A

采用递推的方法，由于要到达棋盘上的一个点，只能从左边或者上边过来，根据加法原则，到达某一点的路径数目，就等于到达其相邻的上点和左点的路径数目的总和。所有海盗能达到的点将其路径数置为0即可。

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

int main()

{

int i,j,x,y,n,m,f[100][100];

long long ans[100][100];

int t;

scanf("%d", &t);

while (t--) {

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

memset(f,1,sizeof(f));

memset(ans,0,sizeof(ans));

ans[0][0]=1;

f[x][y]=0,f[x+1][y+2]=0,f[x-1][y+2]=0;

f[x+1][y-2]=0,f[x-1][y-2]=0,f[x+2][y+1]=0;

f[x-2][y+1]=0,f[x+2][y-1]=0,f[x-2][y-1]=0;

for (i=1; i<=n; i++)

if (f[i][0])

ans[i][0]=1;

else break;

for (i=1; i<=m; i++)

if (f[0][i])

ans[0][i]=1;

else break;

for (i=1; i<=n; i++)

for (j=1; j<=m; j++)

if (f[i][j])

ans[i][j]=ans[i-1][j]+ans[i][j-1];

printf("%lld\n",ans[n][m]);

}

return 0;

}

B

对于任意点K，其与点1之间的最短路有两种情况：

1. 直接从点1走到K；
2. 从1走到点A，通过A直接跳到点B，再从B走到K。

或者说，这条最短路等于min（直接从1走到K的距离，从1走到点A通过A跳到B在从B走到K的距离）。

设点I和J之间的直线距离为DIS[I,J]，则：

在第（1）种情况下，最短路长度为DIS[1,K]；

在第（2）种情况下，最短路长度为DIS[1,A] + DIS[B,K];

由于DIS[1,A]是一个常数（因为A固定），而与K有关的只有B，应直接选择A=1使DIS[1,1] = 0.（也就是说传送门的第一个点一定要建在1点上）。

对当前枚举的第二个传送点位置B,必有唯一一个点C具有如下性质：

DIS[1,C] <= DIS[C,K] 且 DIS[1,C+1]　＞ DIS[C,K]

此时距离1最长的点为C，C+1或N中的一个。

留意到随着B的递增，C是不递减的，所以在O（n）枚举B的同时只需要O（1）就可以找到Ｃ。

#include <cstdio>

#include <cstring>

#include <cstdlib>

#include <algorithm>

using namespace std;

const int maxn = 1100000;

int i, j, r, n, m, ansi, ansj, ansr;

long long ans;

long long p[maxn];

inline void make() {

long long temp;

ans = p[n];

i = j = 1;

while (i <= n) {

while (j < i && p[j] <= p[i] - p[j])

j++;

temp = max(p[i] - p[j], p[j-1]);

temp = max(p[n] - p[i], temp);

if (temp < ans) {

ans = temp;

ansi = i;

ansj = j;

}

i++;

}

}

int main()

{

int T;

scanf("%d", &T);

while (T--) {

memset(p, 0, sizeof(p));

scanf("%d", &n);

for (i = 2; i <= n; i++) {

scanf("%lld", &p[i]);

p[i] += p[i-1];

}

make();

printf("%lld\n", ans);

}

return 0;

}

C

首先需要判断行和列的总和是否相等，因为它们都应该是整个矩阵的所有元素之和。如果他们不相等则这个矩阵肯定不存在。

这道题的贪心策略是，把列和从大到小枚举，对每个列和，按行和从大到小的顺序进行选择填上1，然后该行的行和减去1.这种贪心策略之所以有效，是因为这种策略会使各行的行和趋向于平均，尽可能地使行和减为零的情况推迟发生，而行和减为零意味着，当前可选的行数减少1，因此后面的计算可选择方法肯定比这种贪心的策略要少。

#include <stdio.h>

#include <algorithm>

using namespace std;

const int size=100000; //最大行列数

int a[size],b[size]; //分别保存行和与列和

int main(){

int r,c,i,j;

long long s,t; //枚举时比较的行和与列和总数

while(scanf("%d%d",&r,&c)==2){//输入整数r,c直到文件结束

for(i=0,s=0; i<r; i++){

scanf("%d",&a[i]); //输入行和

s+=a[i]; //累加行和

}

for(i=0,t=0; i<c; i++){

scanf("%d",&b[i]); //输入列和

t+=b[i]; //累加列和

}

if(s!=t){ //如果行和与列和总数不相等

printf("NO\n"); //则不可能有解

continue;

}

sort(a,a+r); //行和排序

sort(b,b+c); //列和排序

for(i=j=0,t=s=0; i<c; i++){//从大到小枚举列和

t+=b[c-i-1]; //当前已枚举的列和总数

s+=r-j; //当前可用的行和总数

while(j<r&&a[j]<i+1){ //如果某行和小于枚举列数

s-=i+1-a[j]; //把行和总数多算出来的部分减去

j++;

}

if(s<t) break; //如果可用行和小于当前列和则不可能有解

}

printf(i==c?"YES\n":"NO\n");//输出答案

}

return 0;

}

D

因为要求出N的因子个数，我们从素数开始讨论。N=1时只有一个因子1，对于任意一个质数p，只有1和p两个因子，所以 ,而对于一个质数的幂，它的因子分别是一共k个，因子的因子数分别是1,2,3,...,k+1个，因此：=.

如果N有两个不同的素因子p1和p2，这时不妨设N=,这时可以把N的因子按照和的最大公约数来分成类，显然每一类的数目都是一样的。注意到一个事实,我们很容易得到：

=

=

采用类似的方法，对N的因子使用数学归纳法，可以证明对任意

#include <stdio.h>

#include <algorithm>

using namespace std;

const int size=100000; //最大行列数

int a[size],b[size]; //分别保存行和与列和

int main(){

int r,c,i,j;

long long s,t; //枚举时比较的行和与列和总数

while(scanf("%d%d",&r,&c)==2){//输入整数r,c直到文件结束

for(i=0,s=0; i<r; i++){

scanf("%d",&a[i]); //输入行和

s+=a[i]; //累加行和

}

for(i=0,t=0; i<c; i++){

scanf("%d",&b[i]); //输入列和

t+=b[i]; //累加列和

}

if(s!=t){ //如果行和与列和总数不相等

printf("NO\n"); //则不可能有解

continue;

}

sort(a,a+r); //行和排序

sort(b,b+c); //列和排序

for(i=j=0,t=s=0; i<c; i++){//从大到小枚举列和

t+=b[c-i-1]; //当前已枚举的列和总数

s+=r-j; //当前可用的行和总数

while(j<r&&a[j]<i+1){ //如果某行和小于枚举列数

s-=i+1-a[j]; //把行和总数多算出来的部分减去

j++;

}

if(s<t) break; //如果可用行和小于当前列和则不可能有解

}

printf(i==c?"YES\n":"NO\n");//输出答案

}

return 0;

}

E

将每个排列利用康托展开压缩为一个整数，采用广度优先搜索的方式不停的搜索直到得到目标状态即可。

#include <stdio.h>

#include <string.h>

const int MAXNODE=362880; //最大状态数

struct State {

char d[9]; //状态中的9个字符

short f; //这个状态到达目标状态的最少操作数

};

//pow[i]表示(i+1)!

int pow[]={1,2,6,24,120,720,5040,40320};

//4种不同的操作的逆操作的置换顺序

int rot[4][9]={{1,4,2,0,3,5,6,7,8},{0,2,5,3,1,4,6,7,8},{0,1,2,4,7,5,3,6,8},{0,1,2,3,5,8,6,4,7}};

short ans[MAXNODE]; //到达每个状态的结果

int head,tail; //广度优先搜索中使用的队列头与尾

State Q[MAXNODE]; //广度优先搜索中使用的队列

//用康托展开把一个状态变换成一个整数

int State2I(State &p)

{

int ret=0;

for(int i=0;i<8;i++) //使用康托展开的公式

for(int j=i+1;j<9;j++) if(p.d[i]>p.d[j]) ret+=pow[7-i];

return ret;

}

//预处理所有状态到达目标状态的最少操作数

void PreCom()

{

memset(ans,255,sizeof(ans)); //清空数组为-1

head=-1,tail=0; //初始化队列头尾

Q[0].f=0; //初始状态操作数为0

for(int i=0;i<9;i++) Q[0].d[i]=i+1;//初始化初始状态

ans[State2I(Q[0])]=0; //初始状态的答案为0

while(head++<tail) { //运算直到队列为空

State &p=Q[head],q;

q.f=p.f+1; //经过一次操作

for(int i=0;i<4;i++) { //枚举4种不同的操作

//按操作的置换顺序变换

for(int j=0;j<9;j++) q.d[j]=p.d[rot[i][j]];

int u=State2I(q); //得到新状态的康托展开值

if(ans[u]<0) { //这个状态并未被扩展

ans[u]=q.f; //更新状态答案

Q[++tail]=q; //新状态放到队列末端

}

}

}

}

//处理输入和输出

void Work()

{

State p;

int x;

while(scanf("%d",&x)==1) { //输入整数x直到文件结束

p.d[0]=x;

for(int i=1;i<9;i++) {

scanf("%d",&x); //共输入9个数字

p.d[i]=x;

}

printf("%d\n",ans[State2I(p)]);//直接查表得到结果并输出

}

}

int main()

{

PreCom(); //预处理

Work(); //处理输入输出

return 0;

}

F

首先把问题简化，给定m和n，求出以[0,m]X[0,n]为包含这个零星的最小矩形的菱形的个数。在这个问题简化方面，只有两种情况，如下所示：

n

n

y

m

m

x

（1）菱形的四个点分别在对角线的四边上，或者（2）菱形的其中一对对角线定点分点在矩形的一对对角线顶点上。

因此我们只需要求出这两个情况下菱形的个数即可。

在情况1中，设变量x，y，有菱形的四边长度相等，得到：

同理在情况2中，可以得到：

上面的两个方程，都要求x，y为整数，且0<=x<=m, 0<=y<=n，这两个方程都可以看做是求解ax + by = c，采用扩展欧几里得算法便可求解，注意当4个点都在矩形上且有两个点在矩形顶点上会出现重复计算，菱形面积为0的点也应该去掉。这样先预处理出所有可能的矩形中菱形的个数，然后离线存储答案。

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

const int maxn=1001; //矩形最大边长

int f[maxn][maxn]; //以[0,m]×[0,n]为包含菱形的最小矩形的菱形的个数

long long g[maxn][maxn];//在[0,m]×[0,n]以m,n为边界的菱形的个数

long long h[maxn][maxn];//在[0,m]×[0,n]中菱形的个数

//扩展欧基里德算法，返回结果为最大公约数d，且ax+by=d

int egcd(int a,int b,long long &x,long long &y)

{

long long k; //临时变量

int d; //最大公约数

if (b==0) //终止条件

{

x=1; //满足终止条件时x的值

y=0; //满足终止条件时y的值

return a; //最大公约数为a

}

else

{

d=egcd(b,a%b,x,y); //递归求解

k=a/b;

k=x-k\*y; //临时变量用于交换两个数

x=y; //扩展欧基里德算法中，从上一层得到x=y

y=k; //扩展欧基里德算法中，从上一层得到y=x-(a/b)\*y

return d; //最大公约数为递归求解结果

}

}

//计算区间的上下界

void cal\_bound(long long x,long long step,long long &l,long long &r,int lb,int rb)

{

int temp;

if (step<0) //当步长为负数时，进行镜像调整使得步长为正

{

x=-x; //x取相反数

step=-step; //步长取相反数

temp=lb;

lb=-rb;

rb=-temp; //把左右边界取相反数并且交换

}

//求最小的l使x+l\*step>=lb

if (lb-x>=0) //左边界在已知解的右边

l=(lb-x+step-1)/step;

else //左边界在已知解的左边

l=(lb-x)/step;

//求最大的r使x+r\*step<=rb

if (rb-x>=0) //右边界在已知解的右边

r=(rb-x)/step;

else //右边界在已知解的左边

r=(rb-x-step+1)/step;

return;

}

//求ax+by=c在lx<=x<=rx且ly<=y<=ry时整数解的个数

int cal(int a,int b,int c,int lx,int rx,int ly,int ry)

{

long long x,y,dx,dy,l1,r1,l2,r2;

int d;

d=egcd(abs(a),abs(b),x,y); //使用扩展欧基里德算法

if (c%d!=0) //不存在解的情况

return 0;

if (a<0) //如果a为负数，则相应调整x

x=-x;

if (b<0) //如果b为负数，则相应调整y

y=-y;

x\*=c/d; //求出其中一个解的x值

y\*=c/d; //求出其中一个解的y值

dx=b/d; //x的变化步长

dy=-a/d; //y的变化步长

cal\_bound(x,dx,l1,r1,lx,rx);//通过x求t的左右边界

cal\_bound(y,dy,l2,r2,ly,ry);//通过y求t的左右边界

if (l1<l2) //取左边界的最大值

l1=l2;

if (r1>r2) //取右边界的最小值

r1=r2;

return r1-l1+1; //返回解的个数

}

int init() //预处理函数

{

int i,j,temp;

for(i=1;i<maxn;i++){ //枚举矩形的其中一边长

for(j=1;j<maxn;j++){ //枚举矩形的另一边长

temp=cal(2\*i,-2\*j,i\*i-j\*j,0,i,0,j);//计算情况(1)的结果

if (i==j) //当矩形为正方形时，正方形重复计算了一次

temp--;

if (temp>0)

f[i][j]+=temp; //将合法解累加到f数组中

temp=cal(2\*i,2\*j,i\*i+j\*j,1,i-1,1,j-1);//计算情况(2)的结果

if (i%2==0&&j%2==0) //减去菱形面积为0的情况

temp--;

if (temp>0)

f[i][j]+=temp; //将合法解累加到f数组中

//从f数组到g数组的转移方程

g[i][j]=g[i-1][j]+g[i][j-1]-g[i-1][j-1]+f[i][j];

//从g数组到h数组的转移方程

h[i][j]=h[i-1][j]+h[i][j-1]-h[i-1][j-1]+g[i][j];

}

}

return 0;

}

int main()

{

int m,n;

init(); //预处理

while(scanf("%d%d",&m,&n)!=EOF){ //输入整数m,n直到文件结束

printf("%I64d\n",h[m][n]); //输出答案

}

return 0;

}

G

利用指针建立二叉树，再进行后续遍历。

直接构造一棵二叉树即可，可以用最后一层节点来保存2^N个值，则他们的父亲结点的字符值就已经由左右儿子的B,I决定了，故不用保存串，只需要记录字符值。

#include <iostream>

#include <cstring>

#include <cstdio>

#include <cmath>

using namespace std;

char s1[2]="0",s2[2]="1";

char s[1200];

struct FBI

{

char s;

FBI \*lchild,\*rchild;

};

void showtree(FBI \*head)//后序遍历树

{

if(head==NULL)

return;

showtree(head->lchild);

showtree(head->rchild);

printf("%c",head->s);

}

void maketree(FBI \*node,char \*p,int len)

{

if(len==1)

{

if(strstr(p,s1)&&strstr(p,s2))

node->s='F';

else if(strstr(p,s1)&&!strstr(p,s2))

node->s='B';

else if(!strstr(p,s1)&&strstr(p,s2))

node->s='I';

node->lchild=NULL;

node->rchild=NULL;

return;

}

char q[520],\*r=new char;

FBI \*z=new FBI;

strncpy(q,p,len/2);

r=p;

int i=len/2;

while(i--)

r++; //将p一分为二 q和r两个字符串

if(strstr(q,s1)&&strstr(q,s2)) //判断左儿子的类型

z->s='F';

else if(strstr(q,s1)&&!strstr(q,s2))

z->s='B';

else if(!strstr(q,s1)&&strstr(q,s2))

z->s='I';

node->lchild=z;

FBI \*c=new FBI;

if(strstr(r,s1)&&strstr(r,s2)) //判断右儿子的类型

c->s='F';

else if(strstr(r,s1)&&!strstr(r,s2))

c->s='B';

else if(!strstr(r,s1)&&strstr(r,s2))

c->s='I';

node->rchild=c;

maketree(z,q,len/2); //递归构建

maketree(c,r,len/2);

}

int main()

{

int T;

scanf("%d", &T);

while(T--)

{

int n;

scanf("%d", &n);

scanf("%s",&s);

if(strlen(s)==1)

{

if(!strcmp(s,"0"))

printf("B\n");

else if(!strcmp(s,"1"))

printf("I\n");

continue;

}

FBI \*head=new FBI;

char s1[2]="0",s2[2]="1";

if(strstr(s,s1)&&strstr(s,s2))

head->s='F';

else if(strstr(s,s1)&&!strstr(s,s2))

head->s='B';

else if(!strstr(s,s1)&&strstr(s,s2))

head->s='I';

maketree(head,s,(int)pow(2.0,(double)n));

showtree(head);

printf("\n");

}

return 0;

}

H

每读入一片雪花，就将该雪花进行哈希操作，并判断哈希表里是否有相同的哈希值，如有相同的哈希值就从链表中一一取出并判断是否同构即可。

#include <iostream>

#include <cstdio>

#include <stdlib.h>

#include <vector>

using namespace std;

const int M = 90001; //myhash函数，取余的数

int snow[100005][6]; //存储雪花信息

vector<int> myhash[M]; //myhash表，表中存储的是snow数组的下标

bool isSame(int a, int b)//判断a与b是否同样

{

for(int i=0;i<6;i++)

{

//顺时针

if((snow[a][0] == snow[b][i] &&

snow[a][1] == snow[b][(i+1)%6] &&

snow[a][2] == snow[b][(i+2)%6] &&

snow[a][3] == snow[b][(i+3)%6] &&

snow[a][4] == snow[b][(i+4)%6] &&

snow[a][5] == snow[b][(i+5)%6])

|| //逆时针

(snow[a][0] == snow[b][i] &&

snow[a][1] == snow[b][(i+5)%6] &&

snow[a][2] == snow[b][(i+4)%6] &&

snow[a][3] == snow[b][(i+3)%6] &&

snow[a][4] == snow[b][(i+2)%6] &&

snow[a][5] == snow[b][(i+1)%6]))

return true;

}

return false;

}

int main()

{

freopen("h.out", "w", stdout);

int T;

cin >> T;

while (T--) {

int ok = 0;

int n;

int i,j;

cin>>n;

for( i = 0; i < n; i++)

for( j = 0; j < 6; j++)

cin>>snow[i][j];

int sum, key;

for(i = 0; i < n; i++)

{

sum = 0;//求出雪花六个花瓣的和

for( j = 0; j < 6; j++)

sum += snow[i][j];

key = sum % M; //求出key

//判断是否与myhash表中myhash[key]存储的雪花相同

for(vector<int>::size\_type j = 0; j < myhash[key].size(); j++)

{

if(isSame(myhash[key][j], i))//如相同

{

cout<<"Twin snowflakes found."<<endl;

ok = 1;

break;

}

}

if (ok) {

break;

}

myhash[key].push\_back(i);//若没找到相同的

}

if (ok == 0)

cout<<"No two snowflakes are alike."<<endl;

}

return 0;

}

I

签到题，直接模拟就好。

#include<fstream>

#include<cstdio>

#include<string>

#include <iostream>

using namespace std;

string s;

char a[5005];

int p;

int main()

{

int T;

scanf("%d", &T);

while (T--) {

int i,len;

cin>>s;

len=s.size();

for(i=0;i<len;++i)

{

if(s[i]=='@')

p=0;

else if(s[i]=='#' && p>0)

--p;

else if(s[i]!='#')

a[++p]=s[i];

}

for(i=1;i<=p;++i)

cout<<a[i];

cout<<'\n';

}

return 0;

}

J

本题乍看可以用树的划分等比较麻烦的方法去做，但实际上考虑到异或的特殊性质，不需要用到这些方法的方法。

首先回想我们是如何计算树上两点之间的距离的，先分别求出根到这两点之间的距离，记为l1, l2， 根到这两点的最近公共祖先的距离为lc.那么这两点距离就是（l1 ＋ l2）- (lc + lc).

然后回到原问题，我们要求的是两点之间的异或距离，同样先求出根到这两点的异或距离，记为x1, x2，根到这两点最近公共祖先距离为xc，那么这两点的异或距离就是x1⊕x2⊕xc⊕xc,所以异或距离就是x1⊕x2，那么我们只需要直到有多少点对，满足根分别到他们的异或距离异或后等于K。

于是我们把问题转换为一个很简单的问题，给出N个数字，问有多少对数字异或后等于K，枚举这些数字，然后统计枚举到的数字和K值出现的次数，加到答案里，问题就解决了。

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <assert.h>

typedef long long LL;

const int MAXN = 500007;

struct Edge {

int to, w;

Edge\* next;

};

Edge edges[MAXN \* 2], \* g[MAXN];

int nEdge;

int open[MAXN];

int a[MAXN];

bool vst[MAXN];

int hash[MAXN \* 2];

int N, K;

inline void addEdge(int x, int y, int w) {

Edge\* p = &edges[nEdge++];

p->to = y;

p->w = w;

p->next = g[x];

g[x] = p;

}

void bfs() {

int i, j, x, m = 0;

Edge\* p;

memset(vst, false, N);

open[m++] = 0;

vst[0] = true;

for (i = 0; i < m; ++i) {

x = open[i];

for (p = g[x]; p; p = p->next) {

if (!vst[p->to]) {

a[p->to] = (a[x] ^ p->w);

vst[p->to] = true;

open[m++] = p->to;

}

}

}

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

++hash[a[i]];

assert(vst[i] == true);

}

}

void input() {

int i, x, y, w;

scanf("%d%d", &N, &K);

assert(2 <= N && N <= 500000);

assert(0 <= K && K <= 500000);

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

g[i] = NULL;

}

nEdge = 0;

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

scanf("%d%d%d", &x, &y, &w);

assert(0 <= x && x < N);

assert(0 <= y && y < N);

assert(1 <= w && w <= 500000);

addEdge(x, y, w);

addEdge(y, x, w);

}

}

void solve() {

int i, j;

LL ans = 0;

bfs();

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

ans += hash[a[i] ^ K];

}

if (K == 0) ans -= N;

ans /= 2;

printf("%I64d\n", ans);

// clear the hash

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

hash[a[i]] = 0;

}

}

int main() {

int T;

scanf("%d", &T);

assert(T <= 50);

while (T--) {

input();

solve();

}

return 0;

}