Dynamic Programming

1.Bitmask+Backtracking

#include <bits/stdc++.h>

using namespace std;

#define MAX 12

int allPoss[MAX+1];

bool col[MAX+5];

bool diaD[40+5];

bool diaU[40+5];

int counter;

vector<int>V[105];

vector<pair<int,int> > pos;

int moveChecker(int x1, int y1, int x2, int y2) {

if(x1 == x2 && y1 == y2) return 0;

if(x1 == x2) return 1;

if(y1 == y2) return 1;

if((x1+y1) == (x2+y2)) return 1;

if((y1-x1) == (y2-x2)) return 1;

else return 2;

}

void nqueenCombinations(int idx){

/\*cout<<idx<<endl;

printf("Combinations\n");

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

printf("%d %d\n",i,allPoss[i]);

}\*/

if(idx==9) {

counter++;

for(int i=1;i<=8;i++) {

V[counter].push\_back(allPoss[i]-1);

}

return;

}

for(int i=1;i<=8;i++) {

//printf("idx = %d i = %d\n",idx,i);

if(col[i] == false && diaD[idx-i] == false && diaU[idx+i] == false) {

col[i]=true;

diaD[idx-i]=true;

diaU[idx+i]=true;

allPoss[idx]=i;

nqueenCombinations(idx+1);

col[i]=false;

diaD[idx-i]=false;

diaU[idx+i]=false;

//printf("came back in %d\n",idx);

}

}

return;

}

int check(int bitmask, int i) {

return bitmask & (1<<i);

}

int \_set(int bitmask,int i) {

return bitmask | (1<<i);

}

int reset(int bitmask,int i){

return bitmask && ~(1<<i);

}

int dp[10][105][1<<9];

int highValue=1<<8;

int func(int idx, int num, int bitmask) {

//the highest range

if(bitmask == (highValue-1)) {

//cout<<"dhuke"<<endl;

return 0;

}

if(dp[idx][num][bitmask] != -1) return dp[idx][num][bitmask];

int ans=10000000;

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

if(check(bitmask,i) == 0) {

int v = moveChecker(pos[idx].first,pos[idx].second,i,V[num][i]);

ans = min(ans,v+func(idx+1,num,\_set(bitmask,i)));

}

}

dp[idx][num][bitmask]=ans;

//cout<<"ans"<<ans<<endl;

return ans;

}

int main(void)

{

counter=0;

for(int i=0;i<=9;i++) col[i]=false;

for(int i=0;i<44;i++) diaD[i]=false,diaU[i]=false;

nqueenCombinations(1);

int T,t;

//freopen("in.txt","r",stdin);

//freopen("out.txt","w",stdout);

scanf("%d",&T);

for(t=1;t<=T;t++) {

char arr[10][10];

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

scanf("%s",arr[i]);

//cout<<arr[i]<<endl;

}

pos.clear();

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

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

if(arr[i][j] == 'q') {

pair<int,int>p;

p.first=i;

p.second=j;

pos.push\_back(p);

}

}

}

memset(dp,-1,sizeof(dp));

int ans=1000000;

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

int v = func(0,i,0);

ans=min(ans,v);

}

printf("Case %d: %d\n",t,ans);

}

return 0;

}

2. LCS DP

int dp[MAX+5][MAX+5];

string a,b;

int lcs(int i,int j){

if(i>=(int)a.size() || j>=(int)b.size()){

return 0;

}

if(dp[i][j] != -1) {

return dp[i][j];

}

if(a[i]==b[j]){

dp[i][j]=1+lcs(i+1,j+1);

return dp[i][j];

}

else{

dp[i][j]=max(lcs(i+1,j),lcs(i,j+1));

return dp[i][j];

}

}

3. LIS DP

int dp[MAX+5][MAX+5];

vector<int>a;

int lis(int i,int j){

if(j>=a.size()) {

return 0;

}

if(dp[i][j] != -1) {

return dp[i][j];

}

if(a[i]<a[j]){

dp[i][j]= max(1+lis(j,j+1),lis(i,j+1));

return dp[i][j];

}

else{

dp[i][j]=max(dp[i][j],lis(i,j+1));

return dp[i][j];

}

}

4. Edit Distance

string a,b;

int edit\_distance(int i,int j){

if(i>=a.size() && j>=b.size()) return 0;

if(i>=a.size()) {

return (b.size()-j);

}

if(j>=b.size()){

return (a.size()-i);

}

if(dp[i][j] != -1) return dp[i][j];

if(a[i]==b[j]){

dp[i][j]=edit\_distance(i+1,j+1);

return dp[i][j];

}

else{

dp[i][j]=min(1+edit\_distance(i,j+1),min(1+edit\_distance(i+1,j),1+edit\_distance(i+1,j+1)));

return dp[i][j];

}

return dp[i][j];

}

5. Minimum Partition Sum Difference

int dp[MAX\_INDEX+1][MAX\_SUM+1];

vector<int>V;

int sum\_total;

int minimum\_partion(int idx,int sum){

if(idx>=V.size()){

//cout<<sum<<endl;

if(sum != sum\_total) {

return (sum\_total-sum);

}

else{

return 100000000;

}

}

if(dp[idx][sum] != -1) return dp[idx][sum];

dp[idx][sum]=min(minimum\_partion(idx+1,sum+V[idx]),minimum\_partion(idx+1,sum));

return dp[idx][sum];

}

6. Ways to cover distance

int dp[MAX+1];

int cover\_distance(int n){

if(n<0) return 0;

if(n==0) return 1;

if(dp[n] != -1) return dp[n];

dp[n]=cover\_distance(n-1)+cover\_distance(n-2)+cover\_distance(n-3);

return dp[n];

}

**7) Counting Tilings Technique**

[Solution Based on Problem Uva 11270]

#include <bits/stdc++.h>

using namespace std;

#define MAX 11

struct Info{

int r; //row

vector<int>V; //combinations

bool operator < (const Info &a) const {

if(r<a.r) {

return true;

}

if(r>a.r){

return false;

}

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

if(V[i]<a.V[i]) return true;

if(V[i]>a.V[i]) return false;

}

return false;

}

};

map<Info,int>M,visit;

int m,n;

vector<vector<int> >global\_vector;

void combination\_generate(int row,vector<int>prev\_comb,vector<int>notun\_comb){

if((int)notun\_comb.size()==n){

//column complete

global\_vector.push\_back(notun\_comb);

return;

}

else{

//try combination

if(row==1){

//nothing's been before

if(m>1){

notun\_comb.push\_back(1);

combination\_generate(row,prev\_comb,notun\_comb);

notun\_comb.pop\_back();

}

if((int)(n-notun\_comb.size())>=2){

notun\_comb.push\_back(3);

notun\_comb.push\_back(4);

combination\_generate(row,prev\_comb,notun\_comb);

notun\_comb.pop\_back();

notun\_comb.pop\_back();

}

}

else{

//somthing can be there

int pos=notun\_comb.size();

if(prev\_comb[pos]==1){

//must use 2

notun\_comb.push\_back(2);

combination\_generate(row,prev\_comb,notun\_comb);

notun\_comb.pop\_back();

}

if(prev\_comb[pos]==2){

//first try with 1

if(row<m){

notun\_comb.push\_back(1);

combination\_generate(row,prev\_comb,notun\_comb);

notun\_comb.pop\_back();

}

//next try with 3

if(pos<n && (prev\_comb[pos+1]==2 || prev\_comb[pos+1] == 3)){

notun\_comb.push\_back(3);

notun\_comb.push\_back(4);

combination\_generate(row,prev\_comb,notun\_comb);

notun\_comb.pop\_back();

notun\_comb.pop\_back();

}

}

if(prev\_comb[pos]==3){

//first try with 1

if(row<m){

notun\_comb.push\_back(1);

combination\_generate(row,prev\_comb,notun\_comb);

notun\_comb.pop\_back();

}

//next try with 3

notun\_comb.push\_back(3);

notun\_comb.push\_back(4);

combination\_generate(row,prev\_comb,notun\_comb);

notun\_comb.pop\_back();

notun\_comb.pop\_back();

}

if(prev\_comb[pos]==4){

//first try with 1

if(row<m){

notun\_comb.push\_back(1);

combination\_generate(row,prev\_comb,notun\_comb);

notun\_comb.pop\_back();

}

//next try with 3

if(pos<n && (prev\_comb[pos+1]==3 || prev\_comb[pos+1]==2)){

notun\_comb.push\_back(3);

notun\_comb.push\_back(4);

combination\_generate(row,prev\_comb,notun\_comb);

notun\_comb.pop\_back();

notun\_comb.pop\_back();

}

}

}

}

return;

}

int counting\_tiles(int row,vector<int>V){

if(row==1){

global\_vector.clear();

vector<int>V;

vector<int>prev;

combination\_generate(row,prev,V);

vector<vector<int>>temp;

for(int i=0;i<(int)global\_vector.size();i++){

temp.push\_back(global\_vector[i]);

}

int res=0;

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

/\*cout<<"shuru kortesi " << endl;

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

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

}

cout<<endl;\*/

int value=counting\_tiles(row+1,temp[i]);

res=res+value;

}

return res;

}

else{

/\*cout<<"row = "<<row<<endl;

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

cout<<V[i]<<" ";

}

cout<<endl;\*/

Info obj;

obj.r=row;

obj.V=V;

if(visit[obj]==1) {

//cout<<"already calculated "<<endl;

/\*cout<<"YES row "<<row<<endl;

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

cout<<V[i]<<" ";

}

cout<<endl;\*/

return M[obj];

}

global\_vector.clear();

vector<int>notun\_comb;

vector<vector<int>>temp;

combination\_generate(row,V,notun\_comb);

for(int i=0;i<(int)global\_vector.size();i++){

temp.push\_back(global\_vector[i]);

}

if(row==m){

/\*cout<<"printing "<<row-1<<" th line " << endl;

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

cout<<V[i]<<" ";

}

cout<<endl;\*/

/\*cout<<" while " << m<< " size = "<<temp.size()<<endl;

cout<<"printing others "<<endl;

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

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

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

}

cout<<endl;

}\*/

Info save;

save.r=row;

save.V=V;

M[save]=temp.size();

return M[save];

}

else{

Info save;

save.r=row;

save.V=V;

int res=0;

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

res=res+counting\_tiles(row+1,temp[i]);

}

M[save]=res;

visit[save]=1;

return M[save];

}

}

}

int main(void){

int T,t;

scanf("%d",&T);

for(t=1;t<=T;t++){

scanf("%d %d",&m,&n);

if(n>m) swap(m,n);

vector<int>V;

int res = counting\_tiles(1,V);

cout<<res<<endl;

M.clear();

visit.clear();

global\_vector.clear();

}

}

**8) LIS Binary Search**

int longest\_increasing\_subsequence\_bs(vector<int>V){

vector<int>sorted;

//vector<int>first\_pos;

//vector<int>second\_pos;

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

if(sorted.size()==0){

sorted.push\_back(V[i]);

//first\_pos.push\_back(i);

//second\_pos.push\_back(i);

}

else{

vector<int>::iterator it=lower\_bound(sorted.begin(),sorted.end(),V[i]);

int p=it-sorted.begin();

if(sorted[p]>=V[i]){

sorted[p]=V[i];

//first\_pos[p]=second\_pos[p];

//second\_pos[p]=i;

}

else if(sorted[p]<V[i]){

sorted.push\_back(V[i]);

//first\_pos.push\_back(i);

//second\_pos.push\_back(i);

}

}

}

/\*for(int i=0;i<(int)first\_pos.size();i++){

cout<<V[first\_pos[i]]<<" ";

}

cout<<endl;\*/

return sorted.size();

}

**9) LIS Segment Tree with path printing**

//LIS segment tree

struct segment\_tree{

int val;

int best;

};

segment\_tree tree[4\*MAX+5];

int prev\_idx[MAX+5];

void build\_segment\_tree(){

memset(tree,0,sizeof(tree));

memset(prev\_idx,-1,sizeof(prev\_idx));

}

void update(int st, int en, int node, int x,int value,int past\_index){

if(st==en){

tree[node].val=value;

tree[node].best=st;

prev\_idx[st]=past\_index;

return;

}

int mid=(st+en)/2;

if(x>mid){

update(mid+1,en,2\*node+1,x,value,past\_index);

}

else{

update(st,mid,2\*node,x,value,past\_index);

}

tree[node].val=max(tree[2\*node].val,tree[2\*node+1].val);

if(tree[node].val == tree[2\*node].val){

tree[node].best=tree[2\*node].best;

}

else{

tree[node].best=tree[2\*node+1].best;

}

return;

}

pair<int,int> query(int st, int en,int node,int x, int y){

if(st>=x && en<=y) {

pair<int,int>res;

res.first=tree[node].val;

res.second=tree[node].best;

return res;

}

if(x>en||y<st) return {0,0};

int mid=(st+en)/2;

pair<int,int> res1=query(st,mid,2\*node,x,y);

pair<int,int> res2=query(mid+1,en,2\*node+1,x,y);

if(res1.first>=res2.first) {

return res1;

}

else{

return res2;

}

}

int string\_to\_num(string s){

int res=0;

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

res=res\*10+(s[i]-'0');

}

return res;

}

vector<int> printing\_path(vector<int>V,int val){

if(val == -1) {

return V;

}

V.push\_back(val);

return printing\_path(V,prev\_idx[val]);

}

int main(void){

//freopen("in.txt","r",stdin);

int T,t;

scanf("%d",&T);

char ch=getchar();

ch=getchar();

for(t=1;t<=T;t++){

vector<int>V;

build\_segment\_tree();

int res;

string s;

vector<int>special;

while (getline(cin, s) && s != "")

{

stringstream ss(s);

int res;

ss >> res;

res += 2;

V.push\_back(res);

special.push\_back(res);

}

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

int max\_val= -1;

int cnt=0;

map<int,int>M,Mrev;

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

if(M[special[i]] == 0){

if(cnt==0) cnt=2;

else cnt++;

M[special[i]]=cnt;

Mrev[cnt]=special[i];

max\_val=max(max\_val,M[special[i]]);

}

}

int ans = 1;

pair<int,int>save;

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

//cout<<V[i]<<" "<<endl;

if(i==0) {

update(1,max\_val,1,M[V[i]],1,-1);

ans=1;

save={1,M[V[i]]};

}

else{

pair<int,int>temp=query(1,max\_val,1,1,M[V[i]]-1);

if(temp.first==0) {

temp.second=-1;

}

update(1,max\_val,1,M[V[i]],temp.first+1,temp.second);

if(ans<(temp.first+1)){

ans=temp.first+1;

save.first=temp.first+1;

save.second=M[V[i]];

}

else if(ans == (temp.first+1)){

if(save.second>M[V[i]]){

save.second=M[V[i]];

}

}

}

}

printf("Max hits: %d\n",ans);

vector<int>notun;

notun=printing\_path(notun,save.second);

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

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

cout<<Mrev[notun[i]]-2<<endl;

}

if(t<T){

cout<<endl;

}

}

return 0;

}