目录

[字符串处理 1](#_Toc516009196)

[KMP算法 1](#_Toc516009197)

[字符串最小表示法 2](#_Toc516009198)

[字符串hash 3](#_Toc516009199)

[图论 4](#_Toc516009200)

[最大流 4](#_Toc516009201)

[最小费用最大流 8](#_Toc516009202)

[生成树计数 10](#_Toc516009203)

[数学 12](#_Toc516009204)

[Euler函数 12](#_Toc516009205)

[FFT 13](#_Toc516009206)

[莫比乌斯反演 14](#_Toc516009207)

[自适应simpson](#_Toc516009208)积分 [15](#_Toc516009208)

[随机素数测试和大数分解(POJ 1811) 15](#_Toc516009209)

[定理 17](#_Toc516009210)

[欧拉定理 17](#_Toc516009211)

[费马平方和定理 17](#_Toc516009212)

[高斯素数 17](#_Toc516009213)

[特殊数列 18](#_Toc516009214)

[第一类斯特林数 18](#_Toc516009215)

[第二类斯特林数 18](#_Toc516009216)

[卡特兰数 18](#_Toc516009217)

[博弈 18](#_Toc516009218)

[威佐夫博奕 18](#_Toc516009219)

[SG函数 19](#_Toc516009220)

[K倍动态减法 19](#_Toc516009221)

[Nim积 20](#_Toc516009222)

[其它 21](#_Toc516009223)

[头文件 21](#_Toc516009224)

[NTT 求模数 22](#_Toc516009225)

# 字符串处理

## KMP算法

/\*

\* next[]的含义：x[i-next[i]...i-1]=x[0...next[i]-1]

\* next[i]为满足x[i-z...i-1]=x[0...z-1]的最大z值（就是x的自身匹配）

\*/

void kmp\_pre (char x[],int m,int Next[]){

int i,j;

j=Next[0]=-1;

i=0;

while (i<m) {

while (-1!=j && x[i]!=x[j])j=Next[j];

Next[++i]=++j;

}

}

/\*

\* 返回x在y中出现的次数，可以重叠

\*/

int Next[10010];

int KMP\_Count (char x[],int m,char y[],int n)

{//x是模式串，y是主串

int i,j;

int ans=0;

//preKMP(x,m,Next);

kmp\_pre(x,m,Next);

i=j=0;

while (i<n)

{

while (-1!=j && y[i]!=x[j])j=Next[j];

i++;j++;

if (j>=m)

{

ans++;

j=Next[j];

}

}

return ans;

}

## 字符串最小表示法

int minsub(char \*s1){

int i = 0,j = 1,k = 0;

int len = strlen(s1);

while(i<len&&j<len&&k<len){

if(i==j) j++;

int p1 = i+k>=len?i+k-len:i+k;

int p2 = j+k>=len?j+k-len:j+k;

if(s1[p1]>s1[p2]) i += k+1,k = 0;

if(s1[p2]>s1[p1]) j += k+1,k = 0;

if(s1[p1]==s1[p2]) k++;

}

return i;

}

## 字符串hash

const int HASH = 10007;

const int MAXN = 2010;

const int SEED = 13331;

typedef unsigned long long LL;

LL P[MAXN],S[MAXN];

struct HASHMAP{

int head[HASH],next[MAXN],size;

LL state[MAXN];

int f[MAXN];

void init(){

size = 0;

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

}

int insert(LL val,int \_id){

int h = val%HASH;

for(int i = head[h]; i != -1;i = next[i])

if(val == state[i]){

int tmp = f[i];

f[i] = \_id;

return tmp;

}

f[size] = \_id;

state[size] = val;

next[size] = head[h];

head[h] = size++;

return 0;

}

}H;

void init(){

P[0] = 1;

for(int i = 1;i < MAXN;i++)

P[i] = P[i-1] \* SEED;

}

char s[2010];

int ans[2010][2010];

int main()

{

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

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

init();

int T,n;

scanf("%d",&T);

while(T--){

scanf("%s",s);

n = strlen(s);

S[0] = 0;

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

S[i] = S[i-1]\*SEED + s[i-1];

memset(ans,0,sizeof ans);

for(int L = 1;L <= n;L++){

H.init();

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

int l = H.insert(S[i+L-1]-S[i-1]\*P[L],i);

ans[i][i+L-1]++;

ans[l][i+L-1]--;

}

}

for(int i = n;i >= 0;i--)

for(int j = i;j <= n;j++)

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

int q,u,v;

scanf("%d",&q);

while(q--){

scanf("%d%d",&u,&v);

printf("%d\n",ans[u][v]);

}

}

return 0;

}

## Manacher 最长回文子串

void Manacher(char s[],int len){

int l=0;

Ma[l++]='$';

Ma[l++]='#';

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

Ma[l++]=s[i];

Ma[l++]='#';

}

Ma[l] = 0;

Mp[0] = 1;

int mx=0,id=0;

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

Mp[i]=mx>i?min(Mp[2\*id-i],mx-i):1;

while(Ma[i+Mp[i]]==Ma[i-Mp[i]])Mp[i]++;

if(i+Mp[i]>mx){

mx=i+Mp[i];

id=i;

}

}

}

# 图论

## 最短路

### Floyd

for(int k = 1;k <= n;k++)

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

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

if(floyd[i][j] > floyd[i][k] + floyd[k][j])

floyd[i][j] = floyd[i][k] + floyd[k][j];}

## 最大流

// ISAP 邻接表形式

const int MAXN = 1000100;//点数的最大值

const int MAXM = 6000100;//边数的最大值

const int INF = 0x3f3f3f3f;

struct Edge{

int to,next,cap,flow;

}edge[MAXM];//注意是MAXM

int tol;

int head[MAXN];

int gap[MAXN],dep[MAXN],pre[MAXN],cur[MAXN];

void init(){

tol = 0;

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

}

//加边，单向图三个参数，双向图四个参数

void addedge(int u,int v,int w,int rw=0){

edge[tol].to = v;edge[tol].cap = w;edge[tol].next = head[u];

edge[tol].flow = 0;head[u] = tol++;

edge[tol].to = u;edge[tol].cap = rw;edge[tol].next = head[v];

edge[tol].flow = 0;head[v]=tol++;

}

//输入参数：起点、终点、点的总数

//点的编号没有影响，只要输入点的总数

int sap(int start,int end,int N){

memset(gap,0,sizeof(gap));

memset(dep,0,sizeof(dep));

memcpy(cur,head,sizeof(head));

int u = start;

pre[u] = -1;

gap[0] = N;

int ans = 0;

while(dep[start] < N){

if(u == end){

int Min = INF;

for(int i = pre[u];i != -1; i = pre[edge[i^1].to])

if(Min > edge[i].cap - edge[i].flow)

Min = edge[i].cap - edge[i].flow;

for(int i = pre[u];i != -1; i = pre[edge[i^1].to]){

edge[i].flow += Min;

edge[i^1].flow -= Min;

}

u = start;

ans += Min;

continue;

}

bool flag = false;

int v;

for(int i = cur[u]; i != -1;i = edge[i].next){

v = edge[i].to;

if(edge[i].cap - edge[i].flow && dep[v]+1 == dep[u]){

flag = true;

cur[u] = pre[v] = i;

break;

}

}

if(flag){

u = v;

continue;

}

int Min = N;

for(int i = head[u]; i != -1;i = edge[i].next)

if(edge[i].cap - edge[i].flow && dep[edge[i].to] < Min){

Min = dep[edge[i].to];

cur[u] = i;

}

gap[dep[u]]--;

if(!gap[dep[u]])return ans;

dep[u] = Min+1;

gap[dep[u]]++;

if(u != start) u = edge[pre[u]^1].to;

}

return ans;

}

// ISAP+bfs 初始化+栈优化

const int MAXN = 1000010;//点数的最大值

const int MAXM = 6000010;//边数的最大值

const int INF = 0x3f3f3f3f;

struct Edge{

int to,next,cap,flow;

}edge[MAXM];//注意是MAXM

int tol;

int head[MAXN];

int gap[MAXN],dep[MAXN],cur[MAXN];

void init(){

tol = 0;

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

}

void addedge(int u,int v,int w,int rw = 0){

edge[tol].to = v; edge[tol].cap = w; edge[tol].flow = 0;

edge[tol].next = head[u]; head[u] = tol++;

edge[tol].to = u; edge[tol].cap = rw; edge[tol].flow = 0;

edge[tol].next = head[v]; head[v] = tol++;

}

int Q[MAXN];

void BFS(int start,int end){

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

memset(gap,0,sizeof(gap));

gap[0] = 1;

int front = 0, rear = 0;

dep[end] = 0;

Q[rear++] = end;

while(front != rear){

int u = Q[front++];

for(int i = head[u]; i != -1; i = edge[i].next){

int v = edge[i].to;

if(dep[v] != -1)continue;

Q[rear++] = v;

dep[v] = dep[u] + 1;

gap[dep[v]]++;

}

}

}

int S[MAXN];

int sap(int start,int end,int N){

BFS(start,end);

memcpy(cur,head,sizeof(head));

int top = 0;

int u = start;

int ans = 0;

while(dep[start] < N){

if(u == end){

int Min = INF;

int inser;

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

if(Min > edge[S[i]].cap - edge[S[i]].flow){

Min = edge[S[i]].cap - edge[S[i]].flow;

inser = i;

}

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

edge[S[i]].flow += Min;

edge[S[i]^1].flow -= Min;

}

ans += Min;

top = inser;

u = edge[S[top]^1].to;

continue;

}

bool flag = false;

int v;

for(int i = cur[u]; i != -1; i = edge[i].next){

v = edge[i].to;

if(edge[i].cap - edge[i].flow && dep[v]+1 == dep[u]){

flag = true;

cur[u] = i;

break;

}

}

if(flag){

S[top++] = cur[u];

u = v;

continue;

}

int Min = N;

for(int i = head[u]; i != -1; i = edge[i].next)

if(edge[i].cap - edge[i].flow && dep[edge[i].to] < Min){

Min = dep[edge[i].to];

cur[u] = i;

}

gap[dep[u]]--;

if(!gap[dep[u]])return ans;

dep[u] = Min + 1;

gap[dep[u]]++;

if(u != start)u = edge[S[--top]^1].to;

}

return ans;

}

## 最小费用最大流

const int MAXN = 10000;

const int MAXM = 100000;

const int INF = 0x3f3f3f3f;

struct Edge{

int to,next,cap,flow,cost;

}edge[MAXM];

int head[MAXN],tol;

int pre[MAXN],dis[MAXN];

bool vis[MAXN];

int N;//节点总个数，节点编号从0~N-1

void init(int n){

N = n;

tol = 0;

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

}

void addedge(int u,int v,int cap,int cost){

edge[tol].to = v;

edge[tol].cap = cap;

edge[tol].cost = cost;

edge[tol].flow = 0;

edge[tol].next = head[u];

head[u] = tol++;

edge[tol].to = u;

edge[tol].cap = 0;

edge[tol].cost = -cost;

edge[tol].flow = 0;

edge[tol].next = head[v];

head[v] = tol++;

}

bool spfa(int s,int t){

queue<int>q;

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

dis[i] = INF;

vis[i] = false;

pre[i] = -1;

}

dis[s] = 0;

vis[s] = true;

q.push(s);

while(!q.empty()){

int u = q.front();

q.pop();

vis[u] = false;

for(int i = head[u]; i != -1;i = edge[i].next){

int v = edge[i].to;

if(edge[i].cap > edge[i].flow &&dis[v] > dis[u] + edge[i].cost ){

dis[v] = dis[u] + edge[i].cost;

pre[v] = i;

if(!vis[v]){

vis[v] = true;

q.push(v);

}

}

}

}

if(pre[t] == -1)

return false;

else return true;

}

//返回的是最大流， cost存的是最小费用

int minCostMaxflow(int s,int t,int &cost){

int flow = 0;

cost = 0;

while(spfa(s,t)){

int Min = INF;

for(int i = pre[t];i != -1;i = pre[edge[i^1].to]){

if(Min > edge[i].cap - edge[i].flow)

Min = edge[i].cap - edge[i].flow;

}

for(int i = pre[t];i != -1;i = pre[edge[i^1].to]){

edge[i].flow += Min;

edge[i^1].flow -= Min;

cost += edge[i].cost \* Min;

}

flow += Min;

}

return flow;

}

## 生成树计数

**Matrix-Tree** 定理**(Kirchhoff** 矩阵**-**树定理**)  
1**、 **G** 的度数矩阵 **D[G]**是一个 **n\*n** 的矩阵，并且满足：当 **i≠j** 时**,dij=0**；当 **i=j** 时， **dij** 等于 **vi** 的度数。

**2**、 **G** 的邻接矩阵 **A[G]**也是一个 **n\*n** 的矩阵， 并且满足：如果 **vi**、 **vj** 之间有边直接相连，则 **aij=1**，否则为 **0**。我们定义 **G** 的 **Kirchhoff** 矩阵**(**也称为拉普拉斯算子**)C[G]**为 **C[G]=D[G]-A[G]**，则 **Matrix-Tree** 定理可以描述为： **G** 的所有不同的生成树的个数等于其 **Kirchhoff** 矩阵 **C[G]**任何一个 **n-1** 阶主子式的行列式的绝对值。所谓 **n-1** 阶主子式，就是对于 **r(1≤r≤n)**，将 **C[G]**的第 **r** 行、第 **r** 列同时去掉后得到的新矩阵，用 **Cr[G]**表示。

//取模

struct Matrix{

int mat[330][330];

void init(){

memset(mat,0,sizeof(mat));

}

int det(int n){//求行列式的值模上MOD，需要使用逆元

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

for(int j = 0;j < n;j++)

mat[i][j] = (mat[i][j]%MOD+MOD)%MOD;

int res = 1;

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

for(int j = i;j < n;j++)

if(mat[j][i]!=0){

for(int k = i;k < n;k++)

swap(mat[i][k],mat[j][k]);

if(i != j)

res = (-res+MOD)%MOD;

break;

}

if(mat[i][i] == 0){

res = -1;//不存在(也就是行列式值为0)

break;

}

for(int j = i+1;j < n;j++){//int mut = (mat[j][i]\*INV[mat[i][i]])%MOD;//打表逆元

int mut = (mat[j][i]\*inv(mat[i][i],MOD))%MOD;

for(int k = i;k < n;k++)

mat[j][k] = (mat[j][k]-(mat[i][k]\*mut)%MOD+MOD)%MOD;

}

res = (res \* mat[i][i])%MOD;

}

return res;

}

};

//不取模

int sgn(double x){

if(fabs(x) < eps)return 0;

if(x < 0)return -1;

else return 1;

}

double b[MAXN][MAXN];

double det(double a[][MAXN],int n){

int i, j, k, sign = 0;

double ret = 1;

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

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

b[i][j] = a[i][j];

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

if(sgn(b[i][i]) == 0){

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

if(sgn(b[j][i]) != 0)

break;

if(j == n) return 0;

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

swap(b[i][k],b[j][k]);

sign++;

}

ret \*= b[i][i];

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

b[i][k]/=b[i][i];

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

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

b[j][k] -= b[j][i]\*b[i][k];

}

if(sign & 1) ret = -ret;

return ret;

}

# 数学

## Euler函数

//直接求解欧拉函数

int euler(int n){ //返回euler(n)

int res=n,a=n;

for(int i=2;i\*i<=a;i++){

if(a%i==0){

res -= res/i;

while(a%i==0) a/=i;

}

}

if(a>1) res=res/a\*(a-1);

return res;

}

//筛选法打欧拉函数表

#define Max 1000001

int euler[Max];

void Init(){

euler[1]=1;

for(int i=2;i<Max;i++)

euler[i]=i;

for(int i=2;i<Max;i++)

if(euler[i]==i)

for(int j=i;j<Max;j+=i)

euler[j]=euler[j]/i\*(i-1);//先进行除法是为了防止中间数据的溢出

}

## 逆元

const LL MAXN = 1e5;

const LL MOD = 1e9+7;

//1到n的逆元

LL inv[MAXN+5],fact[MAXN+5];

void INIT(){

inv[1] = 1;

for (int i = 2;i <= MAXN;i++){

if(i >= MOD) break;

inv[i] = (MOD - MOD / i) \* inv[MOD % i] % MOD;

}

}

//1到n！的逆元(组合数模质数)

LL \_inv(LL x) {

if(x == 1) return 1;

return (MOD - MOD / x) \* \_inv(MOD % x) % MOD;

}

void init() {

fact[0] = 1;

for(int i = 1; i <= MAXN; i++)

fact[i] = fact[i - 1] \* LL(i) % MOD;

inv[MAXN] = \_inv(fact[MAXN]);

for(int i = MAXN-1; i >= 0; i--)

inv[i] = inv[i+1]\*(i+1)%MOD;

}

LL comb(LL a, LL b) {

if(a < b) return 0;

return (fact[a]) \* inv[b] % MOD \* (inv[a - b]) % MOD;

}

LL Lucas(int n, int m) {

LL ans = 1;

while(n&&m&&ans) {

ans = (ans\*comb(n%MOD, m%MOD)) % MOD;

n /= MOD;

m /= MOD;

}

return ans;

}

## FFT

const double PI = acos(-1.0);

//复数结构体

struct Complex{

double x,y;//实部和虚部 x+yi

Complex(double \_x = 0.0,double \_y = 0.0){

x = \_x;

y = \_y;

}

Complex operator -(const Complex &b)const{

return Complex(x-b.x,y-b.y);

}

Complex operator +(const Complex &b)const{

return Complex(x+b.x,y+b.y);

}

Complex operator \*(const Complex &b)const{

return Complex(x\*b.x-y\*b.y,x\*b.y+y\*b.x);

}

};

/\*

\* 进行FFT和IFFT前的反转变换。

\* 位置i和 （ i二进制反转后位置）互换

\* len必须去2的幂

\*/

void change(Complex y[],int len){

int i,j,k;

for(i = 1, j = len/2;i <len-1;i++){

if(i < j)

swap(y[i],y[j]);//交换互为小标反转的元素， i<j保证交换一次

//i做正常的+1， j左反转类型的+1,始终保持i和j是反转的

k = len/2;

while(j >= k){

j -= k;

k /= 2;

}

if(j < k)j += k;

}

}

/\*

\* 做FFT

\* len必须为2^k形式，

\* on==1时是DFT， on==-1时是IDFT

\*/

void fft(Complex y[],int len,int on){

change(y,len);

for(int h = 2; h <= len; h <<= 1){

Complex wn(cos(-on\*2\*PI/h),sin(-on\*2\*PI/h));

for(int j = 0;j < len;j+=h){

Complex w(1,0);

for(int k = j;k < j+h/2;k++){

Complex u = y[k];

Complex t = w\*y[k+h/2];

y[k] = u+t;

y[k+h/2] = u-t;

w = w\*wn;

}

}

}

if(on == -1)

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

y[i].x /= len;

}

## 莫比乌斯反演

void Init()

{

memset(vis,0,sizeof(vis));

mu[1] = 1;

cnt = 0;

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

{

if(!vis[i])

{

prime[cnt++] = i;

mu[i] = -1;

}

for(int j=0; j<cnt&&i\*prime[j]<N; j++)

{

vis[i\*prime[j]] = 1;

if(i%prime[j]) mu[i\*prime[j]] = -mu[i];

else

{

mu[i\*prime[j]] = 0;

break;

}

}

}

}

## 自适应simpson积分

double simpson(double a,double b)

{//simpson公式

double c=a+(b-a)/2;

return (f(a)+4\*f(c)+f(b))\*(b-a)/6;

}

double asr(double a,double b,double eps,double A)

{//自适应部分

double c=a+(b-a)/2;

double L=simpson(a,c);

double R=simpson(c,b);

if(fabs(L+R-A)<=15\*eps) return L+R+(L+R-A)/15.0;//判断是否满足精度

return asr(a,c,eps/2,L)+asr(c,b,eps/2,R);

}

double asr(double a,double b,double eps)

{//积分

return asr(a,b,eps,simpson(a,b));

}

## 随机素数测试和大数分解(POJ 1811)

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Miller\_Rabin 算法进行素数测试

\* 速度快，可以判断一个 < 2^63 的数是不是素数

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

const int S = 8; //随机算法判定次数，一般8~10就够了

// 计算ret = (a\*b)%c a,b,c < 2^63

LL mult\_mod(LL a,LL b,LL c){

a %= c;

b %= c;

LL ret = 0;

LL tmp = a;

while(b){

if(b & 1){

ret += tmp;

if(ret > c)ret -= c;//直接取模慢很多

}

tmp <<= 1;

if(tmp > c)tmp -= c;

b >>= 1;

}

return ret;

}

// 计算 ret = (a^n)%mod

LL pow\_mod(LL a,LL n,LL mod){

LL ret = 1;

LL temp = a%mod;

while(n){

if(n & 1)ret = mult\_mod(ret,temp,mod);

temp = mult\_mod(temp,temp,mod);

n >>= 1;

}

return ret;

}

// 通过 a^(n-1)=1(mod n)来判断n是不是素数

// n-1 = x\*2^t 中间使用二次判断

// 是合数返回true, 不一定是合数返回false

bool check(LL a,LL n,LL x,LL t){

LL ret = pow\_mod(a,x,n);

LL last = ret;

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

ret = mult\_mod(ret,ret,n);

if(ret == 1 && last != 1 && last != n-1)return true;//合数

last = ret;

}

if(ret != 1)return true;

else return false;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Miller\_Rabin算法

// 是素数返回true,(可能是伪素数)

// 不是素数返回false

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

bool Miller\_Rabin(LL n)

{

if( n < 2)return false;

if( n == 2)return true;

if( (n&1) == 0)return false;//偶数

LL x = n - 1;

LL t = 0;

while( (x&1)==0 ){x >>= 1; t++;}

srand(time(NULL));/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/

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

LL a = rand()%(n-1) + 1;

if( check(a,n,x,t) )

return false;

}

return true;

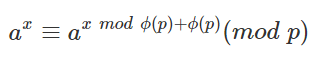
}

## 定理

### 欧拉定理

//欧拉定理

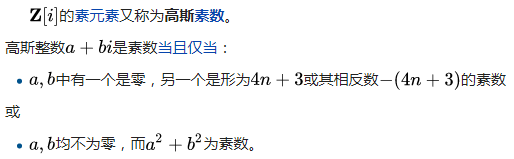
//要求gcd(a,n)==1

//无要求

### 费马平方和定理

奇质数能表示为两个平方数之和的充分必要条件是该素数被4除余1

### 高斯素数



## 特殊数列

### 第一类斯特林数

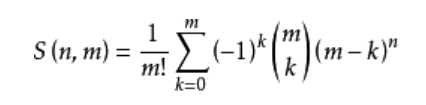
//将 n 个不同元素构成m个圆排列的数目。



### 第二类斯特林数

//将n个不同的元素拆分成m个集合的方案数





### 卡特兰数

//凸多边形三角划分,括号序列的种类 ,

递推关系的解为：

**h(n)=C(2n,n)/(n+1) (n=0,1,2,...)**

递推关系的另类解为：

**h(n)=c(2n,n)-c(2n,n-1)(n=0,1,2,...)**

# 博弈

## 威佐夫博奕

//a[k] = [k（1+√5）/2],b[k] = a[k] +k;

int main(){

int a,b,k,a\_k;

while(scanf("%d%d",&a,&b)!=EOF){

k = abs(a-b);

a = a < b? a : b;

a\_k = floor(k\*(1.0 + sqrt(5.0))/2);

printf("%d\n",a!=a\_k);

//输出为0，说明该点为必败点，1为必胜点

}

return 0;

}

## SG函数

//f[N]:可改变当前状态的方式，N为方式的种类，f[N]要在getSG之前先预处理

//SG[]:0~n的SG函数值

//S[]:为x后继状态的集合

int f[N],SG[MAXN],S[MAXN];

void getSG(int n){

int i,j;

memset(SG,0,sizeof(SG));

//因为SG[0]始终等于0，所以i从1开始

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

//每一次都要将上一状态 的 后继集合 重置

memset(S,0,sizeof(S));

for(j = 0; f[j] <= i && j <= N; j++)

S[SG[i-f[j]]] = 1; //将后继状态的SG函数值进行标记

for(j = 0;; j++) if(!S[j]){ //查询当前后继状态SG值中最小的非零值

SG[i] = j;

break;

}

}

}

## K倍动态减法

int a[N], b[N]; // a 为数列， b 保存 a[0...i] 能构造出的最大的数

int main()

{

int n, k;

int T, icase = 1;

scanf("%d",&T);

while(T--){

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

a[0] = b[0] = 1;

int i = 0, j = 0;

while(n > a[i]){ // 构建数列

i ++;

a[i] = b[i - 1] + 1;

while(a[j + 1] \* k < a[i])

j ++;

if(k \* a[j] < a[i])

b[i] = b[j] + a[i];

else

b[i] = a[i];

}

printf("Case %d: ", icase++);

if(n == a[i])

printf("lose\n");

else{

int ans;

while(n){

if(n >= a[i]){ //构成n的最小的数列中的数字，即为第一次要取的数字

n -= a[i];

ans = a[i];

}

i --;

}

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

}

}

return 0;

}

## Nim积

//一个二维矩阵上，有若干个亮着的灯泡   
//每次选择一个矩阵（右上角的灯泡必须是亮的），改变四个角灯泡的状态

int m[2][2]={0,0,0,1};

int Nim\_Mult\_Power(int x,int y){

if(x<2)

return m[x][y];

int a=0;

for(;;a++)

if(x>=(1<<(1<<a))&&x<(1<<(1<<(a+1))))

break;

int m=1<<(1<<a);

int p=x/m,s=y/m,t=y%m;

int d1=Nim\_Mult\_Power(p,s);

int d2=Nim\_Mult\_Power(p,t);

return (m\*(d1^d2))^Nim\_Mult\_Power(m/2,d1);

}

int Nim\_Mult(int x,int y){

if(x<y)

return Nim\_Mult(y,x);

if(x<2)

return m[x][y];

int a=0;

for(;;a++)

if(x>=(1<<(1<<a))&&x<(1<<(1<<(a+1))))

break;

int m=1<<(1<<a);

int p=x/m,q=x%m,s=y/m,t=y%m;

int c1=Nim\_Mult(p,s),c2=Nim\_Mult(p,t)^Nim\_Mult(q,s),c3=Nim\_Mult(q,t);

return (m\*(c1^c2))^c3^Nim\_Mult\_Power(m/2,c1);

}

int main()

{

int T;

scanf("%d",&T);

while(T--){

int n,ret = 0,x,y;

scanf("%d",&n);

while(n--){

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

ret ^= Nim\_Mult(x,y);

}

if(!ret) puts("Don't waste your time.");

else puts("Have a try, lxhgww.");

}

return 0;

}

# 其它

## 头文件

/\*

ID: sincerelyyin

LANG: C++

TASK: test

\*/

#include<bits/stdc++.h>

typedef long long LL;

using namespace std;

int main()

{

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

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

return 0;

}

#include<iostream>

#include<algorithm>

#include<cctype>

#include<cstdio>

#include<cstring>

#include<string>

#include<map>

#include<vector>

#include<set>

#include<cmath>

#include<queue>

#include<ctime>

#include<bitset>

//cin加速(不可与scanf,printf混用)

std::ios::sync\_with\_stdio(0);

std::cin.tie(0);

## 矩阵快速幂

struct Mat {

LL mat[N][N];

};

Mat operator \* (Mat a, Mat b) {

Mat c;

memset(c.mat, 0, sizeof(c.mat));

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

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

if(a.mat[i][k] == 0) continue; //(针对ZOJ2853)剪枝（加法的运算效率高于乘法，比如Strassen矩阵乘法）

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

c.mat[i][j] = (c.mat[i][j]+a.mat[i][k] \* b.mat[k][j])%MOD;

}

return c;

}

Mat operator ^ (Mat a, int k) {

Mat c;

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

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

c.mat[i][j] = (i == j); //初始化为单位矩阵

for(; k; k >>= 1) {

if(k&1) c = c\*a;

a = a\*a;

}

return c;

}

## NTT 求模数

|  |  |  |  |
| --- | --- | --- | --- |
| r⋅2k+1 | r | k | g |
| 3 | 1 | 1 | 2 |
| 5 | 1 | 2 | 2 |
| 17 | 1 | 4 | 3 |
| 97 | 3 | 5 | 5 |
| 193 | 3 | 6 | 5 |
| 257 | 1 | 8 | 3 |
| 7681 | 15 | 9 | 17 |
| 12289 | 3 | 12 | 11 |
| 40961 | 5 | 13 | 3 |
| 65537 | 1 | 16 | 3 |
| 786433 | 3 | 18 | 10 |
| 5767169 | 11 | 19 | 3 |
| 7340033 | 7 | 20 | 3 |
| 23068673 | 11 | 21 | 3 |
| 104857601 | 25 | 22 | 3 |
| 167772161 | 5 | 25 | 3 |
| 469762049 | 7 | 26 | 3 |
| 998244353 | 119 | 23 | 3 |
| 1004535809 | 479 | 21 | 3 |
| 2013265921 | 15 | 27 | 31 |
| 2281701377 | 17 | 27 | 3 |
| 3221225473 | 3 | 30 | 5 |
| 75161927681 | 35 | 31 | 3 |
| 77309411329 | 9 | 33 | 7 |
| 206158430209 | 3 | 36 | 22 |
| 2061584302081 | 15 | 37 | 7 |
| 2748779069441 | 5 | 39 | 3 |
| 6597069766657 | 3 | 41 | 5 |
| 39582418599937 | 9 | 42 | 5 |
| 79164837199873 | 9 | 43 | 5 |
| 263882790666241 | 15 | 44 | 7 |
| 1231453023109121 | 35 | 45 | 3 |
| 1337006139375617 | 19 | 46 | 3 |
| 3799912185593857 | 27 | 47 | 5 |
| 4222124650659841 | 15 | 48 | 19 |
| 7881299347898369 | 7 | 50 | 6 |
| 31525197391593473 | 7 | 52 | 3 |
| 180143985094819841 | 5 | 55 | 6 |
| 1945555039024054273 | 27 | 56 | 5 |
| 4179340454199820289 | 29 | 57 | 3 |