push(make\_pair(i,j));

helper(table,que);

answer++;

}

}

}

}

int regionsBySlashes(vector<string>& grid) {

int N=grid.size()\*3;

vector<vector<int>> table(N,vector<int>(N,0)); //make table N\*3 X N\*3

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

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

if(grid[i][j]=='/'){

table[i\*3][3\*j+2]=1;

table[i\*3+1][3\*j+1]=1;

table[i\*3+2][3\*j]=1;

}

else if(grid[i][j]=='\\'){

table[i\*3][3\*j]=1;

table[i\*3+1][3\*j+1]=1;

table[i\*3+2][3\*j+2]=1;

}

}

}

int answer=0;

paintRegion(table,answer);

return answer;

}

};

- due to time limit exceeded, I changed my algorithm to DFS.

class Solution {

public:

void paintRegion(vector<vector<int>>& table,int x, int y){

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

table[x][y]=1;

if(x) paintRegion(table,x-1,y);

if(y) paintRegion(table,x,y-1);

if(x!=table.size()-1) paintRegion(table,x+1,y);

if(y!=table.size()-1) paintRegion(table,x,y+1);

}

}

int regionsBySlashes(vector<string>& grid) {

int N=grid.size()\*3;

int answer=0;

vector<vector<int>> table(N,vector<int>(N,0)); //make table N\*3 X N\*3

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

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

if(grid[i][j]=='/'){

table[i\*3][3\*j+2]=1;

table[i\*3+1][3\*j+1]=1;

table[i\*3+2][3\*j]=1;

}

else if(grid[i][j]=='\\'){

table[i\*3][3\*j]=1;

table[i\*3+1][3\*j+1]=1;

table[i\*3+2][3\*j+2]=1;

}

}

}

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

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

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

paintRegion(table,i,j);

answer++;

}

}

}

return answer;

}

};

[78. [**1130] Minimum Cost Tree From Leaf Values – find minimum sum of non-leaf node using the given leaves.]**

- Atually, this problem was super difficult for me. First at all, I thought I sholud have construted a real tree. But it’s quite insane jobs.

- so I tried to make dp but it was not that easy. Finally, I refered to discussion.

- O(N^2) algorithm is below.

> 1. find minimum leaf of the array : val1

> 2. find min(the leaf’s left, the leaf’s right) : val2

> 3. answer+= val1 \* val2

> 4. remove val1 and repeat from 1st.

class Solution {

public:

int mctFromLeafValues(vector<int>& arr) {

int answer=0;

int minimum;

int second;

while(arr.size()>1){

minimum=min\_element(arr.begin(),arr.end())-arr.begin(); //minimum value's index

if(minimum==0) second=1;

else if(minimum==arr.size()-1) second=minimum-1;

else{

if(arr[minimum+1]>arr[minimum-1]) second=minimum-1;

else second=minimum+1;

}

answer+=arr[minimum]\*arr[second];

arr.erase(arr.begin()+minimum);

if(minimum>second) minimum--;

}

return answer;

}

};

- Now I’m trying to understand O(N^3) solution that uses dp.

- The dp solusion from discussion is below.

**Intuition and Algorithm**  
Given an array arr return the smallest possible sum of the values of each non-leaf node.  
The value of each non-leaf node is equal to the product of the largest leaf value in its left and right subtree respectively.

For example:  
arr= [3, 6, 4, 7, 2, 5]



This is one distribution, we are going to solve that from top to bottom with dynammic programming

**Approach 1 (DP)**  
In the image the root, their left subtree contains indexes [0-3] and their right subtree cointains indexes [4-5]. Then their value will be max(arr[0-3])\* max(arr[4-5]).

In general:  
dp(left, right )= min( max(arr[left .. i] ) \* max(arr[i+1 .. right]) + dp(left,i) +dp(i+1,right) ) where i go from left to right-1

class Solution {

public:

int memo[41][41];

int maxi[41][41];

int dp(int left,int right){

if(left==right)return 0; //leaf node

if(memo[left][right]!=-1)return memo[left][right];

int ans = 1<<30;

for(int i=left;i<right;i++)

ans= min(ans, maxi[left][i] \* maxi[i+1][right] + dp(left,i) + dp(i+1,right) );

memo[left][right]=ans;

return ans;

}

int mctFromLeafValues(vector<int>& arr) {

memset(memo,-1,sizeof(memo));

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

maxi[i][i] = arr[i];

for(int j=i+1;j<arr.size();j++)

maxi[i][j] = max(maxi[i][j-1], arr[j]);

}

return dp(0,arr.size()-1);

}

};

[79 [**889] Construct Binary Tree from Preorder and Postorder Traversal – make a binary tree using the given pre and postorder traversals]**

**-** My first algorithm is that the right next node of current node in preoder traversal is root’s left child and the left next node of current node in post traversal is root’s right child.

- But it’s wrong.

- So I tried a divivde and conquer approach.

- What a difficult problem! I spent almost 4 hours for this heck!

- The key point is divide each traversal respectively.

- And if after making left child, mid+1==postr, then just have to return root.

class Solution {

public:

unordered\_map<int,int> table;

TreeNode\* helper(vector<int>& pre, vector<int>& post,int prel,int prer,int postl, int postr){

TreeNode\* root=new TreeNode(pre[prel]);

if(prel==prer) return root;

int mid=table[pre[prel+1]];

int length=mid-postl;

root->left=helper(pre,post,prel+1,prel+length+1,postl,postl+length);

if(mid+1==postr) return root;

root->right=helper(pre,post,prel+length+2,prer,mid+1,postr-1);

return root;

}

TreeNode\* constructFromPrePost(vector<int>& pre, vector<int>& post) {

for(int i=0;i<post.size();i++) table[post[i]]=i;

TreeNode\* root=helper(pre,post,0,pre.size()-1,0,post.size()-1);

return root;

}

};

[80. [**791] Custom Sort String – given sorted array S in a custm way, sort array T so that it is sorted like array S]**

‑ Intuition is below

> 1. count each letter in the array T into an unordered\_map count.

> 2. For a character C from S.begin to S.end, if the map count has the character of S, add it to answer as many as count[C]

> 3. concatenat all the rest charater in the map count.

class Solution {

public:

string customSortString(string S, string T) {

unordered\_map<char,int> count;

for(char c : T)

count[c]++;

string answer;

for(char c : S){

if(count.find(c)!=count.end()){

while(count[c]>0){

answer+=c;

count[c]--;

}

}

}

unordered\_map<char,int>::iterator iter=count.begin();

for(;iter!=count.end();iter++){

while(iter->second>0){

answer+=iter->first;

iter->second--;

}

}

return answer;

}

};

[81. [**442] Find All Duplicates in an Array – find numbers which occur twice]**

- If I just have to problem without concerning memory ans time consuming, it is easy.

- But the point is to use only O(1) memory and O(n) time.

- Algorithm is below

> 1. using numbers in array nums as index and negate it. It’s possible since 1<=nums[i]<=n where 0<=i<n.

> 2. if a number occurs twice, the number’s sign must be + not -.

class Solution {

public:

vector<int> findDuplicates(vector<int>& nums) {

vector<int> answer;

for(int i : nums){

i=abs(i);

nums[i-1]=-nums[i-1];

if(nums[i-1]>0) answer.push\_back(i);

}

return answer;

}

};

[82. [**1343] Number of Sub-arrays of Size K and Average Greater than or Equal to Threshold – find the number of sub-array of size K that has an average greater than or equal to given threshold]**

-Algorithm is below

> 1. From first of the given array, add K-1 elements. => cur\_sum

> 2. iteratively, make cur\_sum has K elements. if K’s average is valid about the given condition. then add 1 to answer

> subtract arr[i-k+1] from cur\_sum and add arr[[i] to cur]sum and repeat the algorithm from 2.

class Solution {

public:

int numOfSubarrays(vector<int>& arr, int k, int threshold) {

int cur\_sum=0;

int answer=0;

for(int i=0;i<k-1;i++)

cur\_sum+=arr[i];

for(int i=k-1;i<arr.size();i++){

cur\_sum+=arr[i];

if(cur\_sum/k >= threshold) answer++;

cur\_sum-=arr[i-k+1];

}

return answer;

}

};

[83. [**1043] Partition Array for Maximum Sum – partition the given array into subarrays of length at most given K. each subarray’s value will be changed the greatest value in the subarrays. return the largest sum of the given arrays after partitioning]**

**-** what the heck? this problem is not a midium level. I think!, Yes, I may not good at DP problem :).

- Anyway, I tried to solve this problem 2d DP.

- row is index, col is the size of partition.

- Algorithm is just >> dp[i][j] = max(dp[i-j]…dp[i-1]) + max(A[i-j+1]….A[i])\*j).



- But tiem limit exceeded occurred.

- see the code.

class Solution {

public:

void printDP(vector<vector<int>> dp){

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

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

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

}

cout<<endl;

}

}

void initializeDP(vector<int> A, int K, vector<vector<int>>& dp){

//vector initialized

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

dp[i][0]=A[i]; //when K=1;

//initialize DP's diagonal.

for(int i=1,j=1;i<K;i++,j++){

dp[i][j]=\*std::max\_element(A.begin(),next(A.begin(),i+1))\*(j+1);

}

//initialize DP's upper triangle.

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

for(int j=i+1;j<K;j++){

dp[i][j]=dp[i][j-1];

}

}

}

int dpMAX(vector<vector<int>>& dp,vector<int> A, int K, int i, int j){

int res=INT\_MIN;

int prev;

int max\_ele=INT\_MIN;

for(int a=i-(j+1);a<i;a++){

prev=dp[a][j]; // a = 1

for(int b=a+1;b<=i;b++){

max\_ele=max(max\_ele,A[b]);

}

res=max(res,prev+max\_ele\*(i-a));

max\_ele=INT\_MIN;

}

return res;

}

int maxSumAfterPartitioning(vector<int>& A, int K) {

if(K==1){

return accumulate(A.begin(),A.end(),0);

}

vector<vector<int>> dp(A.size(),vector<int>(K,0));

initializeDP(A,K,dp);

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

for(int i=j+1;i<A.size();i++){

dp[i][j]=dpMAX(dp,A,K,i,j);

}

}

printDP(dp);

return dp[A.size()-1][K-1];

}

};

- I happened to recognize that I don’t need the 2d DP, since I don’t use previous j(which is 1…K-1) array in the 2d DP. I needed just dp[i][K].

- so I revised my algorithm to an 1d DP.

- speed was 11.42% beats. I know is quite slow, but I’m happy I could solve this problem :).

- see the code.

class Solution {

public:

void initializeDP(vector<int> A, int K, vector<int>& dp){

int max\_ele=INT\_MIN;

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

dp[i]=\*max\_element(A.begin(),next(A.begin(),i+1))\*(i+1);

}

int dpMAX(vector<int>& dp,vector<int> A,int i,int K){

int res=INT\_MIN;

int prev;

int max\_ele=INT\_MIN;

for(int a=i-K;a<i;a++){

prev=dp[a];

for(int b=a+1;b<=i;b++){

max\_ele=max(max\_ele,A[b]);

}

res=max(res,prev+max\_ele\*(i-a));

max\_ele=INT\_MIN;

}

return res;

}

int maxSumAfterPartitioning(vector<int>& A, int K) {

if(K==1){

return accumulate(A.begin(),A.end(),0);

}

vector<int> dp(A.size(),0);

initializeDP(A,K,dp);

for(int i=K;i<A.size();i++){

dp[i]=dpMAX(dp,A,i,K);

}

return dp[A.size()-1];

}

};

[84. [**1219] Path with Maximum Gold – find a path to get maximum gold]**

**-** I used DFS to solve this problem. the algorithm is quite intutive. I think the explanation is not needed. just cafeful to use visit array.

- but time limit exceeded occurred.

- see the code.

class Solution {

public:

int DFS(vector<vector<int>> grid,int x,int y,vector<vector<int>> visit){

visit[x][y]=1;

int cur\_gold=grid[x][y];

vector<int> max\_gold;

if(x && grid[x-1][y]&& !visit[x-1][y]) max\_gold.push\_back(DFS(grid,x-1,y,visit));

if(y && grid[x][y-1]&&!visit[x][y-1]) max\_gold.push\_back(DFS(grid,x,y-1,visit));

if(x+1<grid.size() && grid[x+1][y]&&!visit[x+1][y] ) max\_gold.push\_back(DFS(grid,x+1,y,visit));

if(y+1<grid[0].size() && grid[x][y+1]&&!visit[x][y+1]) max\_gold.push\_back(DFS(grid,x,y+1,visit));

return cur\_gold+ (max\_gold.empty()? 0:\*max\_element(max\_gold.begin(),max\_gold.end()));

}

int getMaximumGold(vector<vector<int>>& grid) {

int answer=0;

vector<vector<int>> visit(grid.size(),vector<int>(grid[0].size(),0));

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

for(int j=0;j<grid[i].size();j++)

if(grid[i][j])

answer=max(answer,DFS(grid,i,j,visit));

return answer;

}

};

- I thought the time limit exceeded happened due to max\_gold vector and visit vector. so I remove these.

- First, I changed max\_gold vector to int and just compare all the possible DFS. Because putting a value into vector and finding maximum value from that vector need quite long time I think.

- Second, I remove visit vector and just use the condition which is, if a cell has 0 golds, then we don’t need to visit the cell. so whenever I visit a cell, I changed the value to 0 like I collected the gold. After all the visiting to neighbor cells, I returned the amount of gold to the cell.

- Finally, I got speed 96.09% beats :).

- see the code.

class Solution {

public:

int DFS(vector<vector<int>>& grid,int x,int y){

int cur\_gold=grid[x][y];

grid[x][y]=0;

int max\_gold=0;

if(x && grid[x-1][y]) max\_gold=max(max\_gold,DFS(grid,x-1,y));

if(y && grid[x][y-1]) max\_gold=max(max\_gold,DFS(grid,x,y-1));

if(x+1<grid.size() && grid[x+1][y]) max\_gold=max(max\_gold,DFS(grid,x+1,y));

if(y+1<grid[0].size() && grid[x][y+1]) max\_gold=max(max\_gold,DFS(grid,x,y+1));

grid[x][y]=cur\_gold;

return cur\_gold+ max\_gold;

}

int getMaximumGold(vector<vector<int>>& grid) {

int answer=0;

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

for(int j=0;j<grid[i].size();j++)

if(grid[i][j])

answer=max(answer,DFS(grid,i,j));

return answer;

}

};

[85. [**877] Stone Game – determine whether alex wins the game or not]**

- the game rule is below.

> 1. player can choose either given array’s first or last element.

> 2. player tries to collect most stone(number) and acts optimally.

- we should determine alex can win or not given array.

- I use DFS to solve this problem. Since each player tries to get more stone than the opponent, we should do brute-force.

- At first, I use the start iterator like below

> piles.erase(start);

- but it occurred a heap-buffer-overflow, I don’t know why actually. So I changed the code to use the current function’s pile vector.

- Finally, I could solve this problem. but the speed was just 49.95% beats.

- see the code.

class Solution {

public:

bool DFS(vector<int> piles, vector<int>::iterator start,vector<int> alex\_lee,int turn){

bool answer=false;

if(piles.empty())

return alex\_lee[0]>alex\_lee[1];

alex\_lee[(turn++)%2]+=\*start;

if(piles.begin()==start)

piles.erase(piles.begin());

else piles.erase(next(piles.end(),-1));

return DFS(piles,piles.begin(),alex\_lee,turn) || DFS(piles,next(piles.end(),-1),alex\_lee,turn);

}

bool stoneGame(vector<int>& piles) {

return DFS(piles,piles.begin(),vector<int>{2,0},0) || DFS(piles,next(piles.end(),-1),vector<int>{2,0},0);

}

};

[86. [**1026] Maximum Difference Between Node and Ancestor – find the greatest difference between an ancestor and decendant]**

- In my algorithm, I should’ve used both DFS and BFS for brute-force.

- The algorithm is quite vivid. For each node, I check differeces between the current node and its decendants.

- see the code. the speed was bad. It was just 5.20% beats.

class Solution {

public:

int DFS(TreeNode\* root,int cur\_val){

if(!root) return 0;

return max(abs(root->val-cur\_val),max(DFS(root->left,cur\_val),DFS(root->right,cur\_val)));

}

int maxAncestorDiff(TreeNode\* root) {

queue<TreeNode\*> que;

TreeNode\* cur;

que.push(root);

int answer=0;

while(!que.empty()){

cur=que.front();

que.pop();

if(cur->left) que.push(cur->left);

if(cur->right) que.push(cur->right);

answer=max(answer,max(DFS(cur->left,cur->val),DFS(cur->right,cur->val)));

}

return answer;

}

};

- In the discussion, there is such a superb approach using 1d vector.

- The algorithm is below.

> 1. finding every paht from root to leaf. and push to vector.

> 2. Then a lefter element is an ancestor of righter element.

> 3. by using this vector, we can find the maximum difference.

- This algorithm was just O(n) where n is the number of nodes.

- Even, we don’t need the 1d vector, we just have to maintain current maximum and minimum value so that when we meet a leaf node, we can use it to calculate a local maximum difference.

- the speed was 69.22% beats. I reckon it quite fast enough.

-see the code.

class Solution {

public:

void DFS(TreeNode\* root, int& answer,int maximum,int minimum){

maximum=max(maximum,root->val);

minimum=min(minimum,root->val);

if(!root->left && !root->right){

answer=max(answer,maximum-minimum);

return;

}

if(root->left) DFS(root->left,answer,maximum,minimum);

if(root->right) DFS(root->right,answer,maximum,minimum);

}

int maxAncestorDiff(TreeNode\* root) {

int answer=0;

DFS(root,answer,INT\_MIN,INT\_MAX);

return answer;

}

};

[87. [**841] Keys and Rooms – each room has a list of keys to enter rooms. return true if it is possible to enter every room]**

- Algorithm is below.

> 1. when I enter a room, inserting every key into queue and recording all the key into can\_visit. Finally, removing all the key in the room.

> 2. pop a key from the queue, if I already visited the key’s room, continue. otherwise repeat from 1.

> 3. when queue is empty, checking whether the given the number of rooms is same to can\_visit.

- the speed was 68.01% beats.

- see the code.

class Solution {

public:

bool canVisitAllRooms(vector<vector<int>>& rooms) {

unordered\_set<int> can\_visit;

queue<int> que;

int cur\_room;

can\_visit.insert(0);

for(int i : rooms[0]){

que.push(i);

can\_visit.insert(i);

}

rooms[0].clear();

while(!que.empty()){

cur\_room=que.front();

que.pop();

if(rooms[cur\_room].empty()) continue;

for(int i : rooms[cur\_room]){

que.push(i);

can\_visit.insert(i);

}

rooms[cur\_room].clear();

}

return rooms.size()==can\_visit.size();

}

};

[88. **[912] Sort an Array – just sorting an array]**

**-** I implement merge sort. But the speed was suck! 5.17% beats.

- so I impelment quick sort as well. But the speed was just 16.25% beats.

- see the code.

class Solution {

public:

vector<int> merge(vector<int> nums){

if(nums.size()==1) return {nums[0]};

vector<int>::iterator mid=next(nums.begin(),nums.size()/2);

vector<int> left;

vector<int> right;

left=merge({nums.begin(),mid});

right=merge({mid,nums.end()});

vector<int> res;

int i=0,j=0;

while(i<left.size() && j<right.size()){

if(left[i]>right[j]){

res.push\_back(right[j]);

j++;

continue;

}

else{

res.push\_back(left[i]);

i++;

continue;

}

}

while(i<left.size()) res.push\_back(left[i++]);

while(j<right.size()) res.push\_back(right[j++]);

return res;

}

void quick(vector<int>& nums, int left, int right){

if(left>=right) return;

int pivot=left;

int i=left+1;

int j=right;

while(i<j){

while(i<j && nums[pivot]>nums[i]) i++;

while(i<j && nums[pivot]<nums[j]) j--;

if(i>=j) break;

int temp=nums[i];

nums[i]=nums[j];

nums[j]=temp;

i++;

j--;

}

if(nums[pivot]>nums[j]){

int temp=nums[j];

nums[j]=nums[pivot];

nums[pivot]=temp;

}

quick(nums,left,j-1);

quick(nums,j,right);

}

vector<int> sortArray(vector<int>& nums) {

quick(nums,0,nums.size()-1);

return nums;

}

};

[89. [**406] Queue Reconstruction by Height – standing people by given a height rule]**

**-** a person have a height and a number which represents the number of person who have a height higher than or equal to the person. by using it, we have sort the given array.

- this problem was quite difficult. Algorithm didn’t happen in my brain.

- Algorithm is below.

> 1. sorting max-height people as a subarray.

> 2. using the value k that each person has as an index, inserting every person to answer vector.

> 3. repeating from 1, until the given array people is empty.

- the speed was 11.52% beats.

- see the code.

class Solution {

public:

vector<vector<int>> MaxSubarray(int cur\_max\_value,vector<vector<int>>& people){

vector<vector<int>> cur\_max;

vector<vector<int>>::iterator iter=next(people.end(),-1);

for(;iter!=next(people.begin(),-1);iter--){

if(cur\_max\_value!=iter[0][0]){

iter++;

break;

}

}

if(iter==next(people.begin(),-1)) iter=people.begin();

cur\_max={iter,people.end()};

people.erase(iter,people.end());

sort(cur\_max.begin(),cur\_max.end());

return cur\_max;

}

vector<vector<int>> reconstructQueue(vector<vector<int>>& people) {

if(people.empty()) return {};

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

vector<vector<int>> cur\_max;

vector<vector<int>> answer;

int cur\_max\_value=people[people.size()-1][0];

answer=MaxSubarray(cur\_max\_value,people);

while(!people.empty()){

cur\_max\_value=people[people.size()-1][0];

cur\_max=MaxSubarray(cur\_max\_value,people);

for(vector<int> person : cur\_max)

answer.insert(next(answer.begin(),person[1]),person);

cout<<endl;

}

return answer;

}

};

[90. [**429] N-ary Tree Level Order Traversal – split given tree based on levels]**

- I think there are two method to solve the problem. one is to use two queues, the other is to use only one queue and to count how many nodes we have to consider in a level

- I chose a second algorithm.

- Algorithm is below.

> 1. For each iteration, memorize how many children are there at a certain level. Then we can use the number as a next level’s the number of nodes.

> 2. And for each iteration, if we met NULL, skip it. otherwise put the node’s children into the queue.

> 3. repeating from 1. until queue becomes empty.

- the speed was 52.03% beats.

-see the code. notice that I added the Node class since it’s not the usual form of normal Node Class.

/\*

// Definition for a Node.

class Node {

public:

int val;

vector<Node\*> children;

Node() {}

Node(int \_val) {

val = \_val;

}

Node(int \_val, vector<Node\*> \_children) {

val = \_val;

children = \_children;

}

};

\*/

class Solution {

public:

void levelOrder(Node\* root,queue<Node\*>& que,vector<int>& cur\_level){

if(!root) return;

for(Node\* i : root->children){

que.push(i);

cur\_level.push\_back(i->val);

}

}

vector<vector<int>> levelOrder(Node\* root) {

if(!root) return {};

queue<Node\*> que;

vector<vector<int>> answer;

vector<int> cur\_level;

que.push(root);

Node\* cur;

int need=1;

answer.push\_back({{root->val}});

while(!que.empty()){

for(int count=0;count<need;count++){

cur=que.front();

que.pop();

levelOrder(cur,que,cur\_level);

}

if(!cur\_level.empty()) answer.push\_back(cur\_level);

need=cur\_level.size();

cur\_level.clear();

}

return answer;

}

};

[91. [**1238] Circular Permutation in Binary Representation – return a permutation such that p[i] and p[i+1] differ by only one bit in their binary representation]**

**-** My algorithm is below.

> 1. put string “0” and “1” into vector table as a initial values. since n’s least limit is 1.

> 2. push\_back the strings in the vector table in a reverse order.

> 3. add 0’s for the number of table.size()/2 and add 1’s for the rest.

> 4. From 1 to n, repeat the algorithm from 2 to 4.

> 5. reverse all the string in the vector table.

> 6. covert all the string in the vector table to int. the method is below.

>>> Notice: usage of stoi

int i = std::stoi("01000101", nullptr, 2);

* The returned value is the converted int value.
* The first argument is the std::string you want to convert.
* The second is a size\_t \* where it'll save the index of the first non digit character.
* The third is an int that corresponds to the base that'll be used for conversion..

> 7. find start position and reconstruct vector to return.

- refer to the picture I drew.



- the speed was not good. It was just 9.85% beats.

class Solution {

public:

void reverseConcatenate(vector<string>& table){

for(int i=table.size()-1;i>=0;i--)

table.push\_back(table[i]);

}

void addBits(vector<string>& table){

for(int i=0;i<table.size()/2;i++)

table[i]+='0';

for(int i=table.size()/2;i<table.size();i++)

table[i]+='1';

}

void reverseStrings(vector<string>& table){

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

reverse(table[i].begin(),table[i].end());

}

vector<int> binaryToInt(vector<string> table){

vector<int> answer;

for(string s : table)

answer.push\_back(std::stoi(s,nullptr,2));

return answer;

}

vector<int> circularPermutation(int n, int start) {

vector<string> table={"0","1"};

for(int i=2;i<=n;i++){

reverseConcatenate(table);

addBits(table);

}

reverseStrings(table);

vector<int> answer=binaryToInt(table);

vector<int>::iterator iter=find(answer.begin(),answer.end(),start);

vector<int> res={iter,answer.end()};

for(vector<int>::iterator res\_iter=answer.begin();res\_iter!=iter;res\_iter++)

res.push\_back(\*res\_iter);

return res;

}

};

[92. [**998] Maximum Binary Tree II – insert the given value into max heap]**

- Algorithm is below.

> 1. if root is exist and root->val > given value, call the helper function with giving root->right as a parameter.

> 2. if root is not exist or root->val<=given value, create a new node of given value.

> 3. Allocate the new node->left = root. And return the node.

- see the code.

class Solution {

public:

TreeNode\* helper(TreeNode\* root, int val){

if(root && root->val>val){

root->right=helper(root->right,val);

return root;

}

TreeNode\* node=new TreeNode(val);

node->left=root;

return node;

}

TreeNode\* insertIntoMaxTree(TreeNode\* root, int val) {

return helper(root,val);

}

};

[93. [**739] Daily Temperatures – determine how many days we have to wait so that a temperatur is wamer than today.]**

- Algorithm is below.

> 1. make stack and push the last element in the given vector T as a pair with count 0 : (T[i],0)

> 2. For i where t.end-2 to 0, if stack’s top is less than T[i], pop and accumulate count.

> 3. if stack is empty, put the T[i] with 0. otherwise put the t[i] with count and put the count into answer[i]

- see the code.

class Solution {

public:

vector<int> dailyTemperatures(vector<int>& T) {

stack<pair<int,int>> stk;

vector<int> answer(T.size(),0);

stk.push(make\_pair(T.back(),0));

for(int i=T.size()-2;i>=0;i--){

int count=1;

while(!stk.empty() && stk.top().first<=T[i]){

count+=stk.top().second;

stk.pop();

}

if(stk.empty()){

stk.push(make\_pair(T[i],0));

}

else{

stk.push(make\_pair(T[i],count));

answer[i]=count;

}

}

return answer;

}

};

[94. [**973] K Closest Points to Origin – calculate Euclidean distance of given coordinates and find K CLosets Points to Origin]**

- I used map. key is Euclidean Distance and value is the coordinates.

- Since there might be same Euclidean Distance but different coordinates. I should’ve used vector<vector<int>> as a value of the map.

- the speed was 27.42% beats.

- I found we can use nth\_element function that sort the list till given index is sorted.

- see the code.

class Solution {

public:

int Euclidean(int x, int y){

return pow(x,2)+pow(y,2);

}

vector<vector<int>> kClosest(vector<vector<int>>& points, int K) {

vector<vector<int>> answer;

map<int,vector<vector<int>>> table;

for(vector<int> cor: points)

table[Euclidean(cor[0],cor[1])].push\_back({cor[0],cor[1]});

int count=0;

map<int,vector<vector<int>>>::iterator iter=table.begin();

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

for(int j=0;j<iter->second.size();j++){

if(count>=K) break;

answer.push\_back({iter->second[j][0],iter->second[j][1]});

count++;

}

iter++;

}

return answer;

}

};

[94. [**1318] Minimum Flips to Make a OR b Equal to c – return how many filps we need to make a OR b == c]**

- Algorithm is below.

> 1. given a,b and c, convert these to bitset.

> 2. if C[i]==1 and A[i]==0 and B[i]==0, then filp++;

> 3. if C[i]==0 and A[i]==1 ,flip++ ,or B[i]==1, filp++; where 0<= i < log2(max(a,b,c)) +1

- the speed was 100% beats.

-see the code.

class Solution {

public:

int minFlips(int a, int b, int c) {

bitset<30> A(a);

bitset<30> B(b);

bitset<30> C(c);

int maximum=max(a,(max(b,c)));

int max\_count=log2(maximum)+1;

int flip=0;

for(int i=0;i<max\_count;i++){

if(C[i] && !A[i] && !B[i]){

flip++;

}

else if(!C[i]){

if(A[i]) flip++;

if(B[i]) flip++;

}

}

return flip;

}

};

[95. [**1306] Jump Game III – determine whether we can reach the index I with value 0 from start position with given rules.]**

**-** I used map to record which index I have to visit so as to visit current value. That is, map’s key is we want to visit, and map’s value is an index we need to visit the index.

- refer to the picture below.



> the red rectangular is start position and key 3 is where 0 is.

- I implemented DFS by using stack so that when we have path to 0, find the answer as soon as possible.

- see the code.

class Solution {

public:

bool canReach(vector<int>& arr, int start) {

if(arr[start]==0) return true;

unordered\_map<int,vector<int>> table;

vector<int> zeros;

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

if(arr[i]==0) zeros.push\_back(i);

if(i+arr[i]<arr.size() && arr[i]) table[i+arr[i]].push\_back(i);

if(i-arr[i]>=0 && arr[i]) table[i-arr[i]].push\_back(i);

}

stack<int> index;

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

int cur;

index.push(zeros[i]);

while(!index.empty()){

cur=index.top();

index.pop();

if(table.find(cur)==table.end()) continue;

for(int j=0;j<table[cur].size();j++){

if(table[cur][j]==start) return true;

index.push(table[cur][j]);

}

table[cur].clear();

}

}

return false;

}

};

[96 [**931] Minimum Falling Path Sum – find minimum sum of given rules]**

**-** At first, I used DFS to consider all the path possible. but time limit exceeded occurred.

- see the code.

class Solution {

public:

int helper(vector<vector<int>> A, int i,int j,int sum=0){

if(j<0 || j>=A[0].size()) return INT\_MAX;

int cur=sum+A[i][j];

if(i==A.size()-1) return cur;

return min(helper(A,i+1,j-1,cur),min(helper(A,i+1,j,cur),helper(A,i+1,j+1,cur)));

}

int minFallingPathSum(vector<vector<int>>& A) {

int minimum=INT\_MAX;

for(int i=0;i<A[0].size();i++){

minimum=min(minimum,helper(A,0,i));

}

return minimum;

}

};

- I changed my algorithm to DP.

- we don’t need to consider previous row after the minimum value of dp[i][j] is allocated.

- the speed was 99.24 % beats.

- see the code.

class Solution {

public:

int minFallingPathSum(vector<vector<int>>& A) {

vector<vector<int>> dp(A.size(),vector<int>(A[0].size(),INT\_MAX));

for(int i=0;i<dp[0].size();i++)

dp[0][i]=A[0][i];

for(int i=0;i<dp.size()-1;i++){

for(int j=0;j<dp[0].size();j++){

if(j-1>=0) dp[i+1][j-1]=min(dp[i+1][j-1],dp[i][j]+A[i+1][j-1]);

dp[i+1][j]=min(dp[i+1][j],dp[i][j]+A[i+1][j]);

if(j+1<dp[0].size()) dp[i+1][j+1]=min(dp[i+1][j+1],dp[i][j]+A[i+1][j+1]);

}

}

return \*min\_element(dp[dp.size()-1].begin(),dp[dp.size()-1].end());

}

};

[97. [**1140] Stone Game II – return the most stones alex can win]**

**-** Finally, I’ve solved this problem!!! I took almost 5 hours…

- I refered to discussion about (sum+helper())/2

- To resolve time limit exceeded, I used dp.

-see the code.

class Solution {

public:

int helper(vector<int>& piles,int x,int M,bool alex,vector<vector<int>>& dp){

if(dp[x][M]!=INT\_MIN) return dp[x][M];

if(piles.size()-x<=2\*M){

int sum=0;

for(int i=x;i<piles.size();i++)

sum+=piles[i];

return sum;

}

int res=INT\_MIN;

int cur\_sum=0;

for(int i=x;i<x+2\*M;i++){

cur\_sum+=piles[i];

res=max(res,cur\_sum-helper(piles,i+1,max(M,i-x+1),!alex,dp));

}

dp[x][M]=res;

return res;

}

int stoneGameII(vector<int>& piles) {

vector<vector<int>> dp(piles.size()+1,vector<int>(piles.size()+1,INT\_MIN));

int sum=std::accumulate(piles.begin(),piles.end(),0);

int aa=helper(piles,0,1,true,dp);

int res=(sum+aa)/2;

return res;

}

};

int stoneGameII(vector<int>& piles) {

vector<vector<int>> dp(piles.size()+1,vector<int>(piles.size()+1,INT\_MIN));

int sum=std::accumulate(piles.begin(),piles.end(),0);

int aa=helper(piles,0,1,true,dp);

int res=(sum+aa)/2;

return res;

}

};

- At first, I took brute force. But the time limit exceeded occurred.

class Solution {

public:

int helper(vector<int>& piles,int x,int M){

//if the rest stones are less then 2M, return all the rest stones

if(piles.size()-x<=2\*M){

int sum=0;

for(int i=x;i<piles.size();i++)

sum+=piles[i];

return sum;

}

//else do brute force.

int sum=INT\_MIN;// this variable will be returned.

int cur\_sum=0;

int i;

for(int m=1;m<=2\*M;m++){ //for every m where 1<=m<=2M, find all the possible case.

cur\_sum=0;

for(i=x;i<x+m;i++)

cur\_sum+=piles[i];

sum=max(sum,cur\_sum-helper(piles,i,max(M,m)));

}

return sum;

}

int stoneGameII(vector<int>& piles) {

int sum=std::accumulate(piles.begin(),piles.end(),0);

// i think I have to do brute force.

return (sum+helper(piles,0,1))/2;

}

};

- I don’t understand the algorithm. I gave up…

- they say like, dp[i][j] is the maximum stones alex can get where starting at index I with M=j.

- fxxking DP!!!

- see the code.

class Solution {

public:

int stoneGameII(vector<int>& piles) {

int length = piles.size();

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

vector<int> sufsum (length + 1, 0);

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

sufsum[i] = sufsum[i + 1] + piles[i];

}

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

dp[i][length] = sufsum[i];

}

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

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

for (int X = 1; X <= 2 \* j && i + X <= length; X++) {

dp[i][j] = max(dp[i][j], sufsum[i] - dp[i + X][max(j, X)]);

}

}

}

return dp[0][1];

}

};

[98. [Samsung SW – A : connect processor]]

- I spent almost 5 hours for debugging…

- see the code.

#include<iostream>

#include<vector>

#include<climits>

using namespace std;

vector<vector<int>> drawTable(int cur\_size, vector<pair<int, int>>& position,int& chips) {

vector<vector<int>> table(cur\_size, vector<int>(cur\_size, 0));

int c;

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

for (int j = 0; j < cur\_size;) {

cin >> c;

if (c == 0 || c == 1) {

table[i][j] = c;

if ((i == 0 || j == 0 || i == cur\_size - 1 || j == cur\_size - 1) && c==1) {

chips++;

}

else if (c == 1) {

position.push\_back(make\_pair(i, j));

}

j++;

}

}

}

return table;

}

void printTable(vector<vector<int>> table) {

int cur\_size = table.size();

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

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

cout << table[i][j];

}

cout << endl;

}

}

vector<int> DFS(vector<vector<int>> table, vector<pair<int, int>>& position, int connect,int check, int lines,int numChips,int cur\_size) {

if (check ==numChips) {

return { connect,lines };

}

bool flag = false;

vector<vector<int>> temp = table;

int x = position[check].first;

int y = position[check].second;

vector<int> res = { 0,100 };

vector<int> cur = { 0,100 };

//north

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

if (i < 0) {

flag = true;

res = DFS(temp, position, connect + 1, check+1,lines + x, numChips, cur\_size);

if (cur[0] < res[0]) {

cur = res;

}

else if (cur[0] == res[0] && cur[1] > res[1])

cur = res;

break;

}

if (temp[i][y] == 0) temp[i][y] = 1;

else break;

}

temp = table;

//south

for (int i = x + 1; i <= cur\_size; i++) {

if (i >= cur\_size) {

flag = true;

res = DFS(temp, position, connect + 1, check+1, lines + cur\_size - (x+1 ), numChips, cur\_size);

if (cur[0] < res[0]) {

cur = res;

}

else if (cur[0] == res[0] && cur[1] > res[1])

cur = res;

break;

}

if (temp[i][y] == 0) temp[i][y] = 1;

else break;

}

temp = table;

//east

for (int i = y + 1; i <= cur\_size; i++) {

if (i >= cur\_size) {

flag = true;

res = DFS(temp, position, connect + 1, check + 1, lines + cur\_size - (y+1), numChips, cur\_size);

if (cur[0] < res[0]) {

cur = res;

}

else if (cur[0] == res[0] && cur[1] > res[1])

cur = res;

break;

}

if (temp[x][i] == 0) temp[x][i] = 1;

else break;

}

temp = table;

//west

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

if (i < 0) {

flag = true;

res = DFS(temp, position, connect + 1, check + 1, lines + y, numChips, cur\_size);

if (cur[0] < res[0]) {

cur = res;

}

else if (cur[0] == res[0] && cur[1] > res[1])

cur = res;

break;

}

if (temp[x][i] == 0) temp[x][i] = 1;

else break;

}

if (!flag) {

cur = DFS(temp, position, connect, check + 1, lines, numChips, cur\_size);

}

return { cur[0],cur[1] };

}

int main(int argc, char\*\* argv) {

int test\_case;

int T;

cin >> T;

int cur\_size;

string cur\_row;

int chips=0;

vector<int> answer;

for (test\_case = 1; test\_case <= T; ++test\_case) {

cin >> cur\_size;

vector<pair<int, int>> position;

chips = 0;

vector<vector<int>> table = drawTable(cur\_size, position,chips);

//printTable(table);

answer = DFS(table, position, chips, 0,0,cur\_size-chips, cur\_size);

cout << "#" << test\_case << " " << answer[1] << endl;

}

return 0;//정상종료시 반드시 0을 리턴해야합니다.

}

[99. [**695] Max Area of Island] – return a maximum size of island]**

**-** just use DFS or BFS. Emprically, DFS is faster than BFS.

- see the code.

class Solution {

public:

int DFS(vector<vector<int>>& grid,int x, int y,int area){

if(!grid[x][y]) return 0;

grid[x][y]=0;

if(x-1>=0 && grid[x-1][y]) area+=DFS(grid,x-1,y,0);

if(y-1>=0 && grid[x][y-1]) area+=DFS(grid,x,y-1,0);

if(x+1<grid.size() && grid[x+1][y]) area+=DFS(grid,x+1,y,0);

if(y+1<grid[0].size() && grid[x][y+1]) area+=DFS(grid,x,y+1,0);

return area+1;

}

int maxAreaOfIsland(vector<vector<int>>& grid) {

int answer=0;

int cur;

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

for(int j=0;j<grid[0].size();j++){

if(grid[i][j]==1){

cur=DFS(grid,i,j,0);

answer=max(answer,cur);

}

}

}

return answer;

}

};

[100. [**46] Permutations – find all the possible permutation]**

**-** The function next\_permutation return true if there is next permutation, and revise given vector to next

permutation. but the function can’t handle negative value.

- So I used DFS. Actually, it’s not such a DFS, but I think it is :).

- Algorithm is below.

> 1. from 0 to nums.size(), input the value into new vector

> 2. remove the value, and repeat from 1.

> 3. if all the value in the given vector nums is removed, push the vector cur into answer vector

- The speed was 61.80% beats.

- see the code.

class Solution {

public:

void dfs(vector<int> nums,vector<vector<int>>& answer,vector<int> cur,int start){

nums.erase(next(nums.begin(),start));

if(nums.size()==0){

answer.push\_back(cur);

return;

}

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

vector<int> temp=cur;

temp.push\_back(nums[i]);

dfs(nums,answer,temp,i);

}

}

vector<vector<int>> permute(vector<int>& nums) {

vector<vector<int>> answer;

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

vector<int> cur;

cur.push\_back(nums[i]);

dfs(nums,answer,cur,i);

}

return answer;

}

};

[101. **[BACKJOON – SAMSUNG SW : Marbles]]**

- I spent 6 hours…

- At first, I used DFS but some test case was not solved.

> I think I solved rightly, but my print method might have been wrong…

- So I refered to a blog and used BFS

- see the code.

#include<iostream>

#include<vector>

#include<algorithm>

#include<climits>

#include<fstream>

#include<queue>

using namespace std;

vector<vector<int>> direction = { {0,-1},{0,1},{-1,0},{1,0} };//left right up dwon

int visit[10][10][10][10];

vector<vector<char>> makeTable2(vector<int>& position) {

std::ifstream in("C:\\Users\\gjsgu\\Desktop\\Algorithm\\test\_case.txt");

string row;

int N;

int M;

if (in.is\_open()) {

in >> N;

in >> M;

}

vector<vector<char>> table(N+2, vector<char>(M+2, '#'));

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

in >> row;

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

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

position[0] = i;

position[1] = j;

}

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

position[2] = i;

position[3] = j;

}

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

position[4] = i;

position[5] = j;

}

table[i][j] = row[j];

}

}

return table;

}

vector<vector<char>> makeTable(vector<int>& position) {

string row;

int N;

int M;

cin >> N;

cin >> M;

vector<vector<char>> table(N+2, vector<char>(M+2, '#'));

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

cin >> row;

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

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

position[0] = i;

position[1] = j;

}

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

position[2] = i;

position[3] = j;

}

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

position[4] = i;

position[5] = j;

}

table[i][j] = row[j];

}

}

return table;

}

void printTable(vector<vector<char>>& table) {

cout << endl;

for (vector<char> row : table) {

for (char c : row) {

cout << c;

}

cout << endl;

}

}

void moveTo(vector<vector<char>>& table,int& x, int& y, int k) {

while (true) {

x += direction[k][0]; y += direction[k][1];

//cout << " x : " << x << " y : " << y << endl;

if (table[x][y] == '#') {

x -= direction[k][0]; y -= direction[k][1];

break;

}

else if (table[x][y] == 'O') break;

}

}

int BFS(vector<vector<char>>& table,int a,int b,int c, int d,int e,int f) {

int count=0;

queue<vector<int>> que;

que.push({ a,b,c,d,0 });

visit[a][b][c][d] = 1;

while (!que.empty()) {

vector<int> cur = que.front(); que.pop();

a = cur[0]; b = cur[1]; c = cur[2]; d = cur[3]; count = cur[4];

//cout << a << " " << b << " " << c << " " << d << endl;

if (count > 10) break; // when the depth is over 10

if (a == e && b == f) return count; // when red ball is placed at th hole.

//ball move

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

int rx = a; int ry = b; int bx = c; int by = d;

moveTo(table,rx, ry, i);

moveTo(table,bx, by, i);

if (bx == e && by == f) continue;

if (rx == bx && ry == by) { //if they are overlapped.

switch (i){

case 0: //left

b < d ? by++ : ry++;

break;

case 1: //right

d < b ? by-- : ry--;

break;

case 2: //up

a < c ? bx++ : rx++;

break;

case 3: //down

c < a ? bx-- : rx--;

break;

}

}

if (!visit[rx][ry][bx][by]) {

//cout << "input que : " << rx << " "<< ry << " " << bx << " " << by << endl;

que.push({ rx,ry,bx,by,count + 1 });

visit[rx][ry][bx][by] = 1;

}

}

}

return -1;

}

int main() {

int answer=-1;

vector<int> position(6,0);

vector<vector<char>> table = makeTable(position);

//vector<vector<char>> table = makeTable2(position);

visit[position[0]][position[1]][position[2]][position[3]] = 1;

//printTable(table);

answer = BFS(table, position[0], position[1], position[2], position[3],position[4],position[5]);

printf("%d", answer);

return 0;

}

[102. **[BACKJOON – SAMSUNG SW : 2048]]**

**-** I used BFS, but BFS is not the matter. move function was the main point.

- I made each direction’s function respectively. so I solved faster than before.

- it took 1 hour and half.

- see the code.

#include<iostream>

#include<vector>

#include<algorithm>

#include<fstream>

#include<queue>

#include<climits>

using namespace std;

vector<vector<int>> makeTable() {

std::ifstream in("C:\\Users\\gjsgu\\Desktop\\Algorithm\\test\_case\\test\_case\_for\_2048.txt");

int N;

int c;

if (in.is\_open()) {

cout << "test\_case is opened." << endl;

in >> N;

}

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

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

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

in >> c;

table[i][j++] = c;

}

}

return table;

}

vector<vector<int>> makeTable2() {

int N;

int c;

cin >> N;

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

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

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

cin >> c;

table[i][j++] = c;

}

}

return table;

}

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

for (vector<int> row : table) {

for (int i : row) {

cout << i << " ";

}

cout << endl;

}

}

int findMax(vector<vector<int>>& table) {

int maximum = 0;

for (vector<int> row : table)

maximum = max(maximum, \*max\_element(row.begin(), row.end()));

return maximum;

}

/\*

3

2 2 2

4 4 4

8 8 8

\*/

// these functions are the main point of this problem.

// move left and right start from column but move up and down start from row.

vector<vector<int>> move\_left(vector<vector<int>> table) {

int cur\_val;

vector<vector<int>> visit(table.size(), vector<int>(table.size(), 0));

for (int col = 1; col < table[0].size(); col++) {

for (int row = 0; row < table.size(); row++) {

cur\_val = table[row][col];

if (cur\_val == 0) continue;

int j = col-1;

while (table[row][j] == 0 && j>0) j--; // finding a cell is not value 0.

table[row][col] = 0;

//if they have a same value.

if (table[row][j] == cur\_val && visit[row][j]==0) {

table[row][j] \*= 2;

visit[row][j] = 1;

}

//if there is no value to be concerned and the cell is not visited yet.

else if (j==0 && table[row][j]==0 && visit[row][j]==0) table[row][0] = cur\_val;

//if they have different values or the cell is already visited.

else if (table[row][j] != cur\_val || visit[row][j]) table[row][j + 1] = cur\_val;

}

}

return table;

}

vector<vector<int>> move\_right(vector<vector<int>> table) {

int cur\_val;

vector<vector<int>> visit(table.size(), vector<int>(table.size(), 0));

for (int col = table[0].size()-2; col>=0; col--) {

for (int row = 0; row < table.size(); row++) {

cur\_val = table[row][col];

if (cur\_val == 0) continue;

//cout << " row : " << row << " col : " << col << " cur\_val : " << cur\_val << endl;

int j = col + 1;

while (table[row][j] == 0 && j < table[0].size()-1) j++; // finding a cell is not value 0.

table[row][col] = 0;

//if they have a same value.

if (table[row][j] == cur\_val && visit[row][j] == 0) {

table[row][j] \*= 2;

visit[row][j] = 1;

}

//if there is no value to be concerned and the cell is not visited yet.

else if (j == table[0].size()-1 && table[row][j] == 0 && visit[row][j] == 0) table[row][table[0].size()-1] = cur\_val;

//if they have different values or the cell is already visited.

else if (table[row][j] != cur\_val || visit[row][j]) table[row][j - 1] = cur\_val;

}

}

return table;

}

vector<vector<int>> move\_up(vector<vector<int>> table) {

int cur\_val;

vector<vector<int>> visit(table.size(), vector<int>(table.size(), 0));

for (int row = 1; row<table.size(); row++) {

for (int col=0; col < table[0].size(); col++) {

cur\_val = table[row][col];

if (cur\_val == 0) continue;

int i = row - 1;

while (table[i][col] == 0 && i >0) i--; // finding a cell is not value 0.

table[row][col] = 0;

//if they have a same value.

if (table[i][col] == cur\_val && visit[i][col] == 0) {

table[i][col] \*= 2;

visit[i][col] = 1;

}

//if there is no value to be concerned and the cell is not visited yet.

else if (i == 0 && table[i][col] == 0 && visit[i][col] == 0) table[0][col] = cur\_val;

//if they have different values or the cell is already visited.

else if (table[i][col] != cur\_val || visit[i][col]) table[i+1][col] = cur\_val;

}

}

return table;

}

vector<vector<int>> move\_down(vector<vector<int>> table) {

int cur\_val;

vector<vector<int>> visit(table.size(), vector<int>(table.size(), 0));

for (int row = table.size()-2; row>=0; row--) {

for (int col = 0; col < table[0].size(); col++) {

cur\_val = table[row][col];

if (cur\_val == 0) continue;

//cout << " row : " << row << " col : " << col << " cur\_val : " << cur\_val << endl;

int i = row + 1;

while (table[i][col] == 0 && i < table.size()-1) i++; // finding a cell is not value 0.

table[row][col] = 0;

//if they have a same value.

if (table[i][col] == cur\_val && visit[i][col] == 0) {

table[i][col] \*= 2;

visit[i][col] = 1;

}

//if there is no value to be concerned and the cell is not visited yet.

else if (i == table.size()-1 && table[i][col] == 0 && visit[i][col] == 0) table[table.size()-1][col] = cur\_val;

//if they have different values or the cell is already visited.

else if (table[i][col] != cur\_val || visit[i][col]) table[i - 1][col] = cur\_val;

}

}

return table;

}

int BFS(vector<vector<int>> table) {

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

que.push(make\_pair(table, 0));

vector<vector<int>> cur\_table;

int depth;

int maximum = 0;

while (!que.empty()) {

cur\_table = que.front().first;

depth = que.front().second;

que.pop();

//printTable(cur\_table);

//cout <<"depth : "<<depth<<endl;

//if the depth is 5, finding the maximum value of it.

if (depth == 5) {

maximum = max(maximum, findMax(cur\_table));

continue;

}

//for 4 directions, moving the current table.

que.push(make\_pair(move\_left(cur\_table), depth + 1));

que.push(make\_pair(move\_right(cur\_table), depth + 1));

que.push(make\_pair(move\_up(cur\_table), depth + 1));

que.push(make\_pair(move\_down(cur\_table), depth + 1));

}

return maximum;

}

int main() {

//vector<vector<int>> table = makeTable();

vector<vector<int>> table = makeTable2();

//printTable(table);

// cout << endl;

int answer=BFS(table);

cout << answer;

return 0;

}

[103. **[BACKJOON – SAMSUNG SW : Snake]]**

**-** I confused due to apple’s corordinate. they gave the corordiantes strating from row 1 and column 1 not 0 and 0.

- to follow tail, I used queue.

- I spent 1 hour and 10~20 minutes.

- see the code.

#include<iostream>

#include<fstream>

#include<vector>

#include<queue>

using namespace std;

pair<queue<pair<int,char>>,vector<vector<int>>> makeTable() {

ifstream in("C:\\Users\\gjsgu\\Desktop\\Algorithm\\test\_case\\test\_case\_for\_snake.txt");

int N; int K;

if (in.is\_open()) {

cout << "file is opened." << endl;

in >> N; in >> K;

}

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

table[0][0] = 2; // 2 is snake's body.

int row, col;

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

in >> row; in >> col;

table[row-1][col-1] = 1; // 1 is apple.

}

int L; // the number of direction change.

in >> L;

int sec; char dir;

queue<pair<int, char>> direction;

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

in >> sec; in >> dir;

direction.push({ sec,dir});

}

return make\_pair(direction, table);

}

pair<queue<pair<int, char>>, vector<vector<int>>> makeTable2() {

int N; int K;

cin >> N; cin >> K;

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

table[0][0] = 2; // 2 is snake's body.

int row, col;

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

cin >> row; cin >> col;

table[row-1][col-1] = 1; // 1 is apple.

}

int L; // the number of direction change.

cin >> L;

int sec; char dir;

queue<pair<int, char>> direction;

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

cin >> sec; cin >> dir;

direction.push({ sec,dir });

}

return make\_pair(direction, table);

}

void printTable(vector<vector<int>> table) {

for (vector<int> row : table) {

for (int i : row) {

cout << i << " ";

}

cout << endl;

}

}

int helper(vector<vector<int>>& table, queue<pair<int,char>>& direction) {

int tx = 0, ty = 0;

int hx = 0, hy = 0;

int second = 0;

vector<vector<int>> dir = { {1,0},{0,-1},{-1,0},{0,1} }; // down left up right

queue<pair<int, int>> tail;

tail.push(make\_pair(0, 0));

int cur\_dir = 3; //start direction

direction.push(make\_pair(1000, 'L')); //At last time, snake has to move to a wall.

while (!direction.empty()) {

int cur\_sec = direction.front().first;

int continue\_sec = cur\_sec - second;

char next\_dir = direction.front().second;

direction.pop();

for (int i = 0; i < continue\_sec; i++) {

second++; // time plused.

hx += dir[cur\_dir][0]; hy += dir[cur\_dir][1];

tail.push(make\_pair(hx, hy));

// if head meets body or wall.

if (hx < 0 || hx >= table.size() || hy < 0 || hy >= table[0].size()||table[hx][hy] == 2 ) return second;

// if head meets an apple.

else if (table[hx][hy] == 1) table[hx][hy] = 2; // notice that even though all the apple is eaten, the game won't be ended.

// if head meets nothing.

else {//tail has to be moved 1 cell.

tx = tail.front().first; ty = tail.front().second;

tail.pop();

table[tx][ty] = 0;

table[hx][hy] = 2;

}

//cout << "hx : " << hx << " hy : " << hy << " tx : " << tx << " ty : " << ty <<" second : "<< second<< endl;

//printTable(table);

}

if (next\_dir == 'L') { // turn left

cur\_dir--;

cur\_dir == -1 ? cur\_dir = 3 : cur\_dir;

}

else cur\_dir = (cur\_dir + 1) % 4; //turn right

}

return second;

}

int main() {

//pair<queue<pair<int, char>>, vector<vector <int >>> input = makeTable();

pair<queue<pair<int, char>>, vector<vector <int >>> input = makeTable2();

vector<vector<int>> table = input.second;

queue<pair<int, char>> direction = input.first;

int answer= helper(table, direction);

cout << answer;

return 0;

}

[104. **[BACKJOON – SAMSUNG SW : Exam Invigilator]]**

**-** this exam’s trap is the range of number. since tue maximum number of people is 1,000,000 \* 1,000,000 and the minimum number of that an invigilator can handle is 1.

- so we have to use long long (int) for the answer and double for the table having the number of people in a class.

- see the code.

#include<iostream>

#include<fstream>

#include<vector>

#include<cmath> // for the function ceil

using namespace std;

pair<vector<double>,int> makeTable() {

std::ifstream in("C:\\Users\\gjsgu\\Desktop\\Algorithm\\test\_case\\test\_case\_for\_exam\_invigilator.txt");

int N;

if (in.is\_open()) {

cout << "file is opened." << endl;

in >> N;

}

vector<double> table(N, 0);

int people;

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

in >> people;

table[i] = people;

}

int B, C;

in >> B; in >> C;

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

return make\_pair(table, C);

}

pair<vector<double>, int> makeTable2() {

int N;

cin >> N;

vector<double> table(N, 0);

int people;

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

cin >> people;

table[i] = people;

}

int B, C;

cin >> B; cin >> C;

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

return make\_pair(table, C);

}

void printTable(vector<double> table) {

for (double i : table)

cout << i << " ";

cout << endl;

}

unsigned long long findMinimum(vector<double> table, int C) {

unsigned long long minimum = table.size();

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

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

minimum += ceil(table[i] / C);

}

return minimum;

}

int main() {

//pair<vector<float>,int> input = makeTable();

pair<vector<double>, int> input = makeTable2();

vector<double> table = input.first;

int C = input.second;

//printTable(table);

unsigned long long answer=findMinimum(table, C); //since 1,000,000 \* 1,000,000 is the maximum number of student and the minimum number of that an invigiltor can handle is 1.

cout << answer;

return 0;

}

[105. **[BACKJOON – SAMSUNG SW : Rolling Dice]]**

- I spent less than 1 hour. it was not that hard to solve.

- see the code.

#include<iostream>

#include<fstream>

#include<vector>

using namespace std;

pair<vector<vector<int>>,vector<int>> makeTable(int& x, int& y) {

ifstream in("C:\\Users\\gjsgu\\Desktop\\Algorithm\\test\_case\\test\_case\_for\_rolling\_dice.txt");

int N, M, K;

if (in.is\_open()) {

cout << "file is opened." << endl;

in >> N; in >> M; in >> x; in >> y; in >> K;

}

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

vector<int> order(K, 0);

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

for (int j = 0; j < M; j++) in >> table[i][j];

for (int i = 0; i < K; i++) in >> order[i];

return make\_pair(table, order);

}

pair<vector<vector<int>>, vector<int>> makeTable2(int& x, int& y) {

int N, M, K;

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

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

vector<int> order(K, 0);

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

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

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

return make\_pair(table, order);

}

void printTable(vector<vector<int>> table) {

for (vector<int> row : table) {

for (int i : row)

cout << i << " ";

cout << endl;

}

}

bool moveTheDice(int a, int b,int& x, int& y,int size\_x, int size\_y){

if (x + a < 0 || x + a >= size\_x || y + b < 0 || y + b >= size\_y) return false;

x = x + a; y = y + b;

return true;

}

void rollTheDice(vector<vector<int>>& dice,int dir) {

int temp;

switch (dir) {

case 1: // east

temp = dice[1][2]; dice[1][2] = dice[1][1]; dice[1][1] = dice[1][0]; dice[1][0] = dice[3][1]; dice[3][1] = temp;

break;

case 2: // west

temp = dice[1][0]; dice[1][0] = dice[1][1]; dice[1][1] = dice[1][2]; dice[1][2] = dice[3][1]; dice[3][1] = temp;

break;

case 3: // 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 4: // 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;

}

}

void paintTheDice(vector<vector<int>>& table, vector<vector<int>>& dice,int x, int y) {

if (!table[x][y]) {

table[x][y] = dice[3][1]; // dice[3][1] is bottom.

return;

}

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

table[x][y] = 0;

return;

}

void followingOrder(vector<vector<int>>& table, vector<int> order, vector<vector<int>> dice,int& x,int& y) {

int dir[4][2] = { {0,1},{0,-1},{-1,0},{1,0} }; //east west north south

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

if (!moveTheDice(dir[order[i] - 1][0], dir[order[i] - 1][1], x, y, table.size(), table[0].size())) continue;

//cout << " x :" << x << " y : " << y << endl;

rollTheDice(dice,order[i]);

paintTheDice(table,dice,x,y);

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

}

}

int main() {

int x, y;

//pair<vector<vector<int>>,vector<int>> input = makeTable(x,y);

pair<vector<vector<int>>, vector<int>> input = makeTable2(x,y);

vector<vector<int>> table = input.first;

vector<int> order = input.second;

vector<vector<int>> dice(4, vector < int>(3,0));

followingOrder(table,order,dice,x,y);

return 0;

}

[106. **[BACKJOON – SAMSUNG SW : Tetromino]]**

**-** I struggled to implement slinding. I spent 1 hour or so.

- it was a quite dirty problem, since I had to make all the possible tetromino with my own hands.

- see the code.

#include<iostream>

#include<fstream>

#include<vector>

#include<climits>

#include<queue>

using namespace std;

vector<vector<int>> makeTable() {

ifstream in("C:\\Users\\gjsgu\\Desktop\\Algorithm\\test\_case\\test\_case\_for\_tetromino.txt");

int N, M;

if (in.is\_open()) {

cout << "file is opened." << endl;

in >> N; in >> M;

}

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

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

for (int j = 0; j < M; j++) in >> table[i][j];

return table;

}

vector<vector<int>> makeTable2() {

int N, M;

cin >> N; cin >> 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];

return table;

}

queue<vector<vector<int>>> makeTetromino() {

queue<vector<vector<int>>> que;

que.push(vector<vector<int>>{{1,1,1,1}});

que.push(vector<vector<int>>{ {1}, { 1 }, { 1 }, {1}});

que.push(vector<vector<int>>{ {1, 1}, { 1,1 }});

que.push(vector<vector<int>>{ {1,0}, { 1,0 }, { 1 ,1}});

que.push(vector<vector<int>>{ {0, 1}, { 0,1 }, {1,1}});

que.push(vector<vector<int>>{ {1, 1}, { 1,0 }, { 1,0 }});

que.push(vector<vector<int>>{ {1, 1}, { 0,1 }, {0,1}});

que.push(vector<vector<int>>{ {1, 1, 1}, { 0,0,1 }});

que.push(vector<vector<int>>{ {1, 1, 1}, {1,0,0}});

que.push(vector<vector<int>>{ {1, 0, 0}, {1,1,1}});

que.push(vector<vector<int>>{ {0, 0, 1}, {1,1,1}});

que.push(vector<vector<int>>{ {1, 0}, { 1,1 }, {0,1}});

que.push(vector<vector<int>>{ {0, 1}, { 1,1 }, {1,0}});

que.push(vector<vector<int>>{ {1, 1, 0}, {0,1,1}});

que.push(vector<vector<int>>{ {0, 1, 1}, {1,1,0}});

que.push(vector<vector<int>>{ {0, 1, 0}, {1,1,1}});

que.push(vector<vector<int>>{ {1, 1, 1}, {0,1,0}});

que.push(vector<vector<int>>{ {0, 1}, { 1,1 }, {0,1}});

que.push(vector<vector<int>>{ {1, 0}, { 1,1 }, {1,0}});

return que;

}

// this function is the key of this problem.

int slideTetro(vector<vector<int>>& table, vector<vector<int>>& tetro) {

int sum = 0;

int maximum = 0;

// we have to get a window.

// window size is tetro.size() X tetro[0].size()

vector<vector<int>> window;

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

for (int j = 0; j+tetro[0].size()<=table[0].size(); j++) {

sum = 0;

//from here, the indice are for window.

for (int k = i; k < i + tetro.size(); k++) {

for (int l = j; l < j + tetro[0].size(); l++) {

if (tetro[k-i][l-j]) sum += table[k][l];

}

}

maximum = max(maximum, sum);

}

}

return maximum;

}

int findMaximum(vector<vector<int>>& table,queue<vector<vector<int>>>& que) {

int maximum = 0;

while (!que.empty()) {

vector<vector<int>> tetro = que.front();

que.pop();

maximum = max(maximum,slideTetro(table,tetro));

}

return maximum;

}

void printTable(vector<vector<int>> table) {

for (vector<int> row : table) {

for (int i : row) {

cout << i << " ";

}

cout << endl;

}

}

int main() {

//vector<vector<int>> table = makeTable();

vector<vector<int>> table = makeTable2();

queue < vector<vector<int>>> que = makeTetromino();

cout<<findMaximum(table, que);

return 0;

}

[107. **[BACKJOON – SAMSUNG SW : Quit The Job]]**

- it was dp problem. I made dp from the last day.

- dp[i] has the maximum money we can earn after ith day.

- the point is that where dp[i], we don’t need to choose ith day if it couldn’t make maximum money.

- dp has table.size()+1 memory, since I started table.size()-1 index and it’s possible we can’t consult at the last day.

- see the code.

#include<iostream>

#include<fstream>

#include<vector>

#include<algorithm>

using namespace std;

vector<vector<int>> makeTable() {

std::ifstream in("C:\\Users\\gjsgu\\Desktop\\Algorithm\\test\_case\\test\_case\_for\_quit\_the\_job.txt");

int N;

if (in.is\_open()) {

cout << "file is opened." << endl;

in >> N;

}

vector<vector<int>> table(N, vector<int>(2, 0));

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

in >> table[i][0]; in >> table[i][1];

}

return table;

}

vector<vector<int>> makeTable2() {

int N;

cin >> N;

vector<vector<int>> table(N, vector<int>(2, 0));

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

cin >> table[i][0]; cin >> table[i][1];

}

return table;

}

void printTable(vector<vector<int>> table) {

for (vector<int> row : table) {

for (int i : row) {

cout << i << " ";

}

cout << endl;

}

}

int findMaximum(vector<vector<int>>& table) {

int last\_day = table.size();

int cur\_max = 0;

vector<int> dp(last\_day+1, 0); // dp is the maximum money we can earn after ith day. whether choosing ith day or not doesn't matter.

for (int i = last\_day - 1; i >= 0; i--) {

// if a consulting continues over the last day, just take dp[i+1]

if (table[i][0] - 1 + i >= last\_day) {

dp[i] = dp[i + 1];

continue;

}

// if not, take maximum between current day's money + dp of next possible day and just dp[i+1]

dp[i] = max(table[i][1] + dp[i + table[i][0]], dp[i+1]);

//cout << "dp[" << i << "] : " << dp[i] << endl;

}

return dp[0];

}

int main() {

//vector<vector<int>> table=makeTable();

vector<vector<int>> table = makeTable2();

cout << findMaximum(table);

return 0;

}

[108. **[BACKJOON – SAMSUNG SW : LAB]]**

- At first, I tried brute force. But it took so much time so I sholud’ve changed the algorithm

- the brute force algorithm is below.

#include<iostream>

#include<fstream>

#include<vector>

#include<cmath>

#include<queue>

using namespace std;

pair<vector<vector<int>>,queue<vector<int>>> makeTable() {

ifstream in("C:\\Users\\gjsgu\\Desktop\\Algorithm\\test\_case\\test\_case\_for\_LAB.txt");

int N, M;

if (in.is\_open()) {

cout << "file is opened." << endl;

in >> N; in >> M;

}

int temp;

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

queue<vector<int>> que;

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

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

in >> table[i][j];

if (table[i][j] == 2) que.push(vector<int>{ {i, j}});

}

}

return make\_pair(table,que);

}

pair<vector<vector<int>>, queue<vector<int>>> makeTable2() {

int N, M;

cin >> N; cin >> M;

int temp;

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

queue<vector<int>> que;

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

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

cin >> table[i][j];

if (table[i][j] == 2) que.push(vector<int>{ {i, j}});

}

}

return make\_pair(table, que);

}

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

for (vector<int> row : table) {

for (int i : row) {

cout << i << " ";

}

cout << endl;

}

}

void BFS(vector<vector<int>>& table,vector<int> virus) {

queue<vector<int>> que;

que.push(virus);

while (!que.empty()) {

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

que.pop();

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

table[x - 1][y] = 2;

que.push(vector<int>{x - 1, y});

}

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

table[x][y - 1] = 2;

que.push(vector<int>{ x,y - 1 });

}

if (x + 1 < table.size() && table[x + 1][y] == 0) {

table[x + 1][y] = 2;

que.push(vector<int>{x + 1, y});

}

if (y + 1 < table.size() && table[x][y + 1] == 0) {

table[x][y + 1] = 2;

que.push(vector<int>{x, y + 1});

}

}

}

int safetyArea(vector<vector<int>> table,queue<vector<int>> que) {

while (!que.empty()) {

BFS(table, que.front());

que.pop();

}

int sum = 0;

for (vector<int> row : table) {

for (int i : row) {

i == 0 ? sum++ : 0;

}

}

return sum;

}

int DFS(vector<vector<int>> table,queue<vector<int>> que,int depth) {

if (depth == 3) {

return safetyArea(table,que);

}

int maximum = 0;

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

for (int j = 0; j < table[0].size(); j++) {

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

table[i][j] = 1;

maximum=max(maximum,DFS(table, que, depth + 1));

table[i][j] = 0;

}

}

return maximum;

}

int main() {

pair<vector<vector<int>>,queue<vector<int>>> input = makeTable();

//pair<vector<vector<int>>, queue<vector<int>>> input = makeTable2();

cout << DFS(input.first,input.second,0);

return 0;

}

- Now, I have to find another solution.

- But most people solve this problem as brute force. so I changed DFS to BFS.

- I have no idea why it’s wrong! at 11%, failure occurs.

- see the code.

#include<iostream>

#include<fstream>

#include<vector>

#include<cmath>

#include<queue>

using namespace std;

pair<vector<vector<int>>,queue<vector<int>>> makeTable(vector<vector<int>>& empty) {

ifstream in("C:\\Users\\gjsgu\\Desktop\\Algorithm\\test\_case\\test\_case\_for\_LAB.txt");

int N, M;

if (in.is\_open()) {

cout << "file is opened." << endl;

in >> N; in >> M;

}

int temp;

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

queue<vector<int>> que;

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

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

in >> table[i][j];

if (table[i][j] == 2) que.push(vector<int>{ {i, j}});

else if (table[i][j] == 0) empty.push\_back({i,j});

}

}

return make\_pair(table,que);

}

pair<vector<vector<int>>, queue<vector<int>>> makeTable2(vector<vector<int>> & empty) {

int N, M;

cin >> N; cin >> M;

int temp;

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

queue<vector<int>> que;

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

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

cin >> table[i][j];

if (table[i][j] == 2) que.push(vector<int>{ {i, j}});

else if (table[i][j] == 0) empty.push\_back({ i,j });

}

}

return make\_pair(table, que);

}

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

for (vector<int> row : table) {

for (int i : row) {

cout << i << " ";

}

cout << endl;

}

}

int safetyArea(vector<vector<int>> & table) {

int sum = 0;

for (vector<int> row : table) {

for (int i : row) {

if (i == 0) sum++;

}

}

return sum;

}

int contagion(vector<vector<int>> table,queue<vector<int>> que) {

while (!que.empty()) {

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

que.pop();

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

table[x - 1][y] = 2;

que.push(vector<int>{x - 1, y});

}

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

table[x][y - 1] = 2;

que.push(vector<int>{ x,y - 1 });

}

if (x + 1 < table.size() && table[x + 1][y] == 0) {

table[x + 1][y] = 2;

que.push(vector<int>{x + 1, y});

}

if (y + 1 < table.size() && table[x][y + 1] == 0) {

table[x][y + 1] = 2;

que.push(vector<int>{x, y + 1});

}

}

return safetyArea(table);

}

int BFS(vector<vector<int>> table, queue<vector<int>> que, vector<vector<int>> empty) {

int maximum = 0;

int depth = 0;

for (int i = 0; i < empty.size()-2; i++) {

table[empty[i][0]][empty[i][1]] = 3;

for (int j = i+1; j < empty.size()-1; j++) {

table[empty[j][0]][empty[j][1]] = 3;

for (int k = j+1; k < empty.size(); k++) {

table[empty[k][0]][empty[k][1]] = 3;

maximum=max(maximum,contagion(table, que));

table[empty[k][0]][empty[k][1]] = 0;

}

table[empty[j][0]][empty[j][1]] = 0;

}

table[empty[i][0]][empty[i][1]] = 0;

}

return maximum;

}

int main() {

vector<vector<int>> empty;

pair<vector<vector<int>>,queue<vector<int>>> input = makeTable(empty);

//pair<vector<vector<int>>, queue<vector<int>>> input = makeTable2(empty);

//printTable(input.first);

cout << BFS(input.first, input.second, empty);

return 0;

}

- Finally, I found the error.

- when I compare the range in the function contagion. I compare y index with table.size() not table[0].size().

- due to [0], I took almost 3 hours. what the fxxk!

- see the final code.

#include<iostream>

#include<fstream>

#include<vector>

#include<cmath>

#include<queue>

using namespace std;

pair<vector<vector<int>>, vector<vector<int>>> makeTable(vector<vector<int>>& empty, queue<vector<int>>& que) {

ifstream in("C:\\Users\\gjsgu\\Desktop\\Algorithm\\test\_case\\test\_case\_for\_LAB.txt");

int N, M;

if (in.is\_open()) {

cout << "file is opened." << endl;

in >> N; in >> M;

}

int temp;

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

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

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

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

in >> table[i][j];

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

que.push(vector<int>{ {i, j}});

//visit[i][j] = 1;

}

else if (table[i][j] == 0) empty.push\_back({i,j});

}

}

return make\_pair(table, visit);

}

pair<vector<vector<int>>, vector<vector<int>>> makeTable2(vector<vector<int>> & empty, queue<vector<int>>& que) {

int N, M;

cin >> N; cin >> M;

int temp;

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

vector<vector<int>> visit(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] == 2) {

que.push(vector<int>{ {i, j}});

//visit[i][j] = 1;

}

else if (table[i][j] == 0) empty.push\_back({ i,j });

}

}

return make\_pair(table, visit);

}

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

for (vector<int> row : table) {

for (int i : row) {

cout << i << " ";

}

cout << endl;

}

}

int safetyArea(vector<vector<int>> & table) {

int sum = 0;

for (vector<int> row : table) {

for (int i : row) {

if (i == 0) sum++;

}

}

return sum;

}

int contagion(vector<vector<int>> table,queue<vector<int>> que, vector<vector<int>> visit) {

while (!que.empty()) {

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

que.pop();

if (visit[x][y] == 1) continue;

visit[x][y] = 1;

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

table[x - 1][y] = 2;

que.push(vector<int>{x - 1, y});

}

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

table[x][y - 1] = 2;

que.push(vector<int>{ x,y - 1 });

}

if (x + 1 < table.size() && table[x + 1][y] == 0) {

table[x + 1][y] = 2;

que.push(vector<int>{x + 1, y});

}

if (y + 1 < table[0].size() && table[x][y + 1] == 0) {

table[x][y + 1] = 2;

que.push(vector<int>{x, y + 1});

}

}

return safetyArea(table);

}

int BFS(vector<vector<int>> table, queue<vector<int>> que, vector<vector<int>> empty,vector<vector<int>> visit) {

int maximum = 0;

int depth = 0;

for (int i = 0; i < empty.size()-2; i++) {

table[empty[i][0]][empty[i][1]] = 3;

for (int j = i+1; j < empty.size()-1; j++) {

table[empty[j][0]][empty[j][1]] = 3;

for (int k = j+1; k < empty.size(); k++) {

table[empty[k][0]][empty[k][1]] = 3;

maximum=max(maximum,contagion(table, que,visit));

table[empty[k][0]][empty[k][1]] = 0;

}

table[empty[j][0]][empty[j][1]] = 0;

}

table[empty[i][0]][empty[i][1]] = 0;

}

return maximum;

}

int main() {

vector<vector<int>> empty;

queue<vector<int>> que;

//pair<vector<vector<int>>,vector<vector<int>>> input = makeTable(empty,que);

pair<vector<vector<int>>, vector<vector<int>>> input = makeTable2(empty,que);

//printTable(input.first);

cout << BFS(input.first, que, empty,input.second);

return 0;

}

[109. **[BACKJOON – SAMSUNG SW : Robot Vacuum Cleaner]]**

- this problem’s hardest point was to determine when cleaning is end.

- without it, not that hard, just followed given rules.

- see the code.

#include<iostream>

#include<fstream>

#include<vector>

using namespace std;

pair<vector<vector<int>>,vector<int>> makeTable() {

ifstream in("C:\\Users\\gjsgu\\Desktop\\Algorithm\\test\_case\\test\_case\_for\_robot\_vacuum\_cleaner.txt");

int N, M;

if (in.is\_open()) {

cout << "file is opened." << endl;

in >> N; in >> M;

}

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

vector<int> pos(3, 0);

for (int i = 0; i < 3; i++) in >> pos[i];

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

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

in >> table[i][j];

}

}

return make\_pair(table,pos);

}

pair<vector<vector<int>>, vector<int>> makeTable2() {

int N, M;

cin >> N; cin >> M;

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

vector<int> pos(3, 0);

for (int i = 0; i < 3; i++) cin >> pos[i];

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

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

cin >> table[i][j];

}

}

return make\_pair(table, pos);

}

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

for (vector<int> row : table) {

for (int i : row) {

cout << i << " ";

}

cout << endl;

}

}

int is\_end(vector<vector<int>>& table,vector<int>& pos) {

int x = pos[0]; int y = pos[1];

vector<vector<int>> dir = { {1,0},{0,-1},{-1,0},{0,1} }; //reverse of north east south west

if ((x - 1 < 0 || table[x - 1][y])&& (y - 1 < 0 || table[x][y - 1])&& (x + 1 >= table.size() || table[x + 1][y])&& (y + 1 >= table[0].size() || table[x][y + 1])) {

int cx = x + dir[pos[2]][0]; int cy = y + dir[pos[2]][1];

if ((cx<0 || cx >= table.size() || cy < 0 || cy >= table[0].size() || table[cx][cy]==1)) return 1;

return 2;

}

return 3;

}

int cleaning(vector<vector<int>>& table,vector<int>& pos,int count) {

vector<vector<int>> dir = { {-1,0},{0,1},{1,0},{0,-1} }; // north east south west

vector<vector<int>> back= { {1,0},{0,-1},{-1,0},{0,1} }; // reverse of north east south west

int x = pos[0]; int y = pos[1]; int head = pos[2];

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

count++;

table[x][y] = 2;

}

int end = is\_end(table, pos);

if (end == 1) return count; //when cleaning is end.

else if (end == 2) { //when vacuum moves backward.

pos[0] = x + back[pos[2]][0]; pos[1] = y + back[pos[2]][1];

return cleaning(table, pos, count);

}

int left=head;

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

left - 1 < 0 ? left = 3 : left--;

int cx = x + dir[left][0]; int cy = y + dir[left][1];

if (cx>=0 && cx<table.size() && cy>=0 && cy<table[0].size() && table[cx][cy]==0) {

pos[0] = cx; pos[1] = cy; pos[2] = left;

return cleaning(table, pos, count);

}

}

return count;

}

int main() {

//pair<vector<vector<int>>,vector<int>> input = makeTable();

pair<vector<vector<int>>, vector<int>> input = makeTable2();

cout << cleaning(input.first, input.second,0);

return 0;

}

[110. **[BACKJOON – SAMSUNG SW : Insert Operator]]**

- by using next\_permutation, we can solve this problem very easily.

- but there was a trap, the function next\_permutation ant in ascending order.

- so we have to insert our operators in following order \*,+,-,/.

- cf.) ascii number of each operator is that \* : 42, + : 43, - : 45, / : 47.

- see the code.

#include<iostream>

#include<fstream>

#include<vector>

#include<algorithm>

#include<climits>

using namespace std;

pair<vector<int>, vector<int>> makeTable() {

ifstream in("C:\\Users\\gjsgu\\Desktop\\Algorithm\\test\_case\\test\_case\_for\_insert\_operator.txt");

int N;

if (in.is\_open()) {

cout << "file is opened." << endl;

in >> N;

}

vector<int> table(N, 0);

vector<int> oper(4, 0);

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

in >> table[i];

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

in >> oper[i];

return make\_pair(table, oper);

}

pair<vector<int>, vector<int>> makeTable2() {

int N;

cin >> N;

vector<int> table(N, 0);

vector<int> oper(4, 0);

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

cin >> table[i];

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

cin >> oper[i];

return make\_pair(table, oper);

}

int calculate(vector<int>& table, vector<char> oper) {

int result = table[0];

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

if (oper[i] == '+') {

result += table[i + 1];

}

else if (oper[i] == '-') {

result -= table[i + 1];

}

else if (oper[i] == '\*') {

result \*= table[i + 1];

}

else if (oper[i] == '/') {

result /= table[i + 1];

}

}

return result;

}

pair<int,int> insertOperator(vector<int>& table, vector<int>& oper) {

vector<char> opers;

// to use next\_permutation operators have to be inserted with following order : \* + - /

//since next\_permutation act as ascending order.

// c.f ) ascii number of operands are 42, 43, 45, 47 in order.

for (int i = 0; i < oper[2]; i++) opers.push\_back('\*');

for (int i = 0; i < oper[0]; i++) opers.push\_back('+');

for (int i = 0; i < oper[1]; i++) opers.push\_back('-');

for (int i = 0; i < oper[3]; i++) opers.push\_back('/');

int minimum = INT\_MAX;

int maximum = INT\_MIN;

int cur\_value;

do {

cur\_value = calculate(table, opers);

minimum = min(minimum, cur\_value);

maximum = max(maximum, cur\_value);

} while (next\_permutation(opers.begin(), opers.end()));

return make\_pair(maximum, minimum);

}

void printTable(vector<int>& table, vector<int>& oper) {

for (int i : table)

cout << i << " ";

cout << endl;

for (int i : oper)

cout << i << " ";

cout << endl;

}

int main() {

//pair<vector<int>, vector<int>> input = makeTable();

pair<vector<int>, vector<int>> input = makeTable2();

pair<int,int> answer= insertOperator(input.first,input.second);

cout << answer.first << endl << answer.second << endl;

return 0;

}

[111. **[BACKJOON – SAMSUNG SW : Start and Link]]**

- at first, I solved using next\_permuation but time limit exceeded occurred.

- so I changed the algorithm to DFS, but it was same.

- code below is DFS.

#include<iostream>

#include<fstream>

#include<climits>

#include<vector>

#include<algorithm>

#include<unordered\_map>

using namespace std;

unordered\_map<string, int> visit;

vector<vector<int>> makeTable() {

ifstream in("C:\\Users\\gjsgu\\Desktop\\Algorithm\\test\_case\\test\_case\_for\_start\_and\_link.txt");

int N;

if (in.is\_open()) {

cout << "file is opened." << endl;

in >> N;

}

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

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

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

in >> table[i][j];

}

}

return table;

}

vector<vector<int>> makeTable2() {

int N;

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];

}

}

return table;

}

void printTeam(vector<int>& team) {

for (int i : team)

cout << i << " ";

cout << endl;

}

int statDifference(vector<vector<int>>& table,vector<int>& red,vector<int>& blue,int middle,string reds) {

int red\_sum = 0; int blue\_sum = 0;

string blues;

for (int i : blue)

blues += i+ '0';

if (visit[blues]) {

return INT\_MAX;

}

visit[reds] = 1;

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

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

red\_sum += table[red[i]-1][red[j]-1];

blue\_sum += table[blue[i]-1][blue[j]-1];

}

}

return abs(red\_sum - blue\_sum);

}

int DFS(vector<vector<int>>& table,vector<int> red, vector<int> blue,int middle,string reds) {

if (red.size() == middle) return statDifference(table,red,blue,middle,reds);

int minimum = INT\_MAX;

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

vector<int> red\_temp = red;

vector<int> blue\_temp = blue;

red\_temp.push\_back(blue[i]);

reds += blue[i] + '0';

blue\_temp.erase(next(blue\_temp.begin(), i));

minimum=min(minimum,DFS(table, red\_temp, blue\_temp, middle,reds));

}

return minimum;

}

// 1 2 3 4 5 6

//1 2 4

// 3 5 6

int makeTeam(vector<vector<int>>& table) {

vector<int> member;

int minimum = INT\_MAX;

int middle = table.size() / 2;

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

member.push\_back(i + 1);

return DFS(table, {}, member, middle,"");

}

int main() {

//vector<vector<int>> table = makeTable();

vector<vector<int>> table = makeTable2();

cout<<makeTeam(table);

return 0;

}

- I should’ve revised the code.

- At first, we don’t need to check all the possible red team.

- Since if red team has player 1,2, and 3, then blue must have 4,5, and 6.

- And instead of moving a player from one to another, just select players to be red team.

- see the code.

#include<iostream>

#include<fstream>

#include<climits>

#include<vector>

#include<algorithm>

#include<unordered\_map>

using namespace std;

vector<vector<int>> makeTable() {

ifstream in("C:\\Users\\gjsgu\\Desktop\\Algorithm\\test\_case\\test\_case\_for\_start\_and\_link.txt");

int N;

if (in.is\_open()) {

cout << "file is opened." << endl;

in >> N;

}

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

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

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

in >> table[i][j];

}

}

return table;

}

vector<vector<int>> makeTable2() {

int N;

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];

}

}

return table;

}

void printTeam(vector<int>& team) {

for (int i : team)

cout << i << " ";

cout << endl;

}

int statDifference(vector<vector<int>>& table,vector<bool>& visit, int middle) {

vector<int> red; vector<int> blue;

//In here, make team

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

visit[i] ? red.push\_back(i) : blue.push\_back(i);

}

int red\_sum = 0; int blue\_sum = 0;

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

for (int j = 0; j < red.size(); j++) {

red\_sum += table[red[i]][red[j]];

blue\_sum += table[blue[i]][blue[j]];

}

}

return abs(red\_sum - blue\_sum);

}

int DFS(vector<vector<int>>& table,vector<bool> visit,int count,int middle,int next) {

if (count == middle) return statDifference(table,visit,middle);

int minimum = INT\_MAX;

// instead of making red and blue vectors and moving player one to another, just select players to be red team.

for (int i = next; i < visit.size(); i++) {

visit[i] = true;

minimum=min(minimum,DFS(table, visit, count + 1, middle,i+1));

visit[i] = false;

}

return minimum;

}

int makeTeam(vector<vector<int>>& table) {

vector<int> member;

vector<bool> visit(table.size(),false);

int minimum = INT\_MAX;

int middle = table.size() / 2;

// we don't neet to check all the possible red team

// since if red choose 1,2,3, blue must choose 4,5,6 and vice verse.

// that is the case of that red choose 4,5,6 isn't needed.

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

visit[i] = true;

minimum = min(minimum, DFS(table, visit, 1, middle, i + 1));

visit[i] = false;

}

return minimum;

}

//1 2 3 4 5 6

//6 5 4 3 2 1

int main() {

//vector<vector<int>> table = makeTable();

vector<vector<int>> table = makeTable2();

cout << makeTeam(table);

return 0;

}

[112. **[BACKJOON – SAMSUNG SW : Ramp]]**

- there was lots of error in this problem. I mean the errors the problem has.

- this problem didn’t describe exactly. Even having wrong information.

- I made an algorithm including all the possible case but I didn’t need to actually.

- However, i didn’t want to spent much time to revise for this problem. I just submited my answer and got marked.

- see the code.

#include<iostream>

#include<fstream>

#include<vector>

#include<algorithm>

using namespace std;

pair<vector<vector<int>>,int> makeTable(){

ifstream in("C:\\Users\\gjsgu\\Desktop\\Algorithm\\test\_case\\test\_case\_for\_ramp.txt");

int N; int L;

if (in.is\_open()) {

cout << "file is opened." << endl;

in >> N; in >> L;

}

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

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

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

in >> table[i][j];

return make\_pair(table,L);

}

pair<vector<vector<int>>, int> makeTable2() {

int N; int L;

cin >> N; cin >> L;

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];

return make\_pair(table, L);

}

// we have to consider row first and column first search.

pair<vector<vector<int>>,int> rowSearch(vector<vector<int>> table,int L, vector<vector<int>> visit,int count) {

int n = table.size();

vector<int> path;

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

int cur\_height = table[i][0];

int length = 1;

bool flag = false;

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

if (table[i][j] == cur\_height + 1) { // higher

if (length >= L) {

for (int s = 0; s < L; s++) if (visit[i][j - 1 - s] == 1) { flag = true; break; }

if (flag) break;

if (j == n - 1) { count++; path.push\_back(i); break; }

for (int s = 0; s < L; s++) visit[i][j - 1 - s] = 1;

length = 1; cur\_height++;

continue;

}

else break;

}

else if (table[i][j] == cur\_height) { // same height

if (j == n - 1) { count++; path.push\_back(i); break; }

length++;

continue;

}

else if (table[i][j]==cur\_height-1) { // lower

if (n - j < L) break;

for (int s = j; s < j + L; s++) if ( visit[i][s] == 1 || table[i][s]!=cur\_height-1) { flag = true; break; }

if (flag) break;

for (int s = j; s < j + L; s++) visit[i][s] = 1;

j += L - 1;

if (j == n - 1) { count++; path.push\_back(i); break; }

length = L;

cur\_height--;

}

else break;

}

}

return make\_pair(visit, count);

}

pair<vector<vector<int>>, int> colSearch(vector<vector<int>> table,int L,vector<vector<int>> visit,int count) {

int n = table.size();

vector<int> path;

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

int cur\_height = table[0][j];

int length = 1;

bool flag = false;

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

if (table[i][j] == cur\_height + 1) { // higher

if (length >= L) {

for (int s = 0; s < L; s++) if (visit[i - 1 - s][j] == 1) { flag = true; break; }

if (flag) break;

if (i == n - 1) { count++; path.push\_back(j); break; }

for (int s = 0; s < L; s++) visit[i - 1 - s][j] = 1;

length = 1; cur\_height++;

continue;

}

else break;

}

else if (table[i][j] == cur\_height) { // same height

if (i == n - 1) { count++; path.push\_back(j); break; }

length++;

continue;

}

else if (table[i][j] == cur\_height - 1) { // lower

if (n - i < L) break;

for (int s = i; s < i + L; s++) if (visit[s][j] == 1|| table[s][j] != cur\_height - 1) { flag = true; break; }

if (flag) break;

for (int s = i; s < i + L; s++) visit[s][j] = 1;

i += L - 1;

if (i == n - 1) { count++; path.push\_back(j); break; }

length = L;

cur\_height--;

}

else break;

}

}

return make\_pair(visit, count);

}

int main() {

//pair<vector<vector<int>>, int> input = makeTable();

pair<vector<vector<int>>, int> input = makeTable2();

vector<vector<int>> visit(input.first.size(), vector<int>(input.first.size(), 0));

pair < vector<vector<int>>, int> row\_first = rowSearch(input.first, input.second, visit, 0);

row\_first = colSearch(input.first, input.second, visit, row\_first.second);

cout<<row\_first.second;

return 0;

}

[113. **[BACKJOON – SAMSUNG SW : Gears ]]**

- I think there are at least 2 method to solve this problem. one is DFS and another one is just to use many if-condintion clause.

- I chose the latter one.

- the trap was that before rotate a gear, we have to check the next gear will be rotated or not.

- see the code.

#include<iostream>

#include<fstream>

#include<vector>

using namespace std;

pair<vector<vector<int>>,vector<vector<int>>> makeTable() {

ifstream in("C:\\Users\\gjsgu\\Desktop\\Algorithm\\test\_case\\test\_case\_for\_gears.txt");

if (in.is\_open()) cout << "file is opened." << endl;

vector<vector<int>> table(4,vector<int>(8,0));

string row;

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

in >> row;

for (int j = 0; j < row.length(); j++) {

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

}

}

int K;

in >> K;

int a; int b;

vector<vector<int>> rotate(K,vector<int>(2,0));

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

in >> a; in >> b;

rotate[i][0] = a; rotate[i][1] = b;

}

return make\_pair(table, rotate);

}

pair<vector<vector<int>>, vector<vector<int>>> makeTable2() {

vector<vector<int>> table(4, vector<int>(8, 0));

string row;

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

cin >> row;

for (int j = 0; j < row.length(); j++) {

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

}

}

int K;

cin >> K;

int a; int b;

vector<vector<int>> rotate(K, vector<int>(2, 0));

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

cin >> a; cin >> b;

rotate[i][0] = a; rotate[i][1] = b;

}

return make\_pair(table, rotate);

}

vector<int> move(vector<int> gear, int dir) {

if (dir == 1) {

int temp = gear[7];

for (int i = gear.size() - 2; i >= 0; i--)

gear[i + 1] = gear[i];

gear[0] = temp;

return gear;

}

else {

int temp = gear[0];

for (int i = 0; i < gear.size() - 1; i++)

gear[i] = gear[i + 1];

gear[7] = temp;

return gear;

}

}

void printTable(vector<vector<int>> gears) {

for (vector<int> row : gears) {

for (int i : row)

cout << i << " ";

cout << endl;

}

}

int tonado(vector<vector<int>> gears,vector<vector<int>> rotate) {

// 2 is east, 6 is west.

int cur\_gear; int dir;

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

cur\_gear = rotate[i][0]-1; dir = rotate[i][1];

gears[cur\_gear] = move(gears[cur\_gear], dir);

int temp\_dir = dir;

//to left

for (int left = cur\_gear - 1; left >= 0; left--) {

if (temp\_dir == 1 && gears[left+1][7]!=gears[left][2]) gears[left] = move(gears[left], temp\_dir=-1);

else if(temp\_dir==-1 && gears[left+1][5]!=gears[left][2]) gears[left] = move(gears[left], temp\_dir=1);

else break;

}

//to right

temp\_dir = dir;

for (int right = cur\_gear + 1; right < gears.size(); right++) {

if(temp\_dir==1 && gears[right-1][3]!=gears[right][6]) gears[right] = move(gears[right], temp\_dir=-1);

else if (temp\_dir ==-1 && gears[right - 1][1] != gears[right][6]) gears[right] = move(gears[right], temp\_dir = 1);

else break;

}

}

return gears[0][0] + gears[1][0] \* 2 + gears[2][0] \* 4 + gears[3][0] \* 8;

}

int main() {

//pair<vector<vector<int>>, vector<vector<int>>> input = makeTable();

pair<vector<vector<int>>, vector<vector<int>>> input = makeTable2();

vector<vector<int>> gears = input.first;

vector<vector<int>> rotate = input.second;

cout << tonado(gears, rotate);

return 0;

}

[114. **[BACKJOON – SAMSUNG SW : Monitoring ]]**

- just following rules, removing blind spot is the main point of this problem.

- DFS is needed.

- see the code.

#include<iostream>

#include<fstream>

#include<vector>

#include<climits>

#include<algorithm>

using namespace std;

pair<vector<vector<int>>, vector<vector<int>>> makeTable() {

ifstream in("C:\\Users\\gjsgu\\Desktop\\Algorithm\\test\_case\\test\_case\_for\_monitoring.txt");

if (in.is\_open()) cout << "file is opened." << endl;

int N; int M;

in >> N; in >> M;

int temp;

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

vector<vector<int>> camera;

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

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

in >> temp;

if (1 <= temp && temp <= 5) camera.push\_back({ i,j,temp });

table[i][j] = temp;

}

}

return make\_pair(table, camera);

}

pair<vector<vector<int>>, vector<vector<int>>> makeTable2() {

int N; int M;

cin >> N; cin >> M;

int temp;

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

vector<vector<int>> camera;

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

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

cin >> temp;

if (1 <= temp && temp <= 5) camera.push\_back({ i,j,temp });

table[i][j] = temp;

}

}

return make\_pair(table, camera);

}

int blindSpot(vector<vector<int>>& table) {

int spot = 0;

for (vector<int> row : table)

for (int i : row)

if (i == 0) spot++;

return spot;

}

vector<vector<int>> East(vector<vector<int>> table, int x, int y) {

for (int j = y; j < table[0].size(); j++) {

if (table[x][j] == 6) break;

if (table[x][j] == 0) table[x][j] = 9; // 9 is where we can monitor.

}

return table;

}

vector<vector<int>> West(vector<vector<int>> table, int x, int y) {

for (int j = y; j >= 0; j--) {

if (table[x][j] == 6) break;

if (table[x][j] == 0) table[x][j] = 9; // 9 is where we can monitor.

}

return table;

}

vector<vector<int>> South(vector<vector<int>> table, int x, int y) {

for (int i = x; i >= 0; i--) {

if (table[i][y] == 6) break;

if (table[i][y] == 0) table[i][y] = 9; // 9 is where we can monitor.

}

return table;

}

vector<vector<int>> North(vector<vector<int>> table, int x, int y) {

for (int i = x; i < table.size(); i++) {

if (table[i][y] == 6) break;

if (table[i][y] == 0) table[i][y] = 9; // 9 is where we can monitor.

}

return table;

}

vector<vector<vector<int>>> Monitoring(vector<vector<int>> table,int x, int y, int dir) {

vector<vector<vector<int>>> res\_table;

vector<vector<int>> temp\_table;

switch (dir) {

case 1:

// East

res\_table.push\_back(East(table, x, y));

// West

res\_table.push\_back(West(table, x, y));

// North

res\_table.push\_back(North(table, x, y));

// South

res\_table.push\_back(South(table, x, y));

return res\_table;

case 2:

//horizontal

temp\_table = table;

temp\_table = East(temp\_table, x, y);

temp\_table = West(temp\_table, x, y);

res\_table.push\_back(temp\_table);

//vertical

temp\_table = table;

temp\_table = South(temp\_table, x, y);

temp\_table = North(temp\_table, x, y);

res\_table.push\_back(temp\_table);

return res\_table;

case 3 :

// North East

temp\_table = table;

temp\_table = North(temp\_table, x, y);

temp\_table = East(temp\_table, x, y);

res\_table.push\_back(temp\_table);

// South East

temp\_table = table;

temp\_table = South(temp\_table, x, y);

temp\_table = East(temp\_table, x, y);

res\_table.push\_back(temp\_table);

// North West

temp\_table = table;

temp\_table = North(temp\_table, x, y);

temp\_table = West(temp\_table, x, y);

res\_table.push\_back(temp\_table);

// South West

temp\_table = table;

temp\_table = South(temp\_table, x, y);

temp\_table = West(temp\_table, x, y);

res\_table.push\_back(temp\_table);

return res\_table;

case 4:

// without South

temp\_table = table;

temp\_table = West(temp\_table, x, y);

temp\_table = North(temp\_table, x, y);

temp\_table = East(temp\_table, x, y);

res\_table.push\_back(temp\_table);

// without West

temp\_table = table;

temp\_table = North(temp\_table, x, y);

temp\_table = East(temp\_table, x, y);

temp\_table = South(temp\_table, x, y);

res\_table.push\_back(temp\_table);

// without North

temp\_table = table;

temp\_table = East(temp\_table, x, y);

temp\_table = South(temp\_table, x, y);

temp\_table = West(temp\_table, x, y);

res\_table.push\_back(temp\_table);

// without East

temp\_table = table;

temp\_table = South(temp\_table, x, y);

temp\_table = West(temp\_table, x, y);

temp\_table = North(temp\_table, x, y);

res\_table.push\_back(temp\_table);

return res\_table;

case 5:

temp\_table = table;

temp\_table = South(temp\_table, x, y);

temp\_table = West(temp\_table, x, y);

temp\_table = North(temp\_table, x, y);

temp\_table = East(temp\_table, x, y);

res\_table.push\_back(temp\_table);

return res\_table;

}

}

int DFS(vector<vector<int>> table, vector<vector<int>> camera,int start) {

if (start == camera.size()) return blindSpot(table);

int minimum = INT\_MAX;

int x = camera[start][0]; int y = camera[start][1]; int dir = camera[start][2];

vector < vector<vector<int>>> res\_table = Monitoring(table, x, y, dir);

for (int i = 0; i < res\_table.size(); i++) minimum=min(minimum,DFS(res\_table[i], camera, start + 1));

return minimum;

}

int main() {

//pair<vector<vector<int>>, vector<vector<int>>> input = makeTable();

pair<vector<vector<int>>, vector<vector<int>>> input = makeTable2();

vector<vector<int>> table = input.first;

vector<vector<int>> camera = input.second;

cout << DFS(table, camera,0);

return 0;

}

[115. **[BACKJOON – SAMSUNG SW : Ladder Rigging]]**

- I spent 5 hour to reach the limit.

- At first, I used DFS but time limit exceeded occurred.

- So I changed my algorithm to BFS but it was same!!!

- so I changed again and make all the thing small and use global variable to reduce the transfer time during calling functions.

- Finally, I got correct mark. I refered to disccusion.

#include<iostream>

#include<fstream>

#include<vector>

#include<queue>

#include<climits>

using namespace std;

int answer = INT\_MAX;

int table[31][11];

int N; int M; int H;

void makeTable() {

ifstream in("C:\\Users\\gjsgu\\Desktop\\Algorithm\\test\_case\\test\_case\_for\_ladder\_rigging.txt");

if (in.is\_open()) cout << "file is opened." << endl;

in >> N; in >> M; in >> H;

int a; int b;

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

in >> a; in >> b;

table[a ][b] = 2;// 2 is horizon line.

}

}

void makeTable2() {

cin >> N; cin >> M; cin >> H;

int a; int b;

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

cin >> a; cin >> b;

table[a ][b] = 2;// 2 is horizon line.

}

}

void printTable() {

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

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

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

cout << endl;

}

}

inline bool isSameEnd() {

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

int pos = j;

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

if (table[i][pos - 1] == 2) pos -= 1;

else if (table[i][pos] == 2) pos += 1;

}

if (j != pos) return false;

}

return true;

}

void DFS(int count, int start) {

if(count >= 4) return;

if (isSameEnd()) {

answer = min(answer, count);

return;

}

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

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

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

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

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

table[i][j] = 2;

DFS( count + 1, j);

table[i][j] = 0;

}

}

}

int main() {

makeTable2();

if (isSameEnd()) {

cout << 0;

return 0;

}

DFS( 0, 1);

answer== INT\_MAX? cout<<-1:cout << answer;

return 0;

}

[116. **[BACKJOON – SAMSUNG SW : Dragon Curve]]**

- this problem’s main point was the way to make the dragon curve with generation.

- I used a stack to record previous directions and take it all when new generation was started with rotating the directions 90 degrees.

- see the code.

#include<iostream>

#include<fstream>

#include<vector>

using namespace std;

int table[101][101] = { 0, };

int N,X, Y, d, g;

int direction[4][2] = { {0,1},{-1,0},{0,-1},{1,0} }; // east north west south

vector<vector<int>> makeTable() {

ifstream in("C:\\Users\\gjsgu\\Desktop\\Algorithm\\test\_case\\test\_case\_for\_dragon\_curve.txt");

if (in.is\_open()) cout << "file is opened." << endl;

in >> N;

vector<vector<int>> curve;

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

in >> X >> Y >> d >> g;

curve.push\_back(vector<int>{X, Y, d, g});

}

return curve;

}

vector<vector<int>> makeTable2() {

cin >> N;

vector<vector<int>> curve;

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

cin >> X >> Y >> d >> g;

curve.push\_back(vector<int>{X, Y, d, g});

}

return curve;

}

void printCurve(vector<vector<int>> curve) {

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

for (int j = 0; j < curve[0].size(); j++)

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

cout << endl;

}

}

void printTable() {

for (int i = 0; i < 101; i++) {

for (int j = 0; j < 101; j++) {

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

}

cout << endl;

}

}

pair<vector<int>, vector<int>> Drawing(vector<int> stack, int ex, int ey) {

vector<int> res = stack;

for (int i = stack.size() - 1; i >= 0; i--) {

int dir = (stack[i] + 1) % 4;

ex += direction[dir][1]; ey += direction[dir][0];

table[ey][ex] = 1;

res.push\_back(dir);

}

return make\_pair(res, vector<int>{ex, ey});

}

void makeDragonCurve(vector<vector<int>> curve) {

//how to make rotate 90 degree?

for (vector<int> row : curve) {

int x = row[0]; int y = row[1]; int dir = row[2]; int generation = row[3];

int ex = x + direction[dir][1]; int ey = y + direction[dir][0];

table[y][x] = 1;

table[ey][ex] = 1;

vector<int> stack = { dir };

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

pair<vector<int>, vector<int>> res = Drawing(stack, ex, ey);

stack = res.first; ex = res.second[0]; ey = res.second[1];

}

}

}

int findAnswer() {

int sum = 0;

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

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

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

}

}

return sum;

}

int main() {

//vector<vector<int>> curve = makeTable();

vector<vector<int>> curve = makeTable2();

makeDragonCurve(curve);

cout<<findAnswer();

return 0;

}

[117. **[BACKJOON – SAMSUNG SW : Chicken Delivery]]**

- DFS was needed to choose which chicken house will be alive.

- BFS was a significant problem, since when I used queue for BFS, time limit exceeded occurred.

- so I throw the fxxking queue out and using vector to recored which chicken houses were chosen.

- From the chosen chicken house, calculated distance between all the houses.

- see the code.

#include<iostream>

#include<fstream>

#include<vector>

#include<climits>

#include<queue>

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

using namespace std;

vector<vector<int>> table;

vector<vector<int>> chicken;

vector<vector<int>> house;

vector<int> chosen;

int N; int M;

void makeTable() {

ifstream in("C:\\Users\\gjsgu\\Desktop\\Algorithm\\test\_case\\test\_case\_for\_chicken\_delivery.txt");

if (in.is\_open()) cout << "file is opened." << endl;

in >> N>>M;

int temp;

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

table.push\_back(vector<int>(N, 0));

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

in >> temp;

if (temp == 2) { chicken.push\_back({ i,j }); continue;}

if (temp == 1) house.push\_back({ i,j });

table[i][j] = 1;

}

}

}

void makeTable2() {

cin >> N >> M;

int temp;

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

table.push\_back(vector<int>(N, 0));

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

cin >> temp;

if (temp == 2) { chicken.push\_back({ i,j }); continue; }

if (temp == 1) house.push\_back({ i,j });

table[i][j] = 1;

}

}

}

int BFS() {

int distance = 0;

for (int k = 0; k < house.size(); k++) {

int hx = house[k][0]; int hy = house[k][1];

int minimum = INT\_MAX;

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

int a = chosen[i];

minimum = min(minimum, abs(hx - chicken[a][0]) + abs(hy - chicken[a][1]));

}

distance += minimum;

}

return distance;

}

int DFS(int start,int count) {

if (count >= M) {

return BFS();

}

int minimum = INT\_MAX;

for (int i = start; i < chicken.size(); i++) {

int x = chicken[i][0]; int y = chicken[i][1];

chosen.push\_back(i);

table[x][y] = 2;

minimum=min(minimum,DFS(i + 1,count+1));

table[x][y] = 0;

chosen.pop\_back();

}

return minimum;

}

int main() {

//makeTable();

makeTable2();

cout << DFS(0,0);

return 0;

}

[118. **[BACKJOON – SAMSUNG SW : Cubing]]**

- it was not a solving algorhthm! I think it was just a time consuming work actually.

- making cube and make it move following the given rules. was suck!

- see the code.

#include<iostream>

#include<fstream>

#include<vector>

using namespace std;

vector<vector<char>> front;

vector<vector<char>> back;

vector<vector<char>> Left;

vector<vector<char>> Right;

vector<vector<char>> up;

vector<vector<char>> down;

vector<vector<string>> rule;

int N;

void Input() {

ifstream in("C:\\Users\\gjsgu\\Desktop\\Algorithm\\test\_case\\test\_case\_for\_cubing.txt");

if (in.is\_open()) cout << "file is opened." << endl;

in >> N;

int temp;

string rotate;

rule.assign(N, vector<string>{});

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

in >> temp;

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

in >> rotate;

rule[i].push\_back(rotate);

}

}

}

void Input2() {

cin >> N;

int temp;

string rotate;

rule.assign(N, vector<string>{});

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

cin >> temp;

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

cin >> rotate;

rule[i].push\_back(rotate);

}

}

}

void printRule() {

for (vector<string> row : rule) {

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

cout << row[i];

}

cout << endl;

}

}

void initCube() {

front = { {'r','r','r'},{'r','r','r'},{'r','r','r'} };

back = { {'o','o','o'},{'o','o','o'},{'o','o','o'} };

Left = { {'g','g','g'},{'g','g','g'},{'g','g','g'} };

Right = { {'b','b','b'},{'b','b','b'},{'b','b','b'} };

up = { {'w','w','w'},{'w','w','w'},{'w','w','w'} };

//up = { {'1','2','3'},{'4','5','6'},{'7','8','9'} };

down = { {'y','y','y'},{'y','y','y'},{'y','y','y'} };

}

vector<vector<char>> moveFront(vector<vector<char>> Left, string dir) {

char temp,temp2,temp3;

if (dir == "-") {

temp = Left[0][0]; temp2 = Left[0][1];

Left[0][0] = Left[0][2]; Left[0][1] = Left[1][2]; Left[0][2] = Left[2][2]; Left[1][2] = Left[2][1]; Left[2][2] = Left[2][0]; Left[2][1] = Left[1][0];

Left[2][0] = temp; Left[1][0] = temp2;

}

else {

temp = Left[0][0]; temp2 = Left[1][0];

Left[0][0] = Left[2][0]; Left[1][0] = Left[2][1]; Left[2][0] = Left[2][2]; Left[2][1] = Left[1][2]; Left[2][2] = Left[0][2]; Left[1][2] = Left[0][1];

Left[0][2] = temp; Left[0][1] = temp2;

}

return Left;

}

void moveSide(char& a, char& b, char& c, char& d, char& e, char& f, char& g, char& h, char& i, char& j, char& l, char& m,string dir) {

char temp = a, temp2 = b, temp3 = c;

if (dir == "-") {

a = d; b = e; c = f; d = g; e = h; f = i; g = j; h = l; i = m; j = temp; l = temp2; m = temp3;

}

else {

a = j; b = l; c = m; j = g; l = h; m = i; g = d; h = e; i = f; d = temp; e = temp2; f = temp3;

}

}

void rotateCube(string side, string dir) {

char temp, temp2, temp3;

//cout << "<<"<<side<<">>" << endl;

if (side == "L") {

Left = moveFront(Left, dir);

moveSide(front[0][0], front[1][0], front[2][0], down[0][0], down[1][0], down[2][0], back[0][0], back[1][0], back[2][0], up[0][0], up[1][0], up[2][0], dir);

}

else if (side == "R") {

Right = moveFront(Right, dir);

moveSide(front[0][2], front[1][2], front[2][2], up[0][2], up[1][2], up[2][2], back[0][2], back[1][2], back[2][2], down[0][2], down[1][2], down[2][2], dir);

}

else if (side == "U") {

up = moveFront(up, dir);

moveSide(front[0][2], front[0][1], front[0][0], Left[0][2], Left[0][1], Left[0][0], back[2][0], back[2][1], back[2][2], Right[0][2], Right[0][1], Right[0][0],dir);

}

else if (side == "D") {

down = moveFront(down, dir);

moveSide(front[2][0], front[2][1], front[2][2], Right[2][0], Right[2][1], Right[2][2], back[0][2], back[0][1], back[0][0], Left[2][0], Left[2][1], Left[2][2], dir);

}

else if (side == "F") {

front = moveFront(front, dir);

moveSide(up[2][0], up[2][1], up[2][2], Right[0][0], Right[1][0], Right[2][0], down[0][2], down[0][1], down[0][0], Left[2][2], Left[1][2], Left[0][2], dir);

}

else if (side == "B") {

back = moveFront(back, dir);

moveSide(up[0][2], up[0][1], up[0][0], Left[0][0], Left[1][0], Left[2][0], down[2][0], down[2][1], down[2][2], Right[2][2], Right[1][2], Right[0][2], dir);

}

}

void printCubeTop() {

for (vector<char> row : up) {

for (char c : row) {

cout << c;

}

cout << endl;

}

}

void printCube() {

cout << "up" << endl;

for (vector<char> row : up) {

for (char c : row) {

cout << c;

}

cout << endl;

}

cout << endl;

cout << "front" << endl;

for (vector<char> row : front) {

for (char c : row) {

cout << c;

}

cout << endl;

}

cout << endl;

cout << "Left" << endl;

for (vector<char> row : Left) {

for (char c : row) {

cout << c;

}

cout << endl;

}

cout << endl;

cout << "Right" << endl;

for (vector<char> row : Right) {

for (char c : row) {

cout << c;

}

cout << endl;

}

cout << endl;

cout << "back" << endl;

for (vector<char> row : back) {

for (char c : row) {

cout << c;

}

cout << endl;

}

cout << endl;

cout << "down" << endl;

for (vector<char> row : down) {

for (char c : row) {

cout << c;

}

cout << endl;

}

}

void followingRules() {

string side; string dir;

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

initCube();

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

side = rule[i][j][0]; dir = rule[i][j][1];

rotateCube(side, dir);

}

printCubeTop();

//printCube();

}

}

int main() {

//Input();

Input2();

followingRules();

return 0;

}

[119. **[Programmers– KAKAO 2020 : String Compression]]**

- we need to be careful when we handle how many substring occurred.

- I used std::to\_string to convert int to string.

- see the code.

#include<iostream>

#include<fstream>

#include<vector>

#include<string>

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

using namespace std;

string s;

void Input() {

ifstream in("C:\\Users\\gjsgu\\Desktop\\Algorithm\\test\_case\\test\_case\_for\_string\_compression.txt");

if (in.is\_open()) cout << "file is opened." << endl;

in >> s;

}

int solution(string s) {

int answer = s.size();

for (int cur\_size = 1; cur\_size < s.size()/2+1; cur\_size++) {

vector<string> res = { s.substr(0,cur\_size) };

vector<int> num = { 1 };

string temp;

int i;

for (i = cur\_size; i + cur\_size <= s.size(); i += cur\_size) {

temp = s.substr(i, cur\_size);

if (res.back() == temp) { num[num.size() - 1]++; continue; }

res.push\_back(temp);

num.push\_back(1);

}

string compression;

for (int j = 0; j < res.size(); j++) {

if (num[j] == 1) compression += res[j];

else {

compression += to\_string(num[j]);

compression += res[j];

}

}

for (; i < s.size(); i++)

compression += s[i];

answer = min(answer, compression.length());

}

return answer;

}

int main() {

Input();

cout << solution(s);

return 0;

}

[120. **[Programmers– KAKAO 2020 : Converting Parenthesis]]**

- this problem’s main point was to follow given rule to make valid parenthesis.

- but the given rule was quite ambiguous. so referring to example in a detailed fashion was important.

- be careful when you handle DFS.

- see the code.

#include <string>

#include <vector>

#include<iostream>

#include<fstream>

using namespace std;

string Input() {

ifstream in("C:\\Users\\gjsgu\\Desktop\\Algorithm\\test\_case\\test\_case\_for\_converting\_parenthesis.txt");

if (in.is\_open()) cout << "file is opened." << endl;

string p; in >> p;

return p;

}

pair<string, string> splitUV(string p) {

string u; string v;

int left = 0; int right = 0;

int i=0;

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

u += p[i];

if (p[i] == '(') left++;

else if (p[i] == ')') right++;

if (left == right) break;

}

for (int j = i + 1; j < p.size(); j++) v += p[j];

return make\_pair(u, v);

}

bool isValid(string u) {

int valid = 0;

for (char c : u) {

if (c == '(') valid++;

else if (c == ')') valid--;

if (valid < 0) return false;

}

return true;

}

string reverse(string p) {

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

p[i] == '(' ? p[i] = ')' : p[i] = '(';

return p;

}

string firstStep(string p) {

string res;

if (p.empty()) return "";

pair<string, string> split = splitUV(p);

string u = split.first; string v = split.second;

if (isValid(u)) { res += u; res += firstStep(v);}

else {

res += "("+ firstStep(v)+")";

u.erase(u.begin()); u.erase(next(u.end(),-1));

u = reverse(u);

res += u;

}

return res;

}

string solution(string p) {

string answer = "";

answer = firstStep(p);

return answer;

}

int main() {

string p= Input();

cout << solution(p);

return 0;

}

[121. **[Programmers– KAKAO 2020 : Lock and Key]]**

- implementing slicing using C++ is quite hard for me.

- I made slicing to determine whether the given key is valid or not.

- see the code.

#include <string>

#include <vector>

#include<iostream>

using namespace std;

vector<vector<int>> makeTable(int k,int l,vector<vector<int>> lock) {

vector<vector<int>> table(k \* 2 - 2 + l, vector<int>(k \* 2 - 2 + l, 2));

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

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

table[i][j] = lock[i - (k - 1)][j - (k - 1)];

}

}

return table;

}

vector<vector<int>> rotateKey(vector<vector<int>> key) {

int k = key.size();

vector<vector<int>> res(k, vector<int>(k));

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

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

res[i][j] = key[k-j-1][i];

}

}

return res;

}

void printKey(vector<vector<int>> key) {

for (vector<int> row : key) {

for (int i : row) {

cout << i << " ";

}

cout << endl;

}

cout << endl;

}

void printTable(vector<vector<int>> table) {

for (vector<int> row : table) {

for (int i : row) {

cout << i << " ";

}

cout << endl;

}

cout << endl;

}

bool isValid(vector<vector<int>> key, vector<vector<int>> table,vector<vector<int>> lock){

int k = key.size(); int t = table.size();

for (int i = 0; i < t-k+1; i++) {

for (int j = 0; j < t-k+1; j++) {

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

bool flag = true;

for (int a = 0; a < k; a++) {

for (int b = 0; b < k; b++) {

if (table[i + a][j + b] == 2) { temp\_lock[i+a][j+b] = 1; continue; }

else if ((table[i + a][j + b] == 0 && key[a][b] == 1) || (table[i + a][j + b] == 1 && key[a][b] == 0)) {

temp\_lock[i+a][j+b] = 1;

continue;

}

else { flag = false; break;}

}

}

if (flag) {

bool flag2 = true;

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

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

if (temp\_lock[i][j] == 0) { flag2 = false; break; }

}

}

if (flag2) {

return true;

}

}

}

}

return false;

}

bool solution(vector<vector<int>> key, vector<vector<int>> lock) {

bool answer = true;

int k = key.size(); int l = lock.size();

vector<vector<int>> table = makeTable(k, l,lock);

//rotate the given key at most 3 times.

if (isValid(key, table,lock)) return true;

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

key=rotateKey(key);

if (isValid(key, table,lock)) return true;

}

return false;

}

int main() {

vector<vector<int>> key = { {0,0},{1,0} };

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

cout<<solution(key, lock);

return 0;

}

[122. **[Programmers– KAKAO 2020 : Lyrics Searcing]]**

- I had solved correctly, but time limit exceeded occurred. so I had to change my algorithm for efficiency.

- below code is before changing the code.

#include <string>

#include <vector>

#include<iostream>

#include<fstream>

#include<unordered\_map>

using namespace std;

pair<vector<string>, vector<string>> Input() {

vector<string> words; vector<string> queries;

ifstream in("C:\\Users\\gjsgu\\Desktop\\Algorithm\\test\_case\\test\_case\_for\_lyrics\_searching.txt");

if (in.is\_open()) cout << "file is opened." << endl;

int N;

in >> N;

string temp;

for (int i = 0; i < N; i++) { in >> temp; words.push\_back(temp); }

while (!in.eof()) { in >> temp; queries.push\_back(temp); }

return make\_pair(words, queries);

}

void printInput(vector<string> input) {

for (string s : input)

cout << s << " ";

cout << endl;

}

void printAnswer(vector<int> answer) {

for (int i : answer)

cout << i << " ";

cout << endl;

}

int numOfValidString(vector<string> words, string quer) {

if (words.empty()) return 0;

int i;

int num = 0;

int size = words[0].size();

if (quer[0] == '?') {

for (i = 0; i < quer.size(); i++) if (quer[i] != '?') break;

int first = i;

int distance = size - first;

for (int j = 0; j < words.size(); j++) {

if (words[j].substr(first, distance) == quer.substr(first, distance)) num++;

}

}

else{

for(i=quer.size()-1; i>=0;i--) if (quer[i] != '?') break;

int last = i;

for (int j = 0; j < words.size(); j++) {

if (words[j].substr(0, last + 1) == quer.substr(0, last + 1)) num++;

}

}

return num;

}

vector<int> solution(vector<string> words, vector<string> queries) {

vector<int> answer;

unordered\_map<int, vector<string>> length;

for (int i = 0; i < words.size(); i++) length[words[i].size()].push\_back(words[i]);

for (int i = 0; i < queries.size(); i++) answer.push\_back(numOfValidString(length[queries[i].size()],queries[i]));

return answer;

}

int main() {

pair<vector<string>, vector<string>> input = Input();

vector<string> words = input.first;

vector<string> queries = input.second;

solution(words, queries);

return 0;

}

- I had to use trie structure to get marked at efficiency test.

- trie structure is often used when we handle string search.

- trie acts like a dictionary. so it has functions like insert, find.

- see the below code. I spent almost 4~5 hours.

#include <string>

#include <vector>

#include<iostream>

#include<fstream>

#include<unordered\_map>

using namespace std;

pair<vector<string>, vector<string>> Input() {

vector<string> words; vector<string> queries;

ifstream in("C:\\Users\\gjsgu\\Desktop\\Algorithm\\test\_case\\test\_case\_for\_lyrics\_searching.txt");

if (in.is\_open()) cout << "file is opened." << endl;

int N;

in >> N;

string temp;

for (int i = 0; i < N; i++) { in >> temp; words.push\_back(temp); }

while (!in.eof()) { in >> temp; queries.push\_back(temp); }

return make\_pair(words, queries);

}

void printInput(vector<string> input) {

for (string s : input)

cout << s << " ";

cout << endl;

}

void printAnswer(vector<int> answer) {

for (int i : answer)

cout << i << " ";

cout << endl;

}

// we have to use trie tree to record words and reversed words.

// we consider trie structure as a dictionary.

class Trie {

private:

Trie\* children[26];

bool terminal;

unordered\_map<int, int> length;

public:

Trie() {

fill(children, children + 26, nullptr);

this->terminal = false;

}

~Trie() {

for (int i = 0; i < 26; i++) if(children[i]!=NULL) delete children[i];

}

void insert(const char\* s,int size) {

if (\*s =='\0') {

terminal = true;

return;

}

length[size] += 1;

if (children[\*s - 'a'] == NULL)

children[\*s - 'a'] = new Trie;

children[\*s - 'a']->insert(s+1, size);

}

int find(const char\* s,int size) {

if (\*s == '\0') return terminal;

int count = 0;

if (\*s == '?') count += length[size];

else {

if (children[\*s - 'a'] == NULL) return 0;

count=children[\*s - 'a']->find(s+1, size);

}

return count;

}

};

vector<int> solution(vector<string> words, vector<string> queries) {

vector<int> answer;

Trie\* prefix = new Trie;

Trie\* suffix = new Trie;

int match;

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

prefix->insert(&words[i][0], words[i].size());

string temp = { words[i].rbegin(),words[i].rend() };

suffix->insert(&temp[0], temp.size());

}

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

if (queries[i][0] == '?') {

string temp = { queries[i].rbegin(), queries[i].rend() };

answer.push\_back(suffix->find(&temp[0], temp.size()));

}

else answer.push\_back(prefix->find(&queries[i][0], queries[i].size()));

}

return answer;

}

int main() {

pair<vector<string>, vector<string>> input = Input();

vector<string> words = input.first;

vector<string> queries = input.second;

printAnswer(solution(words, queries));

return 0;

}

[123. **[Programmers– KAKAO 2020 : Pillar and Beam]]**

- it was a quite dirty and time spending work.

- I had to implement all the possible condition by given rules.

- see the code.

#include <string>

#include <vector>

#include<iostream>

#define pillar 0

#define beam 1

using namespace std;

inline int toY(int y, int n) { return n - y - 1; }

void printTable(vector<vector<vector<int>>>& table, int n) {

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

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

cout << table[i][j][pillar] << "," << table[i][j][beam] << " ";

}

cout << endl;

}

}

void printAnswer(vector<vector<int>>& answer) {

for (vector<int> row : answer) {

for (int i : row) {

cout << i << " ";

}

cout << endl;

}

}

void Create(vector<vector<vector<int>>>& table, int x, int y, bool isBeam,int n) {

if (!isBeam) { //when it is a pillar.

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

else if(table[y][x][beam]) table[y][x][pillar] = 1;

else if(x-1>=0 && table[y][x-1][beam]) table[y][x][pillar] = 1;

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

}

else { // when it is a beam.

if(y+1<n && table[y+1][x][pillar]) table[y][x][beam] = 1;

else if (x + 1 < n && y + 1 < n && table[y + 1][x + 1][pillar]) table[y][x][beam] = 1;

else if (x-1>=0 && table[y][x-1][beam] && x+1<n && table[y][x+1][beam]) table[y][x][beam] = 1;

}

}

void Delete(vector<vector<vector<int>>>& table, int x, int y, bool isBeam,int n) {

if (!isBeam) {

if (y - 1 >= 0 && table[y - 1][x][pillar] && table[y - 1][x][beam] == 0 && (x - 1 <0 || table[y - 1][x - 1][beam] == 0)) return;

else if (y - 1 >= 0 && table[y - 1][x][beam] && (x + 1 >= n || table[y][x + 1][pillar] == 0) && ((x+1>=n || table[y - 1][x + 1][beam] == 0) || (x - 1 < 0 || table[y - 1][x - 1][beam] == 0))) return;

else if (y - 1 >= 0 && x - 1 >= 0 && table[y - 1][x - 1][beam] && table[y][x - 1][pillar] == 0 && ((x - 2 < 0 || table[y - 1][x - 2][beam] == 0) || (table[y - 1][x][beam] == 0))) return;

else table[y][x][pillar] = 0;

}

else {

if (table[y][x][pillar] && (x - 1 < 0 || table[y][x - 1][beam] == 0) && table[y + 1][x][pillar] == 0) return;

else if ((x + 1 < n && table[y][x + 1][pillar]) && table[y + 1][x + 1][pillar] == 0 && table[y][x + 1][beam] == 0) return;

else if ((x - 1 >= 0 && table[y][x - 1][beam] == 1) && table[y + 1][x - 1][pillar] == 0 && table[y + 1][x][pillar] == 0) return;

else if((x+1<n && table[y][x+1][beam]==1) && table[y+1][x+1][pillar]==0 && (x+2>=n || table[y+1][x+2][pillar]==0 )) return;

else table[y][x][beam] = 0;

}

}

vector<vector<int>> solution(int n, vector<vector<int>> build\_frame) {

vector<vector<int>> answer;

// notice that the table's row is y not x. And the origin is bottom-left not top-left.

vector<vector<vector<int>>> table(n, vector<vector<int>>(n, vector<int>(2, 0)));

// make table

for (int i = 0; i < build\_frame.size(); i++) {

int x = build\_frame[i][0]; int y = toY(build\_frame[i][1],n); bool isBeam = build\_frame[i][2]; bool create = build\_frame[i][3];

create ? Create(table, x, y, isBeam,n) : Delete(table, x, y, isBeam,n);

}

printTable(table, n);

// make answer

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

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

if (table[i][j][pillar]) answer.push\_back({ j,toY(i,n),0 });

if (table[i][j][beam]) answer.push\_back({ j,toY(i,n),1 });

}

}

printAnswer(answer);

return answer;

}

int main() {

//vector<vector<int>> input = {0,0,0,1},{2,0,0,1},{4,0,0,1},{0,1,1,1},{1,1,1,1},{2,1,1,1},{3,1,1,1},{2,0,0,0},{1,1,1,0},{2,2,0,1}};

vector<vector<int>> input = { {1,0,0,1},{1,1,1,1},{2,1,0,1},{2,2,1,1},{5,0,0,1},{5,1,0,1},{4,2,1,1},{3,2,1,1} };

//vector<vector<int>> input = { {1,0,0,1},{1,1,1,1},{1,1,0,1},{2,1,0,1},{2,2,1,1},{5,0,0,1},{5,1,0,1},{4,2,1,1},{3,2,1,1} };

solution(6, input);

return 0;

}

[124. **[Programmers– KAKAO 2020 : Outer Wall Check]]**

- I first used DFS to implement a brutr force, but time limit exceeded occurred.

- see the DFS code.

#include <string>

#include <vector>

#include<iostream>

#include<climits>

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

using namespace std;

int minimum = INT\_MAX;

inline bool isCheckedAll(vector<int> table) {

if (find(table.begin(), table.end(), 1) == table.end()) return true;

return false;

}

vector<int> ChecKWall(vector<int> table,int dist, int start,bool left,int n) {

if (left) {

while (dist>=0) {

table[start] = 0;

start - 1 < 0 ? start = n - 1 : start--;

dist--;

}

}

else {

while (dist>=0) {

table[start] = 0;

start = (start + 1) % n;

dist--;

}

}

return table;

}

void DFS(vector<int> table,vector<int>weak, vector<int> dist, int n,int count) {

if (isCheckedAll(table)) { minimum = min(minimum, count); return; }

else if (dist.empty()) return;

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

int start = weak[i];

vector<int> temp\_weak = weak;

temp\_weak.erase(next(temp\_weak.begin(), i));

for (int j = 0; j < dist.size(); j++) {

vector<int> temp\_dist = dist;

temp\_dist.erase(next(temp\_dist.begin(), j));

DFS(ChecKWall(table, dist[j], start, true, n),temp\_weak,temp\_dist,n,count+1);

DFS(ChecKWall(table, dist[j], start, false, n), temp\_weak, temp\_dist, n, count + 1);

}

}

}

int solution(int n, vector<int> weak, vector<int> dist) {

vector<int> table(n,0);

for (int i = 0; i < weak.size(); i++) table[weak[i]] = 1;

DFS(table,weak,dist,n,0);

int answer = minimum;

return answer==INT\_MAX? -1 : answer;

}

- So I have to change for efficiency.

- It was not I had expected. It had a lot of complexity.

- My algorithm is below.

> 1. choose the number of people we use to check from 1 to dist.size() with decending order.

> 2. with chosen people, do permutation and check if they can check all the outer wall. If we can, return the number of people as an answer.

- see the code.

#include <string>

#include <vector>

#include<iostream>

#include<climits>

#include<algorithm>

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

using namespace std;

inline bool isAllOnes(vector<int> check) {

if (find(check.begin(), check.end(), 0) == check.end()) return true;

return false;

}

bool isCheckedAll(int people, vector<int>& dist,vector<int> weak,int n) {

vector<int> chosen = { dist.rbegin(),next(dist.rbegin(),people) };

do {

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

//cout << "<<people :: " << people << endl;

int start = i;

//cout << " weak[start] : " << weak[start] << endl;

vector<int> check(weak.size(), 0);

check[start] = 1;

for (int j = 0; j < chosen.size(); j++) { //start= 10 length = 14 -> 2

//cout << " cur\_start : " << start << endl;

int length = chosen[j] + weak[start];

if (length >= n) {

length %= n;

int k;

for (k = start; k < check.size(); k++) check[k] = 1;

for (k = 0; k < start; k++) {

if (weak[k] <= length) check[k] = 1;

else break;

}

if (isAllOnes(check)) return true;

else start = k % weak.size();

}

else {

int k;

for (k = start; k < weak.size(); k++) {

if (length >= weak[k]) check[k] = 1;

else break;

}

if (isAllOnes(check)) return true;

else start = k % weak.size();

}

}

}

} while (prev\_permutation(chosen.begin(), chosen.end()));

return false;

}

int solution(int n, vector<int> weak, vector<int> dist) {

vector<int> table(n, 0);

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

int minimum = INT\_MAX;

for (int i = 0; i < weak.size(); i++) table[weak[i]] = 1;

// when we need just one person.

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

int length = dist.back() + weak[i];

if (length >= n) {

if (i == 0 || (weak[i - 1] <= length % n)) return 1;

}

else if (length >= weak.back() && i == 0) return 1;

}

// when we need more than one person.

for (int people = 2; people <= dist.size(); people++) {

if (isCheckedAll(people, dist, weak, n)) return people;

}

return -1; //when we failed to check all the outer wall even if we used all the people.

}

int main() {

//cout<<solution(12, { 1,5,6,10 }, { 1,2,3,4 })<<endl;

//cout << solution(12, { 1,3,4,9,10 }, { 3,5,7 }) << endl;

//cout << solution(50, { 1, 5, 10, 12, 22, 25 }, { 4, 3, 2, 1 }) << endl;

cout << solution(50, { 1, 5, 10, 16, 22, 25 }, { 3,4,6 }) << endl;

//cout << solution(50, { 1 }, { 6 }) << endl;

//cout << solution(12, { 1,5 }, { 2,3,7 });

return 0;

}

[125. **[Programmers– KAKAO 2020 : Moving Block]]**

- At first, I used DFS to implement brute force, but I failed lots of test case and time limit exceeded occurred.

- see the code.

#include <string>

#include <vector>

#include<iostream>

#include<climits>

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

using namespace std;

int minimum = INT\_MAX;

typedef struct cell {

bool wall=false;

bool right = false;

bool down = false;

}cell;

void printTable(vector<vector<int>> table) {

for (vector<int> row : table) {

for (int i : row) {

cout << i << " ";

}

cout << endl;

}

}

void DFS(vector<vector<int>> board, vector<vector<cell>> visit, int bs, int x, int y, bool isRight, int count) {

// initialize

int rx; int ry;

if (isRight) { rx = x; ry = y + 1; visit[x][y].right = true; }

else { rx = x + 1; ry = y; visit[x][y].down = true; }

//cout << "x : " << x << " y : " << y <<" Right : "<<isRight<< endl;

// check arriving

if (rx == bs - 1 && ry == bs - 1) { minimum = min(minimum, count); return; }

// DFS

//cout << " x : " << x << " y : " << y <<" rx : "<<rx<<" ry : "<<ry<< " Right : "<<isRight<<" count : "<<count<< endl;

// isRight

if (isRight) {

//cout << "right" << endl;

//move

//right

if (ry + 1 < bs && board[rx][ry + 1] == 0 && visit[rx][ry].right == false) { DFS(board, visit, bs, rx, ry, true, count + 1); }

//down

if (x + 1 < bs && board[x + 1][y] == 0 && board[rx + 1][ry] == 0 && visit[x + 1][y].right == false) { DFS(board, visit, bs, x + 1, y, true, count + 1); }

//left

if (y - 1 >= 0 && board[x][y-1] == 0 && visit[x][y-1].right == false) { DFS(board, visit, bs, x, y-1, true, count + 1); }

//up

if (x - 1 >= 0 && board[x - 1][y] == 0 && board[rx - 1][y] == 0 && visit[x - 1][y].right == false) { DFS(board, visit, bs, x - 1, y, true, count + 1); }

//rotate

//left axis - clockwise

//cout << "1" << endl;

if (rx < bs && ry < bs &&x+1<bs&& board[rx+1][ry] == 0&& board[x+1][y]==0 && visit[x][y].down == false) { DFS(board, visit, bs, x, y, false, count + 1); }

//left axis - anticlockwise

//cout << "2" << endl;

if (rx < bs && ry << bs && x - 1 >= 0 && board[rx - 1][ry] == 0 && board[x - 1][y] == 0 && visit[x - 1][y].down == false) { DFS(board, visit, bs, x - 1, y, false, count + 1); }

//right axis - clockwise

//cout << "3" << endl;

if (x - 1 >= 0 && board[x - 1][y] == 0 && board[rx - 1][ry] == 0 && visit[rx - 1][ry].down == false) { DFS(board, visit, bs, rx - 1, ry, false, count + 1); }

//right axis - anticlockwise

//cout << "4" << endl;

if (x + 1 < bs && board[x + 1][y] == 0 && board[rx + 1][rx] == 0 && visit[rx][ry].down == false) { DFS(board, visit, bs, rx, ry, false, count + 1); }

}

// is Down

else {

//cout << "down" << endl;

//move

//right

if (y + 1 < bs && board[x][y + 1] == 0 && board[rx][ry + 1] == 0 && visit[x][y + 1].down == false) { DFS(board,visit,bs,x,y+1,false,count+1); }

//down

if (rx + 1 < bs && board[rx + 1][y] == 0 && visit[rx][rx].down == false) { DFS(board, visit, bs, rx, ry, false, count + 1); }

//left

if (y - 1 >= 0 && board[x][y - 1] == 0 && board[rx][ry - 1] == 0 && visit[x][y - 1].down == false) { DFS(board, visit, bs, x, y - 1, false, count + 1); }

//up

if (x - 1 >= 0 && board[x - 1][y] == 0 && visit[x - 1][y].down == false) { DFS(board, visit, bs, x - 1, y, false, count + 1); }

//rotate

//top axis - clockwise

//cout << "5" << endl;

if (y - 1 >= 0 && board[x][y - 1] == 0 && board[rx][ry - 1] == 0 && visit[x][y - 1].right == false) { DFS(board, visit, bs, x , y-1, true, count + 1); }

//top axis - anticlockwise

//cout << "6" << endl;

if (y + 1 < bs && board[x][y+1] == 0 && board[rx][ry + 1] == 0 && visit[x][y].right == false) { DFS(board, visit, bs, x, y, true, count + 1); }

//bottom axis - clockwise

//cout << "7" << endl;

if (y + 1 < bs && board[x][y + 1] == 0 && board[rx][ry + 1] == 0 && visit[rx][ry].right == false) { DFS(board, visit, bs, rx, ry, true, count + 1); }

//bottom axis - anticlockwise

//cout << "8" << endl;

if (y - 1 >= 0 && board[x][y - 1] == 0 && board[rx][ry - 1] == 0 && visit[rx][ry - 1].right == false) { DFS(board, visit, bs, rx, ry - 1, true, count + 1); }

}

}

int solution(vector<vector<int>> board) {

int answer = 0;

int bs = board.size();

vector<vector<cell>> visit(bs,vector<cell>(bs));

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

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

if (board[i][j] == 1) visit[i][j].wall = true;

}

}

//printTable(board);

//cout << endl;

DFS(board, visit, bs,0,0,true,0);

answer = minimum;

return answer;

}

int main() {

cout << solution({ {0, 0, 0, 1, 1}, {0, 0, 0, 1, 0}, {0, 1, 0, 1, 1}, {1, 1, 0, 0, 1}, {0, 0, 0, 0, 0} });

return 0;

}

- so I had to change the algorithm to BFS and revise some codes but time limit exceeded occurred also.

- below is the BFS code I used.

int BFS(vector<vector<int>> board, vector<vector<cell>> visit, int bs, int x, int y, bool isRight) {

queue < pair<vector<vector<cell>>, vector<int>>> que;

int count = 0;

int rx, ry;

que.push(make\_pair(visit, vector<int>{x, y, isRight, count}));

while (!que.empty()) {

visit = que.front().first;

vector<int> front = que.front().second;

x = front[0]; y = front[1]; isRight = front[2]; count = front[3];

que.pop();

if (isRight) { rx = x; ry = y + 1; visit[x][y].right = true; }

else { rx = x + 1; ry = y; visit[x][y].down = true; }

if (rx == bs - 1 && ry == bs - 1) return count;

if (isRight) {

//move

//right

if (ry + 1 < bs && board[rx][ry + 1] == 0 && visit[rx][ry].right == false)

{ que.push(make\_pair(visit,vector<int>{ rx, ry, true, count + 1 })); }

//down

if (x + 1 < bs && board[x + 1][y] == 0 && board[rx + 1][ry] == 0 && visit[x + 1][y].right == false)

{ que.push(make\_pair(visit, vector<int>{ x+1, y, true, count + 1 })); }

//left

if (y - 1 >= 0 && board[x][y - 1] == 0 && visit[x][y - 1].right == false)

{ que.push(make\_pair(visit, vector<int>{ x, y-1, true, count + 1 })); }

//up

if (x - 1 >= 0 && board[x - 1][y] == 0 && board[rx - 1][y] == 0 && visit[x - 1][y].right == false)

{ que.push(make\_pair(visit, vector<int>{ x-1, y, true, count + 1 })); }

//rotate

//left axis - clockwise

if (x+1<bs && board[rx + 1][ry] == 0 && board[x + 1][y] == 0 && visit[x][y].down == false)

{ que.push(make\_pair(visit, vector<int>{ x, y, false, count + 1 })); }

//left axis - anticlockwise

if (x - 1 >= 0 && board[rx - 1][ry] == 0 && board[x - 1][y] == 0 && visit[x - 1][y].down == false)

{que.push(make\_pair(visit, vector<int>{ x-1, y, false, count + 1 })); }

//right axis - clockwise

if (x - 1 >= 0 && board[x - 1][y] == 0 && board[rx - 1][ry] == 0 && visit[rx - 1][ry].down == false)

{ que.push(make\_pair(visit, vector<int>{ rx-1, ry, false, count + 1 })); }

//right axis - anticlockwise

if (x + 1 < bs && board[x + 1][y] == 0 && board[rx + 1][ry] == 0 && visit[rx][ry].down == false)

{ que.push(make\_pair(visit, vector<int>{ rx, ry, false, count + 1 }));}

}

else {

//move

//right

if (y + 1 < bs && board[x][y + 1] == 0 && board[rx][ry + 1] == 0 && visit[x][y + 1].down == false)

{ que.push(make\_pair(visit, vector<int>{ x, y+1, false, count + 1 })); }

//down

if (rx + 1 < bs && board[rx + 1][y] == 0 && visit[rx][ry].down == false)

{ que.push(make\_pair(visit, vector<int>{ rx, ry, false, count + 1 })); }

//left

if (y - 1 >= 0 && board[x][y - 1] == 0 && board[rx][ry - 1] == 0 && visit[x][y - 1].down == false)

{ que.push(make\_pair(visit, vector<int>{ x, y-1, false, count + 1 })); }

//up

if (x - 1 >= 0 && board[x - 1][y] == 0 && visit[x - 1][y].down == false)

{ que.push(make\_pair(visit, vector<int>{ x-1, y, false, count + 1 })); }

//rotate

//top axis - clockwise

if (y - 1 >= 0 && board[x][y - 1] == 0 && board[rx][ry - 1] == 0 && visit[x][y - 1].right == false)

{ que.push(make\_pair(visit, vector<int>{ x, y-1, true, count + 1 })); }

//top axis - anticlockwise

if (y + 1 < bs && board[x][y + 1] == 0 && board[rx][ry + 1] == 0 && visit[x][y].right == false)

{ que.push(make\_pair(visit, vector<int>{ x, y, true, count + 1 })); }

//bottom axis - clockwise

if (y + 1 < bs && board[x][y + 1] == 0 && board[rx][ry + 1] == 0 && visit[rx][ry].right == false)

{ que.push(make\_pair(visit, vector<int>{ rx, ry, true, count + 1 })); }

//bottom axis - anticlockwise

if (y - 1 >= 0 && board[x][y - 1] == 0 && board[rx][ry - 1] == 0 && visit[rx][ry - 1].right == false)

{ que.push(make\_pair(visit, vector<int>{ rx, ry-1, true, count + 1 })); }

}

}

return 0;

}

- so I made NewBFS, I got all test cases’ mark but, time limit exceeded occurred at tc 13, 14.

- see the NewBFS code.

int NewBFS(vector<vector<cell>> visit) {

int bs = visit.size();

queue<vector<int>> que;

que.push(vector<int>{0, 0, 1,0});//x,y, isRIght;

while ( !que.empty()) {

int x = que.front()[0]; int y = que.front()[1]; bool isRight = que.front()[2]; int count = que.front()[3];

que.pop();

if (isRight && x == bs - 1 && y + 1 == bs - 1) return count;

else if (x + 1 == bs - 1 && y == bs - 1) return count;

isRight ? visit[x][y].right = false : visit[x][y].down = false;

if (isRight) {

//move

if (y + 2 < bs && visit[x][y + 1].right&& visit[x][y+2].wall) que.push(vector<int>{x, y + 1, true, count + 1});

if (y-1>=0 && visit[x][y-1].right && visit[x][y-1].wall) que.push( vector<int>{x, y -1, true, count + 1});

if(x+1<bs && visit[x+1][y].wall && visit[x+1][y+1].wall && visit[x+1][y].right) que.push(vector<int>{x+1, y, true, count + 1});

if(x-1>=0 && visit[x-1][y].wall && visit[x-1][y+1].wall && visit[x-1][y].right) que.push(vector<int>{x-1, y, true, count + 1});

//rotate

if (x + 1 < bs && visit[x + 1][y].wall && visit[x + 1][y + 1].wall ) {

if( visit[x][y].down) que.push( vector<int>{x, y, false, count + 1});

if( visit[x][y + 1].down) que.push(vector<int>{x, y+1, false, count + 1});

}

if (x - 1 >= 0 && visit[x - 1][y].wall && visit[x - 1][y + 1].wall ) {

if( visit[x - 1][y].down) que.push( vector<int>{x - 1, y, false, count + 1});

if( visit[x - 1][y + 1].down) que.push(vector<int>{x - 1, y + 1, false, count + 1});

}

}

else {

//move

if(y+1<bs && visit[x][y+1].wall && visit[x+1][y+1].wall && visit[x][y+1].down) que.push(vector<int>{x, y+1, false, count + 1});

if(y-1>=0 && visit[x][y-1].wall && visit[x+1][y-1].wall && visit[x][y-1].down) que.push(vector<int>{x, y-1, false, count + 1});

if(x+2<bs && visit[x+2][y].wall && visit[x+1][y].down) que.push(vector<int>{x+1, y, false, count + 1});

if(x-1>=0 && visit[x-1][y].wall && visit[x-1][y].down) que.push(vector<int>{x-1, y, false, count + 1});

//rotate

if (y + 1 < bs && visit[x][y + 1].wall && visit[x + 1][y + 1].wall) {

if( visit[x][y].right) que.push( vector<int>{x, y, true, count + 1});

if( visit[x+1][y].right) que.push( vector<int>{x+1, y, true, count + 1});

}

if (y - 1 >= 0 && visit[x][y - 1].wall && visit[x + 1][y - 1].wall) {

if( visit[x][y-1].right) que.push(vector<int>{x, y-1, true, count + 1});

if( visit[x+1][y-1].right) que.push(vector<int>{x+1, y-1, true, count + 1});

}

}

}

return 0;

}

- I resolved the time limit exceeded at tc 13,14 by changing the location of allocating visit state.

- before I changed it, I allocate visit state when I poped the queue.front().

- but I changed it before I pushed a state into queue.

- see the code.

#include <string>

#include <vector>

#include<iostream>

#include<queue>

using namespace std;

typedef struct cell {

bool wall = true;

bool right = true;

bool down = true;

}cell;

void printTable(vector<vector<int>> table) {

for (vector<int> row : table) {

for (int i : row) {

cout << i << " ";

}

cout << endl;

}

}

int NewBFS(vector<vector<cell>> visit) {

int bs = visit.size();

queue<vector<int>> que;

que.push(vector<int>{0, 0, 1,0});//x,y, isRIght;

while ( !que.empty()) {

int x = que.front()[0]; int y = que.front()[1]; bool isRight = que.front()[2]; int count = que.front()[3];

que.pop();

if (isRight && x == bs - 1 && y + 1 == bs - 1) return count;

else if (x + 1 == bs - 1 && y == bs - 1) return count;

if (isRight) {

//move

if (y + 2 < bs && visit[x][y + 1].right && visit[x][y + 2].wall) {

visit[x][y+1].right = false;

que.push(vector<int>{x, y + 1, true, count + 1});

}

if (y - 1 >= 0 && visit[x][y - 1].right && visit[x][y - 1].wall) {

visit[x][y -1].right = false;

que.push(vector<int>{x, y - 1, true, count + 1});

}

if (x + 1 < bs && visit[x + 1][y].wall && visit[x + 1][y + 1].wall && visit[x + 1][y].right) {

visit[x+1][y].right = false;

que.push(vector<int>{x + 1, y, true, count + 1});

}

if (x - 1 >= 0 && visit[x - 1][y].wall && visit[x - 1][y + 1].wall && visit[x - 1][y].right) {

visit[x -1][y].right = false;

que.push(vector<int>{x - 1, y, true, count + 1});

}

//rotate

if (x + 1 < bs && visit[x + 1][y].wall && visit[x + 1][y + 1].wall ) {

if (visit[x][y].down) {

visit[x][y].down = false;

que.push(vector<int>{x, y, false, count + 1});

}

if (visit[x][y + 1].down) {

visit[x][y+1].down = false;

que.push(vector<int>{x, y + 1, false, count + 1});

}

}

if (x - 1 >= 0 && visit[x - 1][y].wall && visit[x - 1][y + 1].wall ) {

if (visit[x - 1][y].down) {

visit[x-1][y].down = false;

que.push(vector<int>{x - 1, y, false, count + 1});

}

if (visit[x - 1][y + 1].down) {

visit[x - 1][y+1].down = false;

que.push(vector<int>{x - 1, y + 1, false, count + 1});

}

}

}

else {

//move

if (y + 1 < bs && visit[x][y + 1].wall && visit[x + 1][y + 1].wall && visit[x][y + 1].down) {

visit[x][y + 1].down = false;

que.push(vector<int>{x, y + 1, false, count + 1});

}

if (y - 1 >= 0 && visit[x][y - 1].wall && visit[x + 1][y - 1].wall && visit[x][y - 1].down) {

visit[x][y - 1].down = false;

que.push(vector<int>{x, y - 1, false, count + 1});

}

if (x + 2 < bs && visit[x + 2][y].wall && visit[x + 1][y].down) {

visit[x+1][y].down = false;

que.push(vector<int>{x + 1, y, false, count + 1});

}

if (x - 1 >= 0 && visit[x - 1][y].wall && visit[x - 1][y].down) {

visit[x -1][y].down = false;

que.push(vector<int>{x - 1, y, false, count + 1});

}

//rotate

if (y + 1 < bs && visit[x][y + 1].wall && visit[x + 1][y + 1].wall) {

if (visit[x][y].right) {

visit[x ][y].right = false;

que.push(vector<int>{x, y, true, count + 1});

}

if (visit[x + 1][y].right) {

visit[x+1][y].right = false;

que.push(vector<int>{x + 1, y, true, count + 1});

}

}

if (y - 1 >= 0 && visit[x][y - 1].wall && visit[x + 1][y - 1].wall) {

if (visit[x][y - 1].right) {

visit[x ][y-1].right = false;

que.push(vector<int>{x, y - 1, true, count + 1});

}

if (visit[x + 1][y - 1].right) {

visit[x+1][y - 1].right = false;

que.push(vector<int>{x + 1, y - 1, true, count + 1});

}

}

}

}

return 0;

}

int solution(vector<vector<int>> board) {

int answer = 0;

int bs = board.size();

vector<vector<cell>> visit(bs,vector<cell>(bs));

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

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

if (board[i][j] == 1) visit[i][j].wall = false;

}

}

answer = NewBFS(visit);

return answer;

}

int main() {

cout << solution({ {0, 0, 0, 1, 1}, {0, 0, 0, 1, 0}, {0, 1, 0, 1, 1}, {1, 1, 0, 0, 1}, {0, 0, 0, 0, 0} });

return 0;

}

[126. **[Programmers– KAKAO 2019 :Open Chat Room]]**

- it was an easy problem, but I needed to handle complex unordered\_map structure. it was quite confusing.

- see the code.

#include <string>

#include <vector>

#include <sstream>

#include <iostream>

#include<unordered\_map>

using namespace std;

vector<string> res;

int index = 0;

unordered\_map<string, pair<string, vector<int>>> table;

void Tokenize(string s) {

stringstream ss(s);

string token;

getline(ss, token, ' '); string in\_out = token;

getline(ss, token, ' '); string uid = token;

getline(ss, token, ' '); string name = token;

if (in\_out == "Enter") res.push\_back("님이 들어왔습니다.");

else if (in\_out == "Leave") res.push\_back("님이 나갔습니다.");

if (table[uid].second.empty()) {

if(in\_out=="Enter") table[uid] = make\_pair(name, vector<int>{index++});

else if(in\_out=="Change") table[uid] = make\_pair(name, vector<int>{});

}

else {

if (in\_out == "Change") table[uid].first = name;

else if(in\_out=="Enter"){

table[uid].first = name;

table[uid].second.push\_back(index++);

}

else {

table[uid].second.push\_back(index++);

}

}

}

vector<string> solution(vector<string> record) {

vector<string> answer;

for (string s : record) Tokenize(s);

answer.assign(index, "");

for (unordered\_map<string, pair<string, vector<int>>>::iterator iter = table.begin(); iter != table.end(); iter++) {

for (int i : iter->second.second) {

answer[i] = iter->second.first + res[i];

}

}

for (string s : answer)

cout << s << endl;

return answer;

}

int main() {

solution(vector<string>{ "Enter uid1234 Muzi", "Enter uid4567 Prodo","Change uid4567 Ryan", "Leave uid1234", "Enter uid1234 Prodo" });

return 0;

}

[127. **[Programmers– KAKAO 2019 :Fail Rate]]**

- This problem’s point was to find order of index using fail rate.

- I used map, since map dose sort automatically.

- see the code.

#include <string>

#include <vector>

#include<algorithm>

#include <iostream>

#include<map>

using namespace std;

vector<int> makeOrder(vector<double>& failure,vector<int> answer) {

map<double, vector<int>> res;

for (int i = 1; i < failure.size(); i++) {

res[failure[i]].push\_back(i);

}

for (map<double, vector<int>>::reverse\_iterator iter = res.rbegin(); iter != res.rend(); iter++) {

for (int i = 0; i < iter->second.size(); i++) {

answer.push\_back(iter->second[i]);

}

}

return answer;

}

vector<int> solution(int N, vector<int> stages) {

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

vector<double> failure(N+1); //index starts from 1 not 0.

double cur\_pass = 0;

int i;

for (i = stages.size() - 1; i >= 0; i--) {

if (stages[i] == N + 1) cur\_pass++;

else break;

}

double cur\_stage = 0;

for (i; i >= 0;) {

if (stages[i] == N) {

cur\_stage++;

if (i == 0) {

cur\_pass += cur\_stage;

failure[N] = cur\_stage / cur\_pass;

break;

}

i--;

}

else {

cur\_pass += cur\_stage;

failure[N] = cur\_stage / cur\_pass;

N--;

cur\_stage = 0;

}

}

vector<int> answer= makeOrder(failure, vector<int>{});

return answer;

}

int main() {

solution(5, { 2, 1, 2, 6, 2, 4, 3, 3 });

return 0;

}

[128. **[Programmers– KAKAO 2019 :Candidate Key]]**

- it was fxxkcing difficult for me. Indeed, it was not that difficult. but I spent so much time.

- since I had to make all the possible combination first. it was okay even though I spent 1 hour or so.

- But to satisfy minimality, I had to remove duplicate keys, it took a lot of time.

- HOWEVER!!! the main problem that took so much time was!!!! I’ve read problem in a wrong way…

- I thought the problem want me to return maximum size of a possible cadidate key. Sadly, it wans’t.

- I just had to return the number of cadidate keys which is valid….

- oh my poor hand, brain, time… :(

- by the way, I used bitset to remove duplicates and BFS to make all the possible combination. when I made BFS, the queue structure was quite complex, since there were quite many condition.

- see the code.

#include <string>

#include <vector>

#include<iostream>

#include<queue>

#include<unordered\_set>

#include<bitset>

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

using namespace std;

int n; int m;

vector<vector<int>> valid;

vector<vector<int>> all;

bool isValid(vector<string> temp) {

unordered\_set<string> check;

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

check.insert(temp[i]);

//cout << "size : " << check.size() << endl;

if (check.size() == n) return true;

return false;

}

void Combination(vector<vector<string>> table) {

queue<pair<vector<string>,pair<vector<int>,int>>> que;

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

que.push(make\_pair(table[i], make\_pair(vector<int>{ i },i+1)));

}

while (!que.empty()) {

vector<string> temp = que.front().first;

vector<int> indice = que.front().second.first;

int start = que.front().second.second;

que.pop();

/\*cout << "indice : ";

for (int i : indice)

cout << i << " ";

cout << endl;

for (string s : temp)

cout << s << " ";

cout << endl;\*/

if (isValid(temp)) valid.push\_back(indice);

all.push\_back(indice);

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

vector<string> temp\_table = temp;

vector<int> temp\_indice = indice;

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

temp\_table[j] +=" "+ table[i][j];

}

temp\_indice.push\_back(i);

que.push(make\_pair(temp\_table, make\_pair(temp\_indice, i + 1)));

}

}

}

void printValid() {

for (vector<int> row : valid) {

for (int i : row) {

cout << i << " ";

}

cout << endl;

}

}

void printAll() {

for (vector<int> row : all) {

for (int i : row) {

cout << i << " ";

}

cout << endl;

}

}

void Minimalize() {

vector<bitset<20>> bits(valid.size());

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

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

bits[i][valid[i][j]] = 1;

}

}

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

bitset<20> cur\_bit = bits[i];

for (int j = i + 1; j < bits.size();) {

bitset<20> temp = cur\_bit & bits[j];

if (temp == cur\_bit) {

valid.erase(next(valid.begin(), j));

bits.erase(next(bits.begin(), j));

}

else j++;

}

}

}

int solution(vector<vector<string>> relation) {

n = relation.size();

m = relation[0].size();

vector<vector<string>> table(m);

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

vector<string> temp;

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

temp.push\_back(relation[i][j]);

}

table[j] = temp;

}

Combination(table);

Minimalize();

int answer = 0;

return valid.size();

}

int main() {

cout<<solution({

{"100", "ryan", "music", "2"},

{"200", "apeach", "math", "2"},

{"300", "tube", "computer", "3"},

{"400", "con", "computer", "4"},

{"500", "muzi", "music", "3"},

{"600", "apeach", "music", "2"}} );

return 0;

}

[129. **[Programmers– KAKAO 2019 :Muji’s Muckbang Live]]**

- this problem has efficiency test as well.

- as you guess, I passed accuracy tests but not efficiency tests.

- below code is the code passing accuracy tests.

#include <string>

#include <vector>

#include<iostream>

#include<algorithm>

#include<map>

using namespace std;

vector<int> ReduceNumber(vector<int> food\_times,int eat\_number) {

for (long i = 0; i < food\_times.size(); i++) {

if (food\_times[i] == 0) continue;

else food\_times[i] -= eat\_number;

}

return food\_times;

}

void printTable(vector<int> food\_times) {

for (int i : food\_times)

cout << i << " ";

cout << endl<<endl;

}

int solution(vector<int> food\_times, long long k) {

int answer = 0;

int rest\_food = food\_times.size();

long long cur\_index = 0;

long long eat\_number = 0;

map<long long, int> min\_table;

for (int i = 0; i < food\_times.size(); i++) min\_table[food\_times[i]] += 1;

map<long long, int>::iterator iter = min\_table.begin();

long long min\_food = iter->first;

int min\_food\_size = iter->second;

long long accumulated\_eat\_number = 0;

//cout << "Begining" << endl;

//printTable(food\_times);

while (true) {

if (k >= rest\_food) {

eat\_number++;

k -= rest\_food;

if (eat\_number == min\_food) {

accumulated\_eat\_number += eat\_number;

food\_times = ReduceNumber(food\_times, eat\_number);

//printTable(food\_times);

rest\_food -= min\_food\_size;

iter++;

eat\_number = 0;

if (iter != min\_table.end()) {

min\_food = iter->first-accumulated\_eat\_number;

//cout << "next min\_food : " << min\_food << endl;

min\_food\_size = iter->second;

}

else {

//cout << "여기 처리" << endl;

return -1;

}

}

}

else {

long long i;

if(eat\_number) food\_times=ReduceNumber(food\_times, eat\_number);

//cout << "<<Last Traversal>>"<< endl;

//cout << " - Remain k : " << k << " - Remain eat\_number : " << eat\_number << endl;

//printTable(food\_times);

for (i = 0; i <food\_times.size() && k>0; i++) {

if (food\_times[i] == 0) continue;

else {

food\_times[i]--;

k--;

}

}

//cout << "!!!Final!!!" << endl;

//printTable(food\_times);

cur\_index = i%food\_times.size();

bool flag = true;

for (long i = cur\_index; i < food\_times.size(); i++) if (food\_times[i]) { cur\_index = i; flag = false; break; }

if (flag) for (long i = 0; i < food\_times.size(); i++) if (food\_times[i]) { cur\_index = i; flag = false; break; }

if (flag && i == food\_times.size()) return -1;

break;

}

}

return cur\_index+1;

}

int main() {

cout<<solution({ 1,2,3},1);

return 0;

}

- Let’s find another algorithm to pass efficiency test!

- I refered to a site to pass the efficiency tests.

- there are so many geinous…. the point is I don’t need to reduce food’s count. just sort with indice and if Muji can eat a current minimum food, find next food, and so on…

- see the code.

#include <string>

#include <vector>

#include<iostream>

#include<algorithm>

#include<map>

#include<bitset>

using namespace std;

int solution(vector<int> foods, long long k) {

vector<pair<int, int>> food\_times;

for (int i = 0; i < foods.size(); i++) food\_times.push\_back(make\_pair(foods[i], i + 1));

int fs = foods.size();

sort(food\_times.begin(), food\_times.end());

long long accumulated\_min\_food=0;

for (vector<pair<int, int>>::iterator iter = food\_times.begin(); iter != food\_times.end(); iter++) {

long long min\_food = (long long)(iter->first - accumulated\_min\_food) \* fs;

if (min\_food == 0) {

fs--;

continue;

}

else if (min\_food <= k) {

k -= min\_food;

accumulated\_min\_food = iter->first;

fs--;

}

else {

k %= fs;

sort(iter, food\_times.end(), [](pair<int, int> a, pair<int, int> b) {return a.second < b.second; });

return (iter + k)->second;

}

}

return -1;

}

int main() {

cout<<solution({10,6,4,3,2,5,15,9,6,5,4,3,2},60);

return 0;

}

[130. **[Programmers– KAKAO 2019 :Finding Path Game]]**

- this problem’s point was the way to make tree.

- I spent 3 hours or so to make tree.

- I have been stuck since I barely came up with a method to make tree. the given vector sholud be splited before making tree.

- see the code.

#include <string>

#include <vector>

#include<iostream>

#include<algorithm>

#include<queue>

using namespace std;

vector<int> preorder;

vector<int> postorder;

struct Node {

int x;

int y;

int index;

Node\* left=nullptr;

Node\* right=nullptr;

Node(int \_x, int \_y, int \_index) {

x = \_x; y = \_y; index = \_index;

};

~Node() {}

};

pair<vector<pair<vector<int>, int>>, vector<pair<vector<int>, int>>> LRSplit(vector<pair<vector<int>, int>> table,int mid) {

vector < pair<vector<int>, int>> left;

vector < pair<vector<int>, int>> right;

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

if (table[i].first[0] < mid) left.push\_back(table[i]);

else right.push\_back(table[i]);

}

return make\_pair(left, right);

}

void makeTree(Node\* root,vector<pair<vector<int>,int>> table) {

table.erase(table.begin());

if (table.empty()) return;

pair<vector<pair<vector<int>, int>>, vector<pair<vector<int>, int>>> splited = LRSplit(table, root->x);

vector < pair<vector<int>, int>> left = splited.first;

vector < pair<vector<int>, int>> right = splited.second;

if(!left.empty())root->left = new Node(left[0].first[0], left[0].first[1], left[0].second);

if(!right.empty())root->right = new Node(right[0].first[0], right[0].first[1], right[0].second);

if(root->left) makeTree(root->left, left);

if(root->right) makeTree(root->right, right);

}

void printTree(vector < pair<vector<int>, int>> tree) {

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

cout << tree[i].first[0] << "," << tree[i].first[1] << " ";

}

cout << endl;

}

void PreOrderSearch(Node\* root) {

if (!root) return;

preorder.push\_back(root->index);

PreOrderSearch(root->left);

PreOrderSearch(root->right);

}

void PostOrderSearch(Node\* root) {

if (!root) return;

PostOrderSearch(root->left);

PostOrderSearch(root->right);

postorder.push\_back(root->index);

}

void printAnswer(vector<vector<int>>& answer) {

for (vector<int> row : answer) {

for (int i : row) {

cout << i << " ";

}

cout << endl;

}

}

vector<vector<int>> solution(vector<vector<int>> nodeinfo) {

vector<vector<int>> answer;

vector<pair<vector<int>, int>> table;

for (int i = 0; i < nodeinfo.size(); i++) table.push\_back(make\_pair(nodeinfo[i], i + 1));

sort(table.begin(), table.end(), [](pair<vector<int>, int> a, pair<vector<int>, int> b) {return a.first[1] > b.first[1];}); //y first sort

Node\* root = new Node(table[0].first[0], table[0].first[1], table[0].second);

makeTree(root, table);

PreOrderSearch(root);

PostOrderSearch(root);

answer.push\_back(preorder);

answer.push\_back(postorder);

return answer;

}

int main() {

solution({{5, 3}, {11, 5}, {13, 3}, {3, 5}, {6, 1}, {1, 3}, {8, 6}, {7, 2}, {2, 2}} );

return 0;

}

[131. **[Programmers– KAKAO 2019 :Matching Score]]**

- I hate fucking parsing string!!! there is too many condition to check.

- I spent almost 4 hour to parse this fucking problem.

- see the code.

#include <string>

#include <vector>

#include<iostream>

#include<algorithm>

#include<sstream>

#include<map>

#include<unordered\_set>

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

using namespace std;

map<string, pair<unordered\_set<string>,vector<int>>> links; // URL - <{in links},{base,out links,index}>

vector<string> URL\_list;

vector<string> Lower(vector<string> pages) {

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

std::transform(pages[i].begin(), pages[i].end(), pages[i].begin(), ::tolower);

}

return pages;

}

string RemoveDelimeter(string s) {

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

if ('a' <= s[i] && s[i] <= 'z') continue;

else s[i] = ' ';

}

return s;

}

vector<int> FindWord(vector<string> pages,string word) {

vector<int> baseScore(pages.size(),0);

int i = 0;

for (string s : pages) {

s = RemoveDelimeter(s);

stringstream ss(s);

string token;

while (getline(ss, token,' ')){

if (token == word) baseScore[i]++;

}

i++;

}

return baseScore;

}

void printPage(string s) {

cout << s << endl;

}

void GetURL(vector<string> pages) {

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

string cur\_URL = "";

string now = pages[i];

while (cur\_URL == "") {

int start = now.find("<meta") + 5;

now = now.substr(start);

int last = now.find(">");

start = now.find("https://");

if (start > last) continue;

now = now.substr(start);

stringstream ss(now);

string token;

getline(ss, token, '\"');

cur\_URL = token;

URL\_list.push\_back(cur\_URL);

links[cur\_URL] = make\_pair(unordered\_set<string>{}, vector<int>{0, 0, i});

}

}

}

void GetLinks(vector<string> pages) {

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

string now = pages[i];

string cur\_URL = URL\_list[i];

int start = now.find("<body>");

int last = now.find("</body>");

now = now.substr(start, last - start + 7); //partition of body tag.

while(true) {

start = now.find("<a href");

if (start == string::npos) break;

now = now.substr(start);

start = 0;

last = now.find(">");

if (start > last) break;

start = now.find("https://");

if (start != string::npos) links[cur\_URL].second[1]++; //outer ++

now = now.substr(start);

stringstream ss(now);

string token;

getline(ss, token, '\"');

if (links.find(token) != links.end()) links[token].first.insert(cur\_URL);

};

}

}

double GetLinkScore(unordered\_set<string> inLinks) {

unordered\_set<string>::iterator iter = inLinks.begin();

double sum = 0;

for (; iter != inLinks.end(); iter++) {

//cout << links[\*iter].second[0] << "," << links[\*iter].second[1] << endl;

//cout << "result : " << double(links[\*iter].second[0]) / double(links[\*iter].second[1]) << endl;

sum+=double(links[\*iter].second[0]) / double(links[\*iter].second[1]);

}

//cout << sum << endl;

return sum;

}

void printLinks() {

for (auto iter = links.begin(); iter != links.end(); iter++) {

cout << "URL : " << iter->first << endl;

cout << " base score : " << iter->second.second[0];

cout << " out links : " << iter->second.second[1];

cout << " in links : ";

for (auto iter2 = iter->second.first.begin(); iter2 != iter->second.first.end(); iter2++) {

cout << \*iter2 << endl;

}

cout << endl << endl;;

}

}

int solution(string word, vector<string> pages) {

//for (string s : pages) printPage(s);

int answer = 0;

transform(word.begin(), word.end(), word.begin(), ::tolower);

pages = Lower(pages); // make it lowercase

vector<int> baseScore=FindWord(pages,word); // get base score

GetURL(pages);

GetLinks(pages);

map<string, pair<unordered\_set<string>, vector<int>>>::iterator iter = links.begin();

for (; iter != links.end(); iter++) iter->second.second[0] = baseScore[iter->second.second[2]];

//printLinks();

vector<pair<int,double>> res(pages.size());//index, sum of score;

iter = links.begin();

for (int i = 0; iter != links.end();i++, iter++) {

res[i] = make\_pair(iter->second.second[2], iter->second.second[0] + GetLinkScore(iter->second.first));

}

sort(res.begin(), res.end(), [](pair<int, double> a, pair<int, double> b) { return a.second > b.second; });

double maximum = res[0].second;

int index=res[0].first;

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

if (res[i].second != maximum) break;

index = min(index, res[i].first);

}

return index;

}

int main() {

cout<<solution("blind",{ "<html lang=\"ko\" xml:lang=\"ko\" xmlns=\"http://www.w3.org/1999/xhtml\">\n<head>\n <meta charset=\"utf-8\">\n <meta property=\"og:url\" content=\"https://a.com\"/>\n</head> \n<body>\nBlind Lorem Blind ipsum dolor Blind test sit amet, consectetur adipiscing elit. \n<a href=\"https://b.com\"> Link to b </a>\n</body>\n</html>", "<html lang=\"ko\" xml:lang=\"ko\" xmlns=\"http://www.w3.org/1999/xhtml\">\n<head>\n <meta charset=\"utf-8\">\n <meta property=\"og:url\" content=\"https://b.com\"/>\n</head> \n<body>\nSuspendisse potenti. Vivamus venenatis tellus non turpis bibendum, \n<a href=\"https://a.com\"> Link to a </a>\nblind sed congue urna varius. Suspendisse feugiat nisl ligula, quis malesuada felis hendrerit ut.\n<a href=\"https://c.com\"> Link to c </a>\n</body>\n</html>", "<html lang=\"ko\" xml:lang=\"ko\" xmlns=\"http://www.w3.org/1999/xhtml\">\n<head>\n <meta charset=\"utf-8\">\n <meta property=\"og:url\" content=\"https://c.com\"/>\n</head> \n<body>\nUt condimentum urna at felis sodales rutrum. Sed dapibus cursus diam, non interdum nulla tempor nec. Phasellus rutrum enim at orci consectetu blind\n<a href=\"https://a.com\"> Link to a </a>\n</body>\n</html>" })<<endl;

//cout << solution("Muzi", { "<html lang=\"ko\" xml:lang=\"ko\" xmlns=\"http://www.w3.org/1999/xhtml\">\n<head>\n <meta charset=\"utf-8\">\n <meta property=\"og:url\" content=\"https://careers.kakao.com/interview/list\"/>\n</head> \n<body>\n<a href=\"https://programmers.co.kr/learn/courses/4673\"></a>#!MuziMuzi!)jayg07con&&\n\n</body>\n</html>", "<html lang=\"ko\" xml:lang=\"ko\" xmlns=\"http://www.w3.org/1999/xhtml\">\n<head>\n <meta charset=\"utf-8\">\n <meta property=\"og:url\" content=\"https://www.kakaocorp.com\"/>\n</head> \n<body>\ncon%\tmuzI92apeach&2<a href=\"https://hashcode.co.kr/tos\"></a>\n\n\t^\n</body>\n</html>" })<<endl;

return 0;

}

[132. **[Programmers– KAKAO 2019 :Block Game]]**

- Even though it was level 4 problem, it was easy for me.

- I was happy it was not a fucking string parsing :)

- the reason why this problem was easy is there is no time limitation. so I can use some functions roughly and repeatedly.

- see the code.

#include <string>

#include <vector>

#include<iostream>

#include<map>

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

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

using namespace std;

int res = 0;

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

for (vector<int> row : table) {

for (int i : row)

cout << i << " ";

cout << endl;

}

cout << endl;

}

bool isRectangular(vector<vector<int>>& table,int tx,int ty,int bx,int by,int block) {

if (bx == tx && by == ty) return false;

//cout << "변환 전 : " << block << endl;

//cout << tx << ty << " , " << bx << by << endl;

//printTable(table);

vector<vector<int>> temp = table;

for (int i = tx; i <= bx; i++) {

for (int j = ty; j <= by; j++) {

if (block != temp[i][j]) {

//cout << i << "," << j << ","<<temp[i][j] << endl;

return false;

}

temp[i][j] = 0;

}

}

//cout << "변환 후 : "<<block << endl;

//printTable(temp);

table = temp;

res++;

return true;

}

void SetInformation(vector<vector<int>> board, int bs, vector<int>& top,vector<int>& height,map<int,int>& blocks\_top) {

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

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

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

blocks\_top[board[i][j]] = i;

top[j] = board[i][j];

height[j] = i;

}

}

}

void getLimit(int& tx, int& ty, int& bx, int& by, int block,vector<vector<int>>& table) {

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

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

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

tx = min(tx, i); ty = min(ty, j); bx = max(bx, i); by = max(by, j);

}

}

}

}

int solution(vector<vector<int>> board) {

int answer = 0;

int bs = board.size();

vector<int> top(bs, 0); //a columns top block

vector<int> height(bs, 0); // a columns top height

map<int, int> blocks\_top; // block number - the block's top height;

SetInformation(board, bs, top, height,blocks\_top);

for (map<int, int>::iterator iter = blocks\_top.begin(); iter != blocks\_top.end();) {

int tx=999, ty=999, bx=-1, by=-1;

int cur\_block = iter->first;

vector<vector<int>> cur\_table = board;

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

if (top[j] == cur\_block && height[j] > iter->second) {

for (int i = height[j] - 1; i >= iter->second; i--) {

cur\_table[i][j] = cur\_block;

}

}

}

getLimit(tx, ty, bx, by, cur\_block, cur\_table); // get the block's boundary

if (isRectangular(cur\_table, tx, ty, bx, by, cur\_block)) { //check if the block is rectangular

board = cur\_table;

blocks\_top.erase(cur\_block); //if it is rectangular, remove that blcok from the board.

SetInformation(board, bs, top, height, blocks\_top); //reset the information.

iter = blocks\_top.begin();

}

else iter++;

}

return res;

}

int main() {

cout << solution({

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

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

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

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

{0, 0, 0, 0, 0, 0, 4, 0, 0, 0},

{0, 0, 0, 0, 0, 4, 4, 0, 0, 0},

{0, 0, 0, 0, 3, 0, 4, 0, 0, 0},

{0, 0, 0, 2, 3, 0, 0, 0, 5, 5},

{1, 2, 2, 2, 3, 3, 0, 0, 0, 5},

{1, 1, 1, 0, 0, 0, 0, 0, 0, 5}} );

return 0;

}

[133. **[Programmers– KAKAO 2018 :Harvest Fest Traffic]]**

- time handling is the key of this problem.

- finding an efficient algorithm for maximum number of traffic during one second was quite hard but not a terrible.

- but handling time was terrible. First, I handle hour, minute, sec respectively. but I didn’t need that complexity. so I convert all the thing to second. I mean, hour \* 3600000, min \* 60000, sec\* 1000.

since, they gave second to 3 decimal places.

- the finding algorithm I mentioned avobe is below

> 1. take a end time of traffic.

> 2. add 1 sec.

> 3. iterate all the traffic and check there is traffic between the duration.

- see the code. p.s) I didn’t use visual studio for this problem. so there is no color :)

#include <string>

#include <vector>

#include<sstream>

#include<unordered\_map>

#include<iostream>

#include<algorithm>

using namespace std;

vector<pair<int,int>> Input(vector<string>& lines){

vector<pair<int,int>> date(lines.size());

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

stringstream ss(lines[i]);

string hour; string min; string sec;string process;

getline(ss,hour,' ');//remove date

getline(ss,hour,':'); getline(ss,min,':'); getline(ss,sec,' '); getline(ss,process,'s');

date[i].second=stod(hour)\*3600000 + stod(min)\*60000 + stod(sec)\*1000;

date[i].first=date[i].second-stod(process)\*1000+1;

}

return date;

}

int findNumber(vector<pair<int,int>> res){

vector<int> table(res.size());

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

int start=res[i].second;

int end=res[i].second+999;

for(int j=i;j<res.size();j++){

if(res[j].second<start || end<res[j].first) continue;

table[i]++;

}

}

return \*max\_element(table.begin(),table.end());

}

int solution(vector<string> lines) {

int answer = 0;

vector<pair<int,int>> date;

date=Input(lines);

answer=findNumber(date);

return answer;

}

[134. **[Programmers– KAKAO 2018 :News Clustering]]**

- it was just string parsing and make intersection and union set. so not that hard.

- see the code.

#include <string>

#include<iostream>

#include <algorithm>

#include<unordered\_map>

#include<unordered\_set>

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

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

using namespace std;

bool isAlpha(char& s){

if('a'<= s && s<='z') return true;

return false;

}

unordered\_set<string> SplitStr(string s,unordered\_map<string,int>& count){

unordered\_set<string> res;

for(int i=1;i<s.size();i++){

string temp;

if(isAlpha(s[i-1]) && isAlpha(s[i])){

temp=s.substr(i-1,2);

res.insert(temp);

count[temp]++;

}

}

return res;

}

int solution(string str1, string str2) {

int answer = 0;

unordered\_map<string,int> s1; unordered\_map<string,int> s2;

transform(str1.begin(),str1.end(),str1.begin(),::tolower);

transform(str2.begin(),str2.end(),str2.begin(),::tolower);

unordered\_set<string> a=SplitStr(str1,s1); unordered\_set<string> b=SplitStr(str2,s2);

//find intersection

vector<string> inter;

int minimum;

for(unordered\_set<string>::iterator iter=a.begin();iter!=a.end();iter++){

if(s2.find(\*iter)!=s2.end()){

minimum=min(s1[\*iter],s2[\*iter]);

for(int j=0;j<minimum;j++) inter.push\_back(\*iter);

}

}

//find union

vector<string> uni;

unordered\_set<string> temp\_set;

temp\_set.insert(a.begin(),a.end()); temp\_set.insert(b.begin(),b.end());

int maximum;

for(unordered\_set<string>::iterator iter=temp\_set.begin();iter!=temp\_set.end();iter++){

if(s1.find(\*iter)!=s1.end() && s2.find(\*iter)!=s2.end()){

maximum=max(s1[\*iter],s2[\*iter]);

for(int j=0;j<maximum;j++) uni.push\_back(\*iter);

}

else if(s1.find(\*iter)!=s1.end()) for(int j=0;j<s1[\*iter];j++) uni.push\_back(\*iter);

else if(s2.find(\*iter)!=s2.end()) for(int j=0;j<s2[\*iter];j++) uni.push\_back(\*iter);

}

if(inter.empty() && uni.empty()) return 65536;

answer=((double)inter.size()/(double)uni.size())\*65536;

return answer;

}

[135. **[BaekJoon– 1697 : Hide and Seek]]**

- it was just BFS problem, we just sholud be careful of memory using visit hash.

- see the code.

#include<iostream>

#include<queue>

#include<unordered\_map>

using namespace std;

int BFS(int n, int k) {

queue<pair<int, int>> que;

que.push(make\_pair(n, 0));

//vector<bool> visit(100001,false);

unordered\_map<int, bool> visit;

while (!que.empty()) {

int cur\_pos = que.front().first;

int cur\_sec = que.front().second;

visit[cur\_pos] = true;

que.pop();

if (cur\_pos == k) return cur\_sec;

if (cur\_pos + 1 <= 100000 && !visit[cur\_pos+1]) que.push(make\_pair(cur\_pos + 1, cur\_sec + 1));

if (cur\_pos - 1 >= 0 && !visit[cur\_pos-1]) que.push(make\_pair(cur\_pos - 1, cur\_sec + 1));

if (cur\_pos \* 2 <= 100000 && !visit[cur\_pos\*2]) que.push(make\_pair(cur\_pos \* 2, cur\_sec + 1));

}

return -1;

}

int main() {

int n; int k;

cin >> n; cin >> k;

cout<<BFS(n, k);

return 0;

}

[136. **[Programmers– KAKAO 2018 :Shuttle Bus]]**

- handling time is always naughty for me lol :).

- I changed all the bus schedule to unit of minute. <- is this english right? haha.

- see the code.

#include <string>

#include <vector>

#include <algorithm>

#include <map>

#include<iostream>

using namespace std;

string solution(int n, int t, int m, vector<string> timetable) {

string answer = "";

int base = 9\*60;

vector<pair<int,vector<int>>> possible\_time;

for(int i=0;i<n;i++) possible\_time.push\_back(make\_pair(base+(t\*i),vector<int>{0,999999,-1}));

vector<bool> full\_bus(possible\_time.size(),false);

vector<int> min\_table(timetable.size());

for(int i=0; i<timetable.size();i++) min\_table[i]=stoi(timetable[i].substr(0,2))\*60 + stoi(timetable[i].substr(3,2));

sort(min\_table.begin(),min\_table.end());

int start=0;

for(int i=0;i<possible\_time.size();i++){

int bus\_time = possible\_time[i].first;

for(int j=start;j<min\_table.size();j++){

if(min\_table[j]<=bus\_time){

possible\_time[i].second[0]++;

possible\_time[i].second[1]=min(possible\_time[i].second[1],min\_table[j]);

possible\_time[i].second[2]=max(possible\_time[i].second[2],min\_table[j]);

if(possible\_time[i].second[0]>=m){ full\_bus[i]=true; start=j+1; break; }

}

else { start=j; break; }

}

}

int res\_time;

if(full\_bus[n-1]){

if(possible\_time[n-1].second[2]==possible\_time[n-1].second[1]) res\_time=possible\_time[n-1].second[1]-1;

else res\_time=possible\_time[n-1].second[2]-1;

}

else{

res\_time=possible\_time[n-1].first;

}

string hour; string minute;

hour=to\_string(res\_time/60); minute=to\_string(res\_time%60);

if(hour.size()==1) hour="0"+hour; if(minute.size()==1) minute="0"+minute;

answer=hour+":"+minute;

return answer;

}

[137. **[Programmers– KAKAO 2018 :Friends 4 Blocks]]**

- it was a slicing problem.

- but the fucking confusing between m and n, I spend quite much time shit!

- From now, I’m gonna use height and length for m and n to prevent such a confusing situation.

- p.s) unordered\_set looks not able to handle pair<int,int>. so I changed it to set.

- see the code.

#include <string>

#include <vector>

#include<iostream>

#include<set>

using namespace std;

vector<vector<char>> makeTable(int m,int n, vector<string> board){

vector<vector<char>> table(m,vector<char>(n));

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

for(int j=0;j<n;j++)

table[i][j]=board[i][j];

return table;

}

set<pair<int,int>> findBlock(int m,int n, vector<vector<char>>& table){

set<pair<int,int>> remove;

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

for(int j=1;j<n;j++){ //slice

char c=table[i][j];

if(c==0) continue;

bool flag=true;

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

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

if(table[a][b]!=c){

flag=false;

break;

}

}

if(!flag) break;

}

if(flag){

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

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

remove.insert(make\_pair(a,b));

}

}

}

}

}

return remove;

}

vector<vector<char>> newTable(int m, int n,vector<vector<char>> table){

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

//bool flag=true;

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

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

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

//if(a==0) flag=false;

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

else{

table[i][j]=table[a][j];

table[a][j]=0;

break;

}

}

//if(!flag) break;

}

}

return table;

}

int Game(int m,int n, vector<vector<char>> table){

set<pair<int,int>> remove\_index;

int count=0;

while(true){

//find block

remove\_index=findBlock(m,n,table);

if(remove\_index.empty()) break;

count+=remove\_index.size();

//remove block

for(set<pair<int,int>>::iterator iter=remove\_index.begin();iter!=remove\_index.end();iter++){

int x=iter->first; int y = iter->second;

table[x][y]=0;

}

//make new table;

table=newTable(m,n,table);

}

return count;

}

int solution(int m, int n, vector<string> board) {

int answer = 0;

vector<vector<char>> table=makeTable(m,n,board);

answer=Game(m,n,table);

return answer;

}

[138. **[Programmers– KAKAO 2018 :Cache]]**

- Honestly. it was shame I couldn’t solve this problem directly due to LRU.

- Making LRU is quite fresh in a bad way. so I spent 20~30 minutes.

- see the code.

#include <cstring>

#include <vector>

#include<algorithm>

#include<iostream>

using namespace std;

int solution(int cacheSize, vector<string> cities) {

if(cacheSize==0) return cities.size()\*5;

int answer = 0;

//tolower

for(int i=0;i<cities.size();i++) transform(cities[i].begin(),cities[i].end(),cities[i].begin(),::tolower);

//make cache

vector<string> res;

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

vector<string>::iterator iter=find(res.begin(),res.end(),cities[i]);

//if cache doesn't have current city.

if(iter==res.end()){

answer+=5;

res.push\_back(cities[i]);

if(res.size()>cacheSize) res.erase(res.begin());

}

//when cache has the city.

else{

answer++;

res.erase(iter);

res.push\_back(cities[i]);

}

}

return answer;

}

[139. **[Programmers– KAKAO 2018 :Secret Map]]**

- Handling bitset is the main point of this problem.

- So It was easy for me :)

- see the code.

#include <string>

#include <vector>

#include<iostream>

#include<bitset>

using namespace std;

vector<string> solution(int n, vector<int> arr1, vector<int> arr2) {

vector<string> answer;

vector<bitset<16>> table;

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

bitset<16> a; bitset<16> b;

a=arr1[i]; b=arr2[i];

table.push\_back(a|b);

}

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

string temp="";

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

if(table[i][j]==1) temp+="#";

else temp+=" ";

}

answer.push\_back(temp);

}

return answer;

}

[140. **[Programmers– KAKAO 2018 :Dart Game]]**

- String parsing and caculate using given rule is the way to solve this problem.

- parsing part was pretty naughty. since I had to check all the index of the string. but given string was short. So I could follow where an index is and there was no time issue either.

- see the code.

#include <string>

#include<vector>

#include<cmath>

#include<iostream>

using namespace std;

int solution(string dartResult) {

vector<string> res;

string temp="";

//string parsing to make vector

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

if('0'<=dartResult[i] && dartResult[i]<='9'){

temp+=dartResult[i];

}

else{

temp+=dartResult[i];

if(dartResult[i+1]=='\*' || dartResult[i+1]=='#'){

i++;

temp+=dartResult[i];

}

res.push\_back(temp);

temp="";

}

}

//split the score record

vector<pair<int,vector<char>>> score(3);

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

string score\_part="";

int j=0;

for(j=0;j<res[i].size();j++){

if('0'<=res[i][j] && res[i][j]<='9'){

score\_part+=res[i][j];

}

else break;

}

score[i].first=stoi(score\_part);

score[i].second.push\_back(res[i][j]);

if(res[i].back()=='\*' || res[i].back()=='#') score[i].second.push\_back(res[i].back());

}

//let's caculate

vector<int> table(3);

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

if(score[i].second[0]=='S') table[i]=score[i].first;

else if(score[i].second[0]=='D') table[i]=pow(score[i].first,2);

else if(score[i].second[0]=='T') table[i]=pow(score[i].first,3);

}

//option

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

if(score[i].second.size()==1) continue;

if(score[i].second[1]=='#') table[i]\*=-1;

else{

if(i==0) table[i]\*=2;

else{

table[i-1]\*=2;

table[i]\*=2;

}

}

}

int answer = 0;

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

answer+=table[i];

return answer;

}

**[141. [SAMSUNG – SW : Population Movement]]**

- handling visit table and BFS is the point.

- To avoid time limit exceeded, i need to have a habit using call by reference.

- see the code.

#include<iostream>

#include<fstream>

#include<vector>

#include<queue>

#include<cmath>

using namespace std;

int N; int L; int R;

vector<vector<int>> makeTable() {

fstream in("C:\\Users\\gjsgu\\Desktop\\Algorithm\\test\_case\\test\_case\_for\_population\_movement.txt");

if (in.is\_open()) cout << "file is opened." << endl;

in >> N; in >> L; in >> R;

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

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

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

in >> table[i][j];

return table;

}

vector<vector<int>> makeTable2() {

cin >> N; cin >> L; cin >> R;

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

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

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

cin >> table[i][j];

return table;

}

inline bool isValid(int a, int b) {

if (abs(a - b) >= L && abs(a - b) <= R) return true;

return false;

}

vector<pair<int, int>> makeUnion(vector<vector<int>>& table, vector<vector<bool>>& visit, int x, int y,int& sum) {

vector<pair<int, int>> unions;

queue<pair<int, int>> que;

visit[x][y] = true;

que.push(make\_pair(x, y));

while (!que.empty()) {

x = que.front().first; y = que.front().second;

unions.push\_back(make\_pair(x, y));

sum += table[x][y];

que.pop();

if (x - 1 >= 0 && !visit[x - 1][y] && isValid(table[x - 1][y], table[x][y])) {

visit[x - 1][y] = true;

que.push(make\_pair(x - 1, y));

}

if (y - 1 >= 0 && !visit[x][y - 1] && isValid(table[x][y - 1], table[x][y])) {

visit[x][y - 1] = true;

que.push(make\_pair(x, y - 1));

}

if (x + 1 < N && !visit[x + 1][y] && isValid(table[x + 1][y], table[x][y])) {

visit[x + 1][y] = true;

que.push(make\_pair(x + 1, y));

}

if (y + 1 < N && !visit[x][y + 1] && isValid(table[x][y + 1], table[x][y])) {

visit[x][y + 1] = true;

que.push(make\_pair(x, y + 1));

}

}

if (unions.size() == 1) unions.pop\_back();

return unions;

}

void makeMove(vector<vector<int>>& table, vector<pair<int, vector<pair<int, int>>>>& unions) {

for (pair<int, vector<pair<int, int>>> row : unions) {

int people = floor(row.first / row.second.size());

for (pair<int, int> a : row.second) table[a.first][a.second] = people;

}

return;

}

void printTable(vector<vector<int>> table) {

for (vector<int> row : table) {

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

cout << endl;

}

cout << endl;

}

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

int population\_movement = 0;

while (true) {

vector<pair<int,vector<pair<int, int>>>> unions;

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

vector<pair<int, int>> temp;

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

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

int x = i; int y = j;

int sum=0;

if (!visit[x][y]) {

temp = makeUnion(table, visit, x, y,sum);

if (!temp.empty()) unions.push\_back(make\_pair(sum,temp));

}

}

}

if (unions.empty()) break;

else {

makeMove(table, unions);

population\_movement++;

}

}

return population\_movement;

}

int main() {

vector<vector<int>> table = makeTable2();

cout << Move(table);

return 0;

}

**[142. [SAMSUNG – SW : Tree Jaetech]]**

**-** I think, Samsung’s simulation problem is quite dirty.

- they present super rigid time limit.

- In this time also, I spent so much time to reduce running time.

- see the code.

#include<iostream>

#include<fstream>

#include<vector>

#include<algorithm>

#include<unordered\_map>

using namespace std;

int n, m, k;

int dir[8][2] = { {-1 ,-1},{-1,0},{-1,1},{0,-1},{0,1},{1,-1},{1,0},{1,1} }; // From top-left to bottom-right, 8 directions.

vector<vector<int>> energy;

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

vector<vector<int>> table;

vector<vector<int>> dead;

void makeInput() {

ifstream in("C:\\Users\\gjsgu\\Desktop\\Algorithm\\test\_case\\test\_case\_for\_tree\_jaetech.txt");

if (in.is\_open()) cout << "file is opened." << endl;

in >> n; in >> m; in >> k;

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

energy.assign(n, vector<int>(n));

trees.assign(n, vector<vector<int>>(n));

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

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

in >> energy[i][j];

}

}

int a, b, c;

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

in >> a; in >> b; in >> c;

trees[a - 1][b - 1].push\_back(c);

}

}

void makeInput2() {

cin >> n; cin >> m; cin >> k;

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

energy.assign(n, vector<int>(n));

trees.assign(n, vector<vector<int>>(n));

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

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

cin >> energy[i][j];

}

}

int a, b, c;

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

cin >> a; cin >> b; cin >> c;

trees[a - 1][b - 1].push\_back(c);

}

}

inline bool isValid(int x, int y) {

if (x >= 0 && x < n && y >= 0 && y < n) return true;

return false;

}

void spring\_summer() {

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

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

int dead = 0;

vector<int> temp;

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

for (int k = 0; k < trees[i][j].size(); k++) {

int age = trees[i][j][k];

if (table[i][j] < age) { //if there is not enough energy to eat.

dead += age / 2;

continue;

}

else {

table[i][j] -= age;

temp.push\_back(trees[i][j][k]+1); // age +1

}

}

table[i][j] += dead; //summer

trees[i][j].clear();

for (int qq : temp) trees[i][j].push\_back(qq);

}

}

}

void autumn() {

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

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

for (int k = 0; k < trees[i][j].size(); k++) {

int age = trees[i][j][k];

if (age % 5 != 0) continue;

else {

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

int tx = i + dir[t][0]; int ty = j + dir[t][1];

if (isValid(tx, ty)) {

trees[tx][ty].push\_back(1);

}

}

}

}

}

}

}

void winter() {

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

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

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

}

}

}

int main() {

makeInput2();

for (int year = 0; year < k; year++) {

spring\_summer();

autumn();

if (year == k - 1) break; // we don't care last winter

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;

}

**[143. [SAMSUNG – SW : Baby Shark]]**

**-** BFS problem, there is an order of BFS. so I had to sort shells I will visit for each depth.

- and they gave quite a lot of condition, like time, shark size, eating condition, etc.

- see the code.

#include<iostream>

#include<fstream>

#include<vector>

#include<queue>

#include<algorithm>

using namespace std;

int n; int sx; int sy;

vector<vector<int>> table;

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

void Input() {

ifstream in("C:\\Users\\gjsgu\\Desktop\\Algorithm\\test\_case\\test\_case\_for\_baby\_shark.txt");

if (in.is\_open()) cout << "file is opened." << endl;

in >> n;

table.assign(n, vector<int>(n));

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

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

in >> table[i][j];

if (table[i][j] == 9) { table[i][j] = 0; sx = i; sy = j; }

}

}

}

void Input2() {

cin >> n;

table.assign(n, vector<int>(n));

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

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

cin >> table[i][j];

if (table[i][j] == 9) { table[i][j] = 0; sx = i; sy = j; }

}

}

}

void printTable() {

for (vector<int> row : table) {

for (int i : row)

cout << i << " ";

cout << endl;

}

cout << endl;

}

int grownUp() {

queue<vector<int>> que;

int total\_time = 0;

int shark\_size = 2; //initial shark\_size

int eat\_size = 0;

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

que.push(vector<int>{sx, sy, 0});

while (!que.empty()) {

int x = que.front()[0]; int y = que.front()[1]; int time = que.front()[2];

vector<pair<int, int>> order;

while (!que.empty() && que.front()[2] == time) {

order.push\_back(make\_pair(que.front()[0], que.front()[1]));

que.pop();

}

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

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

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

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

if (table[x][y] != 0 && table[x][y] < shark\_size) { //when it is possible to eat.

total\_time += time;

eat\_size++;

if (eat\_size == shark\_size) {

shark\_size++;

eat\_size = 0;

}

table[x][y] = 0;

visit = vector<vector<bool>>(n, vector<bool>(n, false)); //visit re-initialize.

queue<vector<int>> empty\_que;

que.swap(empty\_que); //make the queue empty.

que.push(vector<int>{x, y, 0});

visit[x][y] = true;

break;

}

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 < n && table[tx][ty] <= shark\_size && !visit[tx][ty]) {

visit[tx][ty] = true;

que.push(vector<int>{tx, ty, time + 1});

}

}

}

}

return total\_time;

}

int main() {

Input2();

cout << grownUp();

return 0;

}

**[144. [SAMSUNG – SW : Goodbye Finedust!]]**

**-** it was a simulation problem, but not DFS and BFS.

- following given rules not hard, but have to be careful of robot’s position whether robot is at leftside or middle or rightside.

- see the code.

#include<iostream>

#include<fstream>

#include<vector>

using namespace std;

vector<vector<int>> table;

vector<pair<int,int>> robot;

int rtx, rty, rbx, rby;

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

int r, c, t;

bool lf=false, rt=false;

void Input() {

ifstream in("C:\\Users\\gjsgu\\Desktop\\Algorithm\\test\_case\\test\_case\_for\_goodbye\_finedust.txt");

if (in.is\_open()) cout << "file is opened." << endl;

in >> r; in >> c; in >> t;

table.assign(r, vector<int>(c));

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

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

in >> table[i][j];

if (table[i][j] == -1) robot.push\_back(make\_pair(i, j));

}

}

rtx = robot[0].first; rty = robot[0].second;

rbx = robot[1].first; rby = robot[1].second;

if (rty == 0) lf = true;

else if (rty = c - 1) rt = true;

}

void Input2() {

cin >> r; cin >> c; cin >> t;

table.assign(r, vector<int>(c));

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

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

cin >> table[i][j];

if (table[i][j] == -1) robot.push\_back(make\_pair(i, j));

}

}

rtx = robot[0].first; rty = robot[0].second;

rbx = robot[1].first; rby = robot[1].second;

if (rty == 0) lf = true;

else if (rty = c - 1) rt = true;

}

void dustMove() {

vector<vector<int>> new\_dust;

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

for (int j = 0; j < c; 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 < r && ty < c && 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>>(r,vector<int>(c));

table[rtx][rty] = -1; table[rbx][rby] = -1;

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

}

void printTable() {

for (vector<int> row : table) {

for (int i : row)

cout << i << " ";

cout << endl;

}

cout << endl;

}

void cleanerAct() {

//top

for (int j = rty-1; j >= 1; j--) table[rtx][j] = table[rtx][j-1];

if (lf) for (int i = rtx-1; i >= 1; i--) table[i][0] = table[i - 1][0];

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

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

if(rt) for (int i = 0; i < rtx-1; i++) table[i][c - 1] = table[i + 1][c - 1];

else for (int i = 0; i < rtx; i++) table[i][c - 1] = table[i + 1][c - 1];

for (int j = c - 1; j > rty; j--) table[rtx][j] = table[rtx][j - 1];

table[rtx][rty + 1] = 0;

//bottom

for (int j = rby - 1; j>= 1; j--) table[rbx][j] = table[rbx][j - 1];

if(lf) for (int i = rbx+1; i < r - 1; i++) table[i][0] = table[i + 1][0];

else for (int i = rbx; i < r-1; i++) table[i][0] = table[i + 1][0];

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

if(rt) for (int i = r - 1; i > rbx+1; i--) table[i][c - 1] = table[i - 1][c - 1];

else for (int i = r - 1; i > rbx; i--) table[i][c - 1] = table[i - 1][c - 1];

for (int j = c - 1; j > rby; j--) table[rbx][j] = table[rbx][j - 1];

table[rbx][rby + 1] = 0;

}

int amountFineDust() {

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

dustMove();

cleanerAct();

}

int res = 0;

for (vector<int> row : table)

for (int j = 0; j < c; j++) if(row[j]!=-1) res += row[j];

return res;

}

int main() {

Input2();

cout << amountFineDust();

return 0;

}

**[145. [SAMSUNG – SW :2D Array Calculation]]**

**-** this was weird. when I used unordered\_map, abort occurred at 9% of submission.

- but after I changed the map to int array with same algorithm, it worked well.

- what’s wrong with my previous version??? Even, all the test case they gave or in disccusion session was passed with my unordered\_map algorithm!

- see the code.

#include<iostream>

#include<fstream>

#include<vector>

#include<unordered\_map>

#include<algorithm>

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

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

#define max\_k 101

using namespace std;

int r, c, k;

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

int time = 0;

int row = 3, col = 3;

void Input() {

ifstream in("C:\\Users\\gjsgu\\Desktop\\Algorithm\\test\_case\\test\_case\_for\_2d\_array\_calculation.txt");

if (in.is\_open()) cout << "file is opened." << endl;

in >> r; in >> c; in >> k;

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

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

in >> table[i][j];

}

void Input2() {

cin >> r; cin >> c; cin >> k;

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

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

cin >> table[i][j];

}

void operR() {

int max\_col = 0;

vector<vector<pair<int, int>>> temp(row);

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

int arr[max\_k] = { 0, };

for (int j = 0; j < col; j++) if (table[i][j] != 0) arr[table[i][j]]++;

for (int j = 1; j < max\_k; j++) if (arr[j] != 0) temp[i].push\_back(make\_pair(arr[j], j));

sort(temp[i].begin(), temp[i].end());

max\_col = max(max\_col, temp[i].size() \* 2);

}

col = min(100,max\_col);

table = vector<vector<int>>(row, vector<int>(col,0));

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

for (int j = 0; j < temp[i].size() &&j<50; j++) {

table[i][2\*j] = temp[i][j].second;

table[i][(2\*j) + 1] = temp[i][j].first;

}

}

}

void operC() {

int max\_row = 0;

vector<vector<pair<int,int>>> temp(col);

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

int arr[max\_k] = { 0, };

for (int i = 0; i < row; i++) if(table[i][j]!=0) arr[table[i][j]]++;

for (int i = 1; i < max\_k; i++) if (arr[i] != 0) temp[j].push\_back(make\_pair(arr[i], i));

sort(temp[j].begin(), temp[j].end());

max\_row = max(max\_row, temp[j].size()\*2);

}

row = min(100,max\_row);

table = vector<vector<int>>(row, vector<int>(col,0));

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

for (int i = 0; i < temp[j].size()&& i< 50; i++) {

table[2\*i][j] = temp[j][i].second;

table[(2\*i)+1][j] = temp[j][i].first;

}

}

}

void printTable() {

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

for (int j = 0; j < col; j++)

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

cout << endl;

}

cout << endl;

}

int calculation() {

while (time<=100) {

if (r-1>=0 && c-1>=0 && r-1<row && c-1<col&&table[r - 1][c - 1] == k) return time;

if (row >= col) operR();

else operC();

time++;

}

return -1;

}

int main() {

Input2();

cout<<calculation();

return 0;

}

**[146. [SAMSUNG – SW :Fishing King]]**

**-** I have a so much curiousity of SAMSUNG’s simulation algorithm test cases.

- this problem also has some weird things.

- At first, I solved correctly but time limit exceeded occurred. so I change my code with same algorithm but faster, but it didn’t work. what the heck? how come? algorithm was perfectly same and I just chagned container to reduce traversal time.

- I don’t understand still now.

- Finally, I used my first submission with more efficient shark move.

- see the code.

#include<iostream>

#include<fstream>

#include<vector>

#include<unordered\_map>

#include<algorithm>

#include<map>

using namespace std;

int dir[4][2] = { {-1,0},{1,0},{0,1},{0,-1} }; // up, down, right, left

int r, c, m;

int get\_size = 0;

vector<vector<vector<int>>> table; //speed, direction, size

map<pair<int, int>, vector<vector<int>>> shark\_pos;

void Input() {

ifstream in("C:\\Users\\gjsgu\\Desktop\\Algorithm\\test\_case\\test\_case\_for\_fishing\_king.txt");

if (in.is\_open()) cout << "file is opened." << endl;

in >> r; in >> c; in >> m;

table.assign(r, vector<vector<int>>(c));

int a, b, c, d, e;

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

in >> a; in >> b; in >> c; in >> d; in >> e;

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

}

}

void Input2() {

cin >> r; cin >> c; cin >> m;

table.assign(r, vector<vector<int>>(c));

int a, b, c, d, e;

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

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

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

}

}

void GetShark(int col) {

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

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

get\_size += table[i][col][2];

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

return;

}

}

}

void Moving(int x, int y, int shark\_size) {

int speed = table[x][y][0];

int cur\_dir = table[x][y][1];

// up, down, right, left ==> 0,1,2,3

int temp\_speed;

if (cur\_dir == 0) temp\_speed = speed % ((r - 1) \* 2);

else if (cur\_dir == 1) temp\_speed = speed % ((r - 1) \* 2);

else if (cur\_dir == 2) temp\_speed = speed % ((c - 1) \* 2);

else temp\_speed = speed % ((c - 1) \* 2);

for (int i = 0; i < temp\_speed; i++) {

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

if (tx < 0) {

cur\_dir = 1;

x = 1;

}

else if (ty < 0) {

cur\_dir = 2;

y = 1;

}

else if (tx >= r) {

cur\_dir = 0;

x = r - 2;

}

else if (ty >= c) {

cur\_dir = 3;

y = c - 2;

}

else {

x = tx; y = ty;

}

}

shark\_pos[make\_pair(x, y)].push\_back(vector<int>{speed, cur\_dir, shark\_size});

}

void MoveShark() {

shark\_pos.clear();

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

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

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

Moving(i, j, table[i][j][2]);

}

}

}

table = vector<vector<vector<int>>>(r, vector<vector<int>>(c));

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

sort(iter->second.begin(), iter->second.end(), [](vector<int> a, vector<int> b) {return a[2] > b[2]; });

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

}

}

void printTable() {

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

for (vector<int> srk : row) {

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

else cout << "(" << srk[0] << " " << srk[1] << " " << srk[2] << ")" << " ";

}

cout << endl;

}

cout << endl;

}

void Fishing() {

//printTable();

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

GetShark(i);

//cout << "<<get>>" << endl;

//printTable();

MoveShark();

//cout << "<<move>>" << endl;

//printTable();

}

}

int main() {

Input2();

Fishing();

cout << get\_size;

return 0;

}

**[147. [SAMSUNG – SW : Laboratory 3 ]]**

- huh… so much edge case!

- I used DFS for permutation of viruses and BFS for contagion.

- see the code.

#include<iostream>

#include<fstream>

#include<vector>

#include<queue>

#include<set>

using namespace std;

int lab\_size, number\_of\_virus;

int dir[4][2] = { {-1,0},{0,-1},{1,0},{0,1} }; // up left down right

vector<vector<int>> table;

vector<vector<bool>> visit;

vector<pair<int, int>> virus;

vector<pair<int, int>> chosen\_virus;

set<int> min\_time;

void Input() {

ifstream in("C:\\Users\\gjsgu\\Desktop\\Algorithm\\test\_case\\test\_case\_for\_laboratory3.txt");

if (in.is\_open()) cout << "file is opened." << endl;

in >> lab\_size; in >> number\_of\_virus;

table.assign(lab\_size, vector<int>(lab\_size, 0));

visit.assign(lab\_size, vector<bool>(lab\_size, 0));

int temp\_input;

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

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

in >> temp\_input;

table[i][j]=temp\_input;

visit[i][j] = temp\_input;

if (temp\_input == 2) {

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

visit[i][j] = false;

}

}

}

}

void Input2() {

cin >> lab\_size; cin >> number\_of\_virus;

table.assign(lab\_size, vector<int>(lab\_size, 0));

visit.assign(lab\_size, vector<bool>(lab\_size, 0));

int temp\_input;

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

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

cin >> temp\_input;

table[i][j] = temp\_input;

visit[i][j] = temp\_input;

if (temp\_input == 2) {

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

visit[i][j] = false;

}

}

}

}

void printTable(vector<vector<int>>& temp) {

for (vector<int> row : temp) {

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

cout << endl;

}

cout << endl;

}

bool isContagious(vector<vector<int>>& temp) {

for (int i = 0; i < lab\_size; i++)

for (int j = 0; j < lab\_size; j++) if (temp[i][j] == 0) return false;

//printTable(temp);

return true;

}

int BFS() {

vector<vector<int>> temp = table;

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

queue<pair<int, int>> que;

queue<pair<int, int>> new\_que;

for (pair<int, int> a : chosen\_virus) new\_que.push(a);

//for (pair<int, int> a : chosen\_virus) cout << a.first << "," << a.second << endl; cout << endl;

int time = 0;

while (!new\_que.empty()) {

bool flag = false;

que = new\_que;

new\_que = queue<pair<int, int>>(); // make new\_que empty;

while (!que.empty()) {

int x = que.front().first; int y = que.front().second;

que.pop();

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 < lab\_size && ty < lab\_size && !temp\_visit[tx][ty]) {

if (temp[tx][ty] != 2) flag = true;

temp\_visit[tx][ty] = true; //make visit

temp[tx][ty] = time+10;

new\_que.push(make\_pair(tx, ty));

}

}

}

if (flag) time++;

else if (!isContagious(temp)) time++;

}

if(isContagious(temp)) return time;

return 999;//max lab\_size is 50 so 999 is enough

}

void Contagion(int count,int start) {

if (count == number\_of\_virus) {

int res = BFS();

//cout <<"res : "<< res << endl<<endl;

min\_time.insert(res);

return;

}

for (int i = start; i < virus.size(); i++) {

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

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

visit[x][y] = true;

Contagion(count + 1, i + 1);

visit[x][y] = false;

chosen\_virus.pop\_back();

}

}

int main() {

Input2();

Contagion(0,0);

int answer= \*min\_time.begin();

answer == 999 ? cout<<-1 : cout<<answer;

return 0;

}

**[148. [SAMSUNG – SW : Gerrymandering 2]]**

**-** I can’t understand why SAMSUNG likes r and c representing row and columns, why don’t they use just x and y? Because of it, there always exist a confusion.

- this time as well, I spent a lot of time by index

-see the code.

#include<iostream>

#include<fstream>

#include<vector>

#include<algorithm>

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

using namespace std;

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

vector<vector<int>> table;

int N;

void Input() {

ifstream in("C:\\Users\\gjsgu\\Desktop\\Algorithm\\test\_case\\test\_case\_for\_gerrymandering.txt");

if (in.is\_open()) cout << "file is opened." << endl;

in >> N;

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

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

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

in >> table[i][j];

}

}

}

void Input2() {

cin >> N;

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

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

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

cin >> table[i][j];

}

}

}

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

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

for (int i : row) {

cout << i << " ";

}

cout << endl;

}

cout << endl;

}

void makeDistrict(vector<vector<int>>& temp, int x, int y, int d1, int d2) {

// 1

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

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

if (temp[i][j] == 10) { temp[i][j] = 5; break; }

temp[i][j] = 1;

}

}

// 2

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

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

if (temp[i][j] == 10) { temp[i][j] = 5; break; }

temp[i][j] = 2;

}

}

//3

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

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

if (temp[i][j] == 10) { temp[i][j] = 5; break; }

temp[i][j] = 3;

}

}

//4

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

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

if (temp[i][j] == 10) { temp[i][j] = 5; break; }

temp[i][j] = 4;

}

}

}

int drawLine(int x,int y,int d1,int d2) {

vector<vector<int>> temp(N+1, vector<int>(N+1, 5));

vector<int> exist(5, false);

//top left

for (int i = 0; i <= d1; i++) temp[x+i][y-i] = 10;

//top right

for (int i = 0; i <= d2; i++) temp[x + i][y + i] = 10;

//bottom left

for (int i = 0; i <= d2; i++) temp[x + d1 + i][y - d1 + i] = 10;

//bottom right

for (int i = 0; i <= d1; i++) temp[x + d2 + i][y + d2 - i] = 10;

makeDistrict(temp, x, y, d1, d2);

/\*//count

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

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

exist[temp[i][j] - 1]++;

}

}

for (int i : exist) if (i == 0) return 999999;

//isConnected?

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

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

int cx = i; int cy = j;

if (exist[temp[cx][cy] - 1] == 1) continue;

bool connected = false;

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

int tx = cx + dir[k][0]; int ty = cy + dir[k][1];

if (tx >= 1 && ty >= 1 && tx <= N && ty <= N && temp[tx][ty] == temp[cx][cy]) connected = true;

}

if (!connected) return 999999;

}

}\*/

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

//cout << x << "," << y << " : " << d1 << "," << d2 << endl;

//printTable(temp);

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

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

people[temp[i][j] - 1]+=table[i][j];

}

}

return (\*max\_element(people.begin(), people.end())) - (\*min\_element(people.begin(), people.end()));

}

bool isValidToMakeLine(int x,int y,int d1,int d2) {

if (x + d1 > N || y - d1 < 1) return false;

if (x + d2 > N || y + d2 > N) return false;

if (x + d1 + d2 > N || y - d1 + d2 > N) return false;

if (x + d2 + d1 > N || y + d2 - d1 < 1) return false;

return true;

}

int Gerrymandering() {

//(i< i+d1+d2 && i+d1+d2<N) && (j-d1>=0 && j+d2<N)

int answer = 9999999;

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

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

int d1 = 1, d2 = 1;

for (d1=1; d1<=j; d1++) { //d1

for (d2 = 1; d2<=N-j; d2++) {

if(isValidToMakeLine(i,j,d1,d2)) answer = min(answer, drawLine(i, j, d1, d2));

}

}

}

}

return answer;

}

int main() {

Input2();

cout << Gerrymandering();

return 0;

}

**[149. [SAMSUNG – SW : New Game 2]]**

- it was a stack handling problem. each shell of 2d array has stack for recoding pieces.

- see the code.

#include<iostream>

#include<fstream>

#include<vector>

using namespace std;

vector<vector<int>> table;

vector<vector<vector<int>>> stacks;

vector<vector<int>> pieces;

int N, K;

int dir[4][2] = { {0,1},{0,-1},{-1,0},{1,0} }; //right left up down

void Input() {

ifstream in("C:\\Users\\gjsgu\\Desktop\\Algorithm\\test\_case\\test\_case\_for\_new\_game2.txt");

if (in.is\_open()) cout << "file is opened." << endl;

in >> N; in >> K;

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

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

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

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

in >> table[i][j];

}

}

int a, b, c;

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

in >> a; in >> b; in >> c;

pieces.push\_back(vector<int>{a-1, b-1, c-1,i});

stacks[a - 1][b - 1].push\_back(i);

}

}

void Input2() {

cin >> N; cin >> K;

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

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

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

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

cin >> table[i][j];

}

}

int a, b, c;

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

cin >> a; cin >> b; cin >> c;

pieces.push\_back(vector<int>{a - 1, b - 1, c - 1, i});

stacks[a - 1][b - 1].push\_back(i);

}

}

bool isStackedOver4(int x, int y) {

if (stacks[x][y].size() >= 4) return true;

return false;

}

int Game2() {

int turn = 0;

while (turn<=1000) {

turn++;

bool flag = false;

//move

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

int cur\_dir = pieces[i][2]; int x = pieces[i][0]; int y = pieces[i][1];

//cout << "piece : " << i << " x : " << x << " y : " << y << " dir : "<<cur\_dir<<endl;

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

//cout << " tx : " << tx << " ty : " << ty << endl;

//blue

if (tx < 0 || ty < 0 || tx >= N || ty >= N || table[tx][ty] == 2) {

//cout << "blue" << endl;

if (cur\_dir == 0) cur\_dir = 1;

else if (cur\_dir == 1) cur\_dir = 0;

else if (cur\_dir == 2) cur\_dir = 3;

else cur\_dir = 2;

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

pieces[i][2] = cur\_dir;

//double blue

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

else { i--; continue; }

}

//get index of current stacks;

int bottom;

for (bottom = 0; bottom < stacks[x][y].size(); bottom++) if (stacks[x][y][bottom] == i) break;

// when red

if (table[tx][ty] == 1) {

//cout << "red" << endl;

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

stacks[tx][ty].push\_back(stacks[x][y].back());

pieces[stacks[x][y].back()][0] = tx; pieces[stacks[x][y].back()][1] = ty;

stacks[x][y].pop\_back();

}

}

else {

//cout << "white" << endl;

for (int j = bottom; j < stacks[x][y].size(); j++) {

stacks[tx][ty].push\_back(stacks[x][y][j]);

pieces[stacks[x][y][j]][0] = tx; pieces[stacks[x][y][j]][1] = ty;

}

int size = stacks[x][y].size();

for (int j = bottom; j < size; j++) stacks[x][y].pop\_back();

}

if (isStackedOver4(tx,ty)) { flag = true; break; }

}

if (flag) break;

}

if (turn > 1000) return -1;

return turn;

}

int main() {

Input();

cout << Game2();

return 0;

}

**[150. [SAMSUNG – SW : Circle Rotation]]**

**-** the trick this problem has is a circle they gave is logically same with 2D array without columns act like circle.

- so it was just a simulation problem. but like always, SAMSUNG’s problem is not kind to user.

- there are pretty many ambiguous restrictions.

- see the code.

#include<iostream>

#include<fstream>

#include<vector>

#include<queue>

using namespace std;

int N, M, T;

vector<vector<int>> order;

vector<vector<int>> table;

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

void Input() {

ifstream in("C:\\Users\\gjsgu\\Desktop\\Algorithm\\test\_case\\test\_case\_for\_circle\_rotation.txt");

if (in.is\_open()) cout << "file is opened." << endl;

in >> N; in >> M; in >> T;

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

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

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

in >> table[i][j];

}

}

int a, b, c;

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

in >> a; in >> b; in >> c;

order.push\_back(vector<int>{a,b,c});

}

}

void Input2() {

cin >> N; cin >> M; cin >> T;

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

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

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

cin >> table[i][j];

}

}

int a, b, c;

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

cin >> a; cin >> b; cin >> c;

order.push\_back(vector<int>{a, b, c});

}

}

void Move(int dir,int round,vector<int>& row) {

int temp;

round %= row.size();

if (dir == 0) {

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

temp = row.back();

for (int i = row.size()-1; i >=1; i--) row[i] = row[i - 1];

row[0] = temp;

}

}

else {

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

temp = row[0];

for (int i = 0; i < row.size() - 1;i++)

row[i] = row[i + 1];

row[row.size() - 1] = temp;

}

}

}

bool isEmpty(vector<pair<int, int>>& left\_pos) {

int count = 0;

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

for (int j = 0; j < M; j++) if (table[i][j] != 0) left\_pos.push\_back(make\_pair(i, j));

}

if (left\_pos.empty()) return true;

return false;

}

void printTable() {

for (vector<int> row : table) {

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

cout << endl;

}

cout << endl;

}

int Rotate() {

//cout << "START" << endl;

//printTable();

for (vector<int> ord : order) {

int x = ord[0]; int d = ord[1]; int k = ord[2];

vector<int> row\_index;

//cout << "order : "<<x<<","<<d<<","<<k << endl;

for (int i = 1; i \* x <= N; i++) { //rotate

int circle = (i \* x)-1;

row\_index.push\_back(circle);

Move(d, k,table[circle]);

}

//cout << "After Move" << endl;

//printTable();

vector<pair<int, int>> left\_pos;

if (!isEmpty(left\_pos)) {

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

bool flag = true;

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

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

visit[i][j] = true;

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

int cur = table[i][j];

queue<pair<int, int>> que; que.push(make\_pair(i, j));

while (!que.empty()) {

int x = que.front().first; int y = que.front().second; que.pop();

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

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

if (ty == M) ty = 0; if (ty == -1) ty = M - 1; //they are in shape of circle.

if (tx >= 0 && tx < N&& table[tx][ty] == cur && !visit[tx][ty]) {

visit[tx][ty] = true; que.push(make\_pair(tx, ty));

table[tx][ty] = 0; table[x][y] = 0;

flag = false;

}

}

}

}

}

if (flag) {

//cout << "<<flag happend>>" << endl;

int sum = 0;

for (vector<int> row : table) {

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

}

double average = (double)sum / (double)left\_pos.size();

//cout << "sum : " << sum << " average : " << average << endl;

for (pair<int, int> a : left\_pos) {

if (table[a.first][a.second] > average) table[a.first][a.second]--;

else if(table[a.first][a.second] < average)table[a.first][a.second]++;

}

}

}

//cout << "BFS resulst" << endl;

//printTable();

}

int res = 0;

for (vector<int> row : table) {

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

}

return res;

}

int main() {

Input2();

cout << Rotate();

return 0;

}

**[151. [SAMSUNG – SW : Dice Yut Play]]**

**-** this was the last problem of SAMSUNG SW in Baekjoon. But sadly, my algorithm didn’t work well. so I refered to below URL.

<https://eine.tistory.com/entry/%EB%B0%B1%EC%A4%80-17825%EB%B2%88-%EC%A3%BC%EC%82%AC%EC%9C%84-%EC%9C%B7%EB%86%80%EC%9D%B4-%EB%AC%B8%EC%A0%9C-%ED%92%80%EC%9D%B4>

- In my case, I made all the node of the board and did brute force, but answer was wrong.

- so I took the above URL’s code. he used bit mask for permutation that made using DFS in my case. so genius…

- Futhermore, he made lookup table. I’ve never thought that I could make it as a look up table! I learned a lot from the URL.

- see the code.

#include<iostream>

#include<vector>

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

using namespace std;

#define endl '\n'

typedef long long ll;

typedef pair<int, int> pii;

int dice[10];

#define END 32

//이동 규칙이 복잡할 수 있으므로 Look-up 테이블을 만들어서 사용한다.

//jump[index][0] = 해당 판 점수

//jump[index][1~5] => 주사위 해당 수가 나오면 이동하는 양

int jump[33][6] = {

{0,1,2,3,4,5}, //0번자리

{2,2,3,4,5,9}, //1번자리

{4,3,4,5,9,10}, //2번자리

{6,4,5,9,10,11}, //3번자리

{8,5,9,10,11,12},//4번자리

{10,6,7,8,20,29},//5번자리

{13,7,8,20,29,30}, //6번자리

{16,8,20,29,30,31}, //7번자리

{19,20,29,30,31,32}, //8번자리

{12,10,11,12,13,14}, //9번자리

{14,11,12,13,14,15}, //10번자리

{16,12,13,14,15,16}, //11번자리

{18,13,14,15,16,17}, //12번자리

{20,18,19,20,29,30}, //13번자리

{22,15,16,17,24,25}, //14번자리

{24,16,17,24,25,26}, //15번자리

{26,17,24,25,26,27}, //16번자리

{28,24,25,26,27,28}, //17번자리

{22,19,20,29,30,31}, //18번자리

{24,20,29,30,31,32}, //19번자리

{25,29,30,31,32,32}, //20번자리

{26,20,29,30,31,32}, //21번자리

{27,21,20,29,30,31}, //22번자리

{28,22,21,20,29,30}, //23번자리

{30,23,22,21,20,29}, //24번자리

{32,26,27,28,31,32}, //25번자리

{34,27,28,31,32,32}, //26번자리

{36,28,31,32,32,32}, //27번자리

{38,31,32,32,32,32}, //28번자리

{30,30,31,32,32,32}, //29번자리

{35,31,32,32,32,32}, //30번자리

{40,32,32,32,32,32}, //31번자리

{0,32,32,32,32,32} //32번자리

};

int ans=0;

void check(int bit) {

int score = 0;

vector<int>occupation(35);

vector<int> pos(4);

occupation[0] = 4;

for (int turn = 0; turn < 10; turn++) {

//이번에 옮길 말

int horse = (bit >> (turn \* 2)) & 0x3;

//cout << horse << endl;

//이동하는 거리

int move = dice[turn];

//현재 위치

int& current\_pos = pos[horse];

//이동할 위치

int next\_pos = jump[current\_pos][move];

//이번 이동으로 얻을 점수

int get\_score = jump[next\_pos][0];

//처음이나 끝 위치가 아닌데 말이 겹치는 경우

if (occupation[next\_pos] > 0 && next\_pos && next\_pos != END) {

//불가능한 이동

return;

}

else {

score += get\_score;

occupation[next\_pos]++;

occupation[current\_pos]--;

current\_pos = next\_pos;

}

}

ans = max(ans, score);

}

int main() {

for (int i = 0; i < 10; i++)

cin >> dice[i];

for (int bit = 0; bit < (1 << 20); bit++) {

check(bit);

}

cout << ans << endl;

}

**[152. [KAKAO – 2018 : The Song Just Before]]**

**-** WOW! I have barely solved this problem. At first, I used vector to record notes of a song, but in that case, it would be harder to check whether my song is in there than using string.

- so I changed algorithm. I spent 2 hours or so, I think.

- see the code.

#include <string>

#include<iostream>

#include <vector>

#include <sstream>

#include<algorithm>

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

using namespace std;

string songToInt(string song) {

//cout<<song<<endl;

string res;

bool sharp = false;

for (int i = song.size() - 1; i >= 0; i--) {

if (song[i] == '#') sharp = true;

else {

switch (song[i]) {

case 'C':

if (sharp) res += 'C';

else res += 'c';

break;

case 'D':

if (sharp) res += 'D';

else res += 'd';

break;

case 'E':

res += 'e';

break;

case 'F':

if (sharp) res += 'F';

else res += 'f';

break;

case 'G':

if (sharp) res += 'G';

else res += 'g';

break;

case 'A':

if (sharp) res += 'A';

else res += 'a';

break;

case 'B':

res += 'b';

break;

}

sharp = false;

}

}

//for (int i : res) cout << i << " ";

//cout << endl;

reverse(res.begin(), res.end());

//for (int i : res) cout << i << " ";

//cout << endl;

return res;

}

bool hasMySong(string a, string b) {

int i = 0, j = 0;

bool flag = false;

int start = 0;

while (i<a.size()) {

if (j >= b.size()) {

flag = true;

break;

}

if (a[i] == b[j]) {

i++; j++;

}

else {

start++;

i = start; j = 0;

}

}

return flag;

}

string solution(string m, vector<string> musicinfos) {

string answer = "";

// make list

vector<pair<vector<int>, pair<string, string>>> list; //pair<vector{start,end,length},pair<name,{song : 1,2,12,3...}}>>

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

stringstream ss(musicinfos[i]);

string start; string end; string name; string song;

getline(ss, start, ','); getline(ss, end, ','); getline(ss, name, ','); getline(ss, song, '\n');

string s\_time = ""; s\_time += start[0]; s\_time += start[1];

string s\_temp = ""; s\_temp += start[3]; s\_temp += start[4];

int s\_input = stoi(s\_time) \* 60 + stoi(s\_temp);

string e\_time = ""; e\_time += end[0]; e\_time += end[1];

string e\_temp = ""; e\_temp += end[3]; e\_temp += end[4];

int e\_input = stoi(e\_time) \* 60 + stoi(e\_temp);

string temp = songToInt(song);

int play\_time = e\_input - s\_input;

int temp\_size = temp.size();

if (play\_time > temp\_size) for (int i = temp\_size; i < play\_time; i++) temp+=temp[i % temp\_size];

else if (play\_time < temp\_size) temp = { temp.begin(),next(temp.begin(),play\_time) };

list.push\_back(make\_pair(vector<int>{s\_input, e\_input, e\_input - s\_input - 1, i},

make\_pair(name, temp)));

}

string my\_song = songToInt(m);

vector<pair<string, vector<int>>> find\_list;

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

if (list[i].second.second.find(my\_song)!=string::npos) {

find\_list.push\_back(make\_pair(list[i].second.first, vector<int>{list[i].first[3], list[i].first[2]}));

}

}

if (find\_list.empty()) answer = "(None)";

else {

sort(find\_list.begin(), find\_list.end(), [](pair<string, vector<int>> a, pair<string, vector<int>> b) { return a.second[1] > b.second[1]; });

vector<pair<string, vector<int>>> temp;

int cur = find\_list[0].second[1];

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

if (find\_list[i].second[1] == cur) temp.push\_back(find\_list[i]);

else break;

}

sort(temp.begin(), temp.end(), [](pair<string, vector<int>> a, pair<string, vector<int>> b) { return a.second[0] < b.second[0]; });

answer = temp[0].first;

}

return answer;

}

int main() {

cout<<solution("ABC", { "12:00,12:14,HELLO,C#DEFGAB","13:00,13:05,WORLD,ABCDEF"});

return 0;

}

**[153. [KAKAO – 2018 : Compression]]**

**-** For two days from last problem, I had no time to solve algorithm until today.

- And this problem was quite confusing. Making dictionary was the hardest thing.

- I made a dictionary for each length of word. After that, from max\_length to 1, compare input substring with the dictionary of the substring’s length.

- see the code.

#include <string>

#include <vector>

#include <iostream>

#include<algorithm>

#include<map>

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

using namespace std;

vector<int> solution(string msg) {

vector<int> answer;

map<int, vector<pair<string, int>>> dict;

//dict initialized

dict[1].assign(26, make\_pair("", 0));

for (int i = 0; i < 26; i++) dict[1][i] = make\_pair('A' + i, i + 1);

int max\_length = 1;

int start = 0;

int cur\_index = 27;

bool flag = true;

while (flag) {

bool next = true;

for (int len = max\_length; len >= 1 && flag && next; len--) {

string temp = msg.substr(start, len);

for (int i = 0; i < dict[len].size(); i++) {

if (temp == dict[len][i].first) {

answer.push\_back(dict[len][i].second);

if (start + len < msg.size()) {

start += len;

dict[len + 1].push\_back(make\_pair(temp + msg.substr(start, 1), cur\_index));

max\_length = max(max\_length, len + 1);

cur\_index++;

next = false;

break;

}

else {

flag = false;

break;

}

}

}

}

}

return answer;

}

int main() {

vector<int> answer;

answer = solution("TOBEORNOTTOBEORTOBEORNOT");

//answer = solution("KAKAO");

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

return 0;

}

**[154. [KAKAO – 2018 : Automatic Completion]]**

**-** it was a string parisng problem. so I used Trie tree to make dictionary.

- when I insert a string into the dictionary, I recorded how many word can be parsed from the node with a name has\_word.

- the concept was easy, but making Trie tree was quite naughty :).

- see the code.

#include <string>

#include <vector>

#include<iostream>

using namespace std;

class Trie {

public:

vector<Trie\*> dict; // a : 0 - z : 25

bool terminal = false;

int has\_word = 0;

Trie() {

dict.assign(26, nullptr);

}

~Trie() {

for (Trie\* remove : dict) if (remove != nullptr) remove = nullptr;

}

void insert(const char\* c) {

if (\*c == '\0') {

//has\_word++;

terminal = true;

return;

}

int index = \*c - 'a';

if (dict[index] == nullptr) dict[index] = new Trie();

dict[index]->has\_word++;

dict[index]->insert(c + 1);

}

bool find(const char\* c) {

if (terminal && \*c == '\0') return true;

else if (\*c == '\0') return false;

int index = \*c - 'a';

if (dict[index] != nullptr) {

return dict[index]->find(c + 1);

}

else return false;

}

int length(const char\* c,int count) {

if (has\_word == 1) return count;

else if (has\_word > 1 && \*c == '\0') return count;

int index = \*c - 'a';

return dict[index]->length(c + 1, count + 1);

}

};

int solution(vector<string> words) {

int answer = 0;

Trie\* dict = new Trie();

//make dictionary

for (string s : words) {

const char\* c = s.c\_str(); // to make string to const char\*, we can use string::str.c\_str();

dict->insert(c);

}

for (string s : words) {

const char\* c = s.c\_str();

int temp = dict->length(c, 0);

//cout << "length : "<<temp << endl;

answer += temp;

}

return answer;

}

int main() {

cout<<solution(vector<string>{ "abc", "def", "ghi", "jklm" });

return 0;

}

**[155. [KAKAO – 2018 : Sort Filename]]**

- it was an easy problem, but setting a range of For syntax made a small error. To solve the error, I spent 30 minutes.

- during revise my algorithm, I made my algorithm worse about time complexity, actually I didn’t need to change set to vector, but I have no idea what’s wrong at the time.

- after passing all the test case, it’s quite tiresome returning my alogirhtm to originnal one.

- see the code.

#include <string>

#include <vector>

#include <algorithm>

#include<iostream>

#include<map>

#include<set>

using namespace std;

pair<string,int> getNumber(string s) {

string head;

string temp;

bool flag = true;

for (int i = 0; i < s.size() && flag; i++) {

if ('0' <= s[i] && s[i] <= '9') {

for (int j = i; j <= s.size(); j++) {

if (j == s.size()) {

flag = false;

break;

}

else if ('0' <= s[j] && s[j] <= '9') {

temp += s[j];

}

else {

flag = false;

break;

}

}

}

else head += s[i];

}

int res = stoi(temp);

return make\_pair(head,res);

}

vector<string> solution(vector<string> files) {

vector<string> answer;

vector<string> temp = files;

vector<string> name\_sort;

map<string, vector<pair<string,vector<int>>>> map\_list;

//to make lowercase

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

transform(temp[i].begin(), temp[i].end(), temp[i].begin(), ::tolower);

pair<string,int> head\_number = getNumber(temp[i]);

name\_sort.push\_back(head\_number.first);

//cout << "got name : "<<head\_number.first << endl;

//cout << "got number : " << head\_number.second << endl;

map\_list[head\_number.first].push\_back(make\_pair(files[i], vector<int>{i,head\_number.second}));

}

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

stable\_sort(iter->second.begin(), iter->second.end(), [](pair<string, vector<int>> a, pair<string, vector<int>> b) {return a.second[1] < b.second[1]; });

}

stable\_sort(name\_sort.begin(), name\_sort.end());

for (int i = 0; i < name\_sort.size(); i++) {

answer.push\_back(map\_list[name\_sort[i]][0].first);

map\_list[name\_sort[i]].erase(map\_list[name\_sort[i]].begin());

}

//for (string s : answer)

//cout << s << " ";

return answer;

}

int main() {

solution({ "F-5 Freedom Fighter","fa-36", "B-50 Superfortress", "A-10 Thunderbolt II", "F-14 Tomcat" });

return 0;

}

**[156. [KAKAO – 2018 : Notation Game]]**

**-** this was final problem of KAKAO. By solving this problem, I’ve solved all the KAKAO coding test problem and part of SAMSUNG SW :)

- the way to make string for given notation is the key point of this problem.

- I made whole string first, actually the string was over calculated, but time limit was okay.

- After that, just finding Muzi’s turn, that was all.

- see the code.

#include <string>

#include <vector>

#include<algorithm>

#include<iostream>

using namespace std;

string solution(int n, int t, int m, int p) {

string answer = "";

string alpha="0123456789ABCDEF";

string total\_string="0";

for(int i=1;i<t\*m;i++){

string temp="";

int number=i;

while(number>0){

int res=number%n;

number=number/n;

temp+=alpha[res];

}

reverse(temp.begin(),temp.end());

total\_string+=temp;

}

int count=0;

for(int i=p-1;i<total\_string.size();i+=m){

if(count>=t) break;

answer+=total\_string[i];

count++;

}

return answer;

}

**[157. [SAMSUNG** - **SW : Chocolate and Raisin]**

- it was D4 problem. I mean it was quite hard to solve.

- time was the most difficult one to reach the given condition.

- I used memorization named visit, DFS, and prefix\_sum.

- see the code.

#include<iostream>

#include<vector>

#include<cstring>

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

using namespace std;

vector<vector<int>> table;

vector<vector<int>> prefix\_sum;

int n, m;

int visit[51][51][51][51] = { 0, };

void prefixSum() {

prefix\_sum[0][0] = table[0][0];

for (int i = 1; i < n; i++) prefix\_sum[i][0] = prefix\_sum[i - 1][0] + table[i][0];

for (int j = 1; j < m; j++) prefix\_sum[0][j] = prefix\_sum[0][j - 1] + table[0][j];

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

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

prefix\_sum[i][j] = prefix\_sum[i][j - 1] + prefix\_sum[i - 1][j] - prefix\_sum[i - 1][j - 1] + table[i][j];

}

}

}

int helper(int tx,int ty,int bx,int by) {

if (tx == bx && ty == by) return 0;

else if (visit[tx][ty][bx][by] != -1) return visit[tx][ty][bx][by];

int sum = 0;

if (tx == 0 && ty == 0) sum = prefix\_sum[bx][by];

else if (tx == 0) sum = prefix\_sum[bx][by] - prefix\_sum[bx][ty - 1];

else if (ty == 0) sum = prefix\_sum[bx][by] - prefix\_sum[tx - 1][by];

else sum = prefix\_sum[bx][by] - prefix\_sum[tx - 1][by] - prefix\_sum[bx][ty - 1] + prefix\_sum[tx - 1][ty - 1];

int res = 987654321;

for (int i = tx ; i < bx; i++) {

res=min(res,sum+helper(tx, ty, i, by)+ helper(i + 1, ty, bx, by));

}

for (int j = ty; j < by; j++) {

res=min(res,sum+helper(tx, ty, bx, j) + helper(tx, j+1, bx, by));

}

visit[tx][ty][bx][by] = res;

return res;

}

int main(int argc, char\*\* argv){

int test\_case;

int T;

cin >> T;

for (test\_case = 1; test\_case <= T; ++test\_case) {

cin >> n; cin >> m;

memset(visit, -1, sizeof(visit));

table.assign(n, vector<int>(m));

prefix\_sum.assign(n, vector<int>(m));

for (int i = 0; i < n; i++) for (int j = 0; j < m; j++) cin >> table[i][j]; //make table

prefixSum();

//for (vector<int> row : prefix\_sum) {

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

// cout << endl;

//}

cout << "#" << test\_case << " ";

cout << helper(0, 0, n - 1, m - 1) << endl;

}

return 0;//정상종료시 반드시 0을 리턴해야합니다.

}

**[158. [SAMSUNG** - **SW : Diamond]**

‑ I’ve misunderstood. but it wasn’t only me. so it’s SAMSUNG’s problem haha.

- using set and map, I made an algorithm easily. there was no DFS, DP, etc, but just caculation.

- see the code.

#include<iostream>

#include<unordered\_map>

#include<set>

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

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

using namespace std;

int N, K;

unordered\_map<int, int> diamond;

set<int> diamond\_size;

int maximum = 0;

// K=3

// 1 2 3 5 5 5 5 8 9 9 9 10 15

void helper(set<int>::iterator start) {

int cur\_size = \*start;

maximum = Max(maximum, diamond[cur\_size]);

int res = diamond[cur\_size];

for (set<int>::iterator size\_iter = next(start,1); size\_iter != diamond\_size.end(); size\_iter++) {

int carry = 987654321;

int key\_size = \*size\_iter;

if (cur\_size + K <key\_size) break;

res += diamond[key\_size];

}

maximum = Max(maximum, res);

}

int main(int argc, char\*\* argv){

int test\_case;

int T;

cin >> T;

int input;

for (test\_case = 1; test\_case <= T; ++test\_case){

//get input

cin >> N >> K;

diamond.clear();

diamond\_size.clear();

maximum = 0;

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

cin >> input; diamond[input]++;

diamond\_size.insert(input);

}

//print out

for (set<int>::iterator size\_iter = diamond\_size.begin(); size\_iter != diamond\_size.end(); size\_iter++) {

//cout << \*size\_iter << " ";

helper(size\_iter);

}

cout << "#" << test\_case << " ";

cout << maximum << endl;

}

return 0;

}

**[159. [SAMSUNG** - **SW : Polynomial Calculation]**

**-** why is this prolbem’s level D4? it was so easy.

- I just had to be careful data type and vector iterator.

- see the code.

#include<iostream>

#include<vector>

using namespace std;

vector<long long> getFn(vector<vector<int>> operand, vector<long long> Xs) {

vector<long long> res;

for (const int x : Xs) {

vector<long long> Fn = { 1,x };

for (vector<int> row : operand) {

int t = row[0]; int a = row[1]; int b = row[2];

if (t == 1) {

Fn.emplace\_back((Fn[a]+Fn[b])% 998244353);

}

else if (t == 2) {

Fn.emplace\_back((a \* Fn[b])% 998244353);

}

else if(t==3){

Fn.emplace\_back((Fn[a] \* Fn[b])% 998244353);

}

}

res.emplace\_back(Fn.back());

}

return res;

}

int main(int argc, char\*\* argv)

{

int test\_case;

int T;

cin >> T;

for (test\_case = 1; test\_case <= T; ++test\_case)

{

vector<vector<int>> operand;

vector<long long> Xs;

vector<long long> res;

//get input

int N; cin >> N;

int t, a, b;

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

cin >> t >> a >> b;

operand.emplace\_back(vector<int>{t, a, b});

}

int M; cin >> M;

int k;

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

cin >> k;

Xs.emplace\_back(k);

}

res=getFn(operand,Xs);

cout << "#" << test\_case;

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

cout << " " << res[i];

cout << endl;

}

return 0;

}

**[160. [SAMSUNG** - **SW : The Longest Sequence]**

**-**At first, I used next\_permutation for brute force. but time limit exceeded occurred.

- see the first code.

#include<iostream>

#include<vector>

#include<algorithm>

#include<unordered\_set>

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

using namespace std;

int N;

// 1 3 5 2 4

int getLongestSequenceLength(vector<int>& numbers) {

int res = 0;

int n\_size = numbers.size();

do {

for (int i = n\_size - 1; i >= 1 && i>res; i--) {

if (numbers[i] > numbers[i - 1] && numbers[i] % numbers[i - 1] == 0) {

int temp = 1;

int j;

for ( j = i; j >= 1; j--) {

if (numbers[j] > numbers[j - 1] && numbers[j] % numbers[j - 1] == 0) {

temp++;

}

else break;

}

res = max(res, temp);

i = j+1;

}

}

} while (next\_permutation(numbers.begin(), numbers.end()));

return res;

}

int main(int argc, char\*\* argv)

{

int test\_case;

int T;

cin >> T;

for (test\_case = 1; test\_case <= T; ++test\_case){

//get input

cin >> N;

vector<int> numbers(N);

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

cin >> numbers[i];

}

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

cout << "#" << test\_case << " " << getLongestSequenceLength(numbers) << endl;

}

return 0;

}

- so I change my algorithm to DP.

- Dp[i] has a number of factor i has in the given vector.

- the point is that c style array and memset are fatser than STL array and memset. since when I used STL array and vector, time limit exceeded occurred.

- see the code.

#include<iostream>

#include<vector>

#include<algorithm>

#include<cstring>

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

using namespace std;

int dp[1000001];

int numbers[100001];

int N;

int main(int argc, char\*\* argv)

{

int test\_case;

int T;

cin >> T;

for (test\_case = 1; test\_case <= T; ++test\_case){

//get input

memset(dp, 0, sizeof(dp));

memset(numbers, 0, sizeof(numbers));

cin >> N;

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

cin >> numbers[i];

dp[numbers[i]] = 1;

}

sort(numbers,numbers+N);

int res = 1;

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

for (int j = numbers[i] \* 2; j <= numbers[N-1]; j += numbers[i]) {

if (dp[j] > 0) {

dp[j] = max(dp[j], dp[numbers[i]] + 1);

}

}

}

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

res = max(res, dp[numbers[i]]);

cout << "#" << test\_case << " " << res << endl;

}

return 0;

}

**[161. [SAMSUNG** - **SW : Sewoon will do tomorrow]**

**-** I used DP to check whether there is ovelapped day, but runtime error occurred. I couldn’t find the reason, so I changed my algorithm to easier way.

- see the first code.

#include<iostream>

#include<vector>

#include<cstring>

#include<unordered\_map>

#include<climits>

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

using namespace std;

int getLastDay(vector<pair<int, int>>& table) {

unordered\_map<int, bool> dp;

int min\_day = INT\_MAX;

int t,d;

for (pair<int, int> row : table) {

t = row.first; d = row.second;

int start = (d - t) + 1;

min\_day = min(min\_day, start);

int rest\_day = 0;

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

if (dp.count(i)) rest\_day++;

dp[i]=true;

}

int j = start;

while (rest\_day > 0) {

if (!dp.count(j)) {

dp[j] = true;

rest\_day--;

}

j--;

}

min\_day = min(min\_day, j+1);

}

return min\_day - 1;

}

int main(int argc, char\*\* argv){

ios\_base::sync\_with\_stdio(0);

std::cin.tie(NULL);

std::cout.tie(NULL);

int test\_case;

int T,N;

cin >> T;

for (test\_case = 1; test\_case <= T; ++test\_case){

cin >> N;

vector<pair<int, int>> table(N, make\_pair(0, 0));

int t,d;

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

cin >> t >> d;

table[i] = make\_pair(t, d);

}

cout << "#" << test\_case << " " << getLastDay(table) << endl;

}

return 0;//정상종료시 반드시 0을 리턴해야합니다.

}

- the revised algorithm is based on sorting alogirhtm. after sorting, if there is overlapped day, just moving current homework forward.

- since we took sort, we don’t need to care a case that several days are overlapped.

#include<iostream>

#include<vector>

#include<cstring>

#include<climits>

#include<algorithm>

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

using namespace std;

int getLastDay(vector<pair<int, int>>& table) {

int t,d;

int start\_day = table.back().second - table.back().first + 1;

table.pop\_back();

while(!table.empty()) {

int te = table.back().second; int ts = te- table.back().first+1;

table.pop\_back();

if (te >= start\_day) {

start\_day = ts - (te - start\_day+1);

}

else start\_day = ts;

}

return start\_day - 1;

}

int main(int argc, char\*\* argv){

ios\_base::sync\_with\_stdio(0);

std::cin.tie(NULL);

std::cout.tie(NULL);

int test\_case;

int T,N;

cin >> T;

for (test\_case = 1; test\_case <= T; ++test\_case){

cin >> N;

vector<pair<int, int>> table(N, make\_pair(0, 0));

int t,d;

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

cin >> t >> d;

table[i] = make\_pair(t, d);

}

sort(table.begin(), table.end(), [](pair<int, int> a, pair<int, int> b) {return a.second < b.second; });

cout << "#" << test\_case << " " << getLastDay(table) << endl;

}

return 0;//정상종료시 반드시 0을 리턴해야합니다.

}

**[162. [SAMSUNG** - **SW : Haji Prediction]**

**-** they gave an algorithm and I should dertermine whethere the algorithm would end or not.

- so I recorded numbers appeared during algorithm and if recorded number appear again, break the while syntax and return false.

- but a problem was data type, N could be given up to 10^14. so I used data type LONG LONG, but I needed unsigned long long actually.

- see the code.

#include<iostream>

#include<unordered\_map>

#include<cmath>

using namespace std;

int main(int argc, char\*\* argv){

//what is haji in here by the way?

ios\_base::sync\_with\_stdio(0);

std::cin.tie(NULL);

std::cout.tie(NULL);

int test\_case;

int T;

cin >> T;

unsigned long long N;

for (test\_case = 1; test\_case <= T; ++test\_case){

cin >> N;

bool flag=true;

unordered\_map<unsigned long long, bool> stop;

while (N > 1) {

if (stop.count(N)) {

flag = false;

break;

}

else stop[N] = true;

if (N % 2 == 0) N = N / 2;

else N = 3 \* N + 3;

cout << N << endl;

}

if (flag) cout << "#" << test\_case << " YES" << endl;

else cout << "#" << test\_case << " NO" << endl;

}

return 0;

}

**[163. [SAMSUNG** - **SW : Palindrome Phobia]**

**-** At first, I checked all the possible substings of all the possible strings that can be created during permutation. but time limit exceeded occurred.

- see the first code.

#include<iostream>

#include<algorithm>

using namespace std;

bool isPalindrome(string& s) {

int i = 0, j = s.size() - 1;

while (i < j) {

if (s[i] != s[j]) return false;

i++; j--;

}

return true;

}

bool hasPalindrome(string& s) {

int cur\_size = 2;

int s\_size = s.size();

string temp;

while (cur\_size <= s\_size) {

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

temp = s.substr(i, cur\_size);

if (isPalindrome(temp)) return true;

}

cur\_size++;

}

return false;

}

int main(int argc, char\*\* argv){

int test\_case;

int T;

ios\_base::sync\_with\_stdio(0);

std::cin.tie(NULL);

std::cout.tie(NULL);

cin >> T;

string s;

for (test\_case = 1; test\_case <= T; ++test\_case){

cin >> s;

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

bool flag=false;

do {

if (!hasPalindrome(s)) {

flag = true;

break;

}

} while (next\_permutation(s.begin(), s.end()));

if (flag) cout << "#" << test\_case << " YES" << endl;

else cout << "#" << test\_case << " NO" << endl;

}

return 0;

}

- so I changed my algorithm. the algorithm is that we just have character a,b or c. so to make palindrome, the most number of a character has to be 2 more than the least character.

- e.x) if a=4, b=3, c=2, abc abc aba. but if a=4, b=3, c=3, abc abc abc a.

- with the latter example, we can’t make palindrom. but with the former one we can.

- see the code.

#include<iostream>

#include<algorithm>

#include<climits>

using namespace std;

bool isPalindrome(string& s) {

int i = 0, j = s.size() - 1;

while (i < j) {

if (s[i] != s[j]) return false;

i++; j--;

}

return true;

}

bool hasPalindrome(string& s) {

int cur\_size = 2;

int s\_size = s.size();

string temp;

while (cur\_size <= s\_size) {

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

temp = s.substr(i, cur\_size);

if (isPalindrome(temp)) return true;

}

cur\_size++;

}

return false;

}

int main(int argc, char\*\* argv){

int test\_case;

int T;

ios\_base::sync\_with\_stdio(0);

std::cin.tie(NULL);

std::cout.tie(NULL);

cin >> T;

string s;

for (test\_case = 1; test\_case <= T; ++test\_case){

cin >> s;

int a=0, b=0, c=0;

int minimum = INT\_MAX, maximum = INT\_MIN;

for (char cha : s) {

if (cha == 'a') a++;

else if (cha == 'b') b++;

else c++;

}

minimum = min(a, min(b, c)); maximum = max(a, max(b, c));

if (maximum - minimum > 1) cout << "#" << test\_case << " NO" << endl;

else cout << "#" << test\_case << " YES" << endl;

}

return 0;

}

**[164. [SAMSUNG** - **SW : 3 Dimentional Farmer]**

**-** I did brute force at first, but time was problem, so I made binary search tree, but same problem

occurred. I think I should’ve balanced binary search tree.

- see the first code.

#include<iostream>

#include<vector>

#include<unordered\_map>

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

using namespace std;

class Node {

public:

int value;

Node\* left = nullptr;

Node\* right = nullptr;

Node(int val) :value(val) {}

void insert(int val) {

if (val <= value) {

if (left) left->insert(val);

else left = new Node(val);

}

else {

if (right) right->insert(val);

else right = new Node(val);

}

}

int find(int val,int min\_dis) {

min\_dis = min(min\_dis, abs(val - value));

if(val==value) return 0;

else if (val < value) {

if (left) return left->find(val, min\_dis);

else return min\_dis;

}

else {

if (right) return right->find(val, min\_dis);

else return min\_dis;

}

}

};

int main(int argc, char\*\* argv){

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

cin.tie(NULL);

cout.tie(NULL);

int test\_case;

int T;

cin >> T;

int N, M,c1,c2;

for (test\_case = 1; test\_case <= T; ++test\_case){

cin >> N >> M;

cin >>c1 >> c2;

int x\_distance = abs(c1 - c2);

int rt; cin >> rt;

Node\* root = new Node(rt);

int input;

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

cin >> input;

root->insert(input);

}

int res;

int cnt = 0;

int minimum = 987654321;

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

cin >> input;

res = root->find(input,987654321);

if (x\_distance + res < minimum) {

cnt = 1;

minimum = x\_distance + res;

}

else if (x\_distance + res == minimum) cnt++;

}

cout << "#" << test\_case << " " << minimum << " " << cnt << endl;

}

return 0;

}

- instead of it, I sorted cows’ coordinates and took binary search to get nearest distance

- but it corrected just 22 test case, but there are 5 more test case. I don’t know why it was wrong!

- see the second code.

#include<iostream>

#include<vector>

#include<algorithm>

#include<climits>

#include<unordered\_map>

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

using namespace std;

int N, M, c1, c2;

int bin\_search(const int val,vector<int>& cows){

long long left = 0, right = (long long)N - 1, mid=(left + right) / 2;

if (cows.front() > val) return abs(cows.front() - val);

else if (cows.back() < val) return abs(val-cows.back());

while (left <= right) {

mid = (left + right) / 2;

if (cows[mid] == val) return 0;

else if (cows[mid] > val) right = mid - 1;

else left = mid + 1;

}

if (cows[mid] < val) return min(abs(cows[mid] - val), abs(cows[mid + 1] - val));

else return min(abs(cows[mid] - val), abs(cows[mid - 1] - val));

}

int main(int argc, char\*\* argv){

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

cin.tie(NULL);

cout.tie(NULL);

int test\_case;

int T;

cin >> T;

for (test\_case = 1; test\_case <= T; ++test\_case){

cin >> N >> M;

cin >>c1 >> c2;

unordered\_map<long long, int> cnt;

long long x\_distance = abs(c1 - c2);

vector<int> cows(N);

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

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

long long input;

long long minimum = INT\_MAX;

long long res;

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

cin >> input;

res=bin\_search(input,cows);

int distance = x\_distance + res;

minimum = min(minimum, distance);

cnt[distance]++;

}

cout << "#" << test\_case << " " << minimum << " " << cnt[minimum] << endl;

cnt.clear();

cows.clear();

}

return 0;

}

- so I changed the return value of binary search to index from distance.

- but 3 test case weren’t solved. fuck!

- see the third code.

#include<iostream>

#include<vector>

#include<algorithm>

#include<climits>

#include<unordered\_map>

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

using namespace std;

int N, M, c1, c2;

int bin\_search(const int val,vector<int>& cows){

int left = 0, right = N - 1, mid=(left + right) / 2;

if (cows.front() > val) return 0;

else if (cows.back() < val) return N - 1;

while (left <= right) {

mid = (left + right) / 2;

if (cows[mid] == val) return mid;

else if (cows[mid] > val) right = mid - 1;

else left = mid + 1;

}

if (cows[mid] < val) mid++;

return mid;

}

int main(int argc, char\*\* argv){

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

cin.tie(NULL);

cout.tie(NULL);

int test\_case;

int T;

cin >> T;

for (test\_case = 1; test\_case <= T; ++test\_case){

cin >> N >> M;

cin >>c1 >> c2;

int x\_distance = abs(c1 - c2);

vector<int> cows(N);

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

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

int horse;

int minimum = INT\_MAX;

int idx;

int cnt = 0;

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

cin >> horse;

idx =bin\_search(horse,cows);

if (0 <= idx && idx < N) {

int cow = cows[idx];

int distance = abs(cow - horse);

if (minimum > distance) {

cnt = 1;

minimum = distance;

}

else if (minimum == distance) cnt++;

}

if (0 <= idx && idx < N && cows[idx] != horse) {

int cow = cows[idx - 1];

int distance =abs(cow - horse);

if (minimum > distance) {

cnt = 1;

minimum = distance;

}

else if (minimum == distance) cnt++;

}

}

cout << "#" << test\_case << " " << minimum+x\_distance << " " << cnt << endl;

}

return 0;

}

- I gave up.. I refer to a discussion.

- I think it is almost same and there is no different of time complexity. why it works but my one doesn’t?

- the below code is from there.

#include <iostream>

#include <cstdio>

#include <string>

#include <vector>

#include <algorithm>

#include <cmath>

#include <string.h>

using namespace std;

int TC = 1;

int N, M, C1, C2, H;

int C[500000];

int bSearch(int s, int e) {

if (s > e) return e;

int m = (s + e) / 2;

if (C[m] < H) return bSearch(m + 1, e);

else if (C[m] == H) return m;

else return bSearch(s, m - 1);

}

int main() {

scanf("%d", &TC);

for (int tc = 1; tc <= TC; tc++) {

scanf("%d %d %d %d", &N, &M, &C1, &C2);

int DX = C1 > C2 ? C1 - C2 : C2 - C1;

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

scanf("%d", &C[i]);

sort(C, C + N);

int idx = 0, d, mind = 0x7fffffff;

int cnt = 0;

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

scanf("%d", &H);

idx = bSearch(0, N - 1);

for (int j = idx; j >= 0; j--) {

d = C[j] - H;

if (d < 0) d = -d;

if (d < mind) {

mind = d;

cnt = 1;

}

else if (d == mind)

cnt++;

else break;

}

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

d = C[j] - H;

if (d < 0) d = -d;

if (d < mind) {

mind = d;

cnt = 1;

}

else if (d == mind)

cnt++;

else break;

}

}

printf("#%d %d %d\n", tc, DX + mind, cnt);

}

return 0;

}

- temp record

#include<iostream>

#include<vector>

#include<algorithm>

#include<climits>

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

using namespace std;

int input;

int N, M, c1, c2;

vector<int> cows;

int bSearch(int s, int e) {

if (s > e) return e;

int m = (s + e) / 2;

if (cows[m] < input) return bSearch(m + 1, e);

else if (cows[m] == input) return m;

else return bSearch(s, m - 1);

}

int main(int argc, char\*\* argv) {

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

cin.tie(NULL);

cout.tie(NULL);

int test\_case;

int T;

cin >> T;

for (test\_case = 1; test\_case <= T; ++test\_case) {

cin >> N; cin>> M;

cin >> c1; cin>> c2;

int x\_distance = abs(c1 - c2);

cows.assign(N,0);

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

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

int idx = 0, d, mind = 0x7fffffff;

int cnt = 0;

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

cin >> input;

idx = bSearch(0,N-1);

for (int j = idx; j >= 0; j--) {

d = cows[j] - input;

if (d < 0) d = -d;

if (d < mind) {

mind = d;

cnt = 1;

}

else if (d == mind)

cnt++;

else break;

}

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

d = cows[j] - input;

if (d < 0) d = -d;

if (d < mind) {

mind = d;

cnt = 1;

}

else if (d == mind)

cnt++;

else break;

}

}

cout << "#" << test\_case << " " << mind+x\_distance << " " << cnt << endl;

}

return 0;

}

**[165. [SAMSUNG** - **SW : Palindrome per Word]**

**-** I used DFS and memorization, but time limit exceeded occurred.

- For each k, I count how many valid palindrome per word string appear.

- see the firse code.

#include<iostream>

#include<vector>

using namespace std;

string str;

int s\_size;

vector<vector<int>> visit;

int DFS(const int k,const int s,const int e,int count) {

if (visit[k][s] != -1) {

return visit[k][s];

}

int answer = 0;

if (k <= 1) {

visit[k][s] = 1;

return 1;

}

else if (k == 2) {

if (str.substr(s, (e - s) / 2 + 1) == str.substr(e - (e - s) / 2 + 1, (e - s) / 2)) {

visit[k][s] = 1;

return 1;

}

return 0;

}

for (int i = 1; (s\_size - count) - k + 2 - (2 \* i) >= 0; i++) {

if (str.substr(s, i) == str.substr(e - i+1, i)) {

answer+=DFS(k-2, s + i,e-i,count+(2\*i));

}

else continue;

}

visit[k][s] = answer;

return answer;

}

int splitString() {

int k = 1;

int answer = 0;

bool odd=false; if (s\_size % 2) odd = true;

while (k <= s\_size) {

if (k % 2 == 0 && odd) {

k++;

continue;

}

answer+=DFS(k,0,s\_size-1,0);

k++;

}

return answer;

}

int main(int argc, char\*\* argv){

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

cin.tie(NULL);

cout.tie(NULL);

int test\_case;

int T;

cin >> T;

for (test\_case = 1; test\_case <= T; ++test\_case){

cin >> str;

s\_size = str.size();

visit.assign(s\_size+1, vector<int>(s\_size/2+1, -1));

cout << "#" << test\_case << " " << splitString() << endl;

}

return 0;

}

- I refered to discussions to revise my algorithm.

- Honestly, I didn’t understand the below code well.

#include <iostream>

#include <string.h>

#include <algorithm>

#include <vector>

using namespace std;

#define SMAX 10001

#define MOD 1000000007

int TC;

int main() {

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

cin.tie(NULL);

cout.tie(NULL);

cin >> TC;

for (int tt = 0; tt < TC; tt++) {

char input[SMAX];

cin >> input;

int cnt = 0;

while (input[cnt])

{

cnt++;

}

int count[SMAX] = { 1, };

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

// case 0 one character is center

int ls = i;

int le = i;

int rs = cnt - le - 1;

int re = cnt - ls - 1;

while (ls >= 0 && le <= (cnt - 1) / 2 && input[ls] == input[rs] && input[le] == input[re]) {

count[le + 1] = (count[le + 1] + count[ls]) % MOD;

ls--;

le++;

rs--;

re++;

}

// case 1 two characters are center

ls = i;

le = i + 1;

rs = cnt - le - 1;

re = cnt - ls - 1;

while (ls >= 0 && le <= (cnt - 1) / 2 && input[ls] == input[rs] && input[le] == input[re]) {

count[le + 1] = (count[le + 1] + count[ls]) % MOD;

ls--;

le++;

rs--;

re++;

}

}

int all = 0;

for (int i = 0; i < cnt / 2 + 1; i++) {

all = (all + count[i]) % MOD;

}

cout << "#" << tt + 1 << " " << all << endl;

}

return 0;

}

**[166. [SAMSUNG** - **SW : Odd Medium Pyramid]**

- run time error occurred. I think there is a trick. it’s not a brute force.

- see the first code.

#include<stdio.h>

#include<vector>

#include<algorithm>

using namespace std;

int Mid(int a, int b, int c) {

if ((a > b&& b > c)||(c>b && b>a)) return b;

else if ((b > a&& a > c)||(c>a && a>b)) return a;

else return c;

}

int main() {

int T;

scanf\_s("%d", &T);

int N, X;

for (int test\_case = 1; test\_case <= T; test\_case++) {

scanf\_s("%d %d", &N,&X);

bool flag = false;

vector<int> base(2 \* N - 1);

for (int i = 0; i < base.size(); i++) base[i] = i + 1;

vector<vector<int>> pyramid(N, vector<int>(2 \* N - 1, 0));

do {

int \_size = base.size();

for (int i = 0; i < base.size(); i++) pyramid[0][i] = base[i];

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

for (int i = 2; i < \_size; i++) {

int mid = Mid(pyramid[level-1][i - 2], pyramid[level - 1][i - 1], pyramid[level - 1][i]);

pyramid[level][i - 2] = mid;

}

\_size -= 2;

}

if (pyramid[N - 1][0] == X) flag = true;

} while (next\_permutation(base.begin(), base.end())&& !flag);

if (flag) printf("#%d 1\n", test\_case);

else printf("#%d 0\n", test\_case);

}

return 0;

}

- so then, what would a trick be?

- I found the trick! I searched results of when N is 2~5, and I found 1 and 2n-1 value didn’t appear.

- see the below code.

#include<stdio.h>

int main() {

int T;

scanf\_s("%d", &T);

int N, X;

for (int test\_case = 1; test\_case <= T; test\_case++) {

scanf\_s("%d %d", &N,&X);

if (X != 1 && X != (2 \* N - 1)) printf("#%d 1\n", test\_case);

else printf("#%d 0\n", test\_case);

}

return 0;

}

**[167. [SAMSUNG** - **SW : Addition Problem]**

**-** I felt like level 4 problem’s in SAMSUNG SW expert don’t ask brute force. there is always a tricked rule.

- this problem has trick as well. the key calculation was (b-a) \* (n-2) +1.

- and I had to be careful of data type.

- see the code.

#include<stdio.h>

#include<vector>

#include<algorithm>

#include<unordered\_set>

using namespace std;

int main() {

int T; scanf\_s("%d", &T);

long long n, a, b;

for (int tc = 1; tc <= T; tc++) {

scanf\_s("%lld %lld %lld", &n, &a, &b);

if((n==1 && a!=b)||a>b) printf("#%d 0\n", tc);

else if((n==1 && a==b)||n==2 ) printf("#%d 1\n", tc);

else {

printf("#%d %lld\n", tc, (b - a) \* (n-2)+1);

}

}

return 0;

}

**[168. [SAMSUNG** - **SW : KyeonJae and DaeHwan’s Stone Game]**

**-** we can’t use DFS for this problem due to time complexity.

- so we must find a rule, the rule was if abs(r-b)<2 then, DH win, otherwise KyeongJae win.

- I drew multiple example to find why the logic works well, but honeslty, I wasn’t able to undertand mathmatically.

- see the code.

#include<stdio.h>

#include<cmath>

using namespace std;

int main() {

int T; scanf\_s("%d", &T);

long long r, b;

for (int tc = 1; tc <= T; tc++) {

scanf\_s("%lld %lld", &r, &b);

if (abs(r-b) < 2) printf("#%d DH\n", tc);

else printf("#%d KJ\n", tc);

}

return 0;

}

**[169. [SAMSUNG** - **SW : Add Parenthesis]**

**-** wow… I spent 4 hours and half…

- I misunderstood the problem. I thought I had to consider all the possible pair of parenthesis without limitation of operator a parenthesis can have.

- but each parenthesis can has only one operator.

- From there, I started to revise my algorithm. but wow… I think since I don’t solve this kind of brute force or DFS problem nowdays, it took so much time.

- see the code.

#include<iostream>

#include<string>

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

using namespace std;

string s;

int s\_size;

int N;

int answer=-987654321;

inline int calcul(const int a, const int b, const char c) {

if (c == '+') return a + b;

else if (c == '-') return a - b;

else return a \* b;

}

void DFS(const int idx,int sum) {

if (idx >= N) {

answer = max(answer, sum);

return;

}

//choose current operator

int cur\_choose = calcul(s[idx - 1]-'0', s[idx + 1] - '0', s[idx]);

if (idx > 1) cur\_choose = calcul(sum,cur\_choose , s[idx - 2]);

if (idx + 4 < N - 1) {

DFS(idx + 4, cur\_choose);

}

else if (idx + 2 < N - 1) {

DFS(idx + 4, calcul(cur\_choose, s[idx + 3] - '0', s[idx + 2]));

}

else DFS(idx + 4, cur\_choose);

//otherwise

int not\_choose = 0;

if(idx>1) not\_choose = calcul(sum, s[idx - 1] - '0', s[idx - 2]);

if (idx > 1 && idx+2<N-1) DFS(idx + 2, not\_choose);

else if (idx > 1) DFS(idx+2,calcul(not\_choose, s[idx + 1] - '0', s[idx]));

else if (idx <= 1 && N>3) DFS(idx + 2, s[0] - '0');

}

int main() {

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

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

cin >> N;

cin >> s;

if (N == 1) cout << s[0] - '0';

else {

DFS(1, 0);

cout << answer;

}

return 0;

}

**[170. [SAMSUNG** - **SW : Move Pipe 1]**

**-** it was a simulation problem. given possible directions to move, I had to move the pipe to (N,N).

- I used DFS and memorization using visit vector.

- see the code.

#include<iostream>

#include<vector>

using namespace std;

int N;

vector<vector<int>> table(17,vector<int>(17,0));

vector<vector<vector<int>>> visit(17,vector<vector<int>>(17, vector<int>(3,-1)));

/\*

void printTable(int lx,int ly,int rx,int ry) {

vector<vector<int>> temp = table;

temp[lx][ly] = 2; temp[rx][ry] = 2;

for (vector<int> row : temp) {

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

cout << endl;

}

cout << endl;

}

\*/

int MovePipe(int lx, int ly, int rx, int ry,bool hori, bool verti) {

int answer = 0;

if (rx==N-1 && ry==N-1) {

return 1;

}

int temp;

if (hori) {

if (visit[lx][ly][0] == -1) {

//to horizon

visit[lx][ly][0] = 0;

if (ry+1<N && table[rx][ry + 1] != 1) {

temp = MovePipe(lx, ly + 1, rx, ry + 1, true, false);

visit[lx][ly][0] += temp;

answer += temp;

}

//to diagonal

if (rx+1<N && ry+1< N&&table[rx][ry + 1] != 1 && table[rx + 1][ry] != 1 && table[rx + 1][ry + 1] != 1) {

temp = MovePipe(lx, ly + 1, rx + 1, ry + 1, false, false);

visit[lx][ly][0] += temp;

answer += temp;

}

}

else {

answer += visit[lx][ly][0];

}

}

else if (verti) {

if (visit[lx][ly][1] == -1) {

visit[lx][ly][1] = 0;

//to vertical

if (rx+1<N && table[rx + 1][ry] != 1) {

temp = MovePipe(lx + 1, ly, rx + 1, ry, false, true);

visit[lx][ly][1] += temp;

answer += temp;

}

//to diagonal

if (rx+1<N && ry+1< N &&table[rx + 1][ry] != 1 && table[rx + 1][ry + 1] != 1 && table[rx][ry + 1] != 1) {

temp = MovePipe(lx + 1, ly, rx + 1, ry + 1, false, false);

visit[lx][ly][1] += temp;

answer += temp;

}

}

else {

answer += visit[lx][ly][1];

}

}

else {

if (visit[lx][ly][2] == -1) {

visit[lx][ly][2] = 0;

//to horizon

if (ry+1<N &&table[rx][ry + 1] != 1) {

temp = MovePipe(lx + 1, ly + 1, rx, ry + 1, true, false);

visit[lx][ly][2] += temp;

answer += temp;

}

//to diagonal

if (rx+1<N && ry+1<N && table[rx + 1][ry] != 1 && table[rx + 1][ry + 1] != 1 && table[rx][ry + 1] != 1) {

temp = MovePipe(lx + 1, ly + 1, rx + 1, ry + 1, false, false);

visit[lx][ly][2] += temp;

answer += temp;

}

//to vertical

if (rx+1<N && table[rx + 1][ry] != 1) {

temp = MovePipe(lx + 1, ly + 1, rx + 1, ry, false, true);

visit[lx][ly][2] += temp;

answer+=temp;

}

}

else {

answer += visit[lx][ly][2];

}

}

return answer;

}

int main() {

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

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

//get input

cin >> N;

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

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

bool hori, verti; //if both are false, then dir is diagonal.

//algorithm part

if (table[N - 1][N - 1] == 1) cout << 0;

else cout<<MovePipe(0,0,0,1,true, false);

return 0;

}

**[171. [SAMSUNG** - **SW : Castle Defence]**

**-** it was also a simulation problem.

- there was a limitation of way to kill an enemy by an archer. because of that, it was quite confused.

- the limitation was that an archer can kill a leftmost enemy with the shortest distance from oneself. the leftmost was the key of this problem.

- without above issue, there was nothing to consider hard.

- I used DFS to choose archers’ position

- see the code.

#include<iostream>

#include<vector>

#include<set>

#define MAX\_VALUE 987654321

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

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

using namespace std;

int N, M, D;

vector<vector<int>> table;

vector<int> archers;

int answer = 0;

int max\_enemy;

bool isEmpty(vector<vector<int>>& temp) {

for (vector<int> row : temp)

for (int i : row) if (i == 1) return false;

return true;

}

void printTable(vector<vector<int>>& temp) {

for (vector<int> row : temp) {

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

cout << endl;

}

cout << endl;

}

void moveEnemy(vector<vector<int>>& temp) {

//Enemies come down to the castles.

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

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

temp[i][j] = temp[i - 1][j];

}

}

for (int j = 0; j < M; j++) temp[0][j] = 0;

}

int PlayGame() {

vector<vector<int>> temp = table;

int archer\_x = N;

int res = 0;

while (!isEmpty(temp)) {

vector<pair<int, int>> temp\_remove(3, make\_pair(MAX\_VALUE, MAX\_VALUE));

set<pair<int, int>> remove;

int fst\_dis = MAX\_VALUE; int sec\_dis = MAX\_VALUE; int thr\_dis = MAX\_VALUE;

for (int i = N - 1; i >= 0 && (abs(i-archer\_x)<=D); i--) {

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

if (temp[i][j] == 1) {

int a = (abs(i - archer\_x) + abs(j - archers[0]));

int b = (abs(i - archer\_x) + abs(j - archers[1]));

int c = (abs(i - archer\_x) + abs(j - archers[2]));

//cout<<"f,s,t : " << fst\_dis << "," << sec\_dis << "," << thr\_dis << endl;

//cout << "a,b,c :" << a << "," << b << "," << c << endl;

if ( a <= D && fst\_dis>=a) {

if (fst\_dis == a) {

if (temp\_remove[0].second > j) temp\_remove[0] = make\_pair(i, j);

}

else {

fst\_dis = a;

temp\_remove[0] = make\_pair(i, j);

}

}

if ( b <= D && sec\_dis >= b) {

if (sec\_dis == b) {

if (temp\_remove[1].second > j) temp\_remove[1] = make\_pair(i, j);

}

else {

sec\_dis = b;

temp\_remove[1] = make\_pair(i, j);

}

}

if ( c <= D && thr\_dis >= c) {

if (thr\_dis == c) {

if (temp\_remove[2].second > j) temp\_remove[2] = make\_pair(i, j);

}

else {

thr\_dis = c;

temp\_remove[2] = make\_pair(i, j);

}

}

}

}

}

for (pair<int, int> temp\_re : temp\_remove) if(temp\_re.first!=MAX\_VALUE) remove.insert(temp\_re);

res += remove.size();

//cout << "before killing" << endl;

//printTable(temp);

for (pair<int, int> removed\_enemy : remove) {

temp[removed\_enemy.first][removed\_enemy.second] = 0;

}

//cout << "after killing" << endl;

//printTable(temp);

moveEnemy(temp);

}

return res;

}

void DFS(int start, int count) {

if (count == 3) {

answer=max(answer,PlayGame());

return;

}

// if we got all the enemy by an archers' position, we don't need to do further.

for (int i = start; i < M && max\_enemy!=answer; i++) {

archers.emplace\_back(i);

DFS(i + 1, count + 1);

archers.pop\_back();

}

}

int main() {

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

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

//get input

cin >> N>> M>> D;

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

int input;

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

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

cin >> input;

table[i][j] = input;

if (input == 1) max\_enemy++;

}

}

//algorithm part

if (isEmpty(table)) cout << 0;

else {

DFS(0, 0);

cout << answer;

}

return 0;

}

**[172. [SAMSUNG** - **SW : Attach Sugar Paper]**

- below code is my wrong algorithm. I used BFS to find maximu size of block we can attach with current connected block. but it consider just first max block. so it’s wrong.

- see the code.

#include<iostream>

#include<array>

#include<queue>

using namespace std;

int dir[4][2] = { {-1,0}, {0,-1},{1,0},{0,1} }; // top left bottom right

vector<int> rest = { 5,5,5,5,5,5 };

int answer = 0;

int blockSize(array<array<int, 10>, 10>& table, int x, int y) {

int k=0;

for (k = 1; k < 5 ; k++) {

if (x + k >= 10 || y + k >= 10) return k - 1;

for (int i = x; i <= x + k ; i++) if (table[i][y + k] != 1) return k-1;

for (int j = y; j < y + k ; j++) if (table[x + k][j] != 1) return k-1;

}

if (k == 5) return k - 1;

return k;

}

void attach(array<array<int, 10>, 10>& table, int x, int y, int m) {

table[x][y] = 0;

for (int k = 1; k <= m; k++) {

for (int i = x; i <= x + k && x + k < 10 && y + k < 10; i++) table[i][y + k] = 0;

for (int j = y; j < y + k && y + k < 10 && x + k < 10; j++) table[x + k][j] = 0;

}

}

void findBlock(array<array<int,10>,10>& table, int& number\_of\_one) {

queue<pair<int, int>> que;

int sx, sy;

bool flag = true;

for (int i = 0; i < 10 && flag; i++) {

for (int j = 0; j < 10; j++) {

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

sx = i; sy = j;

flag = false;

break;

}

}

}

que.push(make\_pair(sx,sy));

//BFS

int max\_block\_size = 0; int mx=sx; int my=sy;

vector<vector<int>> visit(10, vector<int>(10, 0));

int temp\_size = 0;

while (!que.empty() && max\_block\_size<5) {

int x = que.front().first; int y = que.front().second; que.pop();

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

visit[x][y] = 1;

temp\_size = blockSize(table, x, y);

if (max\_block\_size < temp\_size && rest[temp\_size]!=0) {

max\_block\_size = temp\_size;

mx = x; my = y;

}

}

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

int dx = x+dir[i][0]; int dy = y+dir[i][1];

if ( dx >= 0 && dy >= 0 && dx < 10 && dy < 10&&visit[dx][dy] == 0 && table[dx][dy] == 1 )

que.push(make\_pair(dx,dy));

}

}

if (rest[max\_block\_size] != 0) {

cout << "mx, my, size : " << mx << "," << my << "," << max\_block\_size + 1 << endl;

attach(table, mx, my, max\_block\_size);

number\_of\_one -= (max\_block\_size + 1) \* (max\_block\_size + 1);

answer++;

rest[max\_block\_size]--;

}

}

int main() {

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

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

array<array<int, 10>, 10> table;

// get input

int number\_of\_one = 0;

for (int i = 0; i < 10; i++)

for (int j = 0; j < 10; j++) {

cin >> table[i][j];

if (table[i][j] == 1) number\_of\_one++;

}

int loof\_count = 0;

//algorithm part

while (number\_of\_one > 0) {

if (loof\_count >= 50) {

answer = -1;

break;

}

findBlock(table, number\_of\_one);

loof\_count++;

}

cout << answer;

return 0;

}

- this is revised code. I used DFS to check all the possible block I can take with current coordinates.

- by recording one’s position, I can have terminated DFS loof.

- see the code.

#include<iostream>

#include<array>

#include<vector>

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

using namespace std;

array<int, 5 > rest\_paper= {5, 5, 5, 5, 5};

int answer = 987654321;

int number\_of\_one = 0;

vector<pair<int, int>> one\_pos;

bool attachPaper(array<array<int, 10>, 10>& temp,int x,int y, int k) {

bool flag = true;

for (int i = x; i <= k+x&&flag; i++) {

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

if (temp[i][j] == 0) {

flag = false;

break;

}

temp[i][j] = 0;

}

}

return flag;

}

void DFS(array<array<int, 10>, 10> temp,int used\_paper,int start,int used\_one) {

if (used\_one == number\_of\_one) {

answer = min(answer, used\_paper);

return;

}

else if (used\_one > number\_of\_one) return;

if (start >= one\_pos.size()) {

return;

}

int x = one\_pos[start].first; int y = one\_pos[start].second;

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

DFS(temp, used\_paper, start + 1, used\_one);

}

for (int j = 0; j < 5; j++) {

if (rest\_paper[j] == 0) continue;

if (x + j >= 10 || y + j >= 10) break;

rest\_paper[j]--;

array<array<int, 10>, 10> next\_table=temp;

bool flag=attachPaper(next\_table, x, y, j);

used\_one += (j + 1) \* (j + 1);

if(flag) DFS(next\_table, used\_paper+1,start+1,used\_one);

used\_one -= (j + 1) \* (j + 1);

rest\_paper[j]++;

}

}

int main() {

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

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

array<array<int, 10>, 10> table;

// get input

for (int i = 0; i < 10; i++)

for (int j = 0; j < 10; j++) {

cin >> table[i][j];

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

number\_of\_one++;

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

}

}

// algorithm part

DFS(table, 0, 0, 0);

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

else cout << answer;

return 0;

}

**[173. [SAMSUNG** - **SW : Baseball]**

**-** What the heck? I spent 4 hours…

- I first misunderstood the problem. since they gave me a rule that once I decide an order of players, I can’t change it even if inning is changed. but I thought I have to find optimum score we can get using find all the path we can reach. so I did brute force for every inning.

- because of that, I spend quite much time.

- second problem was scoring, I tried to make score function iteratively, but it didn’t work well. actually I don’t make sense why it was wrong until now!.

- so I changed that part a hard code.

- see the code.

#include<iostream>

#include<vector>

#include<array>

#include<algorithm>

#include<map>

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

using namespace std;

int N;

vector<vector<int>> table;

vector<int> order;

int answer = 0;

void printInning(vector<int>& inning) {

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

cout << endl;

}

void Game() {

int cur\_player = 0;

int res = 0;

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

array<bool, 4> base = { false,false,false,false };

int outs = 0;

vector<int> temp;

for (int j = 0; j < 8; j++) temp.emplace\_back(table[i][order[j]]);

temp.insert(next(temp.begin(), 3), table[i][0]);

while (outs < 3) {

int cur\_hit = temp[cur\_player];

if (cur\_hit == 0) {

outs++;

cur\_player = (cur\_player + 1) % 9;

continue;

}

else if (cur\_hit == 4) {

for (int c = 3; c >= 1; c--) {

if (base[c]) res++;

base[c] = false;

}

res++;

}

else if (cur\_hit == 1) {

if (base[3]) {

res++;

base[3] = false;

}

if (base[2]) {

base[3] = true; base[2] = false;

}

if (base[1]) {

base[2] = true; base[1] = false;

}

base[1] = true;

}

else if (cur\_hit == 2) {

if (base[3]) {

res++; base[3] = false;

}

if (base[2]) {

res++; base[2] = false;

}

if (base[1]) {

base[3] = true; base[1] = false;

}

base[2] = true;

}

else {

if (base[3]) {

res++; base[3] = false;

}

if (base[2]) {

res++; base[2] = false;

}

if (base[1]) {

res++; base[1] = false;

}

base[3] = true;

}

cur\_player = (cur\_player + 1) % 9;

}

}

answer = max(answer, res);

}

int main() {

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

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

//get input

cin >> N;

table.assign(N,vector<int>(9));

int input;

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

for (int j = 0; j < 9; j++) {

cin >> table[i][j];

}

}

for (int i = 1; i < 9; i++) order.emplace\_back(i);

//algorithm part

do {

Game();

} while (next\_permutation(order.begin(), order.end()));

cout << answer;

return 0;

}

**[174. [SAMSUNG** - **SW : Brainfuck Interpreter]**

**-** like the name, it was a fucking interpreter.

- they gave an ambiguous rule about infinite loop. because of that, I struggled to solve it hard.

- see the code.

#include<iostream>

#include<vector>

#include<map>

#define MAX\_VALUE 256

using namespace std;

map<int, int> parenthesis\_left;

map<int, int> parenthesis\_right;

pair<int,int> loop;

bool flag = false;

void getPair(string brainfuck) {

vector<int> temp;

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

if (brainfuck[i] == '[') {

temp.emplace\_back(i);

}

else if (brainfuck[i] == ']') {

parenthesis\_left[temp.back()] = i;

parenthesis\_right[i] = temp.back();

temp.pop\_back();

}

}

}

void operate(vector<int>& memory,string& brainfuck,int& order\_index,int& pointer,string& input,int& input\_index) {

char order = brainfuck[order\_index];

switch (order) {

case '-':

memory[pointer] -= 1;

if (memory[pointer] < 0) memory[pointer] = 255;

order\_index++;

break;

case '+':

memory[pointer] = (memory[pointer] + 1) % 256;

order\_index++;

break;

case '<':

pointer--;

if (pointer < 0) pointer = memory.size() - 1;

order\_index++;

break;

case '>':

pointer = (pointer + 1) % memory.size();

order\_index++;

break;

case '[':

if (memory[pointer] == 0) {

order\_index = parenthesis\_left[order\_index];

break;

}

order\_index++;

break;

case ']':

loop.first = parenthesis\_right[order\_index];

loop.second = order\_index;

if (memory[pointer] != 0) {

order\_index = parenthesis\_right[order\_index];

break;

}

order\_index++;

break;

case '.':

order\_index++;

break;

case ',':

int temp\_input;

if (input\_index < input.size()) {

temp\_input = (int)input[input\_index];

memory[pointer] = temp\_input;

}

else memory[pointer] = 255;

input\_index++;

order\_index++;

break;

}

}

int main() {

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

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

int t; cin >> t;

//for each test case

int m, c, i;

string brainfuck;

string input;

vector<int> memory;

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

//get input

cin >> m >> c >> i>> brainfuck>> input;

memory.assign(m, 0);

//algoirhtm part

getPair(brainfuck); //get parenthesis pair

int order\_index = 0;

int pointer = 0;

int cnt = 0;

int input\_index = 0;

flag = true;

loop = make\_pair(-1, -1);

int max\_order\_index = 0;

while (cnt< 50000000) {

if (order\_index>=c) {

cout << "Terminates" << endl;

flag = false;

break;

}

operate(memory,brainfuck, order\_index,pointer, input,input\_index);

cnt++;

max\_order\_index = order\_index > max\_order\_index ? order\_index : max\_order\_index;

}

if (flag) {

cout << "Loops " << parenthesis\_right[max\_order\_index] << " " << max\_order\_index << endl;

}

//map clear

parenthesis\_left.clear();

parenthesis\_right.clear();

}

return 0;

}

**[175. [KAKAO 2019** – **Winter Internship : Claw Crane Game]**

**-** this was so easy. I think I don’t need comment for this problem.

- see the code.

#include <string>

#include <vector>

#include<iostream>

using namespace std;

int solution(vector<vector<int>> board, vector<int> moves) {

int answer = 0;

//find top

vector<int> tops(board[0].size(), 0);

for (int j = 0; j < board[0].size(); j++) {

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

if (board[i][j] != 0) {

tops[j] = i;

break;

}

}

}

vector<int> stk;

for (int index : moves) {

if (tops[index - 1] >= board.size()) continue;

int cur\_char = board[tops[index - 1]][index - 1];

tops[index - 1]++;

if (!stk.empty() && stk.back() == cur\_char) {

answer += 2;

stk.pop\_back();

}

else {

stk.push\_back(cur\_char);

}

//[1,5,3,5,1,2,1,4]

}

return answer;

}

int main() {

cout<<solution({ {0, 0, 0, 0, 0},{0, 0, 1, 0, 3},{0, 2, 5, 0, 1},{4, 2, 4, 4, 2},{3, 5, 1, 3, 1} }, { 1, 5, 3, 5, 1, 2, 1, 4 });

return 0;

}

**[176. [KAKAO 2019** – **Winter Internship : Tuple]**

- just testing that I know the use of vector and find algorithm.

- string parsing was the key point.

- see the code.

#include <string>

#include <vector>

#include<unordered\_set>

#include <algorithm>

using namespace std;

vector<int> solution(string s) {

vector<int> answer;

vector<vector<int>> res(501);

int max\_size = 0;

//solution("{{2},{2,1},{2,1,3},{2,1,3,4}}");

for (int i = 2; i < s.size() - 1; i++) {

if (s[i] == '{' || s[i] == ',') continue;

string temp = "";

int input;

vector<int> temp\_res;

int j;

for (j = i; j < s.size() - 1 ; j++) {

if (s[j] == ',' || s[j]=='}') {

input = stoi(temp);

temp\_res.emplace\_back(input);

temp = "";

if (s[j] == '}') break;

}

else {

temp += s[j];

}

}

max\_size = temp\_res.size() > max\_size ? temp\_res.size() : max\_size;

res[temp\_res.size()] = temp\_res;

i = j;

}

vector<int> make\_tuple;

make\_tuple.emplace\_back(res[1][0]);

for (int i = 2; i <= max\_size; i++) {

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

if (find(make\_tuple.begin(), make\_tuple.end(), res[i][j]) == make\_tuple.end()) {

make\_tuple.emplace\_back(res[i][j]);

break;

}

else continue;

}

}

return make\_tuple;

}

int main() {

solution("{{20,111},{111}}");

return 0;

}

**[177. [KAKAO 2019** – **Winter Internship : Banned User]**

**-** doing permutation was the key point.

- thanks to time limit was not that restric. I’ve solved easily using DFS – permutation.

- see the code.

#include <string>

#include <vector>

#include <set>

#include<iostream>

using namespace std;

int answer = 0;

vector<vector<string>> possible;

vector<vector<string>> temp\_answer;

set<set<string>> res\_answer;

int ban\_size;

void DFS(vector<string> current\_set, int index) {

if (index >= ban\_size) {

set<string> res;

for (string tt : current\_set) {

res.insert(tt);

}

if (res.size() == ban\_size) res\_answer.insert(res);

return;

}

for (int i = 0; i < possible[index].size(); i++) {

current\_set.emplace\_back(possible[index][i]);

DFS(current\_set, index + 1);

current\_set.pop\_back();

}

}

int solution(vector<string> user\_id, vector<string> banned\_id) {

possible.assign(banned\_id.size(), vector<string>{});

for (int j = 0; j < banned\_id.size(); j++) {

string ban = banned\_id[j];

for (string user : user\_id) {

if (ban.size() != user.size()) continue;

bool flag = true;

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

if (ban[i] == '\*' || ban[i] == user[i]) continue;

if (ban[i] != user[i]) {

flag = false;

break;

}

}

if (flag) possible[j].emplace\_back(user);

}

}

ban\_size = possible.size();

DFS(vector<string>{}, 0);

for (set<string> ress : res\_answer) {

if (ress.size() != ban\_size) continue;

answer++;

}

return answer;

}

**[178. [KAKAO 2019** – **Winter Internship : Allocate Hotel Room]**

**-** this problem has efficiency test. the key was the union-find structure.

- but without knowing union-find structure, we colud solve it. I’ve solved this problem using recursion with unorderd\_map.

- the map has pair as a value, the first of the pair is prev, the second is next. but I think now, I just need the second. so we just have to make unordered\_map<int,int> structure.

- if map[input\_room] is empty, then we just push the room number into answer vector.

- if the map is not empty, then give the map[input\_room].second to next recursion until we find empty room. after that, we revise each key’s second to new next room.

- it’s hard to describe in written language. just see the code.

#include <string>

#include <vector>

#include <unordered\_map>

using namespace std;

unordered\_map<long long, pair<long long, long long>> next\_room;

vector<long long> answer;

void recursion(long long wanted\_room) {

if (next\_room.find(wanted\_room) == next\_room.end()) {

answer.emplace\_back(wanted\_room);

next\_room[wanted\_room].first = 0;

next\_room[wanted\_room].second = wanted\_room + 1;

return;

}

else {

recursion(next\_room[wanted\_room].second);

long long next = next\_room[next\_room[wanted\_room].second].second;

next\_room[wanted\_room].second = next;

return;

}

}

vector<long long> solution(long long k, vector<long long> room\_number) {

for (long long wanted\_room : room\_number) {

recursion(wanted\_room);

}

return answer;

}

**[179. [KAKAO 2019** – **Winter Internship : Stepping Stones]**

- this problem also has efficiency test. my first try pass all the accuracy test, but not efficiency test.

- see my first code.

#include <string>

#include <vector>

#include<iostream>

using namespace std;

int solution(vector<int> input\_stones, int max\_interval) {

int answer = 0;

vector<pair<int, int>> stones; // pair<value,next\_stone>

stones.emplace\_back(make\_pair(200000001, 1)); //start position

for (int i = 0; i < input\_stones.size(); i++) stones.emplace\_back(make\_pair(input\_stones[i], i + 2));

//algorithm part

while (true) {

bool stop\_flag = false;

for (int i = 1; i < stones.size(); i++) {

//when the stone's value is greater than 1

if (stones[i].first != 0) {

stones[i].first--;

if (stones[i].first == 0) {

int next = stones[i].second;

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

if (stones[j].first != 0) {

stones[j].second = next;

break;

}

else stones[j].second = next;

}

}

}

//when the stone's value is 0

else {

if (stones[i - 1].second - (i-1) > max\_interval) {

stop\_flag = true;

break;

}

i = stones[i - 1].second - 1; // -1 is to make up for 'for' syntax's i++

}

}

if (stop\_flag) break;

else answer++;

}

return answer;

}

int main() {

cout << solution({2, 4, 5, 3, 2, 1, 4, 2, 5, 1},3);

return 0;

}

- let’s figure out to pass the efficiency test.

- I think we don’t need to travel all the stone, since for each person, all the stone’s value is subtracted by 1.

- if we use binary search, we can solve this problem easily. mid is the value we check for each iteration whether the mid number of people can pass the stepping stones or not.

- so if mid number of people can pass, take right half, or take left half.

- see the code.

#include <string>

#include <vector>

#include<iostream>

#define MAX\_VALUE 200000000

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

using namespace std;

int answer = 0;

bool isPossibleToJump(vector<int>& stones,int mid,int max\_interval) {

int cnt = 0;

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

if (stones[i] - (mid-1) <= 0) {

cnt++;

if (cnt >= max\_interval) return false;

}

else cnt = 0;

}

return true;

}

void binarySearch(vector<int>& stones,int left, int right,int max\_interval) {

if (left > right) return;

int mid = (right + left) / 2;

bool flag = isPossibleToJump(stones, mid, max\_interval);

if (flag) {

answer = max(answer, mid);

binarySearch(stones, mid + 1, right, max\_interval);

}

else {

binarySearch(stones, left, mid - 1, max\_interval);

}

}

int solution(vector<int> stones, int max\_interval) {

binarySearch(stones, 0, MAX\_VALUE, max\_interval);

return answer;

}

int main() {

cout << solution({2, 4, 5, 3, 2, 1, 4, 2, 5, 1},3);

return 0;

}

**[180. [KAKAO 2017 : Coloring Book]**

**-** this was a simulation problem. I solved using BFS.

- it’s quite long time since I used BFS lastly. so It took much more time.

- see the code.

#include <vector>

#include<queue>

#include<iostream>

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

using namespace std;

int N, M;

int dir[4][2] = { {-1,0},{0,-1},{1,0},{0,1} }; // north west south east

int BFS(int \_x, int \_y, vector<vector<int>>& table, vector<vector<bool>>& visit) {

int cur\_size = 0;

int color = table[\_x][\_y];

queue<pair<int, int>> que;

que.push(make\_pair(\_x, \_y));

visit[\_x][\_y] = true;

while (!que.empty()) {

int x = que.front().first; int y = que.front().second;

que.pop();

cur\_size++;

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 && table[nx][ny] == color) {

que.push(make\_pair(nx, ny));

visit[nx][ny] = true;

}

}

}

return cur\_size;

}

vector<int> solution(int row\_size, int col\_size, vector<vector<int>> picture) {

N = row\_size; M = col\_size;

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

int area = 0;

int max\_size = 0;

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

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

if (visit[i][j] == false && picture[i][j] != 0) {

area++;

int cur\_size=BFS(i, j, picture,visit);

max\_size = max(max\_size, cur\_size);

}

}

}

vector<int> answer(2);

answer[0] = area;

answer[1] = max\_size;

cout << answer[0] << "," << answer[1] << endl;

return answer;

}

int main() {

solution(6, 4, { {1, 1, 1, 0},{1, 2, 2, 0},{1, 0, 0, 1},{0, 0, 0, 1},{0, 0, 0, 3},{0, 0, 0, 3} });

//solution(6, 4, { {1, 1, 1, 0},{1, 1, 1, 0},{0, 0, 0, 1},{0, 0, 0, 1},{0, 0, 0, 1},{0, 0, 0, 1} });

//solution(4, 6, {{1,1,1,1,0,0}, {1,1,1,0,0,0},{2,2,0,0,3,3}, {2,0,1,0,3,3} });

return 0;

}

**[181. [SAMSUNG SW – Rotate Array 4]**

- this was a simulation problem.

- I had to rotate part of given 2D array clock-wisely, and caclutate minimum value among each row.

- I was able to choose rotating order to minimize the answer value. I implemented a permutation using DFS.

- see the code.

#include<iostream>

#include<vector>

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

using namespace std;

int N, M, K;

int answer = 987654321;

vector<vector<int>> rotates;

vector<bool> visit;

vector<vector<int>> rotateTable(vector<vector<int>> temp\_table,vector<int> temp\_rotate) {

int r = temp\_rotate[0] - 1; int c = temp\_rotate[1] - 1; int s = temp\_rotate[2];

int tx = r - s; int ty = c - s; int bx = r + s; int by = c + s;

//rotating part

while (tx<bx && ty<by) {

int start\_val = temp\_table[tx][ty];

for (int i = tx; i < bx; i++) temp\_table[i][ty] = temp\_table[i + 1][ty];

for (int j = ty; j < by; j++) temp\_table[bx][j] = temp\_table[bx][j + 1];

for (int i = bx; i > tx; i--) temp\_table[i][by] = temp\_table[i - 1][by];

for (int j = by; j > ty + 1; j--) temp\_table[tx][j] = temp\_table[tx][j - 1];

temp\_table[tx][ty + 1] = start\_val;

tx++; ty++; bx--; by--;

}

return temp\_table;

}

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

int minimum = 987654321;

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

int temp\_sum = 0;

for (int i : row) temp\_sum += i;

minimum = min(minimum, temp\_sum);

}

answer = min(answer, minimum);

}

void permutation(vector<vector<int>> temp\_table,int start) {

if (start >= K) {

getMinimum(temp\_table);

return;

}

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

if (visit[i] == false) {

visit[i] = true;

permutation(rotateTable(temp\_table, rotates[i]), start + 1);

visit[i] = false;

}

}

}

int main() {

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

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

// get input

cin >> N >> M >> K;

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];

}

}

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

visit.assign(K, false);

for (int i = 0; i < K; i++) cin >> rotates[i][0] >> rotates[i][1] >> rotates[i][2];

//algorithm part

permutation(table,0);

cout << answer;

return 0;

}

**[182. [SAMSUNG SW – Gerrymandering 1]**

- Checking two districts are connected each other or not was the main point of this problem.

- To split district, I used permutation – DFS, and to check connection, I used queue.

- see the code.

#include<iostream>

#include<vector>

#include<unordered\_set>

#include<algorithm>

#include<climits>

#include<queue>

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

using namespace std;

int N;

vector<vector<int>> table;

vector<int> people;

vector<bool> visit;

int answer = INT\_MAX;

void calcul(vector<int>& dis\_a, vector<int>& dis\_b) {

int sum\_a = 0; int sum\_b = 0;

for (int i : dis\_a) sum\_a += people[i];

for (int i : dis\_b) sum\_b += people[i];

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

}

bool isValid(vector<int>& dis\_a, vector<int>& dis\_b) {

if (dis\_a.empty() || dis\_b.empty()) return false;

int size\_a = dis\_a.size(); int size\_b = dis\_b.size();

queue<int> que;

que.push(dis\_a[0]);

unordered\_set<int> res\_a; unordered\_set<int> res\_b;

vector<bool> visit\_a(N, false); vector<bool> visit\_b(N, false);

visit\_a[que.front()] = true;

res\_a.insert(que.front());

while (!que.empty()) {

int cur\_a = que.front(); que.pop();

for (int i = 0; i < table[cur\_a].size(); i++) {

if (find(dis\_a.begin(), dis\_a.end(), table[cur\_a][i]) != dis\_a.end() && visit\_a[table[cur\_a][i]]==false) {

visit\_a[table[cur\_a][i]] = true;

que.push(table[cur\_a][i]);

res\_a.insert(table[cur\_a][i]);

}

}

}

if (res\_a.size() != size\_a) return false;

que.push(dis\_b[0]);

visit\_b[que.front()] = true;

res\_b.insert(que.front());

while (!que.empty()) {

int cur\_b = que.front(); que.pop();

for (int i = 0; i < table[cur\_b].size(); i++) {

if (find(dis\_b.begin(), dis\_b.end(), table[cur\_b][i]) != dis\_b.end() && visit\_b[table[cur\_b][i]] == false) {

visit\_b[table[cur\_b][i]] = true;

que.push(table[cur\_b][i]);

res\_b.insert(table[cur\_b][i]);

}

}

}

if (res\_b.size() != size\_b) return false;

return true;

}

void DFS(vector<int> dis\_a,int start) {

if (start == N-1) return;

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

if (visit[i] == false) {

vector<int> dis\_b;

visit[i] = true;

dis\_a.emplace\_back(i);

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

if (find(dis\_a.begin(), dis\_a.end(), j) == dis\_a.end()) {

dis\_b.emplace\_back(j);

}

}

if (isValid(dis\_a, dis\_b)) calcul(dis\_a, dis\_b);

DFS(dis\_a, start + 1);

dis\_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>());

people.assign(N, 0);

visit.assign(N, false);

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

int edge;

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

cin >> edge;

table[i].assign(edge, 0);

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

int temp\_input;

cin >> temp\_input;

table[i][j] = temp\_input - 1;;

}

}

//algorithm part

DFS(vector<int>{}, 0);

if (answer == INT\_MAX) cout << -1;

else cout << answer;

return 0;

}

**[182. [SAMSUNG SW – Making Bridge 2]**

**-** At first, I used brute force using permutation but it was hard to check all island was connected.

- see the first code.

#include<iostream>

#include<vector>

#include<unordered\_map>

#include<map>

#include<set>

#include<unordered\_set>

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

using namespace std;

int N, M;

int number\_of\_island = 0;

int needed\_bridge = 0;

vector<vector<int>> table;

int dir[4][2] = { {-1,0},{0,-1},{1,0},{0,1} }; //tx ty bx by size

vector<vector<bool>> visit;

unordered\_map<int, vector<int>> map\_of\_land;

map<pair<int, int>, int> bridge;

int answer = 987654321;

bool isConnected(set<set<int>>& temp\_table) {

unordered\_set<int> res;

for (set<set<int>>::iterator iter = temp\_table.begin(); iter != temp\_table.end(); iter++) {

for (set<int>::iterator inner = iter->begin(); inner != iter->end(); inner++) {

res.insert(\*inner);

}

}

if (res.size() == number\_of\_island) return true;

return false;

}

void nameDistrict(int x, int y,int name) {

table[x][y] = name;

visit[x][y] = true;

if (map\_of\_land[name][2] < x) map\_of\_land[name][2] = x;

if (map\_of\_land[name][3] < y) map\_of\_land[name][3] = 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 && table[nx][ny] == 1 && visit[nx][ny]==false) {

map\_of\_land[name][4]++;

nameDistrict(nx, ny, name);

}

}

}

void findMinimumBridge() {

//horizontal bridge

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

int cur\_land = 1;

int sy = 0;

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

if (table[i][j] != 0 && cur\_land != table[i][j]) {

if (j - sy - 1 >= 2 && cur\_land!=1) {

if (bridge.find(make\_pair(cur\_land, table[i][j])) == bridge.end()) {

bridge[make\_pair(cur\_land, table[i][j])] = j - sy - 1;

}

else bridge[make\_pair(cur\_land, table[i][j])] = min(bridge[make\_pair(cur\_land, table[i][j])], j - sy + 1);

}

cur\_land = table[i][j];

sy = j;

}

else if (cur\_land == table[i][j]) {

sy = j;

}

}

}

//vertical bridge

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

int cur\_land = 1;

int sx = 0;

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

if (table[i][j] != 0 && cur\_land != table[i][j]) {

if (i - sx - 1 >= 2 && cur\_land!=1) {

if (bridge.find(make\_pair(cur\_land, table[i][j])) == bridge.end()) {

bridge[make\_pair(cur\_land, table[i][j])] = i - sx - 1;

}

else bridge[make\_pair(cur\_land, table[i][j])] = min(bridge[make\_pair(cur\_land, table[i][j])], i - sx + 1);

}

cur\_land = table[i][j];

sx = i;

}

else if (cur\_land == table[i][j]) sx = i;

}

}

}

void Permutation(set<set<int>> temp\_table,int start,int sum,map<pair<int,int>,int>::iterator next\_iter) {

if (start >= needed\_bridge) {

if (isConnected(temp\_table)) {

answer = min(answer, sum);

for (set<set<int>>::iterator iter = temp\_table.begin(); iter != temp\_table.end(); iter++) {

for (set<int>::iterator inner = iter->begin(); inner != iter->end(); inner++) {

cout << \*inner << " ";

}

cout << ", ";

}

cout << endl << "answer : " << answer << endl;

}

return;

}

for (map<pair<int, int>, int>::iterator iter = next\_iter; iter != bridge.end(); iter++) {

temp\_table.insert(set<int>{iter->first.first,iter->first.second});

Permutation(temp\_table, start + 1, sum + iter->second, next(iter,1));

temp\_table.erase(temp\_table.find(set<int>{iter->first.first, iter->first.second}));

}

}

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 >> M;

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

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

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

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

//algorithm part

// 1. counting island and make their name

// the name start with 2

int name = 2;

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

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

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

map\_of\_land[name] = vector<int>{ i,j,i,j,1 };

number\_of\_island++;

nameDistrict(i, j, name++);

}

}

}

needed\_bridge = number\_of\_island - 1;

cout << endl;

printTable();

cout << endl;

// 2. making bridge

findMinimumBridge();

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

cout << iter->first.first << " " << iter->first.second << ",";

}

cout << endl;

Permutation(set<set<int>>{},0,0,bridge.begin());

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

else cout << answer;

return 0;

}

- so I changed it to Kurskal algorithm.

- hu.. fucking Kruskal. there are lots of thing to think!

- I spent 2 hours and half… In this stance, I couldn’t get through the SAMSUNG coding test…

- I have to push myself up.

- see the code.

#include<iostream>

#include<vector>

#include<unordered\_map>

#include<map>

#include<set>

#include<unordered\_set>

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

using namespace std;

int N, M;

int number\_of\_island = 0;

int needed\_bridge = 0;

vector<vector<int>> table;

int dir[4][2] = { {-1,0},{0,-1},{1,0},{0,1} }; //tx ty bx by size

vector<vector<bool>> visit;

unordered\_map<int, vector<int>> map\_of\_land;

map<pair<int, int>, int> bridge;

unordered\_set<int> has\_bridge;

int answer = 987654321;

void nameDistrict(int x, int y,int name) {

table[x][y] = name;

visit[x][y] = true;

if (map\_of\_land[name][2] < x) map\_of\_land[name][2] = x;

if (map\_of\_land[name][3] < y) map\_of\_land[name][3] = 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 && table[nx][ny] == 1 && visit[nx][ny]==false) {

map\_of\_land[name][4]++;

nameDistrict(nx, ny, name);

}

}

}

void findMinimumBridge() {

//horizontal bridge

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

int cur\_land = 1;

int sy = 0;

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

if (table[i][j] != 0 && cur\_land != table[i][j]) {

if (j - sy - 1 >= 2 && cur\_land!=1) {

if (bridge.find(make\_pair(cur\_land, table[i][j])) == bridge.end()) {

bridge[make\_pair(cur\_land, table[i][j])] = j - sy - 1;

}

else bridge[make\_pair(cur\_land, table[i][j])] = min(bridge[make\_pair(cur\_land, table[i][j])], j - sy - 1);

}

cur\_land = table[i][j];

sy = j;

}

else if (cur\_land == table[i][j]) {

sy = j;

}

}

}

//vertical bridge

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

int cur\_land = 1;

int sx = 0;

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

if (table[i][j] != 0 && cur\_land != table[i][j]) {

if (i - sx - 1 >= 2 && cur\_land!=1) {

if (bridge.find(make\_pair(cur\_land, table[i][j])) == bridge.end()) {

bridge[make\_pair(cur\_land, table[i][j])] = i - sx - 1;

}

else bridge[make\_pair(cur\_land, table[i][j])] = min(bridge[make\_pair(cur\_land, table[i][j])], i - sx - 1);

}

cur\_land = table[i][j];

sx = i;

}

else if (cur\_land == table[i][j]) sx = i;

}

}

}

int Kruskal(int sum) {

int cur\_size = 1;

while (cur\_size<needed\_bridge) {

int temp\_x; int temp\_y; int temp\_dis = 987654321;

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

if ((has\_bridge.find(iter->first.first) != has\_bridge.end() || has\_bridge.find(iter->first.second) != has\_bridge.end()) &&

!(has\_bridge.find(iter->first.first) != has\_bridge.end() && has\_bridge.find(iter->first.second) != has\_bridge.end())) {

if (iter->second < temp\_dis) {

temp\_x = iter->first.first; temp\_y = iter->first.second; temp\_dis = iter->second;

}

}

}

if (temp\_dis == 987654321) return -1;

bridge.erase(bridge.find(make\_pair(temp\_x, temp\_y)));

//cout << "x : " << temp\_x << " y : " << temp\_y << " dis : " << temp\_dis << endl;

has\_bridge.insert(temp\_x); has\_bridge.insert(temp\_y);

cur\_size++;

sum += temp\_dis;

}

/\*for (unordered\_set<int>::iterator iter = has\_bridge.begin(); iter != has\_bridge.end(); iter++)

cout << \*iter << " ";

cout << endl;\*/

if (has\_bridge.size() != number\_of\_island) return -1;

return sum;

}

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 >> M;

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

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

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

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

//algorithm part

// 1. counting island and make their name

// the name start with 2

int name = 2;

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

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

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

map\_of\_land[name] = vector<int>{ i,j,i,j,1 };

number\_of\_island++;

nameDistrict(i, j, name++);

}

}

}

needed\_bridge = number\_of\_island - 1;

//cout << endl;

//printTable();

//cout << endl;

// 2. making bridge

findMinimumBridge();

/\*for (map<pair<int, int>, int>::iterator iter = bridge.begin(); iter != bridge.end(); iter++) {

cout << iter->first.first << " " << iter->first.second << ",";

}

cout << endl;\*/

int temp\_dis = 987654321;

int temp\_x; int temp\_y;

if (bridge.empty()) {

cout << -1;

return 0;

}

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

if (iter->second < temp\_dis) {

temp\_dis = iter->second;

temp\_x = iter->first.first; temp\_y = iter->first.second;

}

}

bridge.erase(bridge.find(make\_pair(temp\_x, temp\_y)));

has\_bridge.insert(temp\_x); has\_bridge.insert(temp\_y);

cout<<Kruskal(temp\_dis);

return 0;

}

**[183. [SAMSUNG SW – Quit The Job 2]**

- it has been a while since I’ve met DP problem.

- it was not that ambiguous and complex, but one condition has held my back

- I started with end like backpropagation, if (i+ table[i].first)> N+1, that is if he can’t finish the counsel, the most efficient way to get paid is to take money[i]=money[i+1].

- otherwise, max(today pay + next possible counsel pay,money[i+1]) is optimal. since money[i+1] is made as optimal from end of day( the quit day).

- 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>());

int duration; int pay;

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

cin >> duration >> pay;

table[i] = make\_pair(duration, pay);

}

//algorithm part

vector<long long> money(N+2,0);

int max\_money = 0;

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

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

else {

if (table[i].first + i<= N) money[i] = max(table[i].second + money[table[i].first + i], money[i + 1]);

else if (i + 1 <= N) money[i] = max(table[i].second, money[i + 1]);

else money[i] = table[i].second;

}

}

cout << money[1] << endl;

return 0;

}

**[184. [SAMSUNG SW – Chess Board Re-Painting]**

- it was a slicing and brute force problem. but understading the problem is pretty hard. since they took a word “Square”. I’ve been confused whether the “Square” is chess board’s small 1x1 square or 8x8 whole square.

- except that there was nothing to make me confused.

- see the code.

#include<iostream>

#include<vector>

#include<array>

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

using namespace std;

int N, M;

vector<vector<char>> table;

array<array<char, 8>, 8> board;

void Slicing(int x, int y){

for (int i = x; i < x + 8; i++) {

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

board[i-x][j-y] = table[i][j];

}

}

}

int getMinimum() {

//when start with white

int white\_num = 0;

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

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

if ((i + j) % 2 == 0 && board[i][j] == 'B') white\_num++;

if ((i + j) % 2 == 1 && board[i][j] == 'W') white\_num++;

}

}

//when start with black

int black\_num = 0;

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

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

if ((i + j) % 2 == 0 && board[i][j] == 'W') black\_num++;

if ((i + j) % 2 == 1 && board[i][j] == 'B') black\_num++;

}

}

return min(black\_num, white\_num);

}

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];

}

}

//algorithm part

int answer = 987654321;

for (int i = 0; i + 7 < N; i++) {

for (int j = 0; j + 7 < M; j++) {

Slicing(i, j);

answer=min(answer,getMinimum());

}

}

cout << answer;

return 0;

}

**[185. [SAMSUNG SW – Treasure]**

- this problem had a trick that if we use next\_permutation, time limit exceeded occur.

- so we have to find an algorithm but the algorithm was not that hard.

- we should just mulitply a’s minimum value with b’s maximum value for each round.

- see the code.

#include<iostream>

#include<vector>

#include<algorithm>

using namespace std;

int main() {

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

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

//get input

int N; cin >> N;

vector<int> table\_a(N);

vector<int> table\_b(N);

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

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

sort(table\_a.begin(), table\_a.end());

sort(table\_b.begin(), table\_b.end(), [](int a, int b) { return a > b; });

//algorithm part

int i = 0;

int answer = 0;

while (i<N) {

answer += table\_a[i] \* table\_b[i];

i++;

}

cout << answer;

return 0;

}

**[186. [SAMSUNG SW – Teaching]**

- it was a selective permutation problem. all the word in antarctica start with “anta” and end with “tica”. so we teach students at least more than 5 letters. Therefore, if K<=4, return is 0.

- Otherwise, after teaching ‘a’, ‘c’,’ ‘I’, ‘n’, ‘t’, we have to choose which letters we teach next. I used permutation for this section.

- see the code.

#include<iostream>

#include<vector>

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

using namespace std;

int N, K;

vector<string> table;

vector<int> rest\_letter;

vector<bool> letters(26, false);

int answer = 0;

int rest\_size;

bool isPossibleToRead(string s) {

for (int i = 4; i < s.size() - 4; i++) if (letters[s[i] - 'a'] == false) return false;

return true;

}

void Permutation(int learned\_letter,int start) {

if (learned\_letter >= K) {

int temp\_res = 0;

for (string s : table) if (isPossibleToRead(s)) temp\_res++;

answer = max(answer, temp\_res);

return;

}

for (int i = start; i < rest\_size; i++) {

letters[rest\_letter[i]] = true;

Permutation(learned\_letter + 1, i + 1);

letters[rest\_letter[i]] = false;

}

}

int main() {

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

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

// get input

cin >> N >> K;

if (K <= 4) {

cout << 0;

return 0;

}

table.assign(N,"");

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

//algorithm part

letters['a' - 'a'] = true; letters['n' - 'a'] = true; letters['t' - 'a'] = true; letters['c' - 'a'] = true; letters['i' - 'a'] = true;

for (int i = 0; i < 26; i++) if (letters[i] == false) rest\_letter.emplace\_back(i);

rest\_size = rest\_letter.size();

Permutation(5,0);

cout << answer;

return 0;

}

**[187. [SAMSUNG SW – KING]**

- it was an easy simulation problem. I had to just follow given moving order.

- it has an restriction to move king that after moving king, if there is stone, stone also have to be moved. but this restriction was so easy to implement.

- see the code.

#include<iostream>

#include<array>

#include<vector>

#include<map>

using namespace std;

array<array<int, 8>, 8> board;

int main() {

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

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

//get input

string stone; string king; int N;

cin >> king >> stone >> N;

vector<string> orders(N,"");

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

int sy =stone[0] - 'A'; int sx = 8 - (stone[1] - '0');

int ky = king[0] - 'A'; int kx = 8 - (king[1] - '0' );

//algorithm part

map<string, pair<int, int>> direction;

direction["R"] = make\_pair(0, 1); direction["L"] = make\_pair(0, -1); direction["B"] = make\_pair(1, 0); direction["T"] = make\_pair(-1, 0);

direction["RT"] = make\_pair(-1, 1); direction["LT"] = make\_pair(-1, -1); direction["RB"] = make\_pair(1, 1); direction["LB"] = make\_pair(1, -1);

for (string s : orders) {

int dx = direction[s].first; int dy = direction[s].second;

if (sx == dx + kx && sy == dy + ky) {

if (sx + dx < 0 || sy + dy < 0 || sx + dx >= 8 || sy + dy >= 8 || kx + dx < 0 || ky + dy < 0 || kx + dx >= 8 || ky + dy >= 8) continue;

else kx = dx + kx; ky = dy + ky; sx = dx + sx; sy = dy + sy;

}

else if (kx + dx >= 0 && ky + dy >= 0 && kx + dx < 8 && ky + dy < 8) {

kx = dx + kx; ky = dy + ky;

}

}

cout << char('A' + ky) << 8 - kx << endl;

cout << char('A' + sy) << 8 - sx << endl;

return 0;

}

**[188. [SAMSUNG SW – Z]**

- it was a binary search problem. Actually, it was a quaternary search.

- we can divide whole square to 4 district, top\_left, top\_right, bottom\_left and bottom\_right.

- and each value follows below rule.

> top\_left = top\_left, top\_right = top\_left + 4^(n-1),

> bottom\_left = top\_left + 2 \* 4^(n-1), bottom\_right = top\_left + 3 \* 4^(n-1)

- Downsizing the square for each round, we re-determine which district has (r,c).

- if N==1 or top\_left +1 ==top\_right, then end of while.

- see the code.

#include<iostream>

#include<cmath>

using namespace std;

int N, r, c;

long long top\_left, top\_right, bottom\_left, bottom\_right;

void resizing() {

if (r < pow(2, N - 1) && c < pow(2, N - 1)) { //top\_left

}

else if (r < pow(2, N - 1) && c >= pow(2, N - 1)) {//top\_right

top\_left = top\_right;

c -= pow(2, N - 1);

}

else if (r >= pow(2, N - 1) && c < pow(2, N - 1)) {//bottom\_left

top\_left = bottom\_left;

r -= pow(2, N - 1);

}

else {//bottom\_right

top\_left = bottom\_right;

c -= pow(2, N - 1);

r -= pow(2, N - 1);

}

N--;

top\_right = top\_left + pow(4, N - 1);

bottom\_left= top\_left + 2\*pow(4, N - 1);

bottom\_right = top\_left + 3 \* pow(4, N - 1);

}

int main() {

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

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

//get input

cin >> N >> r >> c;

//algorithm part

// tr = 4^(n-1) + top\_left, bl = 2 \* 4^(n-1) + top\_left , br = 3 \* 4^(n-1) + top\_left ,

top\_left = 0, top\_right = pow(4,N-1), bottom\_left = 2 \* pow(4, N-1), bottom\_right = 3 \* pow(4, N-1); //default number

while (N>1) {

resizing();

}

if (r == 0 && c == 0) cout << top\_left;

else if (r == 0 && c == 1) cout << top\_right;

else if (r == 1 && c == 0) cout << bottom\_left;

else cout << bottom\_right;

return 0;

}

**[189. [SAMSUNG SW – Mafia]**

- I felt that I have to read Question more deeply. since I’ve missed a restriction that if rest number of people is even, mafia turn. Otherwise, citizen’s turn.

- because of that, I spent so much time.

- I felt one more thing that it’s more effience that just implement max score rather than using sort algorithm in the algorithm header.

- 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;

//bool end\_game = false;

int N;

int mafia\_pos;

vector<pair<int, bool>> people; //score and is\_dead?

vector<vector<int>> R;

int answer = 0;

void DFS(int rest, int night) {

//if (end\_game) return;

if (people[mafia\_pos].second == true || rest <= 1) {

answer = max(answer, night);

//if (rest == 1 && people[mafia\_pos].second == false) end\_game = true;

return;

}

if (rest % 2 == 0) { //mafia time

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

if (people[i].second == true || i == mafia\_pos) continue;

people[i].second = true;

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

if (people[j].second==true) continue;

people[j].first += R[i][j];

}

DFS(rest - 1, night + 1);

//if (end\_game) return;

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

if (people[j].second == true) continue;

people[j].first -= R[i][j];

}

people[i].second = false;

}

}

else { //citizen time

int max\_score = 0;

int min\_index = 987654321;

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

if (people[i].second == true) continue;

if (people[i].first > max\_score) {

max\_score = people[i].first;

min\_index = i;

}

else if (people[i].first == max\_score) {

min\_index = min(min\_index, i);

}

}

people[min\_index].second = true;

DFS(rest - 1, night);

//if (end\_game) return;

people[min\_index].second = false;

}

}

int main() {

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

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

//get input

cin >> N;

people.assign(N,pair<int,int>());

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

int temp\_input;

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

cin >> temp\_input;

people[i] = make\_pair(temp\_input, false);

}

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

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

cin >> mafia\_pos;

//algoirithm part

DFS(N,0);

cout << answer;

return 0;

}

**[190. [SAMSUNG SW – Game]**

- It was a typical simulation problem. but tricks were there.

- First trick was we had to use memorization to avoid time limit abort.

- Second trick was how to handle an infinite loop, I avoided infinite loop by recording all the path I passed earlier current shell. and then if next shell is in the record. It will be infinite.

- I implemeted this using DFS.

- see the code.

#include<iostream>

#include<vector>

#include<set>

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

using namespace std;

int N, M; //row\_size, col\_size

vector<vector<char>> table;

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

int answer = 0;

set<pair<int, int>> visit;

vector<vector<int>> shell\_visit;

bool flag = false;

int DFS(int x, int y,int cnt) {

if (shell\_visit[x][y] != -1) {

return cnt+shell\_visit[x][y];

}

int move\_num = table[x][y] - '0';

int max\_move = 1;

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

int nx = x + (dir[i][0] \* move\_num); int ny = y + (dir[i][1] \* move\_num);

if (visit.find(make\_pair(nx, ny)) != visit.end()) { // infinite loop

flag = true;

return 0;

}

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

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

max\_move = max(max\_move, DFS(nx,ny,cnt+1)-cnt);

}

visit.erase(visit.find(make\_pair(nx, ny)));

}

shell\_visit[x][y] = max\_move;

return cnt+max\_move;

}

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));

shell\_visit.assign(N, vector<int>(M, -1));

string input\_string;

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

cin >> input\_string;

for (int j = 0; j < M; j++) table[i][j] = input\_string[j];

}

//algorithm part

answer=DFS(0, 0, 0);

visit.insert(make\_pair(0, 0));

if (flag) cout << -1;

else cout << answer;

return 0;

}

**[191. [SAMSUNG SW – Pool]**

- it has been quite long time since I met this kind of a high level problem.

- it was a simulation problem. but it was hard to make it satisfy all the rule.

- I started sied of the pool from 1. if I met less value than current considering value, plus 1 to the shell using BFS.

- that’s meaning is that if we can reach from side with current value, we don’t need to care to fill water.

- so after that, with traveling all the shell, if we meet less value than current considering value, that is we can’t reach from side. so add 1 to answer value and make the shell plused 1.

- see the code.

#include<iostream>

#include<vector>

#include <queue>

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

using namespace std;

int N, M; //<= 50

vector<vector<int>> table;

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

int max\_wall = 0;

void BFS(int height) {

queue<pair<int, int>> que;

que.push(make\_pair(0, 0));

table[0][0] = height;

while (!que.empty()) {

int x = que.front().first; int y = que.front().second; 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 + 1 && ny <= M + 1 && table[nx][ny]<height) {

table[nx][ny] = height;

que.push(make\_pair(nx, ny));

}

}

}

}

int main() {

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

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

int answer = 0;

//get input

cin >> N >> M;

table.assign(N+2, vector<int>(M+2, 0));

string input;

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

cin >> input;

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

table[i][j+1] = input[j] - '0';

max\_wall = max(max\_wall, input[j]-'0');

}

}

//algorithm part;

for (int h = 1; h <= max\_wall; h++) {

BFS(h);

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

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

if (table[i][j] < h) {

answer++;

table[i][j]++;

}

}

}

}

cout << answer;

return 0;

}

**[192. [SAMSUNG SW – Moon is being filled up, Let’s go!]**

- it was a simulation problem with memorization. so I used BFS but there were lots of things to remember. I mean, to memorize in queue.

- so memory size exceeded.

- see the first code.

#include<iostream>

#include<vector>

#include<unordered\_map>

#include<map>

#include<queue>

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

using namespace std;

vector<vector<char>> table;

pair<int, int> start\_pos;

map<pair<int, int>,bool> end\_pos;

int N, M;

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

inline bool doWeHaveTheKey(char door, unordered\_map<char, bool> has\_key) {

switch (door){

case 'A':

if (has\_key.find('a') != has\_key.end()) return true;

return false;

case 'B':

if (has\_key.find('b') != has\_key.end()) return true;

return false;

case 'C':

if (has\_key.find('c') != has\_key.end()) return true;

return false;

case 'D':

if (has\_key.find('d') != has\_key.end()) return true;

return false;

case 'E':

if (has\_key.find('e') != has\_key.end()) return true;

return false;

case 'F':

if (has\_key.find('f') != has\_key.end()) return true;

return false;

default:

break;

}

}

int BFS(int x, int y) {

queue<pair<vector<vector<bool>>,pair<vector<int>,unordered\_map<char,bool>>>> que;

unordered\_map<char, bool> has\_key;

vector<vector<bool>> visit(N, vector<bool>(M, false)); //when we get a key, we have to re-allocate visit matrix.

visit[x][y] = true;

que.push(make\_pair(visit,make\_pair(vector<int>{x,y,0},has\_key)));

int sum = 0;

while (!que.empty()) {

x = que.front().second.first[0]; y = que.front().second.first[1]; sum = que.front().second.first[2];

has\_key = que.front().second.second; visit = que.front().first; que.pop();

if (end\_pos.find(make\_pair(x, y)) != end\_pos.end()) return sum;

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

int nx = x + dir[i][0]; int ny = y + dir[i][1]; unordered\_map<char, bool> temp\_has\_key = has\_key; vector<vector<bool>> temp\_visit = visit;

if (nx >= 0 && ny >= 0 && nx < N && ny < M && table[nx][ny] != '#' && temp\_visit[nx][ny] == false) {

if ('a' <= table[nx][ny] && table[nx][ny] <= 'f') { //when we meet a key

temp\_has\_key[table[nx][ny]] = true;

temp\_visit.assign(N, vector<bool>(M, false));

}

if ('A' <= table[nx][ny] && table[nx][ny] <= 'F' ) { //when we meet a door

if(doWeHaveTheKey(table[nx][ny],has\_key)) table[nx][ny] = '.';

else {

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

continue;

}

}

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

que.push(make\_pair(temp\_visit,make\_pair(vector<int>{nx,ny,sum+1}, temp\_has\_key)));

}

}

}

return -1;

}

int main() {

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

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

//get input

cin >> N >> M;

string input;

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

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

cin >> input;

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

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

if (input[j] == '0') start\_pos = make\_pair(i, j) ;

else if(input[j]=='1') end\_pos[make\_pair(i, j)] = true;

}

}

//algorithm part

int answer = 0;

answer=BFS(start\_pos.first, start\_pos.second);

cout << answer;

return 0;

}

- I had to downsize memory size.

- I resolve memory problem. but time limit problem occurred.

- see the second code.

#include<iostream>

#include<vector>

#include<unordered\_map>

#include<map>

#include<queue>

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

using namespace std;

vector<vector<char>> table;

pair<int, int> start\_pos;

map<pair<int, int>,bool> end\_pos;

vector<vector<bool>> visit;

queue<pair<unordered\_map<char, bool>,vector<int>>> stop\_pos;

int N, M;

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

int answer = 987654321;

inline bool doWeHaveTheKey(char door, unordered\_map<char, bool> has\_key) {

switch (door){

case 'A':

if (has\_key.find('a') != has\_key.end()) return true;

return false;

case 'B':

if (has\_key.find('b') != has\_key.end()) return true;

return false;

case 'C':

if (has\_key.find('c') != has\_key.end()) return true;

return false;

case 'D':

if (has\_key.find('d') != has\_key.end()) return true;

return false;

case 'E':

if (has\_key.find('e') != has\_key.end()) return true;

return false;

case 'F':

if (has\_key.find('f') != has\_key.end()) return true;

return false;

default:

break;

}

}

void DFS(int x, int y, int sum, unordered\_map<char, bool> has\_key) {

if (end\_pos.find(make\_pair(x, y)) != end\_pos.end()) {

answer = min(answer, sum);

return;

}

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 && table[nx][ny] != '#' && visit[nx][ny] == false) {

if ('a' <= table[nx][ny] && table[nx][ny] <= 'f') {

unordered\_map<char, bool> temp\_has\_key = has\_key;

temp\_has\_key[table[nx][ny]] = true;

stop\_pos.push(make\_pair(temp\_has\_key,vector<int>{nx, ny, sum + 1}));

table[nx][ny] = '.';

}

if ('A' <= table[nx][ny] && table[nx][ny] <= 'F') {

if (doWeHaveTheKey(table[nx][ny], has\_key)) table[nx][ny] = '.';

else {

visit[nx][ny] = true;

continue;

}

}

visit[nx][ny] = true;

DFS(nx, ny, sum + 1,has\_key);

visit[nx][ny] = false;

}

}

return;

}

int main() {

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

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

//get input

cin >> N >> M;

string input;

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

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

cin >> input;

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

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

if (input[j] == '0') start\_pos = make\_pair(i, j) ;

else if(input[j]=='1') end\_pos[make\_pair(i, j)] = true;

}

}

//algorithm part

unordered\_map<char, bool> has\_key;

stop\_pos.push(make\_pair(has\_key,vector<int>{start\_pos.first, start\_pos.second, 0}));

while (!stop\_pos.empty()) {

int x = stop\_pos.front().second[0]; int y = stop\_pos.front().second[1]; int sum = stop\_pos.front().second[2];

has\_key = stop\_pos.front().first; stop\_pos.pop();

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

visit[x][y] = true;

DFS(x,y,sum, has\_key);

}

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

else cout << answer;

return 0;

}

- Now, I had to reduce time complexity.

- I solved it using bit mask and 3D visit array. In terms of visit array, if we have different key at current shell, its status is different. so we have to check which key we have.

- it was so hard…

- see the code.

#include<iostream>

#include<vector>

#include<unordered\_map>

#include<queue>

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

using namespace std;

vector<vector<char>> table;

pair<int, int> start\_pos;

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

int N, M;

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

inline bool doWeHaveTheKey(char door, unordered\_map<char, bool> has\_key) {

switch (door) {

case 'A':

if (has\_key.find('a') != has\_key.end()) return true;

return false;

case 'B':

if (has\_key.find('b') != has\_key.end()) return true;

return false;

case 'C':

if (has\_key.find('c') != has\_key.end()) return true;

return false;

case 'D':

if (has\_key.find('d') != has\_key.end()) return true;

return false;

case 'E':

if (has\_key.find('e') != has\_key.end()) return true;

return false;

case 'F':

if (has\_key.find('f') != has\_key.end()) return true;

return false;

default:

break;

}

}

int BFS(int x, int y) {

queue<pair<pair<int, int>, pair<int,int>>> que; //x,y, key,sum

que.push(make\_pair(make\_pair(x, y), make\_pair(0,0)));

visit[x][y][0] = true;

while (!que.empty()) {

x = que.front().first.first; y = que.front().first.second;

int key = que.front().second.first; int sum = que.front().second.second;

que.pop();

if (table[x][y] == '1') return sum;

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

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

int new\_key = key;

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

if (table[nx][ny] == '#') continue;

if ('a' <= table[nx][ny] && table[nx][ny] <= 'f') new\_key=(new\_key | 1 << table[nx][ny] - 'a');

if ('A' <= table[nx][ny] && table[nx][ny] <= 'F') {

if (!(key & (1 << (table[nx][ny] - 'A')))) continue;

}

if (visit[nx][ny][new\_key]) continue;

que.push(make\_pair(make\_pair(nx, ny), make\_pair(new\_key, sum + 1)));

visit[nx][ny][new\_key] = true;

}

}

}

return -1;

}

int main() {

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

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

//get input

cin >> N >> M;

string input;

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

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

cin >> input;

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

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

if (input[j] == '0') start\_pos = make\_pair(i, j);

}

}

//algorithm part

int answer = 0;

visit.assign(N, vector<vector<bool>>(M, vector<bool>(64, false)));

visit[start\_pos.first][start\_pos.second][0] = true;

answer = BFS(start\_pos.first, start\_pos.second);

cout << answer;

return 0;

}

**[193. [SAMSUNG SW – Knight Tour]**

- this problem was so easy for me. I just had to check the condition is possible or not.

- I record possible end position using first postion and for each iteration, I checked a current position can be reached from previous position. that’s it.

- once input ended. I checked all the shell is visited for sure.

- Lastly, if end position is possible position to jump to start position, return true. Otherwise, return false.

- see the code.

#include<iostream>

#include<vector>

#include<map>

using namespace std;

vector<vector<bool>> table(6, vector<bool>(6, false));

map<pair<int, int>, bool> end\_pos;

int dir[8][2] = { {-2,-1},{-1,-2},{-2,1},{-1,2},{2,-1},{1,-2},{2,1},{1,2} };

int main() {

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

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

//make end\_pos

string input;

cin >> input;

int col = input[0] - 'A'; int row = 6 - (input[1] - '0');

table[row][col] = true;

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

int nx = row + dir[i][0]; int ny = col + dir[i][1];

if (nx >= 0 && ny >= 0 && nx < 6 && ny < 6) {

end\_pos[make\_pair(nx, ny)] = true;

}

}

//get input and algorithm part

int prev\_row = row; int prev\_col = col;

for (int i = 0; i < 35; i++) {

cin >> input;

int col = input[0] - 'A'; int row = 6 - (input[1] - '0');

bool flag = false;

//to check whether the Knight can jump to current position from pre-position.

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

int nx = prev\_row + dir[j][0]; int ny = prev\_col + dir[j][1];

if (nx >= 0 && ny >= 0 && nx < 6 && ny < 6 && nx == row && ny == col) {

flag = true;

break;

}

}

if (!flag || table[row][col]) { //if the Knight can not be current position or already visit current position, it's not valid.

cout << "Invalid";

return 0;

}

else {

table[row][col] = true;

}

prev\_row = row; prev\_col = col;

}

// to check all the shell is visited.

for (int i = 0; i < 6; i++) {

for (int j = 0; j < 6; j++) {

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

cout << "Invalid";

return 0;

}

}

}

//to check whether the Knight can jump to start postion from last position

col = input[0] - 'A'; row = 6 - (input[1] - '0');

if (end\_pos.find(make\_pair(row, col)) != end\_pos.end()) {

cout << "Valid";

return 0;

}

else cout << "Invalid";

return 0;

}

**[194. [SAMSUNG SW – Making Maze]**

- this was a simulation problem with a different start position.

- if player move over the limit of array, he can’t make a maze.

- during finishing all the given move, if player have been in the array. it’s a valid maze.

- so I make 50 x 50 array, and give a different start position for each iteration.

- 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;

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

string input;

vector<vector<char>> table;

int main() {

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

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

//get input

cin >> N>>input;

bool flag = false;

int ml, mr, mt, mb;

for (int i = 0; i < 50; i++) {

if (flag) break;

for (int j = 0; j < 50; j++) {

if (flag) break;

flag = true;

table.assign(50, vector<char>(50, '#'));

// allocate start position

table[i][j] = '.';

int cur\_x = i; int cur\_y = j;

int cur\_dir = 0;

ml = j; mr = 0; mt = i; mb = 0;

//take move

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

char cur\_move = input[k];

switch (cur\_move){

case 'L':

cur\_dir--;

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

break;

case 'R':

cur\_dir = (cur\_dir+1) % 4;

break;

case 'F':

cur\_x = cur\_x + dir[cur\_dir][0]; cur\_y = cur\_y + dir[cur\_dir][1];

if (cur\_x < 0 || cur\_y < 0 || cur\_x >= 50 || cur\_y >= 50) {

flag = false;

break;

}

table[cur\_x][cur\_y] = '.';

break;

}

ml = min(ml, cur\_y); mr = max(mr, cur\_y);

mt = min(mt, cur\_x); mb = max(mb, cur\_x);

}

}

}

for (int i = mt; i <= mb; i++) {

for (int j = ml; j <= mr; j++) {

cout << table[i][j];

}

cout << endl;

}

return 0;

}

**[195. [SAMSUNG SW – Downhill]**

- it was an easy simulation problem. just using memozation is the key to satisfy time limit.

- see the code.

#include<iostream>

#include<vector>

using namespace std;

int N, M;

vector<vector<int>> table;

vector<vector<int>> visit;

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

int DFS(int x, int y){

//return part

if (visit[x][y] != -1) return visit[x][y];

if (x == N - 1 && y == M - 1) return 1;

//DFS part

int res = 0;

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 && table[x][y]>table[nx][ny]) {

res += DFS(nx, ny);

}

}

visit[x][y] = res;

return res;

}

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));

visit.assign(N, vector<int>(M, -1));

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

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

cin >> table[i][j];

}

}

//algorithm part

unsigned int answer = 0;

answer=DFS(0, 0);

cout << answer;

return 0;

}

**[196. [SAMSUNG SW – A Monkey Wants To Be A Horse]**

**-** I tried to solve this problem using DFS and DP, but I failed. since there was K value that is the number of the monkey can move like the horse.

- see the first code.

#include<iostream>

#include<vector>

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

using namespace std;

vector<vector<int>> table;

vector<vector<vector<int>>>visit;

vector<vector<bool>> shell\_visit;

int K, N, M;

int k\_dir[8][2] = { {-1,-2},{-2,-1}, {-2,1},{-1,2},{1,2},{2,1},{2,-1},{1,-2} }; //left top start and clock-wise move.

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

int DFS(int x,int y,int cnt) {

if (visit[x][y][K] != -1) return visit[x][y][K];

if (x == N - 1 && y == M - 1) return 0;

if (x == 0 && y == 1) {

cout << "K : "<< K << endl;

}

int nx, ny;

int res = 987654321;

//Knight Move

if (K > 0) {

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

nx = x + k\_dir[i][0]; ny = y + k\_dir[i][1];

if (nx >= 0 && ny >= 0 && nx < N && ny < M && table[nx][ny] != 1 && shell\_visit[nx][ny]==false) {

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

K -= 1;

res= min(res,DFS(nx, ny,cnt+1)+1);

K += 1;

shell\_visit[nx][ny] = false;

}

}

}

//Normal Move

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

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

if (nx >= 0 && ny >= 0 && nx < N && ny < M && table[nx][ny] != 1 && shell\_visit[nx][ny] == false) {

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

res = min(res, DFS(nx, ny, cnt + 1)+1);

shell\_visit[nx][ny] = false;

}

}

if (K > 0) visit[x][y][1] = res;

else visit[x][y][0] = res;

return res;

}

int main(){

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

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

//get input

cin >> K >> M >> N;

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

shell\_visit.assign(N, vector<bool>(M, false));

visit.assign(N, vector<vector<int>>(M,vector<int>(K+1,-1)));

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

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

cin >> table[i][j];

}

}

//algorithm part

shell\_visit[0][0] = true;

int answer = 987654321;

answer=DFS(0,0,0);

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

else cout << answer;

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

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

cout << endl << "i : " << i << " j :" << j << " ";

cout << visit[i][j][0] << " , " << visit[i][j][1] << endl;

}

}\*/

return 0;

}

- so I changed it to BFS.

- BFS also has Memozation. I mean Dp. the reason why we can’t use DP in DFS but BFS is that when we use DP in DFS, a shell can be reached along different ways. so even if the shell’s k value is same, there is a probablity that it has a different res value.

- Therefore, even if we visit a shell with a certain K value. we have to re-check the shell. so the visit array is useless.

- However, BFS always find a shortest path from current shell. so there is no possible way to reach a certain shell with different path. so we can use DP.

- see the code.

#include<iostream>

#include<vector>

#include<queue>

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

using namespace std;

vector<vector<int>> table;

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

vector<vector<bool>> shell\_visit;

int K, N, M;

int k\_dir[8][2] = { {-1,-2},{-2,-1}, {-2,1},{-1,2},{1,2},{2,1},{2,-1},{1,-2} }; //left top start and clock-wise move.

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

int BFS(int x,int y, int k) {

queue<vector<int>> que;

que.push(vector<int>{x, y, 0,0});

visit[x][y][0] = true;

int res = 987654321;

while (!que.empty()) {

x = que.front()[0]; y = que.front()[1]; k = que.front()[2]; int sum = que.front()[3];

que.pop();

if (x == N - 1 && y == M - 1) {

return sum;

}

if (K>k) {

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

int nx = x + k\_dir[i][0]; int ny = y + k\_dir[i][1];

if (nx >= 0 && ny >= 0 && nx < N && ny < M && table[nx][ny] != 1 && visit[nx][ny][k+1] == false) {

visit[nx][ny][k + 1] = true;

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

}

}

}

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 && table[nx][ny] != 1 && visit[nx][ny][k] == false) {

visit[nx][ny][k] = true;

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

}

}

}

if (res == 987654321) return -1;

return res;

}

int main(){

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

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

//get input

cin >> K >> M >> N;

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

visit.assign(N, vector<vector<bool>>(M, vector<bool>(K + 1, false)));

shell\_visit.assign(N, vector<bool>(M, false));

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

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

cin >> table[i][j];

}

}

//algorithm part

shell\_visit[0][0] = true;

int answer = 987654321;

answer=BFS(0,0,K);

cout << answer;

return 0;

}

**[197. [SAMSUNG SW – Robot]**

- It was a simulation problem with some restriction.

- we have to maintain 3D boolean visit array to check whether we visit the shell with a certain direction.

- by maintaining visit array, we don’t need to turn the robot 180 degree at one time. it’s the key point.

- In this kind of simulation problem, memozation is always the hardest point to implement and to come up with.

- I used BFS using queue.

- see the code.

#include<iostream>

#include<vector>

#include<queue>

#include<cmath>

using namespace std;

int N, M;

vector<vector<int>> table;

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

vector<int> start\_pos(3);

vector<int> end\_pos(3);

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

vector<int> rotation = { -1,1 }; //left right back

vector<pair<int, int>> afterPos(int x, int y, int d) {

vector<pair<int, int>> res;

for (int i = 1; i <= 3; i++) {

int nx = x + (dir[d][0] \* i); int ny = y + (dir[d][1] \* i);

if (nx < 0 || ny < 0 || nx >= N || ny >= M || table[nx][ny] != 0 ) return res;

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

}

return res;

}

inline int dirChange(int d) {

if (d == 1) return 3;

else if (d == 2) return 1;

else if (d == 3) return 2;

else if (d == 4) return 0;

}

int BFS() {

queue<vector<int>> que;

que.push(vector<int>{start\_pos[0]-1, start\_pos[1]-1, start\_pos[2], 0});

visit[start\_pos[0] - 1][start\_pos[1] - 1][start\_pos[2]] = true;

int answer = 987654321;

while (!que.empty()) {

int x = que.front()[0]; int y = que.front()[1]; int d = que.front()[2]; int sum = que.front()[3];

//cout << " x : " << x << " y : " << y << " dir : " << d << " sum : " << sum << endl;

que.pop();

if (x == end\_pos[0] - 1 && y == end\_pos[1] - 1 && d == end\_pos[2]) {

answer = min(answer, sum);

continue;

}

for (int r : rotation) {

int nd = d + r;

if (nd < 0) nd = 3;

else if (nd == 4) nd = 0;

if (visit[x][y][nd] == false) {

visit[x][y][nd] = true;

que.push(vector<int>{x, y, nd, sum + abs(r)});

}

}

vector<pair<int, int>> temp = afterPos(x, y, d);

for (pair<int, int> res : temp) {

if (visit[res.first][res.second][d] == false) {

visit[res.first][res.second][d] = true;

que.push(vector<int>{res.first, res.second, d, sum + 1});

}

}

}

return answer;

}

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));

visit.assign(N, vector<vector<bool>>(M,vector<bool>(4,false)));

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

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

cin >> table[i][j];

}

}

cin >> start\_pos[0] >> start\_pos[1] >> start\_pos[2];

cin >> end\_pos[0] >> end\_pos[1] >> end\_pos[2];

//algorithm part

start\_pos[2] = dirChange(start\_pos[2]);

end\_pos[2] = dirChange(end\_pos[2]);

int answer=BFS();

cout << answer;

return 0;

}

**[198. [SAMSUNG SW – Jump]**

- Recently, I’ve been solving lots of simulation problem. so this problem was easy for me.

- I used DFS since I had to find all the path to end position and for that DFS using memozation is the easiest way, I think.

- a precaution is the answer can be 2^63 – 1. so I used unsigned long long data type.

- see the code.

#include<iostream>

#include<vector>

using namespace std;

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

vector<vector<int>> table;

vector<vector<unsigned long long>> memo;

vector<vector<bool>> visit;

int N;

unsigned long long DFS(int x, int y) {

if (x == N - 1 && y == N - 1) return 1;

if (memo[x][y] != -1) return memo[x][y];

if (table[x][y] == 0) return 0;

unsigned long long res = 0;

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

int nx = x + (dir[i][0]\*table[x][y]); int ny = y + (dir[i][1]\*table[x][y]);

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

visit[nx][ny] = true;

res+=DFS(nx, ny);

visit[nx][ny] = false;

}

}

memo[x][y] = res;

return res;

}

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, false));

memo.assign(N, vector<unsigned long long>(N, -1));

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

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

cin >> table[i][j];

}

}

//algorithm part;

visit[0][0] = true;

unsigned long long answer = DFS(0,0);

cout << answer;

return 0;

}

**[199. [SAMSUNG SW – Moving Log]**

**-** this was a simulation problem to find the shortest path to move a log.

- the log’s size is 3, I mean horizontally. so there was a few restriction. but those was okay.

- I used BFS and visit array. but problem occurred when I got input.

- I misread input format. After changing the way to get input. it worked properly.

- see the code.

#include<iostream>

#include<vector>

#include<queue>

using namespace std;

vector<vector<char>> table;

int N;

vector<int> start\_pos(6, 0); //log's start position

vector<int> end\_pos(6, 0); //log's end position

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

bool horizon;

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

int BFS() {

queue<vector<int>> que;//log's position , move count, horizon

que.push(vector<int>{start\_pos[0], start\_pos[1], start\_pos[2], start\_pos[3], start\_pos[4], start\_pos[5], 0,horizon});

visit[start\_pos[2]][start\_pos[3]][horizon] = true;

while (!que.empty()) {

int sx = que.front()[0]; int sy = que.front()[1]; int mx = que.front()[2]; int my = que.front()[3]; int ex = que.front()[4]; int ey = que.front()[5];

int cnt = que.front()[6]; horizon = que.front()[7];

//cout << sx << "," << sy << "," << mx << "," << my << "," << ex << "," << ey << "=="<<cnt<<endl;

if (sx == end\_pos[0] && sy == end\_pos[1] && mx == end\_pos[2] && my == end\_pos[3] && ex == end\_pos[4] && ey == end\_pos[5]) return cnt;

que.pop();

//move

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

int nsx = sx + dir[i][0]; int nsy = sy + dir[i][1]; int nmx = mx + dir[i][0]; int nmy = my + dir[i][1]; int nex = ex + dir[i][0]; int ney = ey + dir[i][1];

if (nsx >= 0 && nsy >= 0 && nex < N && ney < N && table[nsx][nsy] != '1' && table[nmx][nmy] != '1' && table[nex][ney] != '1' && visit[nmx][nmy][horizon] == false) {

visit[nmx][nmy][horizon] = true;

que.push(vector<int>{nsx, nsy, nmx, nmy, nex, ney, cnt + 1, horizon});

}

}

//rotate

if (horizon) {

int nsx = mx - 1; int nex = mx + 1; int nsy = my; int ney = my;

if (nsx >= 0 && nex < N && table[sx - 1][sy] != '1' && table[mx - 1][my] != '1' && table[ex - 1][ey] != '1' &&

table[sx + 1][sy] != '1' && table[mx + 1][my] != '1' && table[ex + 1][ey] != '1' && visit[mx][my][false] == false) {

visit[mx][my][false] = true;

que.push(vector<int>{nsx, nsy, mx, my, nex, ney, cnt + 1, false});

}

}

else {

int nsx = mx; int nex = mx; int nsy = my - 1; int ney = my + 1;

if (nsy >= 0 && ney < N && table[sx][sy - 1] != '1' && table[mx][my - 1] != '1' && table[ex][ey - 1] != '1'

&& table[sx][sy + 1] != '1' && table[mx][my + 1] != '1' && table[ex][ey + 1] != '1' && visit[mx][my][true] == false) {

visit[mx][my][true] = true;

que.push(vector<int>{nsx, nsy, mx, my, nex, ney, cnt + 1, true});

}

}

}

return 0;

}

int main() {

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

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

//get input;

cin >> N;

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

visit.assign(N, vector<vector<bool>>(N, vector<bool>(2,false))); //horizon, vertical for log's middle position

bool flag\_start = true;

bool flag\_end = true;

string input;

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

cin >> input;

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

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

//start position

if (flag\_start && input[j] == 'B') {

flag\_start = false;

if (j + 1 < N && input[j+1] == 'B') {

horizon = true;

start\_pos[0] = start\_pos[2] = start\_pos[4] = i;

start\_pos[1] = j; start\_pos[3] = j + 1; start\_pos[5] = j + 2;

}

else {

start\_pos[0] = i; start\_pos[2] = i + 1; start\_pos[4] = i+2;

start\_pos[1] = start\_pos[3] = start\_pos[5] = j;

horizon = false;

}

}

//end position

if (flag\_end && input[j] == 'E') {

flag\_end = false;

if (j + 1 < N && input[j+1] == 'E') {

end\_pos[0] = end\_pos[2] = end\_pos[4] = i;

end\_pos[1] = j; end\_pos[3] = j + 1; end\_pos[5] = j + 2;

}

else {

end\_pos[0] = i; end\_pos[2] = i + 1; end\_pos[4] = i + 2;

end\_pos[1] = end\_pos[3] = end\_pos[5] = j;

}

}

}

}

//algorithm part

int answer=BFS();

cout << answer;

return 0;

}

**[200. [SAMSUNG SW – Famous 7 Princesses]**

- Wow, I have to say it was quite hard and difficult.

- I thought it was a simulation problem, so if I had used DFS or BFS, I could’ve solved this problem.

- but it was a combination problem. since although I used DFS or BFS, I wasn’t able to search like crossroad.

- Therefore, I had to choose 7 people first using combination. After that, I had to check the member included at least 4 “S” member and if Yes, I also had to check the people were connected each other.

- To determine they were connected, I used DFS.

- see the code.

#include<iostream>

#include<vector>

#include<set>

using namespace std;

vector<vector<char>> table(5, vector<char>(5, 0));

vector<vector<bool>> visit;

vector<vector<int>> temp\_table;

vector<pair<int, int>> s\_pos;

vector<pair<int, int>> cur\_com;

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

int res = 0;

int conneceted\_size;

void DFS(int x, int y) {

conneceted\_size++;

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 < 5 && ny < 5 && temp\_table[nx][ny] == 1 && visit[nx][ny] == false) {

visit[nx][ny] = true;

DFS(nx, ny);

}

}

}

bool isConnected() {

conneceted\_size = 0;

temp\_table.assign(5, vector<int>(5, 0));

visit.assign(5, vector<bool>(5, false));

for (int i = 0; i < 7; i++) temp\_table[cur\_com[i].first][cur\_com[i].second] = 1;

visit[cur\_com[0].first][cur\_com[0].second] = true;

DFS(cur\_com[0].first, cur\_com[0].second);

if (conneceted\_size == 7) return true;

return false;

}

void Combination(int start\_x, int start\_y, int cnt, int member) {

if (cnt == 7 ) {

if (member >= 4) {

if (isConnected()) res += 1;

return;

}

return;

}

for (int i = start\_x; i < 5; i++) {

for (int j = start\_y; j < 5; j++) {

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

bool S\_member=false;

if (table[i][j] == 'S') S\_member = true;

if (j == 4) {

Combination(i + 1, 0, cnt + 1, member + S\_member);

start\_y = 0;

}

else Combination(i, j + 1, cnt + 1, member + S\_member);

cur\_com.pop\_back();

}

}

}

int main() {

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

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

//get input

string input;

for (int i = 0; i < 5; i++) {

cin >> input;

for (int j = 0; j < 5; j++) {

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

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

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

}

}

}

//algorithm part

Combination(0, 0, 0, 0);

cout << res;

return 0;

}

**[201. [SAMSUNG SW – Chess]**

- this was an easy simulation problem.

- I just had to implement the way to move of queen and knight.

- see the code.

#include<iostream>

#include<vector>

using namespace std;

int N, M,q,k,p;

vector<vector<int>> table;

vector<pair<int,int>> queens;

vector<pair<int,int>> knights;

int k\_dir[8][2] = { {-1,-2},{-2,-1},{-2,1},{-1,2},{1,2},{2,1},{2,-1},{1,-2} };

int q\_dir[8][2] = { {-1,0}, {-1,1},{0,1},{1,1},{1,0},{1,-1},{0,-1},{-1,-1} };

int main() {

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

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

//get input

int x, y;

cin >> N >> M;

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

cin >> q;

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

cin >> x >> y;

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

queens.emplace\_back(make\_pair(x-1, y-1));

}

cin >> k;

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

cin >> x >> y;

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

knights.emplace\_back(make\_pair(x-1, y-1));

}

cin >> p;

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

cin >> x >> y;

table[x-1][y-1] = 1;

}

//algorithm part

//for queen

for (int i = 0; i < q; i++) {

for (int j = 0; j < 8; j++) {

int move = 1;

while (true) {

x = queens[i].first + q\_dir[j][0] \* move; y = queens[i].second + q\_dir[j][1] \* move;

if (x < 0 || y < 0 || x >= N || y >= M || table[x][y] == 1) break;

table[x][y] = 2;

move +=1;

}

}

}

//for knight

for (int i = 0; i < k; i++) {

for (int j = 0; j < 8; j++) {

x = knights[i].first + k\_dir[j][0]; y = knights[i].second + k\_dir[j][1];

if (x < 0 || y < 0 || x >= N || y >= M || table[x][y] == 1) continue;

table[x][y] = 2;

}

}

int answer = 0;

for (int i = 0; i < N; i++) {

for (int j = 0; j < M; j++) {

if (table[i][j] == 0) answer++;

}

}

cout << answer;

return 0;

}

**[202. [SAMSUNG SW – Making Bridge]**

- It was a minimum spanning tree problem.

- when I solved this problem. I thought time limit exceeded might have occurred. but it’s not.

- First, I named all the district respectively.

- Second, for each shell that is one of a district, find minimum path to another district using BFS.

- see the code.

#include<iostream>

#include<vector>

#include<queue>

#include<unordered\_set>

#define min(a,b) a>b?b:a

using namespace std;

vector<vector<int>> table;

int dir[4][2] = { {-1,0},{0,-1},{1,0},{0,1} };

int N;

int d\_num = 2;

int answer = 987654321;

void makeDistriction() {

queue<vector<int>> que;

for (int i = 0; i < N; i++) {

for (int j = 0; j < N; j++) {

if (table[i][j] == 1) {

que.push(vector<int>{i,j});

table[i][j] = d\_num;

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 (nx >= 0 && ny >= 0 && nx < N && ny < N && table[nx][ny] == 1) {

table[nx][ny] = d\_num;

que.push(vector<int>{nx, ny});

}

}

}

d\_num += 1;

}

}

}

}

void makeBridge() {

for (int i = 0; i < N; i++) {

for (int j = 0; j < N; j++) {

if (table[i][j] != 0) {

queue<vector<int>> que;

que.push(vector<int>{i, j,0});

int cur\_d = table[i][j];

vector<vector<bool>> visit(N, vector<bool>(N, false));

visit[i][j] = true;

while (!que.empty()) {

int x = que.front()[0]; int y = que.front()[1]; int dis = que.front()[2]; que.pop();

if (table[x][y] != 0 && table[x][y]!=cur\_d) {

answer = min(answer, dis-1);

break;

}

for (int d = 0; d < 4; d++) {

int nx = x + dir[d][0]; int ny = y + dir[d][1];

if (nx >= 0 && ny >= 0 && nx < N && ny < N && visit[nx][ny] == false && table[nx][ny]!=cur\_d) {

visit[nx][ny] = true;

que.push(vector<int>{nx, ny, dis + 1});

}

}

}

}

}

}

}

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

makeDistriction();

makeBridge();

cout << answer;

return 0;

}

**[203. [SAMSUNG SW – Break A Wall and Move]**

**-** It was a simulation problem to find the shortest path from start to end position.

- I used DFS first, but I failed since there is no insurance that a path I found is minimum path when we use DFS.

- so I changed it to BFS and solved it.

- see the code.

#include<iostream>

#include<vector>

#include<queue>

#define min(a,b) a>b?b:a

using namespace std;

int N, M;

vector<vector<int>> table;

vector<vector<vector<bool>>> visit;

int dir[4][2] = { {-1,0},{0,-1},{1,0},{0,1} };

bool flag = false;

int BFS() {

queue<vector<int>> que;

que.push(vector<int>{0, 0, 1,1});

visit[0][0][1] = true;

while (!que.empty()) {

int x = que.front()[0]; int y = que.front()[1]; int punch = que.front()[2]; int sum = que.front()[3];

if (x == N - 1 && y == M - 1) return sum;

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 < M) {

if (table[nx][ny] == 1) {

if (punch && visit[nx][ny][0]==false) {

visit[nx][ny][0] = true;

que.push(vector<int>{nx, ny, 0,sum+1});

}

else continue;

}

else if(visit[nx][ny][punch] == false) {

visit[nx][ny][punch] = true;

que.push(vector<int>{nx, ny, punch,sum+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<int>(M, 0));

visit.assign(N, vector<vector<bool>>(M, vector<bool>(2, false)));

string input;

for (int i = 0; i < N; i++) {

cin >> input;

for (int j = 0; j < M; j++) {

table[i][j] = input[j] - '0';

}

}

//algorithm part

int answer=BFS();

cout << answer;

return 0;

}

**[204. [SAMSUNG SW – Number Board Jump]**

- it was just a DFS problem. there was no restriction so easy.

- see the code.

#include<iostream>

#include<vector>

#include<set>

#include<string>

using namespace std;

vector<vector<int>> table(5,vector<int>(5,0));

set<string> number\_set;

int dir[4][2] = { {-1,0},{0,-1},{1,0},{0,1} };

void DFS(int x, int y, string cur\_set,int cnt) {

if (cnt == 6) {

number\_set.insert(cur\_set);

return;

}

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 < 5 && ny < 5) {

cur\_set += table[nx][ny] + '0';

DFS(nx, ny, cur\_set,cnt+1);

cur\_set.pop\_back();

}

}

}

int main() {

ios\_base::sync\_with\_stdio(false);

cin.tie(NULL); cout.tie(NULL);

//get input

for (int i = 0; i < 5; i++) {

for (int j = 0; j < 5; j++) {

cin >> table[i][j];

}

}

//algorithm part

for (int i = 0; i < 5; i++) {

for (int j = 0; j < 5; j++) {

string cur\_set = to\_string(table[i][j]) + '0';

DFS(i, j, cur\_set, 1);

cur\_set.pop\_back();

}

}

cout << number\_set.size();

return 0;

}

**[205. [SAMSUNG SW – Castle Wall]**

- this was so hard. they gave walls of the castle using numeric value, actually binary value!

- so reaching the given condition was so hard.

- I used BFS, and bit operation.

- see the code.

#include<iostream>

#include<vector>

#include<queue>

#define max(a,b) a>b?a:b

using namespace std;

vector<vector<int>> table;

int N, M;

int dir[4][2] = { {0,-1} ,{-1,0},{0,1},{1,0} };

vector<vector<bool>> visit;

int BFS(int x, int y) {

visit[x][y] = true;

queue<vector<int>> que;

que.push(vector<int>{x, y});

int room\_size = 1;

while (!que.empty()) {

x = que.front()[0]; y = que.front()[1];

que.pop();

int wall = 1;

for (int i = 0; i < 4; i++) {

int nx = x + dir[i][0]; int ny = y + dir[i][1];

if ((table[x][y] & wall)!=wall) {

if (nx >= 0 && ny >= 0 && nx < N && ny < M && visit[nx][ny] == false) {

visit[nx][ny] = true;

que.push(vector<int>{nx, ny});

room\_size += 1;

}

}

wall\*=2;

}

}

return room\_size;

}

int main() {

ios\_base::sync\_with\_stdio(false);

cin.tie(NULL); cout.tie(NULL);

//get input

cin >> M >> N;

table.assign(N, vector<int>(M, 0));

visit.assign(N, vector<bool>(M, false));

for (int i = 0; i < N; i++) {

for (int j = 0; j < M; j++) {

cin >> table[i][j];

}

}

//algorithm part

int num\_of\_rooms = 0;

int max\_size = 0;

for (int i = 0; i < N; i++) {

for (int j = 0; j < M; j++) {

if (visit[i][j] == false) {

int temp = BFS(i,j);

max\_size = max(max\_size, temp);

num\_of\_rooms += 1;

}

}

}

int max\_size2 = 0;

for (int i = 0; i < N; i++) {

for (int j = 0; j < M; j++) {

for (int k = 1; k <= 8; k \*= 2) {

if ((table[i][j]&k)==k) {

visit.assign(N, vector<bool>(M, false));

table[i][j] -= k;

int temp = BFS(i, j);

max\_size2 = max(max\_size2, temp);

table[i][j] += k;

}

}

}

}

cout << num\_of\_rooms << endl;

cout << max\_size << endl;

cout << max\_size2 << endl;

return 0;

}

[206. SAMSUNG SW : Water Bottle]

- it was a simulation problem with BFS.

- But at first, I had no idea how to solve this problem.

- the key was just to implement pouring water from one to another.

- I used BFS to record current bottles’ water volume and visit 3d array to check whether current bottles’ volume is already checked or not.

- I spent 26 minutes and 14 seconds.

- see the code.

#include<iostream>

#include<vector>

#include<set>

#include<queue>

using namespace std;

vector<int> bottle(3, 0);

vector<int> water(3, 0);

bool visit[200][200][200];

set<int> answer;

inline bool isValid(int a,int b,int c) {

if (visit[a][b][c]) return false;

return true;

}

inline bool isFull(int i,int c,int n) {

if ((bottle[i] - c) - n < 0) return true;

return false;

}

void BFS() {

queue<vector<int>> que;

que.push(vector<int>{0, 0, water[2]});

visit[0][0][water[2]] = true;

answer.insert(water[2]);

while (!que.empty()) {

int a = que.front()[0]; int b = que.front()[1]; int c = que.front()[2];

que.pop();

int na, nb, nc;

if (a == 0) answer.insert(c);

if (a != 0) {

if (isFull(1,b,a)) {

na = a - (bottle[1] - b);

nb = bottle[1];

}

else {

na = 0;

nb = a + b;

}

if (visit[na][nb][c] == false) {

visit[na][nb][c] = true;

que.push(vector<int>{na, nb, c});

}

if (isFull(2, c, a)) {

na = a - (bottle[2] - c);

nc = bottle[2];

}

else{

na = 0;

nc = a + c;

}

if (visit[na][b][nc] == false) {

visit[na][b][nc] = true;

que.push(vector<int>{na, b, nc});

}

}

if (b != 0) {

if (isFull(0, a, b)) {

nb = b - (bottle[0] - a);

na = bottle[0];

}

else {

nb = 0;

na = a + b;

}

if (visit[na][nb][c] == false) {

visit[na][nb][c] = true;

que.push(vector<int>{na, nb, c});

}

if (isFull(2, c, b)) {

nb = b - (bottle[2] - c);

nc = bottle[2];

}

else {

nb = 0;

nc = b + c;

}

if (visit[a][nb][nc] == false) {

visit[a][nb][nc] = true;

que.push(vector<int>{a, nb, nc});

}

}

if (c != 0) {

if (isFull(0, a, c)) {

nc = c - (bottle[0] - a);

na = bottle[0];

}

else {

nc = 0;

na = a + c;

}

if (visit[na][b][nc] == false) {

visit[na][b][nc] = true;

que.push(vector<int>{na, b, nc});

}

if (isFull(1, b, c)) {

nc = c - (bottle[1] - b);

nb = bottle[1];

}

else {

nc = 0;

nb = b + c;

}

if (visit[a][nb][nc] == false) {

visit[a][nb][nc] = true;

que.push(vector<int>{a, nb, nc});

}

}

}

}

int main() {

ios\_base::sync\_with\_stdio(false);

cin.tie(NULL); cout.tie(NULL);

//get input;

cin >> bottle[0] >> bottle[1] >> bottle[2];

water[2] = bottle[2];

//algorithm part

BFS();

for (set<int>::iterator iter = answer.begin(); iter != answer.end(); iter++)

cout << \*iter << " ";

return 0;

}

[207. SAMSUNG SW : Seven Dwarfs]

- it was a combination problem. I used DFS to implenmt combination.

- this problem was so easy, I just had to check whether sum of chosen dwarfs height is 100 or not after choosing 7 dwarfs.

- I spent 5 minutes.

- see the code.

#include<iostream>

#include<vector>

#include<algorithm>

using namespace std;

vector<int> table(9, 0);

vector<bool> visit(9, false);

vector<int> chosen;

bool flag = false;

void Combination(int cnt, int start) {

if (cnt == 7) {

int sum = 0;

for (int i : chosen) sum += i;

if (sum == 100) flag = true;

return;

}

for (int i = start; i < 9; i++) {

if (visit[i]) continue;

visit[i] = true;

chosen.emplace\_back(table[i]);

Combination(cnt + 1, i + 1);

if (flag) return;

chosen.pop\_back();

visit[i] = false;

}

}

int main() {

ios\_base::sync\_with\_stdio(false);

cin.tie(NULL); cout.tie(NULL);

//get input

for (int i = 0; i < 9; i++) cin >> table[i];

//algorithm part

Combination(0, 0);

sort(chosen.begin(), chosen.end());

for (int i : chosen) cout << i <<endl;

return 0;

}

[208. SAMSUNG SW : Safety Area]

- It was a simulation problem.

- I used BFS to determine how many safety areas are for each precipitation.

- I spent 11 minutes and 29 seconds.

- see the code.

#include<iostream>

#include<vector>

#include<queue>

#define max(a,b) a>b?a:b

using namespace std;

vector<vector<int>> table;

int dir[4][2] = { {-1,0},{0,-1},{1,0},{0,1} };

int max\_height=0;

int N;

int answer = 1;

void BFS(int height) {

vector<vector<bool>> visit(N, vector<bool>(N, false));

int res = 0;

for (int i = 0; i < N; i++) {

for (int j = 0; j < N; j++) {

if (table[i][j] > height && visit[i][j]==false) {

res += 1;

visit[i][j] = true;

queue<vector<int>> que;

que.push(vector<int>{i, j});

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 (nx >= 0 && ny >= 0 && nx < N && ny<N && visit[nx][ny] == false && table[nx][ny]>height) {

visit[nx][ny] = true;

que.push(vector<int>{nx, ny});

}

}

}

}

}

}

answer = max(answer, res);

}

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];

max\_height = max(max\_height, table[i][j]);

}

}

//algorithm part

for (int i = 1; i < max\_height; i++) BFS(i);

cout << answer;

return 0;

}

[209. SAMSUNG SW : Bus Transfer]

- It was a simulation problem. it was so hard for me.

- I used DFS to find path to destination. but memory limit exceeded.

- see the first code.

#include<iostream>

#include<vector>

#include<map>

#include<set>

#define min(a,b) a>b? b:a

using namespace std;

vector<bool> visit;

vector<vector<int>> shell\_visit;

int sx, sy, dx, dy;

int N, M, K;

map<pair<int, int>, set<int>> table;

vector<vector<short>> bus; //number, x1, y1, x2, y2 ,dir , if dir==1 , then vertical

int answer;

inline int Y(int y) {

return N - y;

}

int DFS(int x, int y) {

if (x == dx && y == dy) {

return 0;

}

if (shell\_visit[y][x] != -1) return shell\_visit[y][x];

int res = 987654321;

for (int b : table[make\_pair(y,x)]) {

if (visit[b]) continue;

visit[b] = true;

//cout << x << " , " << y <<" : "<<b<< endl;

int x1 = bus[b][1]; int y1 = bus[b][2]; int x2 = bus[b][3]; int y2 = bus[b][4]; int d = bus[b][5];

if (d) { //vertical

if (y1 > y2) {

for (int j = y2; j <= y1; j++) {

if(!table[make\_pair(j,x1)].empty()) res = min(res, DFS(x1, j) + 1);

}

}

else {

for (int j = y1; j <= y2; j++) {

if (!table[make\_pair(j, x1)].empty()) res = min(res, DFS(x1, j) + 1);

}

}

}

else { //horizon

if (x1 > x2) {

for (int j = x2; j <= x1; j++) {

if (!table[make\_pair(y1, j)].empty()) res=min(res,DFS(j, y1)+1);

}

}

else{

for (int j = x1; j <= x2; j++) {

if (!table[make\_pair(y1, j)].empty()) res = min(res, DFS(j, y1) + 1);

}

}

}

visit[b] = false;

}

shell\_visit[y][x] = res;

return res;

}

int main() {

ios\_base::sync\_with\_stdio(false);

cout.tie(NULL); cout.tie(NULL);

//get input

cin >> M >> N >> K;

shell\_visit.assign(N, vector<int>(M, -1));

bus.assign(K, vector<short>(6,0));

visit.assign(K, false);

int b;

int x1, x2, y1, y2;

for (int i = 0; i < K; i++) {

cin >> b;

for (int j = 1; j < 5; j++) {

cin >> bus[b-1][j];

//bus[b-1][j] -= 1;

}

bus[b - 1][1] -= 1; bus[b - 1][3] -= 1;

bus[b - 1][2] = Y(bus[b - 1][2]);

bus[b - 1][4] = Y(bus[b - 1][4]);

int temp;

x1 = bus[b - 1][1]; y1 = bus[b - 1][2]; x2 = bus[b - 1][3]; y2 = bus[b - 1][4];

if (x1==x2) { // vertical

bus[b - 1][5] = 1;

if (y1 > y2) {

for (int j = y2; j <= y1; j++) table[make\_pair(j,x1)].insert(b-1);

}

else {

for (int j = y1; j <= y2; j++) table[make\_pair(j, x1)].insert(b - 1);

}

}

else { //horizon

bus[b - 1][5] = 0;

if (x1 > x2) {

for (int j = x2; j <= x1; j++) table[make\_pair(y1, j)].insert(b - 1);

}

else {

for (int j = x1; j <= x2; j++) table[make\_pair(y1, j)].insert(b - 1);

}

}

}

cin >> sx >> sy >> dx >> dy;

sx--; dx--;

sy = Y(sy); dy = Y(dy);

/\*cout << endl;

for (int i = 0; i < K; i++) {

for (int j = 0; j < bus[i].size();j++) cout << bus[i][j] << " ";

cout << endl;

}

cout << endl;\*/

//algorithm part

answer=DFS(sx,sy);

cout << answer;

return 0;

}

- so I searched and find a way.

- the indice ranges are so wide. so I wasn’t able to use 2D array.

- Using BFS, finding a bus I can transfer. the below code is from google.

- I Spent 1 hour and 41 minutes and 38 seconds.

- see the code.

#include <iostream>

#include <queue>

using namespace std;

struct info {

int x1;

int y1;

int x2;

int y2;

bool is\_in(const int& x, const int& y)const {

return (x >= x1 && x <= x2 && y >= y1 && y <= y2);

}

bool is\_in(const info& a)const {

return (a.x1 <= x2 && a.x2 >= x1 && a.y1 <= y2 && a.y2 >= y1);

}

};

void swap(int& a, int& b) {

int tmp = a;

a = b;

b = tmp;

}

int visited[5001];

info bus[5001];

int M, N, K;

int sx, sy, dx, dy;

queue<int> q;

int bfs() {

for (int i = 0; i < K; i++) {

if (bus[i].is\_in(sx, sy)) {

q.push(i);

visited[i] = 1;

}

}

while (!q.empty()) {

int idx = q.front();

if (bus[idx].is\_in(dx, dy)) {

return visited[idx];

}

q.pop();

for (int i = 0; i < K; i++) {

if (visited[i]) continue;

if (bus[idx].is\_in(bus[i])) {

visited[i] = visited[idx] + 1;

q.push(i);

}

}

}

}

int main() {

ios::sync\_with\_stdio(false);

cin.tie(0);

cin >> M >> N >> K;

for (int i = 0; i < K; i++) {

int num, x1, y1, x2, y2;

cin >> num >> x1 >> y1 >> x2 >> y2;

if (y1 > y2) swap(y1, y2);

if (x1 > x2) swap(x1, x2);

bus[i] = { x1,y1,x2,y2 };

}

cin >> sx >> sy >> dx >> dy;

cout << bfs() << "\n";

return 0;

}

[210. SAMSUNG SW : IceBerg]

- it was a simulation problem.

- To melt iceberg. I had to determine how many sea shells are adjacent to current shell.

- To determine how many districts are, I used BFS with visit 2D array.

- I spent 20 minutes or so.

- see the code.

#include<iostream>

#include<vector>

#include<queue>

using namespace std;

int N, M;

vector<vector<int>> table;

vector<vector<bool>> visit;

int dir[4][2] = { {-1,0},{0,-1},{1,0},{0,1} };

void Melting() {

vector<vector<int>> stk;

for (int i = 0; i < N; i++) {

for (int j = 0; j < M; j++) {

if (table[i][j] != 0) {

int res = 0;

for (int d = 0; d < 4; d++) {

int nx = i + dir[d][0]; int ny = j + dir[d][1];

if (nx >= 0 && ny >= 0 && nx < N && ny < M && table[nx][ny] == 0) res++;

}

stk.emplace\_back(vector<int>{i, j, res});

}

}

}

for (vector<int> row : stk) {

int x = row[0]; int y = row[1]; int m = row[2];

table[x][y] -= m;

if (table[x][y] < 0) table[x][y] = 0;

}

}

void BFS(int x, int y) {

queue<vector<int>> que;

que.push(vector<int>{x, y});

visit[x][y] = true;

while (!que.empty()) {

x = que.front()[0]; 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 (nx >= 0 && ny >= 0 && nx < N && ny < M && table[nx][ny] != 0 && visit[nx][ny]==false) {

visit[nx][ny] = true;

que.push(vector<int>{nx, ny});

}

}

}

}

bool isEmpty() {

for (int i = 0; i < N; i++) {

for (int j = 0; j < M; j++) {

if (table[i][j] != 0) return false;

}

}

return 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 >> 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;

bool flag = false;

while (true) {

answer++;

int dis = 0;

visit.assign(N, vector<bool>(M, false));

Melting();

if (isEmpty()) {

flag = true;

break;

}

for (int i = 0; i < N; i++) {

for (int j = 0; j < M; j++) {

if (table[i][j] != 0 && visit[i][j] == false) {

dis++;

BFS(i, j);

}

}

}

if (dis >= 2) break;

}

if (flag) cout << 0;

else cout << answer;

return 0;

}

[211. SAMSUNG SW : Get District]

- It was a BFS problem.

- except for converting given corrodisantes, thers was nothing special.

- I spent 20 minutes and 57 seconds.

- see the code.

#include<iostream>

#include<vector>

#include<algorithm>

#include<queue>

using namespace std;

int N, M, K;

vector<vector<bool>> table;

vector<int> dis;

int answer = 0;

int dir[4][2] = { {-1,0},{0,-1},{1,0},{0,1} };

void DrawBox(int lx,int ly,int rx,int ry) {

for (int i = rx; i <= lx; i++) {

for (int j = ly; j <= ry; j++) {

table[i][j] = true;

}

}

}

void BFS(int x, int y) {

int res = 1;

queue<vector<int>> que;

que.push(vector<int>{x, y});

table[x][y] = true;

while (!que.empty()) {

x = que.front()[0]; 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 (nx >= 0 && ny >= 0 && nx < N && ny < M && table[nx][ny] == false) {

table[nx][ny] = true;

res +=1;

que.push(vector<int>{nx, ny});

}

}

}

dis.emplace\_back(res);

}

int main() {

ios\_base::sync\_with\_stdio(false);

cin.tie(NULL); cout.tie(NULL);

//get input

cin >> N >> M >> K;

table.assign(N, vector<bool>(M, false));

int lx, ly, rx, ry;

for (int i = 0; i < K; i++) {

cin >> ly >> lx >> ry >> rx;

lx = N - lx-1; rx = N - rx;

ry -= 1;

DrawBox(lx, ly, rx, ry);

}

//algorithm part

for (int i = 0; i < N; i++) {

for (int j = 0; j < M; j++) {

if (table[i][j] == false) {

answer += 1;

BFS(i,j);

}

}

}

sort(dis.begin(), dis.end());

cout << answer << endl;

for (int i : dis) cout << i << " ";

return 0;

}

[212. SAMSUNG SW : Tresure Island]

- It was a BFS problem.

- For all the land, I had to search maximum distance from a current shell using BFS.

- I spent 7 minutes and 21 seconds.

- see the code.

#include<iostream>

#include<vector>

#include<queue>

#define max(a,b) a>b?a:b

using namespace std;

vector<vector<char>> table;

int N, M;

int answer = 0;

int dir[4][2] = { {-1,0},{0,-1},{1,0},{0,1} };

int BFS(int x, int y) {

queue<vector<int>> que;

que.push(vector<int>{x, y, 0});

vector<vector<bool>> visit(N, vector<bool>(M, false));

visit[x][y] = true;

int res = 0;

while (!que.empty()) {

x = que.front()[0]; y = que.front()[1]; int cnt = que.front()[2];

que.pop();

res = max(res, cnt);

for (int d = 0; d < 4; d++) {

int nx = x + dir[d][0]; int ny = y + dir[d][1];

if (nx >= 0 && ny >= 0 && nx < N && ny < M && visit[nx][ny] == false && table[nx][ny] == 'L') {

visit[nx][ny] = true;

que.push(vector<int>{nx, ny, cnt + 1});

}

}

}

return res;

}

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];

}

}

//algorithm part

for (int i = 0; i < N; i++) {

for (int j = 0; j < M; j++) {

if (table[i][j] == 'L') {

answer=max(answer,BFS(i, j));

}

}

}

cout << answer;

return 0;

}

[213. SAMSUNG SW : Virus]

- It was a simulation problem.

- To record graph, I used unordered\_map and unorderd\_set.

- To travel connected nodes from current node, I used queue.

- I spent 8 minutes and 13 seconds.

- see the code.

#include<iostream>

#include<vector>

#include<unordered\_map>

#include<unordered\_set>

#include<queue>

using namespace std;

unordered\_map<int, unordered\_set<int>> table;

unordered\_map<int, bool> visit;

int N, K;

int main() {

ios\_base::sync\_with\_stdio(false);

cin.tie(NULL); cout.tie(NULL);

//get input

cin >> N >> K;

int a, b;

for (int i = 0; i < K; i++) {

cin >> a >> b;

table[a].insert(b);

table[b].insert(a);

}

//algorihtm part

visit[1] = true;

queue<int> que;

que.push(1);

int answer = 0;

while (!que.empty()) {

int cur\_com = que.front(); que.pop();

for (unordered\_set<int>::iterator iter = table[cur\_com].begin(); iter != table[cur\_com].end(); iter++) {

if (visit[\*iter] == false) {

visit[\*iter] = true;

answer += 1;

que.push(\*iter);

}

}

}

cout << answer;

return 0;

}

[214. SAMSUNG SW : Horizontal Vertical Puzzle]

- it was a permuation problem.

- At first, I didn’t understand what the problem requires. so I spent 5 minutes or so to understand it.

- I used DFS for permutation.

- To check whether the puzzle table is valid, for each word in the table, I compared it to the given words.

- if it exists, erase it, if not, return false and keep permutate.

- I spent 20 minutes and 49 seconds.

- see the code.

#include<iostream>

#include<vector>

#include<algorithm>

using namespace std;

vector<string> words(6,"");

vector<string> table(3, "");

vector<int> chosen;

vector<bool> visit(6, false);

bool flag = false;

bool isPuzzle() {

vector<string> chosen\_words;

for (int i = 0; i < 3; i++) {

table[i] = words[chosen[i]];

chosen\_words.emplace\_back(table[i]);

}

for (int j = 0; j < 3; j++) {

string temp = "";

for (int i = 0; i < 3; i++) {

temp += table[i][j];

}

chosen\_words.emplace\_back(temp);

}

vector<string> temp\_words = words;

for (string cur : chosen\_words) {

if (find(temp\_words.begin(), temp\_words.end(), cur) != temp\_words.end()) {

temp\_words.erase(find(temp\_words.begin(), temp\_words.end(), cur));

}

else return false;

}

return true;

}

void Combination(int cnt) {

if (cnt == 3) {

if (isPuzzle()) {

flag = true;

return;

}

return;

}

for (int i = 0; i < 6; i++) {

if (visit[i] == false) {

visit[i] = true;

chosen.emplace\_back(i);

Combination(cnt + 1);

if (flag) return;

chosen.pop\_back();

visit[i] = false;

}

}

}

int main() {

ios\_base::sync\_with\_stdio(false);

cin.tie(NULL); cout.tie(NULL);

//get input

for (int i = 0; i < 6; i++) cin >> words[i];

//algorithm part

Combination(0);

if (!flag) cout << 0;

else {

for (string cur : table) {

cout << cur << endl;

}

}

return 0;

}

[215. SAMSUNG SW : Postman Han Sang Duck]

- At first, I thought it was just a BFS problem.

- But it was a DP Problem. I encounterd a quite difficult one for a long time.

- First, sort the given heights.

- Second, Find a minimum maximum height that Han Sang Duck can delivery all the post.

- Third, Find a maximum minimum height that Han Sang Duck can delivery all the post.

- If we find a valid height, calculate their difference.

- Go back to Second until the right pointer of sorted list reaches the list’s end.

- I spent 1 hours and 22 minutes and 48 seconds.

- see the code.

#include<iostream>

#include<vector>

#include<set>

#define max(a,b) a>b? a:b

#define min(a,b) a>b? b:a

using namespace std;

int N, house;

int px, py;

vector<vector<int>> house\_pos;

vector<vector<char>> table;

vector<vector<int>> height;

vector<vector<bool>> visit;

int dir[8][2] = { {-1,0},{-1,-1},{0,-1},{1,-1},{1,0},{1,1},{0,1},{-1,1} }; // start from north, gose by antiClock-wise

set<int> fatigue;

set<int>::iterator s\_left;

set<int>::iterator s\_right;

void DFS(int x, int y,int min\_h, int max\_h) {

if (x < 0 || y < 0 || x >= N || y >= N || visit[x][y] || height[x][y]<min\_h || height[x][y]>max\_h) return;

visit[x][y] = true;

for (int d = 0; d < 8; d++) DFS(x+dir[d][0], y+dir[d][1], min\_h, max\_h);

}

bool isValid() {

for (vector<int> row : house\_pos) if (visit[row[0]][row[1]] == false) return false;

return true;

}

int main() {

ios\_base::sync\_with\_stdio(false);

cin.tie(NULL); cout.tie(NULL);

//get input

cin >> N;

table.assign(N, vector<char>(N, 0));

height.assign(N, vector<int>(N, 0));

string input;

for (int i = 0; i < N; i++) {

cin >> input;

for (int j = 0; j < N; j++) {

table[i][j]=input[j];

if (table[i][j] == 'K') {

house += 1;

house\_pos.emplace\_back(vector<int>{i, j});

}

if (table[i][j] == 'P') {

px = i; py = j;

}

}

}

for (int i = 0; i < N; i++) {

for (int j = 0; j < N; j++) {

cin >> height[i][j];

fatigue.insert(height[i][j]);

}

}

s\_left = fatigue.begin();

s\_right = fatigue.begin();

//algorithm part

int answer = 987654321;

while (s\_right!=fatigue.end()) {

while (s\_left!=fatigue.end()) {

visit.assign(N, vector<bool>(N, false));

DFS(px, py, \*s\_left, \*s\_right);

if (!isValid()) break;

answer = min(answer, \*s\_right - \*s\_left);

s\_left++;

}

s\_right++;

}

cout << answer;

return 0;

}

[216. SAMSUNG SW : Cheeze 1]

- It was a simulation problem.

- Before make cheeze melt, I should determine shells that are adjacent to air.

- To do that, I used BFS.

- After that, for each shell, I checked whether it is cheeze and the shell is adjacent to air.

- if so, I pushed its x,y corrdinates to a stk. Finally, remove the shells.

- During Above operation, I also counted how many cheeze shells left.

- I spent 14 minutes and 19 seconds.

- see the code.

#include<iostream>

#include<vector>

#include<queue>

using namespace std;

int N, M;

vector<vector<short>> table;

vector<vector<bool>> air;

int dir[4][2] = { {-1,0},{0,-1},{1,0},{0,1} };

int num\_of\_cheeze = 0;

void AirVisit() {

air.assign(N, vector<bool>(M, false));

queue<vector<int>> que;

que.push(vector<int>{0, 0});

air[0][0] = 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 (nx >= 0 && ny >= 0 && nx < N && ny < M && table[nx][ny] == 0 && air[nx][ny]==false) {

air[nx][ny] = true;

que.push(vector<int>{nx, ny});

}

}

}

}

void Melting() {

vector<pair<int, int>> stk;

num\_of\_cheeze = 0;

for (int i = 0; i < N; i++) {

for (int j = 0; j < M; j++) {

if (table[i][j] == 1) {

num\_of\_cheeze += 1;

for (int d = 0; d < 4; d++) {

int nx = i + dir[d][0]; int ny = j + dir[d][1];

if (nx >= 0 && ny >= 0 && nx < N && ny < M && air[nx][ny] == true) {

stk.emplace\_back(make\_pair(i, j));

break;

}

}

}

}

}

for (pair<int, int> row : stk) {

table[row.first][row.second] = 0;

}

}

void PrintTable() {

cout << endl;

for (vector<short> row : table) {

for (short i : row) cout << i << " ";

cout << endl;

}

cout << endl;

}

bool isEmpty() {

for (vector<short> row : table) {

for (short 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;

table.assign(N, vector<short>(M, 0));

for (int i = 0; i < N; i++) {

for (int j = 0; j < M; j++) {

cin >> table[i][j];

}

}

//algorithm part

int sec = 0;

while (true) {

AirVisit();

Melting();

sec += 1;

if (isEmpty()) break;

}

cout << sec << endl;

cout << num\_of\_cheeze << endl;

return 0;

}

[217. SAMSUNG SW : Cheeze 2]

- It was a simulation problem like a just previous problem.

- it required more conditions. Without those things, they were almost same.

- I spent 12 minutes and 33 seconds.

- see the code.

#include<iostream>

#include<vector>

#include<queue>

using namespace std;

int N, M;

vector<vector<short>> table;

vector<vector<bool>> air;

int dir[4][2] = { {-1,0},{0,-1},{1,0},{0,1} };

void AirVisit() {

air.assign(N, vector<bool>(M, false));

queue<vector<int>> que;

que.push(vector<int>{0, 0});

air[0][0] = 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 (nx >= 0 && ny >= 0 && nx < N && ny < M && table[nx][ny] == 0 && air[nx][ny] == false) {

air[nx][ny] = true;

que.push(vector<int>{nx, ny});

}

}

}

}

void Melting() {

vector<pair<int, int>> stk;

for (int i = 0; i < N; i++) {

for (int j = 0; j < M; j++) {

if (table[i][j] == 1) {

int num\_of\_air = 0;

for (int d = 0; d < 4; d++) {

int nx = i + dir[d][0]; int ny = j + dir[d][1];

if (nx >= 0 && ny >= 0 && nx < N && ny < M && table[nx][ny] == 0 && air[nx][ny] == true) num\_of\_air += 1;

}

if (num\_of\_air >= 2) stk.emplace\_back(make\_pair(i, j));

}

}

}

for (pair<int, int> row : stk) {

table[row.first][row.second] = 0;

}

}

bool isEmpty() {

for (vector<short> row : table) {

for (short c : row) if (c != 0) return false;

}

return true;

}

int main() {

ios\_base::sync\_with\_stdio(false);

cin.tie(NULL); cout.tie(NULL);

//get input

cin >> N >> M;

table.assign(N, vector<short>(M, false));

for (int i = 0; i < N; i++) {

for (int j = 0; j < M; j++) {

cin >> table[i][j];

}

}

//algorithm part

int sec = 0;

while (!isEmpty()) {

AirVisit();

Melting();

sec += 1;

}

cout << sec;

return 0;

}

[218. SAMSUNG SW : Monomino Domino]

- WOW. I’ve suddenly encountered a super strong enemy for a long time.

- It was a simulation problem.

- To tell you a trap in this problem, 2 x 1 or 1 x 2 “BLOCK”

- they move together, but one of a 1 x 1 block in 2 x 1 or 1 x 2 block can be removed respectively.

- In those case, I had to revise their friend block to themselves.

- I used two 10 x 4 table, so I had to re-allocate blue table’s corrodinates.

- When I remove a row, I had to revise their friend block as I mentioned above.

- I spent 3 hours ans 8 minutes and 13 seconds.

- see the code.

#include<iostream>

#include<vector>

#define EM 999 // empty

using namespace std;

int N;

vector<vector<vector<int>>> blue(10,vector<vector<int>>(4, vector<int>(2,EM)));

vector<vector<vector<int>>> green(10, vector<vector<int>>(4, vector<int>(2,EM)));

vector<vector<int>> order;

int answer = 0;

int num\_shell = 0;

void PutAndMove(int t, int x, int y) {

int i;

if (t == 1) {

//green

for (i = 2; i < 11; i++) if (i >= 10 || green[i][y][0]!=EM) break;

green[i - 1][y][0] = i - 1; green[i - 1][y][1] = y;

//blue

for (i = 2; i < 11; i++) if (i >= 10 || blue[i][x][0]!=EM) break;

blue[i-1][x][0] = i-1; blue[i - 1][x][1] = x;

}

else if (t == 2) {

//green

for (i = 2; i < 11; i++) if (i >= 10 || green[i][y][0] != EM || green[i][y + 1][0] != EM) break;

green[i-1][y][0] = i-1; green[i - 1][y][1] = y+1;

green[i - 1][y + 1][0] = i-1; green[i - 1][y + 1][1] = y;

//blue

for (i = 2; i < 11; i++) if (i+1 >= 10 ||(i+1<10 && blue[i+1][x][0] != EM)) break;

blue[i-1][x][0] = i; blue[i - 1][x][1] = x; blue[i][x][0] = i-1; blue[i][x][1]=x;

}

else if(t==3){

//green

for (i = 2; i < 11; i++) if (i+1 >= 10 ||(i+1<10 && green[i+1][y][0] != EM)) break;

green[i-1][y][0] = i; green[i - 1][y][1] = y;

green[i][y][0] = i-1; green[i][y][1] = y;

//blue

for (i = 2; i < 11; i++) if (i >= 10 || blue[i][x][0] != EM || blue[i][x+1][0] != EM) break;

blue[i-1][x][0] = i-1; blue[i - 1][x][1] =x+1;

blue[i-1][x+1][0] = i-1; blue[i - 1][x + 1][1] = x;

}

}

void RemoveRow() {

for (int i = 6; i < 10; i++) {

int g\_num = 0; int b\_num = 0;

for (int j = 0; j < 4; j++) {

if (green[i][j][0] != EM) g\_num += 1;

if (blue[i][j][0] != EM) b\_num += 1;

}

if (g\_num == 4) {

answer += 1;

for (int j = 0; j < 4; j++) { //remove friend

if (green[i][j][0] < 10 && green[i][j][1] < 4) {

green[green[i][j][0]][green[i][j][1]][0] = green[i][j][0];

green[green[i][j][0]][green[i][j][1]][1] = green[i][j][1];

}

green[i][j][0] = EM; green[i][j][1] == EM;

}

}

if (b\_num == 4) {

answer += 1;

for (int j = 0; j < 4; j++) { //remove friend

if (blue[i][j][0] < 10 && blue[i][j][1] < 4) {

blue[blue[i][j][0]][blue[i][j][1]][0] = blue[i][j][0];

blue[blue[i][j][0]][blue[i][j][1]][1] = blue[i][j][1];

}

blue[i][j][0] = EM; blue[i][j][1] = EM;

}

}

}

}

void MoveDown() {

for (int i = 8; i >= 4; i--) {

for (int j = 0; j < 4; j++) {

if (green[i][j][0]!=EM) {

int g\_empty = 0;

if (green[i][j][0] == i && green[i][j][1] == j) { //has no friend

for (int k = i + 1; k < 10; k++) {

if (green[k][j][0]==EM) g\_empty += 1;

else break;

}

green[i][j][0] = EM; green[i][j][1] = EM;

green[i + g\_empty][j][0] = i + g\_empty;

green[i + g\_empty][j][1] = j;

}

else { //has friend

int fx = green[i][j][0]; int fy = green[i][j][1];

if (i != fx) { //vertical friend

int nx = i > fx ? i : fx;

for (int k = nx + 1; k < 10; k++) {

if (green[k][j][0] == EM) g\_empty += 1;

else break;

}

if (g\_empty >= 1) {

green[i][j][0] = EM; green[i][j][1] = EM;

green[fx][fy][0] = EM; green[fx][fy][1] = EM;

green[nx + g\_empty][j][0] = nx + g\_empty - 1;

green[nx + g\_empty][j][1] = j;

green[nx + g\_empty - 1][j][0] = nx + g\_empty;

green[nx + g\_empty - 1][j][1] = j;

}

}

else { //horizon friend

for (int k = i + 1; k < 10; k++) {

if (green[k][j][0]!=EM || green[k][fy][0]!=EM) break;

g\_empty += 1;

}

if (g\_empty >= 1) {

green[i][j][0] = EM; green[i][j][1] = EM;

green[i][fy][0] = EM; green[i][fy][1] = EM;

green[i + g\_empty][j][0] = i + g\_empty;

green[i + g\_empty][j][1] = fy;

green[i + g\_empty][fy][0] = i + g\_empty;

green[i + g\_empty][fy][1] = j;

}

}

}

}

if (blue[i][j][0]!=EM) {

int b\_empty = 0;

if (blue[i][j][0] == i && blue[i][j][1] == j) { //has no friend

for (int k = i + 1; k < 10; k++) {

if (blue[k][j][0]==EM) b\_empty += 1;

else break;

}

blue[i][j][0] = EM; blue[i][j][1] = EM;

blue[i + b\_empty][j][0] = i + b\_empty;

blue[i + b\_empty][j][1] = j;

}

else { //has friend

int fx = blue[i][j][0]; int fy = blue[i][j][1];

if (i != fx) { //vertical friend

int nx = i > fx ? i : fx;

for (int k = nx + 1; k < 10; k++) {

if (blue[k][j][0]==EM) b\_empty += 1;

else break;

}

if (b\_empty >= 1) {

blue[i][j][0] = EM; blue[i][j][1] = EM;

blue[fx][fy][0] = EM; blue[fx][fy][1] = EM;

blue[nx + b\_empty][j][0] = nx + b\_empty - 1;

blue[nx + b\_empty][j][1] = j;

blue[nx + b\_empty - 1][j][0] = nx + b\_empty;

blue[nx + b\_empty - 1][j][1] = j;

}

}

else { //horizon friend

for (int k = i + 1; k < 10; k++) {

if (blue[k][j][0]!=EM || blue[k][fy][0]!=EM) break;

b\_empty += 1;

}

if (b\_empty >= 1) {

blue[i][j][0] = EM; blue[i][j][1] = EM;

blue[i][fy][0] = EM; blue[i][fy][0] = EM;

blue[i + b\_empty][j][0] = i + b\_empty;

blue[i + b\_empty][j][1] = fy;

blue[i + b\_empty][fy][0] = i + b\_empty;

blue[i + b\_empty][fy][1] = j;

}

}

}

}

}

}

}

void Border() {

int b\_num = 0; int g\_num = 0;

for (int i = 4; i <= 5; i++) {

for (int j = 0; j < 4; j++) {

if (green[i][j][0]!=EM) {

g\_num += 1;

break;

}

}

for (int j = 0; j < 4; j++) {

if (blue[i][j][0]!=EM) {

b\_num += 1;

break;

}

}

}

if (g\_num >= 1) {

for (int i = 9; i >= 10 - g\_num; i--) {

for (int j = 0; j < 4; j++) {

if (green[i][j][0] != EM) {

if (green[i][j][0] < 10 && green[i][j][1] < 4) {

green[green[i][j][0]][green[i][j][1]][0] = green[i][j][0];

green[green[i][j][0]][green[i][j][1]][1] = green[i][j][1];

}

}

green[i][j][0] = EM; green[i][j][1] = EM;

}

}

}

if (b\_num >= 1) {

for (int i = 9; i >= 10 - b\_num; i--) {

for (int j = 0; j < 4; j++) {

if (blue[i][j][0] != EM) {

if (blue[i][j][0] < 10 && blue[i][j][1] < 4) {

blue[blue[i][j][0]][blue[i][j][1]][0] = blue[i][j][0];

blue[blue[i][j][0]][blue[i][j][1]][1] = blue[i][j][1];

}

}

blue[i][j][0] = EM; blue[i][j][1] = EM;

}

}

}

}

void getScore() {

for (int i = 4; i < 10; i++) {

for (int j = 0; j < 4; j++) {

if (green[i][j][0]!=EM) num\_shell += 1;

if (blue[i][j][0]!=EM) num\_shell += 1;

}

}

}

void PrintTable() {

cout << "### GREEN ###" << endl;

for (vector<vector<int>> row : green) {

for (vector<int> i : row) {

if (i[0] == EM) cout << "0" << " ";

else cout << "1" << " ";

}

cout << endl;

}

cout << endl;

cout << "### BLUE ###" << endl;

for (vector<vector<int>> row : blue) {

for (vector<int> i : row) {

if (i[0] == EM) cout << "0" << " ";

else cout << "1" << " ";

}

cout << endl;

}

cout << endl;

}

int main() {

ios\_base::sync\_with\_stdio(false);

cin.tie(NULL); cout.tie(NULL);

//get input

cin >> N;

order.assign(N, vector<int>(3, 0));

for (int i = 0; i < N; i++) {

cin >> order[i][0] >> order[i][1] >> order[i][2];

}

//algorithm part

int sec = 0;

for (vector<int> row : order) {

int t = row[0]; int x = row[1]; int y = row[2];

PutAndMove(t, x, y);

RemoveRow();

MoveDown();

Border();

MoveDown();

RemoveRow();

MoveDown();

}

getScore();

cout << answer << endl;

cout << num\_shell << endl;

return 0;

}

[219. SAMSUNG SW : Juvenile Shark]

- It was a simulation problem.

- I thought it was easy, actually until now. but I’ve been trapped by something so I took so much time.

- the trap was to syncronize fishes' position with table.

- I had used fishes' position vector as global variable. So the program didn’t work well.

- To find this trap, I spent so much time :(

- I didn’t used BFS. Because of that, I think it devoured much time

- I spent 2 hours and half or so.

- see the code.

#include<iostream>

#include<vector>

#include<array>

#include<algorithm>

#define max(a,b) a>b?a:b

using namespace std;

array<array <int, 2>, 8> dir = { { {-1,0},{-1,-1},{0,-1},{1,-1},{1,0},{1,1},{0,1},{-1,1} } };

vector<bool> visit(16,false);

int answer = 0;

void PrintTable(vector<vector<int>>& table) {

cout << endl;

for (vector<int> row : table) {

for (int i : row) cout << i+1 << " ";

cout << endl;

}

cout << endl;

}

void FishSwap(vector<vector<int>>& table, vector<vector<int>>& fish, int a, int b) {

int ax = fish[a][2]; int ay = fish[a][3]; int bx = fish[b][2]; int by = fish[b][3];

fish[a][2] = bx; fish[a][3] = by; fish[b][2] = ax; fish[b][3] = ay;

table[ax][ay] = b; table[bx][by] = a;

}

void MoveFish(vector<vector<int>>& table,vector<vector<int>>& fish, int sx, int sy) {

for (int f = 0; f < 16; f++) {

if (visit[f] == true) continue;

int x = fish[f][2]; int y = fish[f][3]; int d = fish[f][1];

int rotate\_num = 0;

while (rotate\_num<8) {

int nx = x + dir[d][0]; int ny = y + dir[d][1];

if (nx < 0 || ny < 0 || nx >= 4 || ny >= 4 || (sx == nx && sy == ny)) {

d = (d + 1) % 8;

rotate\_num += 1;

continue;

}

else {

FishSwap(table,fish, f, table[nx][ny]);

fish[f][1] = d;

break;

}

}

}

}

void MoveShark(vector<vector<int>> table,vector<vector<int>> fish,int sx, int sy, int sd, int sum,int cnt ) {

MoveFish(table,fish,sx,sy);

int nx = sx + dir[sd][0]; int ny = sy + dir[sd][1];

while (nx >= 0 && ny >= 0 && nx < 4 && ny < 4 ) {

if (visit[table[nx][ny]] == false) {

visit[table[nx][ny]] = true;

MoveShark(table, fish, nx, ny, fish[table[nx][ny]][1], sum + table[nx][ny] + 1, cnt + 1);

visit[table[nx][ny]] = false;

}

nx = nx + dir[sd][0]; ny = ny + dir[sd][1];

}

answer = max(answer, sum);

}

int main() {

ios\_base::sync\_with\_stdio(false);

cin.tie(NULL); cout.tie(NULL);

//get input

vector<vector<int>> table(4, vector<int>(4, 0));

vector<vector<int>> fish(16, vector<int>(4, 0));

for (int i = 0; i < 4; i++) {

for (int j = 0; j < 4; j++) {

cin >> fish[4 \* i + j][0] >> fish[4 \* i + j][1];

fish[4 \* i + j][0] -= 1;

fish[4 \* i + j][1] -= 1; //dir index starts at 0.

fish[4 \* i + j][2] = i; fish[4 \* i + j][3] = j; //x y position

table[i][j] = fish[4 \* i + j][0];

}

}

visit[fish[0][0]] = true;

//algorithm part

int sx = 0; int sy = 0;

int sd; int sum = 0;

sd = fish[0][1]; sum = fish[0][0] + 1;

sort(fish.begin(), fish.end()); //sort by number

MoveShark(table,fish,sx,sy,sd,sum,0);

cout << answer;

return 0;

}

[220. Leetcode 31 : Next Permutation]

- it was a quite difficult problem. Since I had to satisfy memory restriction.

- I was able to use only a given vector with extra constant memory.

- Logic is below.

- First, Finding first element from last index in the given vector that is lower than its right elements.

- Second, Finding the smallest element just right after the first element that is greater than the first element.

- Third, Swap the smallest element with the first element.

- Fourth, sort the range, I means, from just right after first element to end of the vector.

- see the code.

#include<iostream>

#include<vector>

#include<algorithm>

#define max(a,b) a>b? a:b

using namespace std;

bool flag = false;

int main() {

vector<int> table = { 6,7,5,3,5,6,2,9,1,2,7,0,9 }; // 1234 1243 1324 1342 1423 1432

// we can guess that a next number is always greater than a current number.

int changed\_index = 0;

int right\_index = 0;

//To check it is last or not.

int i;

int after\_num = 987654321;

for ( i = 0; i < table.size()-1; i++) if (table[i] < table[i + 1]) break;

if (i == table.size()-1) reverse(table.begin(), table.end());

else {

// it must be changed from last index.

for (int i = table.size()-1; i >= 0; i--) {

for (int j = i + 1; j < table.size(); j++) {

if (table[i] < table[j]) {

changed\_index = i;

flag = true;

if (table[j] < after\_num) {

after\_num = table[j];

right\_index = j;

}

//4 1 3 2 // 3 4 1 2 // 2 4 3 1 <<= this one have to occur.

}

}

if (flag) break; // if something is changed by above if syntax, we don't need to go further.

}

swap(table[changed\_index], table[right\_index]);

sort(table.begin() + 1 + changed\_index , table.end());

}

for (int i : table) cout << i << " ";

return 0;

}

[221. SAMSUNG SW – Gas Pipe]

- It has been such a long time I’ve met a time limit exceed.

- It was a simulation problem. I’ve struggle to reach the time limit.

- Checking whether all the pipe is used consumed lots of computational time.

- So I chagend the logic just to find a possible condition to put a pipe of a certain shape.

- First, Finding the stolen shell.

- Second, Checking a possible pipe to put the shell.

- I spent 2 hours and 40 minutes and 20 seconds.

- see the code.

#include<iostream>

#include<vector>

using namespace std;

int N, M;

vector<vector<char>> table;

int dir[4][2] = { {-1,0},{0,-1},{1,0},{0,1} };

char answer;

int nx, ny,d;

void FindStartPosition(int& sx, int& sy) {

//find start direction

for (d = 0; d < 4; d++) {

nx = sx + dir[d][0]; ny = sy + dir[d][1];

if (nx >= 0 && ny >= 0 && nx < N && ny < M && table[nx][ny] != '.') {

if (nx == sx) { //horizontal

if (table[nx][ny] == '-' || table[nx][ny] == '+') break;

else if (ny < sy && (table[nx][ny] == '1' || table[nx][ny] == '2')) break;

else if (ny > sy && (table[nx][ny] == '2' || table[nx][ny] == '3')) break;

}

else { //vertical

if (table[nx][ny] == '|' || table[nx][ny] == '+') break;

else if (nx < sx && (table[nx][ny] == '1' || table[nx][ny] == '4')) break;

else if (nx > sx && (table[nx][ny] == '2' || table[nx][ny] == '3')) break;

}

}

}

sx = nx; sy = ny;

}

void Move() {

while (table[nx][ny] != 'Z' && table[nx][ny] != '.') {

switch (table[nx][ny]) {

case '-':

if (d == 1) ny -= 1;

else if (d == 3) ny += 1;

else return;

break;

case '|':

if (d == 0) nx -= 1;

else if (d == 2) nx += 1;

else return;

break;

case '+':

if (d == 0) nx -= 1;

else if (d == 2) nx += 1;

else if (d == 1) ny -= 1;

else if (d == 3) ny += 1;

else return;

break;

case '1':

if (d == 1) {

nx += 1;

d = 2;

}

else if (d == 0) {

ny += 1;

d = 3;

}

else return;

break;

case '2':

if (d == 2) {

ny += 1;

d = 3;

}

else if (d == 1) {

nx -= 1;

d = 0;

}

else return;

break;

case '3':

if (d == 3) {

nx -= 1;

d = 0;

}

else if (d == 2) {

ny -= 1;

d = 1;

}

else return;

break;

case '4':

if (d == 3) {

nx += 1;

d = 2;

}

else if (d == 0) {

ny -= 1;

d = 1;

}

else return;

break;

}

}

}

void FindPipe() {

vector<bool> around(4, false);

int x = nx; int y = ny;

for (int d = 0; d < 4; d++) {

nx = x + dir[d][0]; ny = y + dir[d][1];

if (nx >= 0 && ny >= 0 && nx < N && ny < M && table[nx][ny] != '.') {

if (d == 0 && (table[nx][ny] == '|' || table[nx][ny] == '+' || table[nx][ny] == '1' || table[nx][ny] == '4')) around[d] = true;

else if(d==2 && (table[nx][ny] == '|' || table[nx][ny] == '+' || table[nx][ny] == '2' || table[nx][ny] == '3')) around[d] = true;

else if (d == 1 && (table[nx][ny] == '-' || table[nx][ny] == '+' || table[nx][ny] == '1' || table[nx][ny] == '2')) around[d] = true;

else if (d == 3 && (table[nx][ny] == '-' || table[nx][ny] == '+' || table[nx][ny] == '3' || table[nx][ny] == '4')) around[d] = true;

}

}

if (around[0] && around[1] && around[2] && around[3]) answer = '+';

else if (around[0] && around[1]) answer = '3';

else if (around[0] && around[3]) answer = '2';

else if (around[2] && around[1]) answer = '4';

else if (around[2] && around[3]) answer = '1';

else if (around[0] && around[2]) answer = '|';

else if (around[1] && around[3]) answer = '-';

}

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));

int sx, sy, dx, dy;

string input;

for (int i = 0; i < N; i++) {

cin >> input;

for (int j = 0; j < M; j++) {

table[i][j] = input[j];

if (table[i][j] == 'M') {

sx = i; sy = j;

}

if (table[i][j] == 'Z') {

dx = i; dy = j;

}

}

}

//algorithm part

int csx, csy;

FindStartPosition(sx, sy);

Move();

csx = nx; csy = ny;

FindPipe();

cout << csx + 1 << " " << csy + 1 << " " << answer << endl;

return 0;

}

[222. SAMSUNG SW – Adult Shark]

- It was a same problem with what I’ve solved at SAMSUNG Codingtest 2020.

- it was a simulation problem with direction priorities.

- First, I made priority 3D array as a look up table.

- Second, Finding an adjacent empty shell of a shark.

- Third, If so, push the shark to temprorary vector. if not, finding an adjacent shell that has the shark’s pheromone.

- In terms of the temporary vector, if more than 2 shark have been located a certain shell, remove greater number of shark.

- Fourth, Reduce pheromone.

- Actually, I spent 1 hours and 6 minutes at the real test. Now, 49 minutes and 5 seconds.

- But I think the logic is slightly different with what I’ve done at the real test.

- so I’m quite nervous now.

- see the code.

#include<iostream>

#include<vector>

using namespace std;

int N, M, K;

vector<vector<pair<int,int>>> table; //shark\_number , remain time of pheromone

vector<vector<vector<int>>> prior; // shark\_number -> current\_direction -> next\_direction's priority

vector<vector<int>> shark; // x, y, direction

int dir[4][2] = { {-1,0},{1,0},{0,-1},{0,1} };

vector<bool> visit;

void SharkMove() {

vector<vector<int>> stk;

for (int i = 0; i < M;i++) {

if (visit[i] == true) continue;

int x = shark[i][0]; int y = shark[i][1]; int cur\_d = shark[i][2];

bool flag = false;

//finding empty shell

for (int d = 0; d < 4; d++) {

int nx = x + dir[prior[i][cur\_d][d]][0]; int ny = y + dir[prior[i][cur\_d][d]][1];

if (nx >= 0 && ny >= 0 && nx < N && ny < N && table[nx][ny].first == 0) { //is it empty shell?

stk.emplace\_back(vector<int>{nx, ny, i+1}); //x ,y, shark\_number

flag = true;

shark[i][0] = nx; shark[i][1] = ny; shark[i][2] = prior[i][cur\_d][d];

break;

}

}

//move to their own shell

if (flag) continue;

for (int d = 0; d < 4; d++) {

int nx = x + dir[prior[i][cur\_d][d]][0]; int ny = y + dir[prior[i][cur\_d][d]][1];

if (nx >= 0 && ny >= 0 && nx < N && ny < N && table[nx][ny].first == i+1) { // is it my shell?

shark[i][0] = nx; shark[i][1] = ny; shark[i][2] = prior[i][cur\_d][d];

table[nx][ny].second = K; //reset pheromone

break;

}

}

}

//determining empty shell

for (vector<int> row : stk) {

int x = row[0]; int y = row[1]; int number = row[2];

if (table[x][y].first == 0) {

table[x][y].first = number;

table[x][y].second = K; //set phoeromon

}

else {

if (table[x][y].first < number) {

visit[number - 1] = true;

continue;

}

visit[table[x][y].first - 1] = true;

table[x][y].first = number;

table[x][y].second = K; //set phoeromon

}

}

}

void Remove() {

for (int i = 0; i < N; i++) {

for (int j = 0; j < N; j++) {

if (table[i][j].first != 0 && table[i][j].second == 0) table[i][j].first = 0;

else if (table[i][j].first != 0) {

table[i][j].second -= 1;

}

}

}

}

bool isOnlyOneAlive() {

if (visit[0] == false) {

for (int i = 1; i < M; i++) {

if (visit[i] == false) return false;

}

return true;

}

return false;

}

void PrintTable() {

cout << endl;

for (int i = 0; i < N; i++) {

for (int j = 0; j < N; j++) {

cout << table[i][j].second << " ";

}

cout << endl;

}

cout << endl;

}

int main() {

ios\_base::sync\_with\_stdio(false);

cin.tie(NULL); cout.tie(NULL);

//get input

cin >> N >> M >> K;

table.assign(N, vector<pair<int,int>>(N,pair<int,int>(0,0)));

shark.assign(M, vector<int>(3,0));

visit.assign(M, false);

for (int i = 0; i < N; i++) {

for (int j = 0; j < N; j++) {

cin >> table[i][j].first;

if (table[i][j].first != 0) {

shark[table[i][j].first-1][0] = i;

shark[table[i][j].first-1][1] = j;

table[i][j].second = K; //Pheromone time.

}

}

}

for (int i = 0; i < M; i++) {

cin >> shark[i][2];

shark[i][2] -= 1;

}

prior.assign(M, vector<vector<int>>(4, vector<int>(4, 0)));

int input;

for (int i = 0; i < M; i++) {

for (int j = 0; j < 4; j++) {

for (int k = 0; k < 4; k++) {

cin >> input;

prior[i][j][k]=input-1; // dir starts at 0.

}

}

}

//algorithm part

int answer = 0;

while (!isOnlyOneAlive() && answer<=1000) {

Remove();

SharkMove();

answer += 1;

}

if (answer >= 1001) answer = -1;

cout << answer;

return 0;

}

[223. SAMSUNG SW – Start Taxi]

- What a dirty problem!!!

- Because of one edge condition, I spent so much time.

- The edge condition is that a shell can be both of starting point and destination.

- It was one of the problem that I’ve solved at SAMSUNG codingtest 2020.

- I thought I had solved well, but I changed my mind to that I might have been wrong after resolving this problem.

- First, Finding a closest passanger. For that, I record people whom the taxi can reach with spending same fuel.

- Second, Determining a passanger from the record people.

- Third, Bring the person to a destination.

- While moving taxi, if fuel becomes 0, stop the algorithm.

- I spent 2 hours and 41 minutes and 31 seconds.

- see the code.

#include<iostream>

#include<vector>

#include<queue>

#include<set>

#include<map>

using namespace std;

int N, M, K;

vector<vector<int>> table;

map<pair<int,int>,pair<int,int>> people; // sx,sy dx,dy

int dir[4][2] = { {-1,0},{0,-1},{1,0},{0,1} };

int tx, ty;

vector<vector<bool>> visit;

vector<vector<bool>> done\_person;

int fuel;

int finish\_num = 0;

bool stop = true;

void BFS() {

visit.assign(N, vector<bool>(N, false));

queue<vector<int>> que;

que.push(vector<int>{tx, ty, fuel}); //tx, ty, fuel

visit[tx][ty] = true;

set<vector<int>> stk;

stop = true;

int cur\_fuel;

//find a closest passanger

while (!que.empty()) {

cur\_fuel = que.front()[2];

while (!que.empty()) {

int x = que.front()[0]; int y = que.front()[1]; fuel = que.front()[2];

if (cur\_fuel != fuel) break;

que.pop();

if (done\_person[x][y]==false && people.find(make\_pair(x, y))!=people.end()) {

stk.insert(vector<int>{x, y});

stop = false;

}

if (!stop) continue;

for (int d = 0; d < 4; d++) {

int nx = x + dir[d][0]; int ny = y + dir[d][1];

if (nx < 0 || ny < 0 || nx >= N || ny >= N) continue;

if (table[nx][ny] ==0 && visit[nx][ny] == false) {

visit[nx][ny] = true;

que.push(vector<int>{nx, ny, fuel - 1});

}

}

}

if (!stop) break;

}

if (stop) return;

//determining passanger

vector<int> temp = \*stk.begin();

tx = temp[0]; ty = temp[1]; fuel = cur\_fuel;

int dx = people[make\_pair(tx, ty)].first; int dy = people[make\_pair(tx, ty)].second;

//bring him to destination.

stop = true;

que = queue<vector<int>>{};

visit.assign(N, vector<bool>(N, false));

visit[tx][ty] = true;

que.push(vector<int>{tx, ty, fuel, 0});

while (!que.empty()) {

int x = que.front()[0]; int y = que.front()[1]; fuel = que.front()[2]; int cnt = que.front()[3];

if (x == dx && y == dy) {

done\_person[tx][ty] = true;

tx = x; ty = y;

fuel += cnt \* 2;

finish\_num += 1;

stop = false;

return;

}

que.pop();

if (fuel == 0) continue;

for (int d = 0; d < 4; d++) {

int nx = x + dir[d][0]; int ny = y + dir[d][1];

if (nx < 0 || ny < 0 || nx >= N || ny >= N) continue;

if (table[nx][ny] ==0 && visit[nx][ny] == false) {

visit[nx][ny] = true;

que.push(vector<int>{nx, ny, fuel - 1, cnt + 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,0));

done\_person.assign(N, vector<bool>(N,false));

for (int i = 0; i < N; i++) {

for (int j = 0; j < N; j++) {

cin >> table[i][j];

}

}

cin >> tx >> ty;

tx -= 1; ty -= 1;

int a, b, c, d;

for (int i = 0; i < M; i++) {

cin >> a >> b >> c >> d;

a -= 1; b -= 1; c -= 1; d -= 1;

people[make\_pair(a, b)] = make\_pair(c, d);

}

fuel = K;

//algorithm part;

while (finish\_num <M) {

BFS();

if (stop) {

fuel = -1;

break;

}

//cout << fuel << endl;

}

cout << fuel;

return 0;

}

[224. SAMSUNG SW – Mineral]

- It was a simulation problem with having quite confusing elements.

- They Described the given cave R x C Array. but it was like a wall not ground.

- I mean, the given array was a standing state. oh.. how can I say? my English is So Sxxk..

- I hope you understand, Let’s take next step.

- First, Thorwing a stick from left to right or right to left in turn. if the stick meets a mineral, remove the mineral.

- Second, Re-clustring the minerals, during checking whether a cluster is on the ground or in the air.

- Third, if a cluster is in the air, we have to drop it down until it meets a different cluster or ground.

- I spent 1 hours and 13 minutes and 2 seconds.

- see the code.

#include<iostream>

#include<vector>

#include<queue>

#define EM 0

#define NEM 9

using namespace std;

int N, M,K;

vector<vector<int>> table;

vector<int> sticks;

int in\_the\_air = 0;

int num\_of\_sticks;

int dir[4][2] = { {-1,0},{0,-1},{1,0},{0,1} };

inline int reverseX(int x) { return N - x; }

void Throw(bool left,int height) {

if (left) {

for (int j = 0; j < M; j++) {

if (table[height][j] !=EM) {

table[height][j] = EM;

return;

}

}

}

else {

for (int j = M - 1; j >= 0; j--) {

if (table[height][j] != EM) {

table[height][j] = EM;

return;

}

}

}

}

void NumberingCluster() {

int cur\_number = 1;

in\_the\_air = 0;

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] != EM && visit[i][j]==false) {

bool ground = false;

queue<vector<int>> que;

que.push(vector<int>{i, j, cur\_number});

visit[i][j] = true;

table[i][j] = cur\_number;

while (!que.empty()) {

int x = que.front()[0]; int y = que.front()[1]; int cur = que.front()[2]; que.pop();

for (int d = 0; d < 4; d++) {

int nx = x + dir[d][0]; int ny = y + dir[d][1];

if ( nx >= N ) { ground = true; continue; }

if (nx < 0 || ny < 0 || ny >= M|| visit[nx][ny] || table[nx][ny] == EM) continue;

visit[nx][ny] = true;

table[nx][ny] = cur;

que.push(vector<int>{nx, ny, cur});

}

}

if (!ground) in\_the\_air = cur\_number;

cur\_number++;

}

}

}

}

void DropCluster() {

bool stop = false;

while (!stop) {

for (int i = N - 1; i >= 0; i--) {

for (int j = 0; j < M; j++) {

if ((table[i][j] == in\_the\_air) && (i + 1 >= N || (table[i + 1][j] != EM && table[i + 1][j] != in\_the\_air))) stop = true;

}

}

if (!stop) {

for (int i = N - 1; i >= 0; i--) {

for (int j = 0; j < M; j++) {

if (table[i][j] == in\_the\_air) {

table[i + 1][j] = table[i][j];

table[i][j] = EM;

}

}

}

}

}

}

void PrintTable() {

for (vector<int> row : table) {

for (int i : row) {

if (i != EM) cout << 'x';

else cout << '.';

}

cout << endl;

}

}

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));

string input;

for (int i = 0; i < N; i++) {

cin >> input;

for (int j = 0; j < M; j++) {

if (input[j] == 'x') table[i][j] = NEM;

else table[i][j] = EM;

}

}

cin >> K;

sticks.assign(K,0);

for (int i = 0; i < K; i++) cin >> sticks[i];

num\_of\_sticks = K;

//algorithm part

for (int i = 0; i < K; i += 2) {

Throw(true, reverseX(sticks[i])); //remove Mineral from left

NumberingCluster();

if (in\_the\_air != 0) DropCluster();

//PrintTable();

//cout << endl;

if (i + 1 < K) { //from right

Throw(false, reverseX(sticks[i + 1]));

NumberingCluster();

if (in\_the\_air != 0) DropCluster();

//PrintTable();

//cout << endl;

}

}

PrintTable();

return 0;

}

[225. SAMSUNG SW – Ant]

- It was an easy simulation problem.

- Just using double nested for syntax, we could solve it.

- First, Getting input of a first groub of ants and Rerversing it.

- Second, Appending a second groub of ants to the reversed first groub.

- Third, from index=0 to N1+N2-1, if we meet different direction when we compare [i] vs [i+1], swap the two ants only if [i]’s direction is right.

- I spent 19 minutes and 38 seconds.

- see the code.

#include<iostream>

#include<vector>

#include<algorithm>

using namespace std;

int N1, N2,T;

vector<pair<char, int>> ants;

int main() {

ios\_base::sync\_with\_stdio(false);

cin.tie(NULL); cout.tie(NULL);

//get input

cin >> N1 >> N2;

ants.assign(N1,pair<char,int>(0,3)); // 3 is right 1 is left

int i = 0;d

for (i = 0; i < N1; i++) cin >> ants[i].first;

reverse(ants.begin(), ants.end());

ants.resize(N1+N2);

int j = i;

for (int i = 0; i < N2; i++,j++) {

cin >> ants[j].first;

ants[j].second = 1;

}

cin >> T;

//algorithm part

for (int time = 0; time < T; time++) {

for (int i = 0; i < N1 + N2-1; i++) {

if (ants[i].second != ants[i + 1].second && ants[i].second==3) {

swap(ants[i], ants[i + 1]);

i += 1;

}

}

}

for (pair<char, int> row : ants) cout << row.first;

return 0;

}

[226. SAMSUNG SW – Escape]

- it was an easy simulation problem.

- First, Spreading water.

- Second, Pushing all the possible next position of the hedgehog to a queue from a current position by spending same time.

- Third, During Pushing next position, if the hedgehog meets the destination, print the spending time.

- Fourth, if queue becomes empty without reaching the destination, print ”KAKTUS”.

- I spent 16 minutes and 19 seconds.

- see the code.

#include<iostream>

#include<vector>

#include<queue>

using namespace std;

int N, M;

vector<vector<char>> table;

int sx, sy;

vector<vector<bool>> visit;

queue<vector<int>> que;

vector<vector<int>> water;

bool safe = false;

int dir[4][2] = { {-1,0},{0,-1},{1,0},{0,1} };

void Flood() {

int \_size = water.size();

for (int i = 0; i < \_size; i++) {

int x = water[i][0]; int y = water[i][1];

for (int d = 0; d < 4; d++) {

int nx = x + dir[d][0]; int ny = y + dir[d][1];

if (nx < 0 || ny < 0 || nx >= N || ny >= M || table[nx][ny]=='\*' || table[nx][ny]=='X' || table[nx][ny]=='D') continue;

table[nx][ny] = '\*';

water.emplace\_back(vector<int>{nx,ny});

}

}

}

void MoveHedgehog(){

int cur\_time = que.front()[2];

while (!que.empty()) {

int x = que.front()[0]; int y = que.front()[1]; int time = que.front()[2];

if (cur\_time != time) break;

if (table[x][y] == 'D') {

safe = true;

return;

}

que.pop();

for (int d = 0; d < 4; d++) {

int nx = x + dir[d][0]; int ny = y + dir[d][1];

if (nx < 0 || ny < 0 || nx >= N || ny >= M || table[nx][ny] == '\*' || table[nx][ny] == 'X' ||visit[nx][ny]==true) continue;

visit[nx][ny] = true;

que.push(vector<int>{nx, ny, time + 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, '.'));

visit.assign(N, vector<bool>(M, false));

for (int i = 0; i < N; i++) {

for (int j = 0; j < M; j++) {

cin >> table[i][j];

if (table[i][j] == 'S') {

sx = i; sy = j;

}

if (table[i][j] == '\*') water.emplace\_back(vector<int>{i, j});

}

}

//algorithm part

que.push(vector<int>{sx, sy, 0}); // x,y, minutes

visit[sx][sy] = true;

while (!que.empty() && !safe) {

Flood();

MoveHedgehog();

}

if (safe) cout << que.front()[2];

else cout << "KAKTUS";

return 0;

}

[227. SAMSUNG SW – Candy Game]

- It was an easy simulation problem.

- Even thought it was easy, I’ve stucked by an edge condition that I had to check the origin array.

- First, swapping 2 elements in a row and checking those columns.

- Seoncd, Swapping 2 elements in a column and checking those rows.

- I spent 32 minutes and 5 seconds by the edge condition.

- see the code.

#include<iostream>

#include<vector>

#define max(a,b) a>b? a:b

using namespace std;

vector<vector<char>> table;

int N;

int answer = 1;

void LongestCandy(bool isRow,int ax1,int ax2) {

if (isRow) {

int temp1 = 1; int temp2 = 1;

char cur1 = table[0][ax1]; char cur2 = table[0][ax2];

for (int i = 1; i < N; i++) {

if (table[i][ax1] == cur1) temp1 += 1;

else {

answer = max(answer, temp1);

cur1 = table[i][ax1];

temp1 = 1;

}

if (table[i][ax2] == cur2) temp2 += 1;

else {

answer = max(answer, temp2);

cur2 = table[i][ax2];

temp2 = 1;

}

}

answer = max(answer, temp1);

answer = max(answer, temp2);

}

else {

int temp1 = 1; int temp2 = 1;

char cur1 = table[ax1][0]; char cur2 = table[ax2][0];

for (int j = 1; j < N; j++) {

if (table[ax1][j] == cur1) temp1 += 1;

else {

answer = max(answer, temp1);

cur1 = table[ax1][j];

temp1 = 1;

}

if (table[ax2][j] == cur2) temp2 += 1;

else {

answer = max(answer, temp2);

cur2 = table[ax2][j];

temp2 = 1;

}

}

answer = max(answer, temp1);

answer = max(answer, temp2);

}

}

int main() {

ios\_base::sync\_with\_stdio(false);

cin.tie(NULL); cout.tie(NULL);

//get input

cin >> N;

table.assign(N, vector<char>(N, ' '));

for (int i = 0; i < N; i++) {

for (int j = 0; j < N; j++) {

cin>>table[i][j];

}

}

//algorithm part

for (int i = 0; i < N - 1; i++) {

LongestCandy(false, i, i + 1);

}

for (int j = 0; j < N - 1; j++) {

LongestCandy(true, j, j + 1);

}

for (int i = 0; i < N; i++) {

for (int j = 0; j < N-1; j++) {

swap(table[i][j], table[i][j + 1]);

LongestCandy(true, j, j + 1);

swap(table[i][j], table[i][j + 1]);

if (i == N - 1) continue;

swap(table[i][j], table[i + 1][j]);

LongestCandy(false, i, i + 1);

swap(table[i][j], table[i + 1][j]);

}

if (i == N - 1) break;

swap(table[i][N -1], table[i + 1][N - 1]);

LongestCandy(false, i, i + 1);

swap(table[i][N - 1], table[i + 1][N - 1]);

}

cout << answer;

return 0;

}

[228. SAMSUNG SW – Bakery]

- It was a simulation problem. it looks quite difficult. But it’s not.

- Since we can’t put a pipe if a shell already has another pipe.

- But we don’t need to care of it. using greedy search, we can get our answer.

- Because, if we check up-right first, we can travel all the shell.

- First, Using DFS, finding a path from column 0 to columns M-1.

- Second, for all the row, repeating First.

- I spent 51 minutes and 46 seconds due to the above trick.

- see the code.

#include<iostream>

#include<vector>

#define max(a,b) a>b?a:b

using namespace std;

int N, M;

vector<vector<char>> table;

vector<vector<bool>> visit;

int dir[3][2] = { {-1,1} ,{0,1},{1,1} };

int answer = 0;

int sum = 0;

bool flag = false;

void DFS(int x, int y) {

if (y == M - 1) {

answer += 1;

flag = true;

return;

}

for (int d = 0; d < 3; d++) {

int nx = x+ dir[d][0]; int ny = y+dir[d][1];

if (nx < 0 || nx >= N || visit[nx][ny] == true || table[nx][ny]=='x') continue;

visit[nx][ny] = true;

DFS(nx, ny);

if (flag) return;

}

}

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));

visit.assign(N, vector<bool>(M, false));

string input;

for (int i = 0; i < N; i++) {

cin >> input;

for (int j = 0; j < M; j++) {

table[i][j]=input[j];

}

}

//algorithm part

for (int i = 0; i < N; i++) {

visit[i][0] = true;

flag = false;

DFS(i, 0);

}

cout << answer;

return 0;

}

[229. SAMSUNG SW – Sheep]

- it was an easy simultaion problem.

- Using BFS we can solve it easily.

- First, traveling all the shell, if we meet non-visit shell, start BFS from the shell.

- Second, During BFS, count sheeps and wolfs in the area.

- Thrid, if number of sheep is greater than wolf, subtract number of wolf from sum\_of wolf or Vice versa.

- I spent 11 minutes and 52 seconds.

- see the code.

#include<iostream>

#include<vector>

#include<queue>

using namespace std;

int N, M;

vector<vector<char>> table;

vector<vector<bool>> visit;

int dir[4][2] = { {-1,0},{0,-1},{1,0},{0,1} };

int sum\_of\_wolf = 0;

int sum\_of\_sheep = 0;

void BFS(int x, int y){

queue<vector<int>> que;

int num\_of\_sheep = 0; int num\_of\_wolf = 0;

if (table[x][y] == 'v') num\_of\_wolf += 1;

else if (table[x][y] == 'o') num\_of\_sheep += 1;

que.push(vector<int>{x, y});

visit[x][y] = true;

while (!que.empty()) {

x = que.front()[0]; 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 (nx < 0 || ny < 0 || nx >= N || ny >= M || table[nx][ny] == '#' || visit[nx][ny]==true) continue;

visit[nx][ny] = true;

if (table[nx][ny] == 'v') num\_of\_wolf += 1;

else if (table[nx][ny] == 'o') num\_of\_sheep += 1;

que.push(vector<int>{nx, ny});

}

}

if (num\_of\_sheep > num\_of\_wolf) sum\_of\_wolf -= num\_of\_wolf;

else sum\_of\_sheep -= num\_of\_sheep;

}

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));

visit.assign(N, vector<bool>(M, false));

for (int i = 0; i < N; i++) {

for (int j = 0; j < M; j++) {

cin >> table[i][j];

if (table[i][j] == 'v') sum\_of\_wolf += 1;

else if (table[i][j] == 'o') sum\_of\_sheep += 1;

}

}

//algorithm part

for (int i = 0; i < N; i++) {

for (int j = 0; j < M; j++) {

if (table[i][j] != '#' && visit[i][j] == false) {

BFS(i, j);

}

}

}

cout << sum\_of\_sheep << " " << sum\_of\_wolf;

return 0;

}

[230. SAMSUNG SW – Lake of Swan]

- it was a BFS problem. But it has a rigid time limit.

- My first trial was to use naïve BFS. But I’ve stuck by time.

- Second trial was binary search. For that, I’ve preprocessed how many day will be needed to melt each ice. Then, I used binary search to find minimum day that was needed to meet two swans. But I’ve stuck by time and some test-cases were not passed.

- Third trial was using four queue. For each round, I’ve recorded possible swan’s positions and possible ice that can be melt.

- Then, for those position, I pushed adjacent shells that can be next position of a swan and ice can be melt at next round.

- I spent 1 hours and 43 minutes and 26 seconds.

- see the code.

#include<iostream>

#include<vector>

#include<queue>

#define EM 0

#define ICE 987654321

using namespace std;

int N, M;

queue<vector<int>> waterQ;

queue<vector<int>> swanQ;

vector<vector<int>> table;

vector<vector<bool>> visit;

vector<pair<int,int>> swan; //sx sy, dx dy

int dir[4][2] = { {-1,0},{0,-1},{1,0},{0,1} };

int sx, sy, dx, dy;

void PrintTable() {

cout << endl;

for (vector<int> row : table) {

for (int c : row) cout << c << " ";

cout << endl;

}

cout << endl;

}

void MeltingPreprocessing() {

queue<vector<int>> water = waterQ;

waterQ = queue<vector<int>>{};

while (!water.empty()) {

int x = water.front()[0]; int y = water.front()[1]; water.pop();

for (int d = 0; d < 4; d++) {

int nx = x + dir[d][0]; int ny = y + dir[d][1];

if (nx < 0 || ny < 0 || nx >= N || ny >= M) continue;

if (table[nx][ny] == ICE) {

table[nx][ny] = EM;

waterQ.push(vector<int>{nx, ny});

}

}

}

}

bool Travel() {

queue<vector<int>> que=swanQ;

swanQ = queue<vector<int>>{};

visit[que.front()[0]][que.front()[1]] = 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 (nx < 0 || ny < 0 || nx >= N || ny >= M || visit[nx][ny]==true) continue;

if (table[nx][ny]==ICE) {

swanQ.push(vector<int>{nx, ny});

visit[nx][ny] = true;

continue;

}

if (nx == dx && ny == dy) return true;

visit[nx][ny] = true;

que.push(vector<int>{nx, ny});

}

}

return false;

}

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,EM));

visit.assign(N, vector<bool>(M, false));

string input;

for (int i = 0; i < N; i++) {

cin >> input;

for (int j = 0; j < M; j++) {

if (input[j] == '.') {

table[i][j] = EM;

waterQ.push(vector<int>{i, j});

}

else if (input[j] == 'L') {

swan.emplace\_back(make\_pair(i, j));

waterQ.push(vector<int>{i, j});

table[i][j] = EM;

}

else table[i][j] = ICE;

}

}

sx = swan[0].first; sy = swan[0].second;

dx = swan[1].first; dy = swan[1].second;

//algorithm part

swanQ.push(vector<int>{sx, sy});

int answer = 0;

while (!Travel()) {

answer += 1;

MeltingPreprocessing();

}

cout << answer;

return 0;

}

[231. SAMSUNG SW – Pioneer of Catan]

- It was a difficult simulation problem. Since a shell has 6 neighbours.

- So implement 6 neighbout was the key of this problem.

- The second difficult thing was the way to dertermine next position.

- There are 6 possible next position. Let’s say current position is (x,y). Then possible next positions are (x-1,y+1), (x-2,y), (x-1,y-1), (x+1,y-1), (x+2,y), (x+1,y+1).

- To determine a next position, I drew some example. From the example, I’ve found a such rule.

- First round, we have to move above coordinates with {1,0,1,1,1,1} time respectively.

- Second round, {2,1,2,2,2,2}. Third round, {3,2,3,3,3,3} and so on.

- Finding next number was not that difficult. I just had to find minimum number that has minimum frequency.

- In addition, determining 2D array size was one of the key. If I took too small size, run time error occurred. But if I took too big size, I’ve stuck by memory limitation.

- Finally I found an adequate size that was 600 x 300.

- I spent 1 hours or so.

- see the code.

#include<iostream>

#include<vector>

using namespace std;

vector<vector<int>> table;

int C,N;

int sx, sy;

const int dir[6][2] = { {-1,1},{-2,0},{-1,-1},{1,-1},{2,0},{1,1} };

vector<int> count\_num; // To count number of apearance of each number.

vector<int> num\_of\_move;

int FindNextNumber(int x, int y) {

vector<bool> visit(6,false); // index 0 won't be used.

for (int d = 0; d < 6; d++) { //to check adjacent numbers

int nx = x + dir[d][0]; int ny = y + dir[d][1];

if (nx < 0 || ny < 0 || nx >= 600 || ny >= 300) continue;

visit[table[nx][ny]] = true;

}

vector<int> will\_check;

for (int i = 1; i < 6; i++) if (visit[i] == false) will\_check.emplace\_back(i);

int min\_num = count\_num[will\_check[0]];

int min\_index = will\_check[0];

for (int i = 1; i < will\_check.size(); i++) {

if (min\_num > count\_num[will\_check[i]]) {

min\_index = will\_check[i];

min\_num = count\_num[will\_check[i]];

}

}

count\_num[min\_index] += 1;

return min\_index;

}

int Travel(int x,int y) {

int cur\_round = 1;

table[x][y] = 1;

int idx = 0;

while (true) {

for (int d = 0; d < 6; d++) {

for (int move\_num = 0; move\_num < num\_of\_move[d]+idx; move\_num++) {

cur\_round += 1;

x = x + dir[d][0]; y = y + dir[d][1];

table[x][y]=FindNextNumber(x,y);

if (cur\_round == N) return table[x][y];

}

}

idx += 1;

}

}

int main() {

ios\_base::sync\_with\_stdio(false);

cin.tie(NULL); cout.tie(NULL);

//get input

cin >> C;

//algorithm part

for (int t = 0; t < C; t++) {

cin >> N;

if (N == 1) cout << 1 << endl;

else {

table.assign(600, vector<int>(300, 0));

num\_of\_move = { 1,0,1,1,1,1 };

count\_num = { 0,1,0,0,0,0 };

sx = 300, sy = 150;

int answer = Travel(sx, sy);

cout << answer << endl;

}

}

return 0;

}

[232. SAMSUNG SW – Magic Star]

- It was a permutation problem. Thanks to this problem, I used DFS for a long time.

- First, I had to record empty position of magic star and had to find missing numbers.

- Second, For each empty position, I’ve inserted a possible value in turn with lexicographical order.

- I’ve forgot to check spending time. But I guess almost 40 minutes or so.

- see the code.

#include<iostream>

#include<vector>

using namespace std;

vector<vector<int>> table(5, vector<int>(9, -1));

vector<bool> alpha(12, 0);

vector<int> need\_number;

vector<pair<int, int>> pos;

vector<bool> visit;

bool found = false;

int empty\_size = 0;

bool IsMagicStar() {

int sum = 0;

for (int i = 0, j = 4; i < 4; i++, j--) sum += table[i][j]+1;

if (sum != 26) return false;

sum = 0;

for (int i = 0, j = 4; i < 4; i++, j++) sum += table[i][j]+1;

if (sum != 26) return false;

sum = 0;

for (int j = 1; j < 9; j += 2) sum += table[1][j]+1;

if (sum != 26) return false;

sum = 0;

for (int j = 1; j < 9; j += 2) sum += table[3][j]+1;

if (sum != 26) return false;

sum = 0;

for (int i = 4, j = 4; i > 0; i--, j--) sum += table[i][j]+1;

if (sum != 26) return false;

sum = 0;

for (int i = 4, j = 4; i > 0; i--, j++) sum += table[i][j]+1;

if (sum != 26) return false;

return true;

}

void DFS(int insert\_pos, int cnt) {

if (cnt == empty\_size) {

if (IsMagicStar()) {

found = true;

}

return;

}

int x = pos[insert\_pos].first; int y = pos[insert\_pos].second;

for (int i = 0; i < empty\_size; i++) {

if (visit[i] == true) continue;

table[x][y] = need\_number[i];

visit[i] = true;

DFS(insert\_pos+1, cnt + 1);

if (found) return;

visit[i] = false;

table[x][y] = -1;

}

}

int main() {

ios\_base::sync\_with\_stdio(false);

cin.tie(NULL); cout.tie(NULL);

//get input

string input;

for (int i = 0; i < 5; i++) {

cin >> input;

for (int j = 0; j < 9; j++) {

if (input[j] == 'x') pos.emplace\_back(make\_pair(i, j));

else if ('A' <= input[j] && input[j] <= 'L') {

table[i][j] = input[j] - 'A';

alpha[input[j] - 'A'] = true;

}

}

}

empty\_size = pos.size();

for (int i = 0; i < 12; i++) if (alpha[i] == false) need\_number.emplace\_back(i);

visit.assign(need\_number.size(), false);

//algorithm part

DFS(0, 0);

for (int i = 0; i < 5; i++) {

for (int j = 0; j < 9; j++) {

if (table[i][j] != -1) cout << char(table[i][j] + 'A');

else cout << '.';

}

cout << endl;

}

}

[233. SAMSUNG SW – Fire!]

- It was a simulation problem.

- They gave an ambiguous condition that there is no mention which is first between person and fire.

- First, I moved person first but it was wrong. so I swap the order.

- For person’s move, I used double while syntax. Since after moving 1 shell, fire has to be spreaded.

- For Fire’s move, I used two queue to remember currently spreaded fires.

- I spent 32 minutes and 2 seconds.

- see the code.

#include<iostream>

#include<vector>

#include<queue>

#define FAIL 987654321

using namespace std;

int N, M;

vector<vector<char>> table;

int sx, sy;

const int dir[4][2] = { {-1,0},{0,-1},{1,0},{0,1} };

vector<vector<bool>> visit;

vector<vector<bool>> fire\_visit;

queue<vector<int>> fire\_que;

void PrintTable() {

cout << endl;

for (vector<char> row : table) {

for (char c : row) cout << c << " ";

cout << endl;

}

cout << endl;

}

void MoveFire() {

queue<vector<int>> NQ = fire\_que;

fire\_que = queue<vector<int>>{};

while (!NQ.empty()) {

int x = NQ.front()[0]; int y = NQ.front()[1]; NQ.pop();

if (fire\_visit[x][y]==false) {

fire\_visit[x][y] = true;

for (int d = 0; d < 4; d++) {

int nx = x + dir[d][0]; int ny = y + dir[d][1];

if (nx < 0 || ny < 0 || nx >= N || ny >= M || table[nx][ny] == '#') continue;

table[nx][ny] = 'F';

fire\_que.push(vector<int>{nx, ny});

}

}

}

}

int MovePerson() {

queue<vector<int>> que;

que.push(vector<int>{sx, sy, 1});

visit[sx][sy] = true;

while (!que.empty()) {

MoveFire();

int cur\_cnt = que.front()[2];

while (!que.empty()) {

int x = que.front()[0]; int y = que.front()[1]; int cnt = que.front()[2];

if (x == 0 || y == 0 || x == N - 1 || y == M - 1) return cnt;

if (cur\_cnt != cnt) break;

que.pop();

for (int d = 0; d < 4; d++) {

int nx = x + dir[d][0]; int ny = y + dir[d][1];

if (nx < 0 || ny < 0 || nx >= N || ny >= M || table[nx][ny] == '#'|| table[nx][ny]=='F' || visit[nx][ny]) continue;

visit[nx][ny] = true;

que.push(vector<int>{nx, ny, cnt + 1});

}

}

}

return FAIL;

}

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));

fire\_visit.assign(N, vector<bool>(M, false));

visit.assign(N, vector<bool>(M, false));

for (int i = 0; i < N; i++) {

for (int j = 0; j < M; j++) {

cin >> table[i][j];

if (table[i][j] == 'J') {

sx = i; sy = j;

}

else if (table[i][j] == 'F') {

fire\_que.push(vector<int>{i, j});

}

}

}

int answer=MovePerson();

if (answer == FAIL) cout << "IMPOSSIBLE ";

else cout << answer;

return 0;

}

[234. SAMSUNG SW – Degree of Kinship]

- It was a travelling problem composed of tree structure.

- I used unorderd\_map to represent edges between nodes.

- First, Recording a given node’s parent.

- Second, for two node we need to find their degree of kinship, making vectors to record their parent.

- Third, Finding a closest same parent.

- I spent 19 minutes and 58 seconds.

- see the code.

#include<iostream>

#include<vector>

#include<array>

#include<unordered\_map>

using namespace std;

int N, S, D,M;

unordered\_map<int, int> parent;

void PrintTable(vector<int>& left, vector<int>& right) {

cout << endl;

for (int i : left) cout << i << " ";

cout << endl;

for (int i : right) cout << i << " ";

cout << endl;

}

int main() {

ios\_base::sync\_with\_stdio(false);

cin.tie(NULL); cout.tie(NULL);

//get input

cin >> N>>S>>D>>M;

int x, y;

for (int i = 0; i < M; i++) {

cin >> x >> y;

parent[y] = x;

}

//algorithm part

vector<int> left, right;

left.emplace\_back(S); right.emplace\_back(D);

while (parent[S] != 0) {

left.emplace\_back(parent[S]);

S = parent[S];

}

while (parent[D] != 0) {

right.emplace\_back(parent[D]);

D = parent[D];

}

int answer = -1;

for (int i = 0; i < right.size(); i++) {

for (int j = 0; j < left.size(); j++) {

if (right[i] == left[j]) {

answer = i + j;

break;

}

}

if (answer != -1) break;

}

cout << answer;

return 0;

}

[235. SAMSUNG SW – Number of Island]

- It was an easy BFS problem. But they gave me row and columns size oppositely.

- Due to that, I spent little more time.

- the algorithm is that During traveling all the shell of 2D array, if we meet a non-visit ground, start BFS to visit the ground’s island.

- I spent 11 minutes and 45 seconds.

- see the code.

#include<iostream>

#include<vector>

#include<queue>

using namespace std;

int N, M;

vector<vector<int>> table;

vector<vector<int>> visit;

int dir[8][2] = { {-1,0},{-1,-1},{0,-1},{1,-1},{1,0},{1,1},{0,1},{-1,1} };

void BFS(int x,int y, int cnt) {

queue<vector<int>> que;

que.push(vector<int>{x, y});

visit[x][y] = cnt;

while (!que.empty()) {

x = que.front()[0]; y = que.front()[1]; que.pop();

for (int d = 0; d < 8; d++) {

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 || visit[nx][ny] == cnt) continue;

visit[nx][ny] = cnt;

que.push(vector<int>{nx, ny});

}

}

}

int main() {

ios\_base::sync\_with\_stdio(false);

cin.tie(NULL); cout.tie(NULL);

int cnt = 0;

do {

//get input

cin >> M >> N;

if (N == 0 && M == 0) break;

table.assign(N, vector<int>(M, 0));

visit.assign(N, vector<int>(M, -1));

for (int i = 0; i < N; i++) {

for (int j = 0; j < M; j++) {

cin >> table[i][j];

}

}

//algorithm part

int num\_of\_island = 0;

for (int i = 0; i < N; i++) {

for (int j = 0; j < M; j++) {

if (table[i][j] == 1 && visit[i][j] != cnt) {

BFS(i, j,cnt);

num\_of\_island+=1;

}

}

}

cout << num\_of\_island << endl;

cnt += 1;

} while (true);

return 0;

}

[236. SAMSUNG SW – Start Link]

- It was a BFS problem.

- Just using queue, pushing current floor + U or – D floor to the queue.

- If we meet the target floor, return the number of move.

- I spent 8 minutes and 22 seconds.

- see the code.

#include<iostream>

#include<queue>

#include<unordered\_map>

#define FAIL -1

using namespace std;

int F, S, G, U, D;

int Queueing() {

queue<vector<int>> que;

que.push(vector<int>{S, 0}); // start pos , number of needed move.

unordered\_map<int, bool> visit;

visit[S] = true;

while (!que.empty()) {

int cur = que.front()[0]; int cnt = que.front()[1]; que.pop();

if (cur == G) return cnt;

if (visit[cur + U] == false && cur+U<=F) {

visit[cur + U] = true;

que.push(vector<int>{cur + U, cnt + 1});

}

if (visit[cur - D] == false && cur-D>=1) {

visit[cur - D] = true;

que.push(vector<int>{cur - D, cnt + 1});

}

}

return -1;

}

int main() {

ios\_base::sync\_with\_stdio(false);

cin.tie(NULL); cout.tie(NULL);

//get input

cin >> F >> S >> G >> U >> D;

//algorithm part

int answer=Queueing();

if (answer == FAIL) cout << "use the stairs";

else cout << answer;

return 0;

}

[237. SAMSUNG SW – Connection]

- It was a BFS problem. But there was restrictions.

- We have to minimize two path without overlapping a line on another line.

- First, using queue, I found A’s minimum path and record the path.

- Second, Considering the recorded path, I found B’s minimum path.

- Third, after swapping A and B, repeating First and Second. Since we have to consider which one have to be first line to find minimum.

- I spent 43 minutes and 31 seconds.

- see the code.

#include<iostream>

#include<vector>

#include<queue>

#define MAX 987654321

using namespace std;

int N, M;

vector<vector<int>> table;

int asx, asy, adx, ady;

int bsx, bsy, bdx, bdy;

int answer = MAX;

int dir[4][2] = { {-1,0},{0,-1},{1,0},{0,1} };

void BFS(int a, int b, int c, int d, int e, int f, int g, int h) {

queue <pair<vector<int>, vector<pair<int,int>>>> que;

que.push(make\_pair(vector<int>{a, b,0}, vector<pair<int,int>>{make\_pair(a,b)})); //pair of (current position vector with move count) , (all the shell passed)

vector<vector<bool>> visit(N,vector<bool>(M,false));

visit[a][b] = true;

visit[e][f] = true;

visit[g][h] = true;

vector<pair<int, int>> record;

int a\_dis = MAX; int b\_dis = MAX;

while (!que.empty()) {

int x = que.front().first[0]; int y = que.front().first[1]; int cnt = que.front().first[2]; vector<pair<int, int>> path = que.front().second;

if (x == c && y == d) {

record = path;

a\_dis = cnt;

break;

}

que.pop();

for (int d = 0; d < 4; d++) {

int nx = x + dir[d][0]; int ny = y + dir[d][1];

if (nx < 0 || ny < 0 || nx >= N || ny >= M || visit[nx][ny] == true) continue;

path.emplace\_back(make\_pair(nx, ny));

que.push(make\_pair(vector<int>{nx, ny, cnt + 1}, path));

path.pop\_back();

visit[nx][ny] = true;

}

}

queue<vector<int>> que2;

que2.push(vector<int>{e, f, 0});

visit.assign(N, vector<bool>(M, false));

for (pair<int, int> row : record) visit[row.first][row.second] = true;

visit[e][f] = true;

visit[a][b] = true;

visit[c][d] = true;

while (!que2.empty()) {

int x = que2.front()[0]; int y = que2.front()[1]; int cnt = que2.front()[2];

if (x == g && y == h) {

b\_dis = cnt;

break;

}

que2.pop();

for (int d = 0; d < 4; d++) {

int nx = x + dir[d][0]; int ny = y + dir[d][1];

if (nx < 0 || ny < 0 || nx >= N || ny >= M || visit[nx][ny] == true) continue;

que2.push(vector<int>{nx, ny, cnt + 1});

visit[nx][ny] = true;

}

}

answer = min(answer, a\_dis + b\_dis);

return;

}

int main() {

ios\_base::sync\_with\_stdio(false);

cin.tie(NULL); cout.tie(NULL);

//get input

cin >> M >> N;

cin >> asy >> asx >> ady >> adx;

cin >> bsy >> bsx >> bdy >> bdx;

N += 1; M += 1;

table.assign(N, vector<int>(M, 0));

//algorithm part

BFS(asx, asy, adx, ady, bsx, bsy, bdx, bdy);

BFS(bsx, bsy, bdx, bdy, asx, asy, adx, ady);

if (answer == MAX) cout << "IMPOSSIBLE";

else cout << answer;

return 0;

}

[238. SAMSUNG SW – Private Lesson Man]

- It was a difficult BFS problem.

- A shell has 2 numbers and we can move when a shell has same number with its adjacent shell and the two numbers are adjacent directly.

- I also had to record path to destination. Morever, when we can not arrive destination, we had to return the greatest shell we can reach with the path.

- First, I took input as a 2D array, like given example.

> 1 2 2 4 4 5

> 2 4 5 3 …

- Second, I made a vector “visit” to check a shell we’ve already visit.

- Third, I create a function “ResIndex” that it returns a shell’s number corresponding to the shell’s index.

- Fourth, I create a function “ResPair” that it returns a pair of coordinates which occupy same shell.

- Fifth, I used queue to implemet BFS, the que has current coordinates, count of move, and path.

> Using the coordinates and using ResPair function, we can get the frined’s coordinate and number.

> the two given coordinates, I visited adjancet shells using BFS.

- Sixth, If we can get destinaion, return the count of move and path. if not and que becomes empty, return the greatest number of shell with its path.

- I spent 1 hours and 20 minutes.

- see the code.

#include<iostream>

#include<vector>

#include<queue>

using namespace std;

int N,M;

vector<vector<int>> table;

vector<bool> visit;

int dir[4][2] = { {-1,0},{0,-1},{1,0},{0,1} };

int destination;

vector<int> max\_path;

int cur\_max = 0;

int cur\_answer = 0;

int answer;

bool flag = false;

//N+(N-1)

inline int ResIndex(int x, int y) {

int res = 0;

if (x % 2 == 0) {

res = (N + N - 1) \* (x/2) + 1;

res += (y / 2);

}

else {

res = N;

res += (N + N - 1) \* (x / 2) + 1;

res += (y - 1) / 2;

}

return res;

}

inline pair<int,int> ResPair(int x, int y) {

if (x % 2 == 0) {

if (y % 2 == 0) return make\_pair(x, y + 1);

else return make\_pair(x, y - 1);

}

else {

if (y % 2 == 0) return make\_pair(x, y - 1);

else return make\_pair(x, y + 1);

}

}

void PrintTable() {

cout << endl;

for (vector<int> row : table) {

for (int i : row) cout << i << " ";

cout << endl;

}

cout << endl;

}

vector<int> BFS() {

queue<pair<vector<int>, vector<int>>> que;

que.push(make\_pair(vector<int>{0, 0, 1}, vector<int>{1}));

max\_path = vector<int>{ 1 };

visit[ResIndex(0, 0)] = true;

while (!que.empty()) {

int fx = que.front().first[0]; int fy = que.front().first[1]; int cnt = que.front().first[2]; vector<int> path = que.front().second;

pair<int, int> cor = ResPair(fx, fy); int lx = cor.first; int ly = cor.second;

que.pop();

if (ResIndex(fx, fy) == destination) {

answer = cnt;

flag = true;

return path;

}

if (cur\_max < ResIndex(fx, fy)) {

max\_path = path;

cur\_max = ResIndex(fx, fy);

cur\_answer = cnt;

}

for (int d = 0; d < 4; d++) {

int nfx = fx + dir[d][0]; int nfy = fy + dir[d][1]; int nlx = lx + dir[d][0]; int nly = ly + dir[d][1];

if (nfx < 0 || nfy < 0 || nlx < 0 || nly < 0 || nfx>=N || nlx>=N || nfy>=M || nly>=M) continue;

if (table[nfx][nfy] == table[fx][fy] && visit[ResIndex(nfx, nfy)] == false) {

path.emplace\_back(ResIndex(nfx, nfy));

visit[ResIndex(nfx, nfy)] = true;

que.push(make\_pair(vector<int>{nfx, nfy, cnt + 1}, path));

path.pop\_back();

}

if (table[nlx][nly]==table[lx][ly] && visit[ResIndex(nlx,nly)]==false) {

path.emplace\_back(ResIndex(nlx, nly));

visit[ResIndex(nlx, nly)] = true;

que.push(make\_pair(vector<int>{nlx, nly, cnt + 1}, path));

path.pop\_back();

}

}

}

// return max path

return max\_path;

}

int main() {

ios\_base::sync\_with\_stdio(false);

cin.tie(NULL); cout.tie(NULL);

//get input

cin >> N;

M = 2 \* N;

table.assign(N, vector<int>( M, -1));

int a, b;

for (int i = 0; i < N; i++) {

if (i % 2 == 0) {

for (int j = 0; j < M; j+=2) {

cin >> table[i][j] >> table[i][j + 1];

}

}

else {

for (int j = 1; j < M - 1; j+=2) {

cin >> table[i][j] >> table[i][j + 1];

}

}

}

if (N % 2 == 1) {

destination = (N + N - 1) \* (N / 2) + N;

visit.assign((N + N - 1) \* (N / 2) + N+1, 0);

}

else {

destination = (N + N - 1) \* (N / 2);

visit.assign((N + N - 1) \* (N / 2)+1, 0);

}

//algorithm part

vector<int> res = BFS();

if (flag) {

cout << answer << endl;

for (int i : res) cout << i << " ";

}

else {

cout << cur\_answer<<endl;

for (int i : max\_path) cout << i << " ";

}

return 0;

}

[239. SAMSUNG SW – Fire]

- It was a simulation problem.

- First, fire has to be spreaded. So I used 2 queue, a queue for recording new position of fire, another queue for implemting BFS.

- Second, person has to move to edge of the building. So I used 2 queue, like above. Since fire speads every second, we have to record current position to move at next second.

- I spent 16 minutes and 14 seconds.

- see the code.

#include<iostream>

#include<vector>

#include<queue>

#define FAIL 987654321

using namespace std;

int N, M, T;

vector<vector<char>> table;

int sx, sy;

queue<pair<int, int>> fire;

queue<vector<int>> person;

const int dir[4][2] = { {-1,0},{0,-1},{1,0},{0,1} };

void MoveFire() {

queue<pair<int, int>> que = fire;

fire = queue<pair<int, int>>{};

while (!que.empty()) {

int x = que.front().first; int y = que.front().second; que.pop();

for (int d = 0; d < 4; d++) {

int nx = x + dir[d][0]; int ny = y + dir[d][1];

if (nx < 0 || ny < 0 || nx >= N || ny >= M || table[nx][ny] == '#' || table[nx][ny]=='\*') continue;

fire.push(make\_pair(nx, ny));

table[nx][ny] = '\*';

}

}

}

int BFS() {

vector<vector<bool>> visit(N, vector<bool>(M, false));

visit[sx][sy] = true;

while (!person.empty()) {

queue<vector<int>> que = person;

person = queue<vector<int>>{};

MoveFire();

while (!que.empty()) {

int x = que.front()[0]; int y = que.front()[1]; int cnt = que.front()[2]; que.pop();

if (x == 0 || y == 0 || x == N - 1 || y == M - 1) return cnt;

for (int d = 0; d < 4; d++) {

int nx = x + dir[d][0]; int ny = y + dir[d][1];

if (nx < 0 || ny < 0 || nx >= N || ny >= M || table[nx][ny] == '#' || table[nx][ny] == '\*' || visit[nx][ny]==true) continue;

visit[nx][ny] = true;

person.push(vector<int>{nx, ny, cnt + 1});

}

}

}

return FAIL;

}

int main() {

ios\_base::sync\_with\_stdio(false);

cin.tie(NULL); cout.tie(NULL);

cin >> T;

for (int test\_case = 0; test\_case < T; test\_case++) {

//get input

cin >> M >> N;

table.assign(N, vector<char>(M, ' '));

for (int i = 0; i < N; i++) {

for (int j = 0; j < M; j++) {

cin >> table[i][j];

if (table[i][j] == '@') {

sx = i; sy = j;

}

if (table[i][j] == '\*') fire.push(make\_pair(i, j));

}

}

person.push(vector<int>{sx, sy, 1});

//algorithm part

int answer=BFS();

if (answer == FAIL) cout << "IMPOSSIBLE"<<endl;

else cout << answer << endl;

person = queue<vector<int>>{};

fire = queue<pair<int, int>>{};

}

return 0;

}

[240. SAMSUNG SW – チーズ]

- It was a simulation problem. But it was written in japanese.

- So I guessed the problem using input and output.

- In my interpretation, we have to find minimum path from start position to destinations.

- The number of destination can be 9 at most. And we have to reach with ascending order.

- I spent 12 minutes and 4 seconds.

- see the code.

#include<iostream>

#include<vector>

#include<queue>

#include<string>

using namespace std;

int N, M, T;

vector<vector<char>> table;

vector<vector<int>> visit;

const int dir[4][2] = { {-1,0},{0,-1},{1,0},{0,1} };

int sx, sy;

int answer=0;

void BFS(int bfs\_cnt, int target) {

queue<vector<int>> que;

que.push(vector<int>{sx, sy, 0});

visit[sx][sy] = bfs\_cnt;

while (!que.empty()) {

int x = que.front()[0]; int y = que.front()[1]; int cnt = que.front()[2]; que.pop();

if (table[x][y]-'0' == target) {

answer += cnt;

sx = x; sy = y;

return;

}

for (int d = 0; d < 4; d++) {

int nx = x + dir[d][0]; int ny = y + dir[d][1];

if (nx < 0 || ny < 0 || nx >= N || ny >= M || table[nx][ny] == 'X' || visit[nx][ny] == bfs\_cnt) continue;

que.push(vector<int>{nx, ny, cnt + 1});

visit[nx][ny] = bfs\_cnt;

}

}

}

int main() {

ios\_base::sync\_with\_stdio(false);

cin.tie(NULL); cout.tie(NULL);

//get input

cin >> N >> M >> T;

table.assign(N, vector<char>(M, 0));

visit.assign(N, vector<int>(M, -1));

for (int i = 0; i < N; i++) {

for (int j = 0; j < M; j++) {

cin >> table[i][j];

if (table[i][j] == 'S') {

sx = i; sy = j;

}

}

}

//algorithm part

int target = 1;

while (target <= T) {

BFS(target, target);

target += 1;

}

cout << answer;

return 0;

}

[240. SAMSUNG SW – Dice Game]

- It was an easy simulation problem.

- they gave us a board and numbers from rolled dice.

- we just should simulate the game

- First, move as much as a number from rolled dice.

- Second, move again as much as a number on the board.

- Iterating first and second, when we arrive N(destination) or over N, return how many move we took.

- I spent 15 minutes and 13 seconds.

- see the code.

#include<iostream>

#include<vector>

using namespace std;

int N, M;

vector<int> table;

vector<int> dice;

int main() {

ios\_base::sync\_with\_stdio(false);

cin.tie(NULL); cout.tie(NULL);

//get input

cin >> N >> M;

table.assign(N, 0);

dice.assign(M, 0);

for (int i = 0; i < N; i++) cin >> table[i];

for (int i = 0; i < M; i++) cin >> dice[i];

//algorithm part

int j = 0;

int cur = 0;

while (j < M) {

if (cur+dice[j]>=N-1 || cur + dice[j] + table[cur + dice[j]] >= N-1) {

cout << j+1;

return 0;

}

cur += dice[j] + table[cur + dice[j]];

j += 1;

}

return 0;

}

[241. SAMSUNG SW – Laser Communication]

- It was a difficult simulation problem for a long time.

- Since I had to remember all the possible number of mirror a shell can have.

- At first, I tried to approach just using BFS. But It was not that easy.

- So I started to check all the shell’s possible mirror number.

- I worried since given memory size may be not enough to remember queue’s elements.

- But I had no choice without it.

- I spent 1 hours and 30 minutes and 12 seconds.

- see the code.

#include<iostream>

#include<vector>

#include<queue>

using namespace std;

int N, M;

vector<vector<char>> table;

vector<pair<int, int>> start;

int answer = 987654321;

const int dir[4][2] = { {-1,0},{0,-1},{1,0},{0,1} };

void BFS() {

queue<vector<int>> que;

vector<vector<int>> visit(N, vector<int>(M, 987654321));

//for initial direction

for (int d = 0; d < 4; d++) {

que.push(vector<int>{start[0].first,start[0].second,0, d});

}

visit[start[0].first][start[0].second] = true;

while (!que.empty()) {

int x = que.front()[0]; int y = que.front()[1]; int cnt = que.front()[2]; int hori = que.front()[3]; que.pop();

for (int d = 0; d < 4; d++) {

int nx = x + dir[d][0]; int ny = y + dir[d][1]; int cur\_cnt = cnt;

if (nx < 0 || ny < 0 || nx >= N || ny >= M || table[nx][ny] == '\*') continue;

if (hori != d) cur\_cnt += 1;

if (visit[nx][ny] >= cur\_cnt) {

visit[nx][ny] = cur\_cnt;

que.push(vector<int>{nx, ny, cur\_cnt, d});

}

}

}

answer = visit[start[1].first][start[1].second];

}

int main() {

ios\_base::sync\_with\_stdio(false);

cin.tie(NULL); cout.tie(NULL);

//get input

cin >> M >> N;

table.assign(N, vector<char>(M, 0));

for (int i = 0; i < N; i++) {

for (int j = 0; j < M; j++) {

cin >> table[i][j];

if (table[i][j] == 'C') {

start.emplace\_back(make\_pair(i, j));

}

}

}

//algorithm part

BFS();

cout << answer;

return 0;

}

[242. SAMSUNG SW – Move of Knight]

- It was an easy simulation problem.

- Using BFS with a given moving rules of knight, we can solve it very easily.

- I spent 6 minutes and 5 seconds.

- see the code.

#include<iostream>

#include<vector>

#include<queue>

using namespace std;

int N, T;

vector<vector<bool>> table;

const int dir[8][2] = { {-2,-1},{-1,-2},{1,-2},{2,-1},{2,1},{1,2},{-1,2},{-2,1} };

int sx, sy, dx, dy;

int BFS() {

queue<vector<int>> que;

que.push(vector<int>{sx, sy, 0});

table[sx][sy] = true;

while (!que.empty()) {

int x = que.front()[0]; int y = que.front()[1]; int cnt = que.front()[2]; que.pop();

if (x == dx && y == dy) return cnt;

for (int d = 0; d < 8; d++) {

int nx = x + dir[d][0]; int ny = y + dir[d][1];

if (nx < 0 || ny < 0 || nx >= N || ny >= N || table[nx][ny] == true) continue;

que.push(vector<int>{nx, ny, cnt + 1});

table[nx][ny] = true;

}

}

return 0;

}

int main() {

ios\_base::sync\_with\_stdio(false);

cin.tie(NULL); cout.tie(NULL);

cin >> T;

for (int test = 0; test < T; test++) {

//get input;

cin >> N;

table.assign(N, vector<bool>(N, false));

cin >> sx >> sy >> dx >> dy;

cout<<BFS()<<endl;

}

return 0;

}

[243. SAMSUNG SW – Tomato]

- It was a simulation problem with 3D matrix.

- Except that a given matrix is 3D, it was a simliar problem I’ve solved till now.

- Just using BFS with 6 directions that are up, down, north, west, south, east, we can solve it easily.

- I used two queues to record new riped tomatos.

- I spent 13 minutes and 13 seconds.

- see the code.

#include<iostream>

#include<vector>

#include<queue>

using namespace std;

int N, M, H;

vector<vector<vector<int>>> table;

int not\_tomato = 0;

const int dir[6][3] = { {-1,0,0},{0,-1,0},{0,0,-1},{1,0,0},{0,1,0},{0,0,1} };

queue<vector<int>> BFS(queue<vector<int>> que) {

queue<vector<int>> res;

while(!que.empty()) {

int h = que.front()[0]; int x = que.front()[1]; int y = que.front()[2]; que.pop();

for (int d = 0; d < 6; d++) {

int nh = h + dir[d][0]; int nx = x + dir[d][1]; int ny = y + dir[d][2];

if (nh < 0 || nx < 0 || ny < 0 || nh >= H || nx >= N || ny >= M || table[nh][nx][ny] == 1 || table[nh][nx][ny] == -1) continue;

table[nh][nx][ny] = 1;

not\_tomato -= 1;

res.push(vector<int>{nh, nx, ny});

}

}

return res;

}

int main() {

ios\_base::sync\_with\_stdio(false);

cin.tie(NULL); cout.tie(NULL);

//get input

queue<vector<int>> que;

cin >> M >> N >> H;

table.assign(H, vector<vector<int>>(N, vector<int>(M, 0)));

for (int h = H - 1; h >= 0; h--) {

for (int i = 0; i < N; i++) {

for (int j = 0; j < M; j++) {

cin >> table[h][i][j];

if (table[h][i][j] == 0) not\_tomato += 1;

if (table[h][i][j] == 1) que.push(vector<int>{h, i, j});

}

}

}

//algorithm part

int answer = 0;

while (not\_tomato > 0) {

answer += 1;

int cur\_tomato = not\_tomato;

que = BFS(que);

if (cur\_tomato == not\_tomato) {

cout << -1;

return 0;

}

}

cout << answer;

return 0;

}

[244. SAMSUNG SW – Walking with Drinking Beer]

- I thought it was a BFS-simulation problem. But it was not.

- Since to solve this problem using BFS, it takes so much time.

- So I changed my algorithm. it was easy.

- First, pushing every beer shop or destination where we can reach with 20 beers from a current position to a queue.

- Second, Using the queue, iterating “First”, and if we can reach destination, return true. or return false.

- I spent 39 minutes and 29 seconds. Because of wrong thought at first.

- see the code.

#include<iostream>

#include<vector>

#include<queue>

#define ORIGIN 32768

using namespace std;

int N, T;

int sx, sy, dx, dy;

vector<bool> visit;

vector<pair<int, int>> beer;

bool BFS() {

queue<vector<int>> que;

que.push(vector<int>{sx, sy});

while (!que.empty()) {

int x = que.front()[0]; int y = que.front()[1]; que.pop();

if (x == dx && y == dy) return true;

for (int i = 0; i < N+1; i++) {

if (visit[i] == true) continue;

int nx = beer[i].first; int ny = beer[i].second;

if (abs(x -nx) + abs(y - ny) <= 1000) {

que.push(vector<int>{nx,ny});

visit[i] = true;

}

}

}

return false;

}

int main() {

ios\_base::sync\_with\_stdio(false);

cin.tie(NULL); cout.tie(NULL);

cin >> T;

for (int test = 0; test < T; test++) {

//get input

cin >> N;

cin >> sx >> sy;

visit.assign(N+1, false);

beer.assign(N+1, pair<int, int>());

for (int i = 1; i < N+1; i++) cin >> beer[i].first >> beer[i].second;

cin >> dx >> dy;

beer[0].first = dx; beer[0].second = dy;

//algorithm part

if (BFS()) cout << "happy" << endl;

else cout << "sad" << endl;

}

return 0;

}

[245. SAMSUNG SW – Peg Solitaire]

- It was DFS problem for a long time.

- For all the pin, we have to simulate all the possible move.

- I used DFS for backtracking.

- First, record all the pin’s position.

- Second, for each pin, move if possible. and revise the given board.

- Third, return a maximum DFS depth.

- I spent 25 minutes and 55 seconds.

- see the code.

#include<iostream>

#include<vector>

#include<algorithm>

using namespace std;

const int N = 5;

const int M = 9;

const int dir[4][2] = { {-1,0},{0,-1},{1,0},{0,1} };

vector<vector<char>> table;

vector<vector<int>> pins;

int num\_pin = 0;

int pin\_size;

int answer;

void DFS(int cnt) {

answer = max(answer, cnt);

for (int i = 0; i < pin\_size; i++) {

int x = pins[i][0]; int y = pins[i][1];

if (table[x][y] == '.') continue;

for (int d = 0; d < 4; d++) {

int nx = x + dir[d][0]; int ny = y + dir[d][1];

int nnx = x + dir[d][0] \* 2; int nny = y + dir[d][1] \* 2;

if (nnx<0 || nny<0 || nnx>=N || nny>=M) continue;

if (table[nx][ny] == 'o' && table[nnx][nny] == '.') {

table[x][y] = '.'; table[nx][ny] = '.'; table[nnx][nny] = 'o';

pins[i][0] = nnx; pins[i][1] = nny;

DFS(cnt + 1);

pins[i][0] = x; pins[i][1] = y;

table[x][y] = 'o'; table[nx][ny] = 'o'; table[nnx][nny] = '.';

}

}

}

}

int main() {

ios\_base::sync\_with\_stdio(false);

cin.tie(NULL); cout.tie(NULL);

int T;

cin >> T;

for (int t = 0; t < T; t++) {

num\_pin = 0;

pins.clear();

pins.shrink\_to\_fit();

//get input

table.assign(N, vector<char>(M, 0));

for (int i = 0; i < N; i++) {

for (int j = 0; j < M; j++) {

cin >> table[i][j];

if (table[i][j] == 'o') {

num\_pin += 1;

pins.emplace\_back(vector<int>{i, j});

}

}

}

pin\_size = pins.size();

answer = 0;

//algorithm part

DFS(0);

cout << pin\_size - answer <<" "<< answer << endl;

}

}

[246. SAMSUNG SW – Key]

- WOW, it was a quite difficult traveling problem. Since I had to renew keys I had currently.

- I used BFS and if I got a new key, I push the key to key-table(map hash table).

- After BFS, if key-table’s size was increased, I re-tried the BFS with new keys.

- the code was quite dirty, because I made it just adding anything that I needed at the time.

- Morever, there was hidden edge conditions. To catch those things, I spent much time.

- I spent 49 minutes and 17 seconds.

- see the code.

#include<iostream>

#include<vector>

#include<map>

#include<queue>

#define max(a,b) a>b ? a:b

using namespace std;

const int dir[4][2] = { {-1,0},{0,-1},{1,0},{0,1} };

int N, M;

vector<vector<char>> table;

map<char, bool> keys;

vector<pair<int, int>> possible\_entrance;

int answer = 0;

vector<vector<bool>> visit\_key;

void BFS(int sx, int sy, vector<pair<char, pair<int, int>>> given\_key) {

queue<vector<int>> que;

vector<vector<bool>> visit(N, vector<bool>(M, false));

que.push(vector<int>{sx, sy});

visit[sx][sy] = true;

vector<pair<char, pair<int, int>>> new\_keys = given\_key;

int n\_key\_size = new\_keys.size();

int m\_cnt=0;

while (!que.empty()) {

int x = que.front()[0]; int y = que.front()[1];que.pop();

if (table[x][y] == '$' && visit\_key[x][y] == false) {

visit\_key[x][y] = true;

answer += 1;

}

for (int d = 0; d < 4; d++) {

int nx = x + dir[d][0]; int ny = y + dir[d][1];

if (nx < 0 || ny < 0 || nx >= N || ny >= M || table[nx][ny] == '\*' || visit[nx][ny] == true) continue;

if (('A' <= table[nx][ny] && table[nx][ny] <= 'Z') && keys.find(tolower(table[nx][ny])) == keys.end()) continue;

else if (('A' <= table[nx][ny] && table[nx][ny] <= 'Z') && keys.find(tolower(table[nx][ny])) != keys.end()) {

que.push(vector<int>{nx, ny});

visit[nx][ny] = true;

}

else if (('a' <= table[nx][ny] && table[nx][ny] <= 'z')) {

if (keys.find(table[nx][ny]) == keys.end()) {

new\_keys.emplace\_back(make\_pair(table[nx][ny], make\_pair(nx, ny)));

keys[tolower(table[nx][ny])] = true;

}

que.push(vector<int>{nx, ny});

visit[nx][ny] = true;

}

else {

que.push(vector<int>{nx, ny});

visit[nx][ny] = true;

}

}

}

if (n\_key\_size != new\_keys.size()) BFS(new\_keys[n\_key\_size].second.first, new\_keys[n\_key\_size].second.second, new\_keys);

return;

}

vector<pair<int, int>> Entrance() {

vector<pair<int, int>> possible\_entrance;

for (int i = 0; i < N; i++) {

if (table[i][0] == '.' || table[i][0] == '$' || keys.find(tolower(table[i][0])) != keys.end()) possible\_entrance.emplace\_back(make\_pair(i, 0));

if (table[i][M - 1] == '.' || table[i][M - 1] == '$' || keys.find(tolower(table[i][M - 1])) != keys.end())

possible\_entrance.emplace\_back(make\_pair(i, M - 1));

}

for (int j = 0; j < M; j++) {

if (table[0][j] == '.' || table[0][j] == '$' || keys.find(tolower(table[0][j])) != keys.end()) possible\_entrance.emplace\_back(make\_pair(0, j));

if (table[N - 1][j] == '.' || table[N - 1][j] == '$' || keys.find(tolower(table[N - 1][j])) != keys.end()) possible\_entrance.emplace\_back(make\_pair(N - 1, j));

}

return possible\_entrance;

}

int main() {

ios\_base::sync\_with\_stdio(false);

cin.tie(NULL); cout.tie(NULL);

int T;

cin >> T;

for (int t = 0; t < T; t++) {

answer = 0;

keys.clear();

possible\_entrance.clear();

//get input

cin >> N >> M;

table.assign(N, vector<char>(M, 0));

visit\_key.assign(N, vector<bool>(M, false));

for (int i = 0; i < N; i++) {

for (int j = 0; j < M; j++) {

cin >> table[i][j];

}

}

string input\_key;

cin >> input\_key;

if(input\_key[0]!='0') for (int i = 0; i < input\_key.size(); i++) keys[input\_key[i]] = true;

possible\_entrance = Entrance();

int n\_key\_size = keys.size();

//algorithm part

do {

n\_key\_size = keys.size();

for (int ent = 0; ent < possible\_entrance.size(); ent++)

BFS(possible\_entrance[ent].first, possible\_entrance[ent].second, vector<pair<char, pair<int, int>>>{});

possible\_entrance = Entrance();

} while (n\_key\_size != keys.size());

cout << answer << endl;

}

return 0;

}

[247. SAMSUNG SW – SangGeun’s Traveling]

- I thought it was a DFS problem. But it was not.

- there was a trick that if we can use plains that we already used, we can travel all the city just by using N-1 plain.

- I spent 8 minutes and 40 seconds.

- see the code.

#include<iostream>

#include<map>

using namespace std;

int T, N, M;

int main() {

ios\_base::sync\_with\_stdio(false);

cin.tie(NULL); cout.tie(NULL);

cin >> T;

for (int t = 0; t < T; t++) {

//get input

cin >> N >> M;

for (int i = 0; i < M; i++) {

int s, d;

cin >> s >> d;

}

//algorithm part

cout << N - 1<<endl;

}

return 0;

}