

**ACM模板**



by tokitsukaze

2018.11.21

目录

[1.数据结构 4](#_Toc23088)

[1.1 RMQ 4](#_Toc4750)

[1.1.1 一维RMQ 4](#_Toc14918)

[1.1.2 下标RMQ 5](#_Toc25046)

[1.1.3 二维RMQ 5](#_Toc7022)

[1.2 并查集 7](#_Toc30849)

[1.2.1 dsu 7](#_Toc13221)

[1.3 树状数组 7](#_Toc31567)

[1.3.1 一维bit 7](#_Toc10369)

[1.3.2 二维bit 8](#_Toc1544)

[1.4 线段树 9](#_Toc10505)

[1.4.1 一维线段树 9](#_Toc3188)

[1.4.2 动态开点线段树 11](#_Toc16219)

[1.4.3 多棵动态开点线段树的分裂与合并 12](#_Toc2018)

[1.5 Trie 15](#_Toc24711)

[1.5.1 Trie 15](#_Toc32282)

[1.5.2 01Trie 16](#_Toc13387)

[1.5.3 动态开点01Trie 17](#_Toc5295)

[1.6 主席树 19](#_Toc7051)

[1.7 Treap 21](#_Toc15370)

[1.8 LCA 24](#_Toc29635)

[1.8.1 dfs+ST O(nlogn)预处理,O(1)查询 24](#_Toc6055)

[1.8.2 树链剖分 O(n)预处理,O(logn)查询 25](#_Toc18111)

[1.8.3 离线Tarjan O(n) 25](#_Toc13656)

[1.9 树链剖分 26](#_Toc20830)

[1.10 KD-tree 29](#_Toc29580)

[1.11 k叉哈夫曼树 34](#_Toc24879)

[2.字符串 35](#_Toc14064)

[2.1 KMP 35](#_Toc18465)

[2.1.1 kmp 35](#_Toc20957)

[2.1.2 exkmp 36](#_Toc24234)

[2.2 manacher 37](#_Toc21618)

[2.2.1 插分隔符 37](#_Toc4098)

[2.2.2 不插分隔符 38](#_Toc29261)

[2.3 AC自动机 39](#_Toc2411)

[2.4 hash 41](#_Toc21312)

[2.4.1 hash 41](#_Toc11430)

[2.4.2 BKDRHash 42](#_Toc3700)

[2.4.3 Hash\_map 43](#_Toc9977)

[2.5 回文树 44](#_Toc19413)

[3.图论 45](#_Toc22895)

[3.1 链式前向星 45](#_Toc9667)

[3.2 最短路 45](#_Toc25675)

[3.2.1 Dijkstra 45](#_Toc30040)

[3.2.2 spfa 46](#_Toc30390)

[3.3 floyd求最小环 hdu1599 47](#_Toc17853)

[3.4 最小生成树 48](#_Toc19760)

[3.4.1 prim 48](#_Toc20336)

[3.4.2 kruskal 49](#_Toc23637)

[3.5 最小树形图(待补) 50](#_Toc8804)

[3.6 二分图匹配 50](#_Toc21230)

[3.6.1 匈牙利算法 50](#_Toc7771)

[3.7 网络流 51](#_Toc13673)

[3.7.1 Dinic 51](#_Toc4850)

[3.7.2 ISAP 53](#_Toc9769)

[3.7.3 High Level Preflow Push 56](#_Toc27962)

[3.8 费用流 60](#_Toc18774)

[3.8.1 Min Cost Max Flow 60](#_Toc18122)

[3.9 强连通分量 62](#_Toc26631)

[3.10 双连通分量 63](#_Toc2735)

[3.10.1 边双连通-桥-割点 63](#_Toc22027)

[3.11 2-sat 65](#_Toc18094)

[3.11.1 染色法 输出字典序最小的解 O(n\*m) 65](#_Toc30354)

[3.10.2 强连通缩点法 输出任意一组解 O(n+m) 66](#_Toc13226)

[4.数论 69](#_Toc24455)

[4.1 素数筛与分解质因数 69](#_Toc5682)

[4.1.1 埃氏筛 69](#_Toc22622)

[4.1.2 线性筛 70](#_Toc24704)

[4.1.3 区间筛 70](#_Toc6299)

[4.1.4 分解质因数 71](#_Toc2417)

[4.2 Miller\_Rabin + Pollard\_rho 71](#_Toc2923)

[4.3 exgcd 73](#_Toc26686)

[4.3.1 exgcd 73](#_Toc18674)

[4.3.2 xa+yb=c 74](#_Toc8958)

[4.4 逆元 74](#_Toc15180)

[4.4.1 费马小定理 74](#_Toc28220)

[4.4.2 扩展欧几里得 74](#_Toc21584)

[4.4.3 公式 75](#_Toc3614)

[4.4.4 逆元打表 75](#_Toc8841)

[4.5 中国剩余定理 75](#_Toc19957)

[4.6 欧拉函数 75](#_Toc18199)

[4.6.1 直接求某个数的欧拉函数 O(sqrt(n)) 75](#_Toc2356)

[4.6.2 线性筛 O(n) 76](#_Toc6979)

[4.7 莫比乌斯函数 77](#_Toc30997)

[4.8 组合数 77](#_Toc3087)

[4.8.1 小范围 77](#_Toc21628)

[4.8.2 大范围 78](#_Toc8982)

[4.9 Lucas定理 78](#_Toc31700)

[4.10 第二类Stirling数 79](#_Toc12134)

[4.11 线性基 79](#_Toc30972)

[4.12 Berlekamp-Massey 81](#_Toc22084)

[4.13 原根 83](#_Toc7043)

[4.13.1 原根性质 83](#_Toc21238)

[4.13.2 指标法则 83](#_Toc31672)

[4.13.3 求素数原根与指标表 83](#_Toc9743)

[4.14 ex Baby-Step-Giant-Step a^x≡b (mod c) 85](#_Toc17252)

[5.多项式 86](#_Toc14279)

[5.1 FFT 86](#_Toc20774)

[5.2 NTT 87](#_Toc20070)

[5.3 FWT 89](#_Toc13247)

[5.4 拉格朗日插值 90](#_Toc213)

[6.矩阵 92](#_Toc9441)

[6.1 矩阵基本操作 92](#_Toc20164)

[6.2 矩阵快速幂 93](#_Toc10034)

[6.3 高斯消元 93](#_Toc4883)

[6.3.1 同余方程 mod=2时 异或加速 93](#_Toc18849)

[7.博弈论 94](#_Toc27222)

[7.1 威佐夫博弈 94](#_Toc25161)

[7.2 SG函数 94](#_Toc13999)

[7.2.1 sg表 94](#_Toc25963)

[7.2.2 记忆化搜索求sg表 95](#_Toc29532)

[7.3 阶梯博弈 96](#_Toc24911)

[7.4 SJ定理 96](#_Toc23960)

[8.dp 96](#_Toc11781)

[8.1 LIS 96](#_Toc26065)

[8.2 LPS 96](#_Toc6243)

[8.3 数位dp 97](#_Toc13118)

[9.Other 98](#_Toc18861)

[9.1 FastIO 98](#_Toc5454)

[9.1.1 fast input 98](#_Toc31182)

[9.1.2 FastIO 98](#_Toc22937)

[9.2 网格整数点共有多少个正方形 101](#_Toc10250)

[9.3 模拟退火 102](#_Toc29228)

[9.3.1 简单版 ->模拟退火求费马点 102](#_Toc13971)

[9.3.2 复杂版 102](#_Toc30572)

[9.4 矩形面积并 104](#_Toc26467)

[9.5 判断星期几 106](#_Toc10729)

[9.6 hash\_map 106](#_Toc27976)

[9.7 O(1)快速乘 107](#_Toc31691)

[9.8 快速模 107](#_Toc6671)

[9.9 离散化 108](#_Toc9932)

[NTT常用mod 108](#_Toc15141)

[斐波那契数列性质 110](#_Toc15067)

# **1.**数据结构

## **1.1** RMQ

1.1.1 一维RMQ

int v[MAX],maxx[MAX][22],minn[MAX][22];

void RMQ(int n)

{

int i,j;

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

{

maxx[i][0]=minn[i][0]=v[i];

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

{

maxx[i][j]=0;

minn[i][j]=INF;

}

}

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

{

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

{

int t=1<<(j-1);

maxx[i][j]=max(maxx[i][j-1],maxx[i+t][j-1]);

minn[i][j]=min(minn[i][j-1],minn[i+t][j-1]);

}

}

}

int query(int l,int r)

{

int j=(int)(log10(r-l+1)/log10(2))+1;

int i=r-(1<<(j-1))+1;

return max(maxx[l][j-1],maxx[i][j-1]);

// return min(minn[l][j-1],minn[i][j-1]);

}

1.1.2 下标RMQ

int v[MAX],maxx[MAX][22],minn[MAX][22];

int pmax(int a,int b){return v[a]>v[b]?a:b;}

int pmin(int a,int b){return v[a]<v[b]?a:b;}

void RMQ(int n)

{

int i,j;

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

{

maxx[i][0]=minn[i][0]=i;

}

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

{

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

{

int t=1<<(j-1);

maxx[i][j]=pmax(maxx[i][j-1],maxx[i+t][j-1]);

minn[i][j]=pmin(minn[i][j-1],minn[i+t][j-1]);

}

}

}

int query(int l,int r)

{

int j=(int)(log10(r-l+1)/log10(2))+1;

int i=r-(1<<(j-1))+1;

return pmax(maxx[l][j-1],maxx[i][j-1]);

// return pmin(minn[l][j-1],minn[i][j-1]);

}

1.1.3 二维RMQ

int v[302][302];

int maxx[302][302][9][9],minn[302][302][9][9];

void RMQ(int n,int m)

{

int i,j,ii,jj;

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

{

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

{

maxx[i][j][0][0]=minn[i][j][0][0]=v[i][j];

}

}

for(ii=0;(1<<ii)<=n;ii++)

{

for(jj=0;(1<<jj)<=m;jj++)

{

if(ii+jj)

{

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

{

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

{

if(ii)

{

minn[i][j][ii][jj]=min(minn[i][j][ii-1][jj],minn[i+(1<<(ii-1))][j][ii-1][jj]);

maxx[i][j][ii][jj]=max(maxx[i][j][ii-1][jj],maxx[i+(1<<(ii-1))][j][ii-1][jj]);

}

else

{

minn[i][j][ii][jj]=min(minn[i][j][ii][jj-1],minn[i][j+(1<<(jj-1))][ii][jj-1]);

maxx[i][j][ii][jj]=max(maxx[i][j][ii][jj-1],maxx[i][j+(1<<(jj-1))][ii][jj-1]);

}

}

}

}

}

}

}

int query(int x1,int y1,int x2,int y2)

{

int k1=0;

while((1<<(k1+1))<=x2-x1+1) k1++;

int k2=0;

while((1<<(k2+1))<=y2-y1+1) k2++;

x2=x2-(1<<k1)+1;

y2=y2-(1<<k2)+1;

return max(max(maxx[x1][y1][k1][k2],maxx[x1][y2][k1][k2]),max(maxx[x2][y1][k1][k2],maxx[x2][y2][k1][k2]))

// return min(min(minn[x1][y1][k1][k2],minn[x1][y2][k1][k2]),min(minn[x2][y1][k1][k2],minn[x2][y2][k1][k2]));

}

## **1.2** 并查集

1.2.1 dsu

struct dsu

{

int pre[MAX];

void init(int n)

{

int i;

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

{

pre[i]=i;

}

}

int find(int x)

{

if(pre[x]!=x) pre[x]=find(pre[x]);

return pre[x];

}

void merge(int a,int b)

{

int ra,rb;

ra=find(a);

rb=find(b);

if(ra!=rb) pre[ra]=rb;

}

}dsu;

## **1.3** 树状数组

1.3.1 一维bit

struct Fenwick\_Tree

{

#define type int

type bit[MAX];

int n;

void init(int \_n){n=\_n;mem(bit,0);}

int lowbit(int x){return x&(-x);}

void insert(int x,type v)

{

while(x<=n)

{

bit[x]+=v;

x+=lowbit(x);

}

}

type get(int x)

{

type res=0;

while(x)

{

res+=bit[x];

x-=lowbit(x);

}

return res;

}

type query(int l,int r)

{

return get(r)-get(l-1);

}

#undef type

}tr;

1.3.2 二维bit

struct Fenwick\_Tree

{

#define type int

type bit[MAX][MAX];

int n,m;

void init(int \_n,int \_m){n=\_n;m=\_m;mem(bit,0);}

int lowbit(int x){return x&(-x);}

void update(int x,int y,type v)

{

int i,j;

for(i=x;i<=n;i+=lowbit(i))

{

for(j=y;j<=m;j+=lowbit(j))

{

bit[i][j]+=v;

}

}

}

type get(int x,int y)

{

type i,j,res=0;

for(i=x;i>0;i-=lowbit(i))

{

for(j=y;j>0;j-=lowbit(j))

{

res+=bit[i][j];

}

}

return res;

}

type query(int x1,int x2,int y1,int y2)

{

x1--;

y1--;

return get(x2,y2)-get(x1,y2)-get(x2,y1)+get(x1,y1);

}

#undef type

}tr;

**1.4** 线段树

1.4.1 一维线段树

struct Segment\_Tree

{

#define type int

#define ls (id<<1)

#define rs (id<<1|1)

int n,ql,qr;

type a[MAX],v[MAX<<2],tag[MAX<<2],qv;

void pushup(int id)

{

}

void pushdown(int id)

{

if(!tag[id]) return;

}

void build(int l,int r,int id)

{

tag[id]=0;

if(l==r)

{

v[id]=a[l];

return;

}

int mid=(l+r)>>1;

build(l,mid,ls);

build(mid+1,r,rs);

pushup(id);

}

void update(int l,int r,int id)

{

if(l>=ql&&r<=qr)

{

return;

}

pushdown(id);

int mid=(l+r)>>1;

if(ql<=mid) update(l,mid,ls);

if(qr>mid) update(mid+1,r,rs);

pushup(id);

}

type query(int l,int r,int id)

{

type res=0;

if(l>=ql&&r<=qr) return v[id];

pushdown(id);

int mid=(l+r)>>1;

if(ql<=mid) res+=query(l,mid,ls);

if(qr>mid) res+=query(mid+1,r,rs);

return res;

}

void build(int \_n){n=\_n;build(1,n,1);}

void upd(int l,int r,int v)

{

ql=l;

qr=r;

qv=v;

update(1,n,1);

}

type ask(int l,int r)

{

ql=l;

qr=r;

return query(1,n,1);

}

#undef type

#undef ls

#undef rs

}tr;

1.4.2 动态开点线段树

//空间大小是nlogm,n为插入的节点总数,m为区间长度

struct Segment\_Tree

{

#define type int

int root,tot,ls[MAX\*20],rs[MAX\*20],ql,qr;

type v[MAX\*20],tag[MAX\*20],qv;

void init()

{

root=0;

ls[0]=rs[0]=0;

v[0]=0;

tag[0]=-1;

tot=1;

}

int newnode()

{

ls[tot]=rs[tot]=0;

v[tot]=0;

tag[tot]=-1;

return tot++;

}

void pushdown(int id)

{

if(tag[id]==-1) return;

if(!ls[id]) ls[id]=newnode();

if(!rs[id]) rs[id]=newnode();

tag[id]=-1;

}

void pushup(int id)

{

}

void update(int l,int r,int &id)

{

if(!id) id=newnode();

if(l>=ql&&r<=qr)

{

v[id]=(r-l+1)\*qv;

tag[id]=qv;

return;

}

pushdown(id);

int mid=(l+r)>>1;

if(ql<=mid) update(l,mid,ls[id]);

if(qr>mid) update(mid+1,r,rs[id]);

pushup(id);

}

type query(int l,int r,int &id)

{

if(!id) return 0;

if(l>=ql&&r<=qr) return v[id];

int mid=(l+r)>>1;

type res=0;

if(ql<=mid) res+=query(l,mid,ls[id]);

if(qr>mid) res+=query(mid+1,r,rs[id]);

return res;

}

#undef type

}tr;

1.4.3 多棵动态开点线段树的分裂与合并

//空间大小是nlogm,n为插入的节点总数,m为区间长度

struct Segment\_Tree

{

#define type int

int s[MAX\*20],top;//内存池

int root[MAX],tot,ls[MAX\*20],rs[MAX\*20],ql,qr,n;

type v[MAX\*20],tag[MAX\*20],qv;

void init()

{

top=0;

mem(root,0);

ls[0]=rs[0]=0;

v[0]=0;

tot=1;

}

int newnode()

{

int t;

if(top) t=s[--top];

else t=tot++;

ls[t]=rs[t]=0;

v[t]=0;

return t;

}

void delnode(int x)

{

s[top++]=x;

}

void pushup(int id)

{

v[id]=v[ls[id]]+v[rs[id]];

}

void pushdown(int id)

{

if(tag[id]==-1) return;

if(!ls[id]) ls[id]=newnode();

if(!rs[id]) rs[id]=newnode();

tag[id]=-1;

}

int split(int l,int r,int &id)

{

if(!id) return 0;

if(ql<=l&&r<=qr)

{

int temp=id;

id=0;

return temp;

}

int t=newnode();

int mid=(l+r)>>1;

if(ql<=mid) ls[t]=split(l,mid,ls[id]);

if(qr>mid) rs[t]=split(mid+1,r,rs[id]);

pushup(t);

pushup(id);

return t;

}

int merge(int a,int b)

{

if(!a||!b) return a+b;

ls[a]=merge(ls[a],ls[b]);

rs[a]=merge(rs[a],rs[b]);

if(!ls[a]&&!rs[a])

{

v[a]+=v[b];//把b的信息合并给a

}

else

{

pushup(a);

//此处可能需要更新其他东西

}

delnode(b);

return a;

}

void update(int l,int r,int &id)

{

if(!id) id=newnode();

if(l>=ql&&r<=qr)

{

v[id]=(r-l+1)\*qv;

tag[id]=qv;

return;

}

pushdown(id);

int mid=(l+r)>>1;

if(ql<=mid) update(l,mid,ls[id]);

if(qr>mid) update(mid+1,r,rs[id]);

pushup(id);

}

type query(int l,int r,int &id)

{

if(!id) return 0;

if(l>=ql&&r<=qr) return v[id];

int mid=(l+r)>>1;

type res=0;

if(ql<=mid) res+=query(l,mid,ls[id]);

if(qr>mid) res+=query(mid+1,r,rs[id]);

return res;

}

#undef type

}tr;

## **1.5** Trie

1.5.1 Trie

struct Trie

{

#define type int

struct trie

{

int v;

trie \*next[26];

trie()

{

v=0;

for(int i=0;i<26;i++) next[i]=NULL;

}

}\*root;

void insert(trie \*p,char \*s)

{

int i=0,t;

while(s[i])

{

t=s[i]-'a';

if(p->next[t]==NULL) p->next[t]=new trie;

p=p->next[t];

p->v++;//按情况改

i++;

}

}

int find(trie \*p,char \*s)

{

int i=0,t;

while(s[i])

{

t=s[i]-'a';

if(p->next[t]==NULL) return 0;

p=p->next[t];

i++;

}

return p->v;//按情况改

}

//删除前缀为s的字符串

void del(char \*s)

{

int i=0,t,temp;

trie \*p,\*pre;

pre=p=root;

while(s[i])

{

t=s[i]-'a';

if(p->next[t]==NULL) return;

if(!s[i+1])

{

temp=p->next[t]->v;

p->next[t]=NULL;

break;

}

pre=p;

p=p->next[t];

i++;

}

i=0;

p=root;

while(s[i])

{

t=s[i]-'a';

if(p->next[t]==NULL) return;

p=p->next[t];

p->v-=temp;

i++;

}

}

#undef type

}tr;

1.5.2 01Trie

/\*

数组大小(x+1)\*MAX:插入的值的最大值<2^x<MAX

Trie.Insert(1,x,v);

Trie.Delete(1,x,v);

Trie.query(1,x,v);

Trie.clear(1,17);

\*/

struct Trie

{

int cnt[32\*MAX],val[32\*MAX];

void Insert(int x,int pos,int v)

{

if(pos<0)

{

cnt[x]++;

val[x]=v;

return;

}

Insert((x<<1)|((v>>pos)&1),pos-1,v);

cnt[x]=cnt[x<<1]+cnt[x<<1|1];

}

void Delete(int x,int pos,int v)

{

if(pos<0)

{

cnt[x]--;

return;

}

Delete((x<<1)|((v>>pos)&1),pos-1,v);

cnt[x]=cnt[x<<1]+cnt[x<<1|1];

}

void clear(int x,int pos)

{

cnt[x]=0;

if(pos<0) return;

clear(x<<1,pos-1);

clear(x<<1|1,pos-1);

}

int query(int x,int pos,int v)//查询与v异或的最大值 并返回

{

if(pos<0) return val[x];

int temp=(v>>pos)&1;

temp|=x<<1;

if(cnt[temp^1]) return query(temp^1,pos-1,v);

return query(temp,pos-1,v);

}

}tr;

1.5.3 动态开点01Trie

/\*

数组大小(x+1)\*MAX:插入的值的最大值<2^x<MAX

Trie.Insert(1,x,v);

Trie.Delete(1,x,v);

Trie.query(1,x,v);

Trie.clear(1,x);

\*/

struct Trie

{

int root,tot,ls[MAX\*31],rs[MAX\*31],val[MAX\*31],cnt[MAX\*31];

void init()

{

root=0;

ls[0]=rs[0]=0;

val[0]=cnt[0]=0;

tot=1;

}

int newnode()

{

ls[tot]=rs[tot]=0;

val[tot]=0;

return tot++;

}

void Insert(int &x,int pos,int v)

{

if(!x) x=newnode();

if(pos<0)

{

cnt[x]++;

val[x]=v;

return;

}

if((v>>pos)&1) Insert(rs[x],pos-1,v);

else Insert(ls[x],pos-1,v);

cnt[x]=cnt[ls[x]]+cnt[rs[x]];

}

void Delete(int x,int pos,int v)

{

if(pos<0)

{

cnt[x]--;

return;

}

if((v>>pos)&1) Delete(rs[x],pos-1,v);

else Delete(ls[x],pos-1,v);

cnt[x]=cnt[ls[x]]+cnt[rs[x]];

}

int query(int x,int pos,int v)

{

if(pos<0) return val[x];

int temp=(v>>pos)&1;

if(temp)

{

if(cnt[rs[x]]) return query(rs[x],pos-1,v);

else return query(ls[x],pos-1,v);

}

else

{

if(cnt[ls[x]]) return query(ls[x],pos-1,v);

else return query(rs[x],pos-1,v);

}

}

}tr;

## **1.6** 主席树

struct president\_tree

{

#define type int

int root[MAX],ls[40\*MAX],rs[40\*MAX],tot,ql,qr;

type sum[40\*MAX],qv;

void init()

{

mem(root,0);

tot=1;

ls[0]=rs[0]=sum[0]=0;

}

int newnode(int x)

{

ls[tot]=ls[x];

rs[tot]=rs[x];

sum[tot]=sum[x];

return tot++;

}

void insert(int l,int r,int &id,int pre) //set(ql,ql,v)

{

id=newnode(pre);

sum[id]+=qv;

if(l==r) return;

int mid=(l+r)>>1;

if(ql<=mid) insert(l,mid,ls[id],ls[pre]);

else insert(mid+1,r,rs[id],rs[pre]);

}

int kindcnt(int l,int r,int id) //set(ql,qr)

{

if(ql<=l&&r<=qr) return sum[id];

int mid=(l+r)>>1;

int res=0;

if(ql<=mid) res+=kindcnt(l,mid,ls[id]);

if(qr>=mid+1) res+=kindcnt(mid+1,r,rs[id]);

return res;

}

int kthsmall(int l,int r,int id,int pre,int k)

{

if(l==r) return l;

int mid=(l+r)>>1;

int temp=sum[ls[id]]-sum[ls[pre]];

if(temp>=k) return kthsmall(l,mid,ls[id],ls[pre],k);

else return kthsmall(mid+1,r,rs[id],rs[pre],k-temp);

}

int kthbig(int l,int r,int id,int pre,int k)

{

if(l==r) return l;

int mid=(l+r)>>1;

int temp=sum[rs[id]]-sum[rs[pre]];

if(temp>=k) return kthbig(mid+1,r,rs[id],rs[pre],k);

else return kthbig(l,mid,ls[id],ls[pre],k-temp);

}

void set(int l,int r,type v=0){ql=l;qr=r;qv=v;}

}pt;

## **1.7** Treap

struct Treap

{

#define type ll

struct node

{

int ch[2],fix,sz,w;

type v;

node(){}

node(type x)

{

v=x;

fix=rand();

sz=w=1;

ch[0]=ch[1]=0;

}

}t[MAX];

int tot,root,tmp;

void init()

{

srand(unsigned(new char));

root=tot=0;

t[0].sz=t[0].w=0;

mem(t[0].ch,0);

}

inline void maintain(int k)

{

t[k].sz=t[t[k].ch[0]].sz+t[t[k].ch[1]].sz+t[k].w ;

}

inline void rotate(int &id,int k)

{

int y=t[id].ch[k^1];

t[id].ch[k^1]=t[y].ch[k];

t[y].ch[k]=id;

maintain(id);

maintain(y);

id=y;

}

void insert(int &id,type v)

{

if(!id) t[id=++tot]=node(v);

else

{

if(t[id].sz++,t[id].v==v)t[id].w++;

else if(insert(t[id].ch[tmp=v>t[id].v],v),t[t[id].ch[tmp]].fix>t[id].fix) rotate(id,tmp^1);

}

}

void erase(int &id,type v)

{

if(!id)return;

if(t[id].v==v)

{

if(t[id].w>1) t[id].w--,t[id].sz--;

else

{

if(!(t[id].ch[0]&&t[id].ch[1])) id=t[id].ch[0]|t[id].ch[1];

else

{

rotate(id,tmp=t[t[id].ch[0]].fix>t[t[id].ch[1]].fix);

t[id].sz--;

erase(t[id].ch[tmp],v);

}

}

}

else

{

t[id].sz--;

erase(t[id].ch[v>t[id].v],v);

}

}

type kth(int k)//k small

{

int id=root;

if(id==0) return 0;

while(id)

{

if(t[t[id].ch[0]].sz>=k) id=t[id].ch[0];

else if(t[t[id].ch[0]].sz+t[id].w>=k) return t[id].v;

else

{

k-=t[t[id].ch[0]].sz+t[id].w;

id=t[id].ch[1];

}

}

}

int find(type key,int f)

{

int id=root,res=0;

while(id)

{

if(t[id].v<=key)

{

res+=t[t[id].ch[0]].sz+t[id].w;

if(f&&key==t[id].v) res-=t[id].w;

id=t[id].ch[1];

}

else id=t[id].ch[0];

}

return res;

}

type find\_pre(type key)

{

type res=-LLINF;

int id=root;

while(id)

{

if(t[id].v<key)

{

res=max(res,t[id].v);

id=t[id].ch[1];

}

else id=t[id].ch[0];

}

return res;

}

type find\_suc(type key)

{

type res=LLINF;

int id=root;

while(id)

{

if(t[id].v>key)

{

res=min(res,t[id].v);

id=t[id].ch[0];

}

else id=t[id].ch[1];

}

return res;

}

void insert(type v){insert(root,v);}

void erase(type v){erase(root,v);}

int upper\_bound\_count(type key){return find(key,0);}//the count >=key

int lower\_bound\_count(type key){return find(key,1);}//the count >key

int rank(type key){return lower\_bound\_count(key)+1;}

#undef type

}t; //t.init();

## **1.8** LCA

1.8.1 dfs+ST O(nlogn)预处理,O(1)查询

struct node{int to;int w;node(){}node(int \_to,int \_w):to(\_to),w(\_w){}};

int dis[MAX];

int path[2\*MAX],deep[2\*MAX],first[MAX],tot;

int dp[2\*MAX][28];

vector<node> mp[MAX];

void dfs(int x,int pre,int h)

{

int i;

path[++tot]=x;

first[x]=tot;

deep[tot]=h;

for(i=0;i<mp[x].size();i++)

{

int to=mp[x][i].to;

if(to==pre) continue;

dis[to]=dis[x]+mp[x][i].w;

dfs(to,x,h+1);

path[++tot]=x;

deep[tot]=h;

}

}

void ST(int n)

{

int i,j,x,y;

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

{

dp[i][0]=i;

}

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

{

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

{

x=dp[i][j-1];

y=dp[i+(1<<(j-1))][j-1];

dp[i][j]=deep[x]<deep[y]?x:y;

}

}

}

int query(int l,int r)

{

int len,x,y;

len=(int)log2(r-l+1);

x=dp[l][len];

y=dp[r-(1<<len)+1][len];

return deep[x]<deep[y]?x:y;

}

int LCA(int x,int y)

{

int l,r,pos;

l=first[x];

r=first[y];

if(l>r) swap(l,r);

pos=query(l,r);

return path[pos];

}

int getdist(int a,int b)

{

return dis[a]+dis[b]-2\*dis[LCA(a,b)];

}

void work(int n)

{

for(int i=0;i<=n;i++) dis[i]=0;

tot=0;

dfs(1,0,0);

ST(2\*n-1);

}

1.8.2 树链剖分 O(n)预处理,O(logn)查询

1.8.3 离线Tarjan O(n)

## **1.9** 树链剖分

/\*

size[]数组，以x为根的子树节点个数

top[]数组，当前节点的所在链的顶端节点

son[]数组，重儿子

deep[]数组，当前节点的深度

fa[]数组，当前节点的父亲

idx[]数组，树中每个节点剖分后的新编号

rnk[]数组，idx的逆，表示线段上中当前位置表示哪个节点

\*/

struct HLD

{

#define type int

struct edge{int a,b;type v;edge(int \_a,int \_b,type \_v=0):a(\_a),b(\_b),v(\_v){}};

struct node{int to;type w;node(){}node(int \_to,type \_w):to(\_to),w(\_w){}};

vector<int> mp[MAX];

vector<edge> e;

int deep[MAX],fa[MAX],size[MAX],son[MAX];

int rnk[MAX],top[MAX],idx[MAX],tot;

int n,rt;

void init(int \_n)

{

n=\_n;

for(int i=0;i<=n;i++) mp[i].clear();

e.clear();

e.pb(edge(0,0));

}

void add\_edge(int a,int b,type v=0)

{

e.pb(edge(a,b,v));

mp[a].pb(b);

mp[b].pb(a);

}

void dfs1(int x,int pre,int h)

{

int i,to;

deep[x]=h;

fa[x]=pre;

size[x]=1;

for(i=0;i<sz(mp[x]);i++)

{

to=mp[x][i];

if(to==pre) continue;

dfs1(to,x,h+1);

size[x]+=size[to];

if(son[x]==-1||size[to]>size[son[x]]) son[x]=to;

}

}

void dfs2(int x,int tp)

{

int i,to;

top[x]=tp;

idx[x]=++tot;

rnk[idx[x]]=x;

if(son[x]==-1) return;

dfs2(son[x],tp);

for(i=0;i<sz(mp[x]);i++)

{

to=mp[x][i];

if(to!=son[x]&&to!=fa[x]) dfs2(to,to);

}

}

void work(int \_rt)

{

int i;

rt=\_rt;

mem(son,-1);

tot=0;

dfs1(rt,0,0);

dfs2(rt,rt);

}

int LCA(int x,int y)

{

while(top[x]!=top[y])

{

if(deep[top[x]]<deep[top[y]]) swap(x,y);

x=fa[top[x]];

}

if(deep[x]>deep[y]) swap(x,y);

return x;

}

//node

void init\_node()

{

build(n);

}

void modify\_node(int x,int y,type val)

{

while(top[x]!=top[y])

{

if(deep[top[x]]<deep[top[y]]) swap(x,y);

update(idx[top[x]],idx[x],val);

x=fa[top[x]];

}

if(deep[x]>deep[y]) swap(x,y);

update(idx[x],idx[y],val);

}

type query\_node(int x,int y)

{

type res=0;

while(top[x]!=top[y])

{

if(deep[top[x]]<deep[top[y]]) swap(x,y);

res+=query(idx[top[x]],idx[x]);

x=fa[top[x]];

}

if(deep[x]>deep[y]) swap(x,y);

res+=query(idx[x],idx[y]);

return res;

}

//path

void init\_path()

{

v[idx[rt]]=0;

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

{

if(deep[e[i].a]<deep[e[i].b]) swap(e[i].a,e[i].b);

v[idx[e[i].a]]=e[i].v;

}

build(n);

}

void modify\_edge(int id,type val)

{

if(deep[e[id].a]>deep[e[id].b]) update(idx[e[id].a],idx[e[id].a],val);

else update(idx[e[id].b],idx[e[id].b],val);

}

void modify\_path(int x,int y,type val)

{

while(top[x]!=top[y])

{

if(deep[top[x]]<deep[top[y]]) swap(x,y);

update(idx[top[x]],idx[x],val);

x=fa[top[x]];

}

if(deep[x]>deep[y]) swap(x,y);

if(x!=y) update(idx[x]+1,idx[y],val);

}

type query\_path(int x,int y)

{

type res=0;

while(top[x]!=top[y])

{

if(deep[top[x]]<deep[top[y]]) swap(x,y);

res+=query(idx[top[x]],idx[x]);

x=fa[top[x]];

}

if(deep[x]>deep[y]) swap(x,y);

if(x!=y) res+=query(idx[x]+1,idx[y]);

return res;

}

#undef type

}hld;

/\*\*\*\*\*\*\*\*\*\*\*attention!\*\*\*\*\*\*\*\*\*\*\*\*/

//hld.init(n)

//hld.add\_edge(): undirected edge.

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

## **1.10** KD-tree

namespace kd\_tree

{

const double alpha=0.75;

const int dim=2;

#define type int

const type NONE=INF; //初始值

struct kdtnode

{

bool exist;

int l,r,sz,fa,dep,x[dim],mx[dim],mn[dim];

type v,tag;

kdtnode(){}

void initval()

{

sz=exist;tag=v;

if(exist) for(int i=0;i<dim;i++) mn[i]=mx[i]=x[i];

}

void null()

{

exist=sz=0;

v=tag=NONE;

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

{

mx[i]=-INF;

mn[i]=INF;

}

}

void newnode(int x0,int x1,type val=NONE)

{

x[0]=x0;

x[1]=x1;

l=r=fa=0;

exist=1;

v=val;

initval();

}

kdtnode(int a,int b,type d=NONE){newnode(a,b,d);}

};

struct KDT

{

#define ls t[id].l

#define rs t[id].r

kdtnode t[MAX];

int tot,idx,root;

inline void pushup(int id)

{

t[id].initval();

t[id].sz+=t[ls].sz+t[rs].sz;

t[id].tag=min({t[ls].tag,t[rs].tag,t[id].tag});

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

{

if(ls)

{

t[id].mx[i]=max(t[id].mx[i],t[ls].mx[i]);

t[id].mn[i]=min(t[id].mn[i],t[ls].mn[i]);

}

if(rs)

{

t[id].mx[i]=max(t[id].mx[i],t[rs].mx[i]);

t[id].mn[i]=min(t[id].mn[i],t[rs].mn[i]);

}

}

}

bool isbad(int id){return t[id].sz\*alpha+3<max(t[ls].sz,t[rs].sz);}

int st[MAX],top;

void build(int &id,int l,int r,int fa,int dep=0)

{

id=0;if(l>r) return;

int m=(l+r)>>1; idx=dep;

nth\_element(st+l,st+m,st+r+1,[&](int x,int y){return t[x].x[idx]<t[y].x[idx];});

id=st[m];

build(ls,l,m-1,id,(dep+1)%dim);

build(rs,m+1,r,id,(dep+1)%dim);

pushup(id);

t[id].dep=dep;

t[id].fa=fa;

}

inline void init(int n=0)

{

root=0;

t[0].null();

for(int i=1;i<=n;i++) st[i]=i;

if(n) build(root,1,n,0);

tot=n;

}

void travel(int id)

{

if(!id) return;

if(t[id].exist) st[++top]=id;

travel(ls);

travel(rs);

}

void rebuild(int &id,int dep)

{

top=0;travel(id);

build(id,1,top,t[id].fa,dep);

}

void insert(int &id,int now,int fa,int dep=0)

{

if(!id)

{

id=now;

t[id].dep=dep;

t[id].fa=fa;

return;

}

if(t[now].x[dep]<t[id].x[dep]) insert(ls,now,id,(dep+1)%dim);

else insert(rs,now,id,(dep+1)%dim);

pushup(id);

if(isbad(id)) rebuild(id,t[id].dep);

t[id].dep=dep;

t[id].fa=fa;

}

inline void insert(kdtnode x){t[++tot]=x;insert(root,tot,0,0);}

inline void del(int id)

{

if(!id) return;

t[id].null();

int x=id;

while(x)

{

pushup(x);

x=t[x].fa;

}

if(isbad(id))

{

x=t[id].fa;

rebuild(root==id?root:(t[x].l==id?t[x].l:t[x].r),t[id].dep);

}

}

kdtnode q;

ll dist(ll x,ll y){return x\*x+y\*y;}

ll getdist(int id)//点q离区域t[id]最短距离

{

if(!id) return LLINF;

ll res=0;

if(q.x[0]<t[id].mn[0]) res+=dist(q.x[0]-t[id].mn[0],0);

if(q.x[1]<t[id].mn[1]) res+=dist(q.x[1]-t[id].mn[1],0);

if(q.x[0]>t[id].mx[0]) res+=dist(q.x[0]-t[id].mx[0],0);

if(q.x[1]>t[id].mx[1]) res+=dist(q.x[1]-t[id].mx[1],0);

return res;

}

kdtnode a,b;

inline int check(kdtnode &x)//x在矩形(a,b)内

{

int ok=1;

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

{

ok&=(x.x[i]>=a.x[i]);

ok&=(x.x[i]<=b.x[i]);

}

return ok;

}

inline int allin(kdtnode &x)//x的子树全在矩形(a,b)内

{

int ok=1;

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

{

ok&=(x.mn[i]>=a.x[i]);

ok&=(x.mx[i]<=b.x[i]);

}

return ok;

}

inline int allout(kdtnode &x)//x的子树全不在矩形(a,b)内

{

int ok=0;

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

{

ok|=(x.mx[i]<a.x[i]);

ok|=(x.mn[i]>b.x[i]);

}

return ok;

}

type res;

void query(int id)

{

if(!id) return;

if(allout(t[id])||t[id].sz==0) return;

if(allin(t[id]))

{

res=min(res,t[id].tag);

return;

}

if(check(t[id])&&t[id].exist) res=min(res,t[id].v);

int l,r;

l=ls;

r=rs;

if(t[l].tag>t[r].tag) swap(l,r);

if(t[l].tag<res) query(l);

if(t[r].tag<res) query(r);

}

inline type query(kdtnode \_a,kdtnode \_b)

{

a=\_a;b=\_b;

res=INF;

query(root);

return res;

}

}kd;

#undef type

#undef ls

#undef rs

}

using namespace kd\_tree;

## **1.11** k叉哈夫曼树

/\*

用两个队列代替优先队列 复杂度On

注意：小的先进 原数组记得先排序

\*/

int a[MAX];

int Huffman(int k)

{

int i,res,s;

queue<int> q,d;

s=((n-1)%(k-1)?k-1-(n-1)%(k-1):0);//计算要补多少个0

while(s--) q.push(0);

for(i=1;i<=n;i++) q.push(a[i]);

res=0;

while(sz(q)+sz(d)>1)

{

s=0;

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

{

if(sz(q)&&sz(d))

{

if(q.front()<d.front())

{

s+=q.front();

q.pop();

}

else

{

s+=d.front();

d.pop();

}

}

else if(sz(q))

{

s+=q.front();

q.pop();

}

else if(sz(d))

{

s+=d.front();

d.pop();

}

}

res+=s;

d.push(s);

}

return res;

}

# **2.**字符串

## **2.1** KMP

2.1.1 kmp

int Next[MAX];

void getnext(char \*b,int \*Next,int len)

{

int i,j;

i=0;

j=Next[0]=-1;

while(i<len)

{

if(j==-1||b[i]==b[j]) Next[++i]=++j;

else j=Next[j];

}

}

int KMP(char \*a,char \*b,int lena,int lenb)

{

int i,j;

getnext(b,Next,lenb);

i=j=0;

while(i<lena)

{

if(j==-1||a[i]==b[j])

{

i++;

j++;

}

else j=Next[j];

if(j==lenb) break;//successful match

}

return j==-1?0:j;

}

2.1.2 exkmp

struct exKMP

{

int next[MAX];

void getnext(char \*s)

{

int i,j,pos,len;

next[i=0]=len=strlen(s);

while(s[i]==s[i+1]&&i+1<len) i++;

next[1]=i;

pos=1;

for(i=2;i<len;i++)

{

if(next[i-pos]+i<next[pos]+pos) next[i]=next[i-pos];

else

{

j=max(0,next[pos]+pos-i);

while(i+j<len&&s[j]==s[j+i]) j++;

next[i]=j;

pos=i;

}

}

}

void work(char \*a,char \*b,int \*ex)

{

int i=0,j,pos,lena,lenb;

getnext(b);

lena=strlen(a);

lenb=strlen(b);

i=0;

while(a[i]==b[i]&&i<lenb&&i<lena) i++;

ex[0]=i;

pos=0;

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

{

if(next[i-pos]+i<ex[pos]+pos) ex[i]=next[i-pos];

else

{

j=max(0,ex[pos]+pos-i);

while(i+j<lena&&j<lenb&&a[j+i]==b[j]) j++;

ex[i]=j;

pos=i;

}

}

}

}exkmp;

## **2.2** manacher

2.2.1 插分隔符

struct Manacher

{

int p[MAX<<1];

char s[MAX<<1];

int work(char \*a)

{

int len,i,mid,r,res=0;

len=strlen(a+1);

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

{

p[i]=0;

s[2\*i-1]='%';

s[2\*i]=a[i];

}

s[len=len\*2+1]='%';

mid=r=0;

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

{

if(i<r) p[i]=min(p[2\*mid-i],r-i);

else p[i]=1;

while(i-p[i]>=1&&i+p[i]<=len&&s[i-p[i]]==s[i+p[i]]) p[i]++;

if(i+p[i]>r)

{

r=i+p[i];

mid=i;

}

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

}

return res;

}

}la;

2.2.2 不插分隔符

struct Manacher

{

int p[MAX];

int work(char \*s)//return max length of palindrome

{

int r,mid,i,len,res=0;

len=strlen(s+1);

//odd

r=mid=0;

mem(p,0);

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

{

//substring s[i,i]

if(r>i) p[i]=min(p[2\*mid-i],r-i);

while(i+p[i]+1<=len&&s[i+p[i]+1]==s[i-p[i]-1])

{

//substring s[i-p[i]-1,i+p[i]+1]

p[i]++;

}

if(i+p[i]>r)

{

r=i+p[i];

mid=i;

}

res=max(res,p[i]\*2+1);

}

//even

r=mid=0;

mem(p,0);

for(i=2;i<=len;i++)

{

if(r>i) p[i]=min(p[2\*mid-i],r-i+1);

while(i+p[i]<=len&&s[i+p[i]]==s[i-p[i]-1])

{

//substring s[i-p[i]-1,i+p[i]]

p[i]++;

}

if(i+p[i]-1>r)

{

r=i+p[i]-1;

mid=i;

}

res=max(res,p[i]\*2);

}

return res;

}

}la;

## **2.3** AC自动机

struct AC\_Automaton

{

static const int K=26;//may need change

int next[MAX][K],fail[MAX],cnt[MAX],last[MAX];

int root,tot;

inline int getid(char c)//may need change

{

return c-'a';

}

int newnode()

{

mem(next[tot],0);

fail[tot]=0;

cnt[tot]=0;

return tot++;

}

void init()

{

tot=0;

root=newnode();

}

void insert(char \*s)

{

int len,now,i;

len=strlen(s);

now=root;

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

{

int t=getid(s[i]);

if(!next[now][t]) next[now][t]=newnode();

now=next[now][t];

}

cnt[now]++;

}

void setfail()

{

int i,now;

queue<int>q;

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

{

if(next[root][i]) q.push(next[root][i]);

}

while(!q.empty())

{

now=q.front();

q.pop();

//suffix link

if(cnt[fail[now]]) last[now]=fail[now];

else last[now]=last[fail[now]];

/\*

may need add something here:

cnt[now]+=cnt[fail[now]];

\*/

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

{

if(next[now][i])

{

fail[next[now][i]]=next[fail[now]][i];

q.push(next[now][i]);

}

else next[now][i]=next[fail[now]][i];

}

}

}

int query(char \*s)

{

int len,now,i,res;

len=strlen(s);

now=root;

res=0;

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

{

int t=getid(s[i]);

now=next[now][t];

int tmp=now;

while(tmp&&cnt[tmp]!=-1)

{

res+=cnt[tmp];

cnt[tmp]=-1;

tmp=last[tmp];

}

}

return res;

}

//build fail tree

vector<int> mp[MAX];

void build\_tree()

{

for(int i=0;i<=tot;i++) mp[i].clear();

for(int i=1;i<tot;i++) mp[fail[i]].pb(i);

}

}ac;

## **2.4** hash

2.4.1 hash

//seed通常rand()一个

//p通常用1e9+7 或者1e9+9

//ull比mod快

struct hash\_table

{

ll seed,p;

ll Hash[MAX],tmp[MAX];

void set(ll \_seed,ll \_p)

{

seed=\_seed;

p=\_p;

}

void work(char \*s,int n)

{

tmp[0]=1;

Hash[0]=0;

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

{

tmp[i]=tmp[i-1]\*seed%p;

Hash[i]=(Hash[i-1]\*seed+(s[i]-'a'))%p;//may need change

}

}

ll get(int l,int r)

{

return ((Hash[r]-Hash[l-1]\*tmp[r-l+1])%p+p)%p;

}

};

2.4.2 BKDRHash

struct BKDRHash

{

static const ull seed=1313131;//31,131,1313,13131,131313

static const int p=2000007;

ull Hash[MAX],tmp[MAX];

ull val[MAX];

int last[p+10],nex[MAX],cnt;

void init()//clear hash table

{

mem(last,0);

cnt=0;

}

bool insert(ull x)

{

int u=x%p;

for(int i=last[u];i;i=nex[i])

{

if(val[i]==x) return 1;

}

nex[++cnt]=last[u];

last[u]=cnt;

val[cnt]=x;

return 0;

}

void work(char \*s,int n)

{

tmp[0]=1;

Hash[0]=0;

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

{

tmp[i]=tmp[i-1]\*seed;

Hash[i]=Hash[i-1]\*seed+s[i];

}

}

ull get(int l,int r)

{

return Hash[r]-Hash[l-1]\*tmp[r-l+1];

}

}bkdr; //bkdr.init();

2.4.3 Hash\_map

struct Hash\_map

{

static const int p=999917;

ll val[MAX],w[MAX];

int tot,head[p],nex[MAX];

int top,st[MAX];

void clear(){tot=0;while(top) head[st[top--]]=0;}

void add(int x,ll y){val[++tot]=y;nex[tot]=head[x];head[x]=tot;w[tot]=0;}

bool count(ll y)

{

int x=y%p;

for(int i=head[x];i;i=nex[i])

{

if(y==val[i]) return 1;

}

return 0;

}

ll& operator [](ll y)

{

int x=y%p;

for(int i=head[x];i;i=nex[i])

{

if(y==val[i]) return w[i];

}

add(x,y);

st[++top]=x;

return w[tot];

}

}mp;

## **2.5** 回文树

struct Palindrome\_Tree

{

int len[MAX],next[MAX][26],fail[MAX],last,s[MAX],tot,n;

int cnt[MAX],deep[MAX];

int newnode(int l)

{

mem(next[tot],0);

fail[tot]=0;

deep[tot]=cnt[tot]=0;

len[tot]=l;

return tot++;

}

void init()

{

tot=n=last=0;

newnode(0);

newnode(-1);

s[0]=-1;

fail[0]=1;

}

int get\_fail(int x)

{

while(s[n-len[x]-1]!=s[n]) x=fail[x];

return x;

}

void add(int t)//attention the type of t is int

{

int id,now;

s[++n]=t;

now=get\_fail(last);

if(!next[now][t])

{

id=newnode(len[now]+2);

fail[id]=next[get\_fail(fail[now])][t];

deep[id]=deep[fail[id]]+1;

next[now][t]=id;

}

last=next[now][t];

cnt[last]++;

}

void count()

{

for(int i=tot-1;~i;i--) cnt[fail[i]]+=cnt[i];

}

}pam; //pam.init();

# **3.**图论

## **3.1 链式前向星**

//注意 这里MAX指的是边数 双向边要\*2

int head[MAX],tot;

struct node

{

int to,v,next;

}mp[MAX];

void init()

{

mem(head,-1);

tot=0;

}

void add(int x,int y,int v)

{

mp[tot].v=v;

mp[tot].to=y;

mp[tot].next=head[x];

head[x]=tot++;

}

## **3.2 最短路**

3.2.1 Dijkstra

struct node

{

int id;

int v;

node(){}

node(int a,int b) :id(a),v(b){}

friend bool operator <(node a,node b){return a.v>b.v;}

};

vector<node> mp[MAX];

bool flag[MAX];

int dis[MAX];

void dij(int s)

{

priority\_queue<node> q;

node t,to;

mem(dis,0x3f);

mem(flag,0);

dis[s]=0;

q.push(node(s,0));

while(!q.empty())

{

t=q.top();

q.pop();

if(flag[t.id]) continue;

flag[t.id]=1;

for(int i=0;i<sz(mp[t.id]);i++)

{

to=mp[t.id][i];

if(dis[to.id]>dis[t.id]+to.v)

{

dis[to.id]=dis[t.id]+to.v;

q.push(node(to.id,dis[to.id]));

}

}

}

}

3.2.2 spfa

//最长路 dis变为-INF 松弛改成<

struct node

{

int id;

int v;

node(){}

node(int a,int b) :id(a),v(b){}

};

vector<node> mp[MAX];

int dis[MAX];

bool flag[MAX];

void spfa(int s)

{

queue<node> q;

node t,to;

mem(dis,0x3f);

mem(flag,0);

dis[s]=0;

flag[s]=1;

q.push(node(s,dis[s]));

while(!q.empty())

{

t=q.front();

q.pop();

flag[t.id]=0;

for(int i=0;i<sz(mp[t.id]);i++)

{

to=mp[t.id][i];

if(dis[to.id]>dis[t.id]+to.v)

{

dis[to.id]=dis[t.id]+to.v;

if(!flag[to.id])

{

q.push(node(to.id,dis[to.id]));

flag[to.id]=1;

}

}

}

}

}

## **3.3 floyd求最小环 hdu1599**

int mp[111][111],dis[111][111],ans,n;

void floyd()

{

int i,j,k;

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

{

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

{

if(mp[k][i]==INF) continue;

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

{

if(mp[k][j]==INF) continue;

ans=min(ans,mp[k][i]+mp[k][j]+dis[i][j]);

}

}

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

{

if(dis[i][k]==INF) continue;

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

{

if(dis[k][j]==INF) continue;

dis[i][j]=min(dis[i][j],dis[i][k]+dis[k][j]);

}

}

}

}

int main()

{

int m,i,a,b,w;

while(~scanf("%d%d",&n,&m))

{

mem(mp,0x3f);

mem(dis,0x3f);

ans=INF;

while(m--)

{

scanf("%d%d%d",&a,&b,&w);

mp[a][b]=mp[b][a]=dis[a][b]=dis[b][a]=min(mp[a][b],w);

}

floyd();

if(ans==INF) puts("It's impossible.");

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

}

return 0;

}

## **3.4 最小生成树**

3.4.1 prim

struct node

{

int id;

int v;

node(){}

node(int a,int b) :id(a),v(b){}

friend bool operator <(node a,node b){return a.v>b.v;}

};

vector<node> mp[MAX];

bool flag[MAX];

int dis[MAX],ans;

void prim()

{

node t,to;

priority\_queue<node> q;

mem(dis,0x3f);

mem(flag,0);

dis[1]=0;

q.push(node(1,dis[1]));

ans=0;

while(!q.empty())

{

t=q.top();

q.pop();

if(flag[t.id]) continue;

flag[t.id]=1;

ans+=dis[t.id];

for(int i=0;i<sz(mp[t.id]);i++)

{

to=mp[t.id][i];

if(!flag[to.id]&&dis[to.id]>to.v)

{

dis[to.id]=to.v;

q.push(node(to.id,dis[to.id]));

}

}

}

}

3.4.2 kruskal

int pre[MAX],ans;

struct node

{

int x,y,v;

node(){}

node(int a,int b,int c):x(a),y(b),v(c){}

friend bool operator<(node a,node b)

{

return a.v<b.v;

}

}a[MAX];

void init(int n)

{

for(int i=1;i<=n;i++) pre[i]=i;

}

int find(int x)

{

if(pre[x]!=x) pre[x]=find(pre[x]);

return pre[x];

}

void merge(node s)

{

int rx,ry;

rx=find(s.x);

ry=find(s.y);

if(rx!=ry)

{

pre[rx]=ry;

ans+=s.v;

}

}

void kruskal(int n,int m,node \*s)

{

init(n);

ans=0;

sort(s,s+m);

for(int i=0;i<m;i++) merge(s[i]);

}

## **3.5 最小树形图**(待补)

## **3.6 二分图匹配**

最小点覆盖的点数=最大匹配数

最小路径覆盖的边数=顶点数n-最大匹配数

最大独立集=最小路径覆盖=顶点数n-最大匹配数

3.6.1 匈牙利算法

vector<int> mp[MAX];

int link[MAX],used[MAX];

int dfs(int x)

{

int i,to;

for(i=0;i<sz(mp[x]);i++)

{

to=mp[x][i];

if(!used[to])

{

used[to]=1;

if(link[to]==-1||dfs(link[to]))

{

link[to]=x;

return 1;

}

}

}

return 0;

}

int hungary(int n)//返回最大匹配数

{

mem(link,-1);

int i,res=0;

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

{

mem(used,0);

if(dfs(i)) res++;

}

return res;

}

## **3.7 网络流**

3.7.1 Dinic

struct Dinic

{

static const int N=1010;

struct node

{

int from,to,cap,flow;

node(int u,int v,int c,int f) :from(u),to(v),cap(c),flow(f){}

};

int s,t;

vector<node> edge;

vector<int> mp[N];

int vis[N],dist[N],id[N];

void init(int n)

{

edge.clear();

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

{

mp[i].clear();

id[i]=dist[i]=vis[i]=0;

}

}

void add(int from,int to,int cap)

{

edge.pb(node(from,to,cap,0));

edge.pb(node(to,from,0,0));

int m=edge.size();

mp[from].pb(m-2);

mp[to].pb(m-1);

}

bool bfs()

{

int i,x;

mem(vis,0);

queue<int>q;

q.push(s);

dist[s]=0;

vis[s]=1;

while(!q.empty())

{

x=q.front();

q.pop();

for (i=0;i<mp[x].size();i++)

{

node &e=edge[mp[x][i]];

if(!vis[e.to]&&e.cap>e.flow)

{

vis[e.to]=1;

dist[e.to]=dist[x]+1;

q.push(e.to);

}

}

}

return vis[t];

}

int dfs(int x,int a)

{

if(x==t||!a)return a;

int flow = 0, f;

for(int &i=id[x];i<mp[x].size();i++)

{

node &e=edge[mp[x][i]];

if(dist[x]+1==dist[e.to]&&(f=dfs(e.to,min(a,e.cap-e.flow)))>0)

{

e.flow+=f;

edge[mp[x][i]^1].flow-=f;

flow+=f;

a-=f;

if(!a) break;

}

}

return flow;

}

int maxflow(int \_s,int \_t)

{

s=\_s;

t=\_t;

int res=0;

while(bfs())

{

mem(id,0);

res+=dfs(s,INF);

}

return res;

}

}dc;

3.7.2 ISAP

struct ISAP

{

static const int N=100010;

struct node

{

int from,to,cap,flow;

node(){}

node(int u,int v,int c,int f):from(u),to(v),cap(c),flow(f){}

};

int p[N],num[N],cur[N],d[N];

int t,s,n;

bool vis[N];

vector<int> mp[N];

vector<node> edge;

void init(int \_n)

{

n=\_n;

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

{

mp[i].clear();

d[i]=INF;

vis[i]=num[i]=cur[i]=0;

}

edge.clear();

}

void add(int from,int to,int cap)

{

edge.pb(node(from,to,cap,0));

edge.pb(node(to,from,0,0));

int m=edge.size();

mp[from].pb(m-2);

mp[to].pb(m-1);

}

bool bfs()

{

queue<int> q;

d[t]=0;

vis[t]=1;

q.push(t);

while(!q.empty())

{

int u=q.front();

q.pop();

for(int i=0;i<mp[u].size();i++)

{

node &e=edge[mp[u][i]^1];

if(!vis[e.from]&&e.cap>e.flow)

{

vis[e.from]=true;

d[e.from]=d[u]+1;

q.push(e.from);

}

}

}

return vis[s];

}

int augment()

{

int u=t,flow=INF;

while(u!=s)

{

node &e=edge[p[u]];

flow=min(flow,e.cap-e.flow);

u=edge[p[u]].from;

}

u=t;

while(u!=s)

{

edge[p[u]].flow+=flow;

edge[p[u]^1].flow-=flow;

u=edge[p[u]].from;

}

return flow;

}

int maxflow(int \_s,int \_t)

{

s=\_s;

t=\_t;

int flow=0;

bfs();

if(d[s]>=n) return 0;

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

{

if(d[i]<INF) num[d[i]]++;

}

int u=s;

while(d[s]<n)

{

if(u==t)

{

flow+=augment();

u=s;

}

bool ok=false;

for(int i=cur[u];i<mp[u].size();i++)

{

node &e=edge[mp[u][i]];

if(e.cap>e.flow&&d[u]==d[e.to]+1)

{

ok=true;

p[e.to]=mp[u][i];

cur[u]=i;

u=e.to;

break;

}

}

if(!ok)

{

int mn=n-1;

for(int i=0;i<mp[u].size();i++)

{

node &e=edge[mp[u][i]];

if(e.cap>e.flow) mn=min(mn,d[e.to]);

}

if(--num[d[u]]==0) break;

num[d[u]=mn+1]++;

cur[u]=0;

if(u!=s) u=edge[p[u]].from;

}

}

return flow;

}

}isap;

3.7.3 High Level Preflow Push

struct High\_Level\_Preflow\_Push

{

static const int N=10010;

struct node{int v,cap,index;};

vector<node> edge[N];

vector<int> List[N];

vector<list<int>::iterator> listit;

list<int> dlist[N];

int highest,highestActive,vis[N],excess[N],height[N],n;

void init(int \_n)

{

n=\_n;

for(int i=0;i<=n;i++) edge[i].clear();

}

void add(int u,int v,int cap)

{

edge[u].push\_back(node{v,cap,edge[v].size()});

edge[v].push\_back(node{u,0,edge[u].size()-1});

}

void globalRelabel(int t)

{

int u,i,hp,v,index;

queue<int> q;

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

{

height[i]=n;

List[i].clear();

dlist[i].clear();

vis[i]=0;

}

height[t]=0;

q.push(t);

while(!q.empty())

{

u=q.front();

q.pop();

for(i=0;i<edge[u].size();i++)

{

v=edge[u][i].v;

index=edge[u][i].index;

if(height[v]==n&&edge[v][index].cap>0)

{

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

vis[height[v]]++;

q.push(hp=v);

}

}

}

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

{

if(height[i]<n)

{

listit[i]=dlist[height[i]].insert(dlist[height[i]].begin(),i);

if(excess[i]>0) List[height[i]].push\_back(i);

}

}

highest=height[hp];

highestActive=height[hp];

}

void push(int u,node &e)

{

int v,df;

v=e.v;

df=min(excess[u],e.cap);

e.cap=e.cap-df;

edge[v][e.index].cap=edge[v][e.index].cap+df;

excess[u]=excess[u]-df;

excess[v]=excess[v]+df;

if(excess[v]>0&&excess[v]<=df) List[height[v]].push\_back(v);

}

void discharge(int u)

{

int i,nh,v,cap,h;

nh=n;

for(i=0;i<edge[u].size();i++)

{

v=edge[u][i].v;

cap=edge[u][i].cap;

if(cap>0)

{

if(height[u]==height[v]+1)

{

push(u,edge[u][i]);

if(excess[u]==0) return;

}

else nh=min(nh,height[v]+1);

}

}

h=height[u];

if(vis[h]==1)

{

for(i=h;i<=highest;i++)

{

for(list<int>::iterator it=dlist[i].begin();it!=dlist[i].end();it++)

{

vis[height[\*it]]--;

height[\*it]=n;

}

dlist[i].clear();

}

highest=h-1;

}

else

{

vis[h]--;

listit[u]=dlist[h].erase(listit[u]);

height[u]=nh;

if(nh==n) return;

vis[nh]++;

listit[u]=dlist[nh].insert(dlist[nh].begin(),u);

highestActive=nh;

highest=max(highest,highestActive);

List[nh].push\_back(u);

}

}

int maxflow(int s,int e)

{

int i,u;

if(s==e) return 0;

highestActive=0;

highest=0;

for(i=0;i<=n;i++) height[i]=vis[i]=excess[i]=0;

height[s]=n;

listit.resize(n);

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

{

if(i!=s)

{

listit[i]=dlist[height[i]].insert(dlist[height[i]].begin(),i);

}

}

vis[0]=n-1;

excess[s]=INF;

excess[e]=-INF;

for(i=0;i<edge[s].size();i++) push(s,edge[s][i]);

globalRelabel(e);

while(highestActive>=0)

{

if(List[highestActive].empty()==1)

{

highestActive--;

continue;

}

u=List[highestActive].back();

List[highestActive].pop\_back();

discharge(u);

}

return excess[e]+INF;

}

}hlpp;

## **3.8 费用流**

3.8.1 Min Cost Max Flow

struct MCMF

{

#define type int //int->INF ll->LLINF

static const int N=2005;

struct node

{

int from,to;

type cap,flow,cost;

node(){}

node(int u,int v,type c,type f,type co):from(u),to(v),cap(c),flow(f),cost(co){}

};

int n,s,t;

vector<node> edge;

vector<int> mp[N];

int vis[N],id[N];

type d[N],a[N];

void init(int \_n)

{

n=\_n;

for(int i=0;i<=n;i++) mp[i].clear();

edge.clear();

}

void add(int from,int to,type cap,type cost=0)

{

edge.pb(node(from,to,cap,0,cost));

edge.pb(node(to,from,0,0,-cost));

int m=edge.size();

mp[from].pb(m-2);

mp[to].pb(m-1);

}

bool spfa(type& flow,type& cost)

{

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

{

d[i]=INF;

vis[i]=0;

}

d[s]=0;vis[s]=1;id[s]=0;a[s]=INF;

queue<int> q;

q.push(s);

while(!q.empty())

{

int x=q.front();

q.pop();

vis[x]=0;

for(int i=0;i<mp[x].size();i++)

{

node& e=edge[mp[x][i]];

int to=e.to;

if(e.cap>e.flow&&d[to]>d[x]+e.cost)

{

d[to]=d[x]+e.cost;

a[to]=min(a[x],e.cap-e.flow);

id[to]=mp[x][i];

if(!vis[to])

{

vis[to]=1;

q.push(to);

}

}

}

}

if(d[t]==INF) return false;

flow+=a[t];

cost+=a[t]\*d[t];

int x=t;

while(x!=s)

{

edge[id[x]].flow+=a[t];

edge[id[x]^1].flow-=a[t];

x=edge[id[x]].from;

}

return true;

}

pair<type,type> mincost\_maxflow(int \_s,int \_t)

{

type flow=0,cost=0;

s=\_s;

t=\_t;

while(spfa(flow,cost));

return MP(cost,flow);

}

#undef type

}mcmf;

## **3.9 强连通分量**

int scc,top,tot;

vector<int> mp[MAX];

int low[MAX],dfn[MAX],belong[MAX];

int stk[MAX],flag[MAX];

void init(int n)

{

int i;

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

{

mp[i].clear();

low[i]=0;

dfn[i]=0;

stk[i]=0;

flag[i]=0;

}

scc=top=tot=0;

}

void tarjan(int x)

{

int to,i,temp;

stk[top++]=x;

flag[x]=1;

low[x]=dfn[x]=++tot;

for(i=0;i<mp[x].size();i++)

{

to=mp[x][i];

if(!dfn[to])

{

tarjan(to);

low[x]=min(low[x],low[to]);

}

else if(flag[to]) low[x]=min(low[x],dfn[to]);

}

if(low[x]==dfn[x])

{

scc++;

do

{

temp=stk[--top];

flag[temp]=0;

belong[temp]=scc;

}while(temp!=x);

}

}

## **3.10 双连通分量**

3.10.1 边双连通-桥-割点

namespace Tarjan

{

int bcc,top,tot,n;

vector<int> mp[MAX];

vector<PII > bridge;

int low[MAX],dfn[MAX],belong[MAX],fa[MAX];

int stk[MAX];

int cut[MAX],add\_block[MAX];

void dfs(int x,int pre)

{

int to,i,tmp,k,son;

stk[top++]=x;

low[x]=dfn[x]=++tot;

fa[x]=pre;

son=k=0;

for(auto to:mp[x])

{

if(to==pre&&!k)

{

k++;

continue;

}

if(!dfn[to])

{

son++;

dfs(to,x);

low[x]=min(low[x],low[to]);

if(x!=pre&&low[to]>=dfn[x])

{

cut[x]=1;

add\_block[x]++;

}

if(low[to]>dfn[x]) bridge.pb(MP(x,to));

}

else low[x]=min(low[x],dfn[to]);

}

if(x==pre&&son>1)

{

cut[x]=1;

add\_block[x]=son-1;

}

if(low[x]==dfn[x])

{

bcc++;

do

{

tmp=stk[--top];

belong[tmp]=bcc;

}while(tmp!=x);

}

}

void work(int \_n,vector<int> e[])

{

n=\_n;

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

{

mp[i]=e[i];

low[i]=dfn[i]=fa[i]=stk[i]=0;

cut[i]=add\_block[i]=0;

}

bcc=top=tot=0;

bridge.clear();

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

{

if(!dfn[i]) dfs(i,i);

}

}

void rebuild(vector<int> e[])

{

int i,t;

for(i=1;i<=n;i++) e[i].clear();

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

{

t=fa[i];

if(belong[i]!=belong[t])

{

e[belong[i]].pb(belong[t]);

e[belong[t]].pb(belong[i]);

}

}

}

}

## **3.11 2-sat**

3.11.1 染色法 输出字典序最小的解 O(n\*m)

vector<int> mp[MAX];

bool flag[MAX];

int cnt,s[MAX];

void init(int n)

{

int i;

for(i=0;i<2\*n;i++)

{

mp[i].clear();

}

mem(flag,0);

}

bool dfs(int x)

{

int i;

if(flag[x^1]) return 0;

if(flag[x]) return 1;

s[cnt++]=x;

flag[x]=1;

for(i=0;i<sz(mp[x]);i++)

{

if(!dfs(mp[x][i])) return 0;

}

return 1;

}

void twosat(int n)

{

int i;

for(i=0;i<2\*n;i++)

{

if(!flag[i]&&!flag[i^1])

{

cnt=0;

if(!dfs(i))

{

while(cnt) flag[s[--cnt]]=0;

if(!dfs(i^1))//无解

{

puts("NO");

return;

}

}

}

}

for(i=0;i<2\*n;i+=2)

{

if(flag[i]) printf("%d\n",i+1);

else printf("%d\n",i+2);

}

}

3.10.2 强连通缩点法 输出任意一组解 O(n+m)

int scc,top,tot;

vector<int> mp[MAX];

int low[MAX],dfn[MAX],belong[MAX];

int stk[MAX],flag[MAX];

int pos[MAX],degree[MAX],ans[MAX],outflag[MAX],cnt;

vector<int> dag[MAX];

void init(int n)

{

int i;

for(i=0;i<2\*n;i++)

{

mp[i].clear();

dag[i].clear();

low[i]=0;

dfn[i]=0;

stk[i]=0;

flag[i]=0;

degree[i]=0;

outflag[i]=0;

}

scc=top=tot=0;

}

void tarjan(int x)

{

int to,i,temp;

stk[top++]=x;

flag[x]=1;

low[x]=dfn[x]=++tot;

for(i=0;i<mp[x].size();i++)

{

to=mp[x][i];

if(!dfn[to])

{

tarjan(to);

low[x]=min(low[x],low[to]);

}

else if(flag[to]) low[x]=min(low[x],dfn[to]);

}

if(low[x]==dfn[x])

{

scc++;

do

{

temp=stk[--top];

flag[temp]=0;

belong[temp]=scc;

}while(temp!=x);

}

}

void add(int x,int y)

{

mp[x].pb(y);

}

void topsort(int n)

{

int i,t;

queue<int> q;

cnt=0;

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

{

if(degree[i]==0) q.push(i);

outflag[i]=0;

}

while(!q.empty())

{

t=q.front();

q.pop();

if(outflag[t]==0)

{

outflag[t]=1;

outflag[pos[t]]=2;

}

for(i=0;i<sz(dag[t]);i++)

{

int to=dag[t][i];

degree[to]--;

if(degree[to]==0) q.push(to);

}

}

}

void builddag(int n)

{

int i,j,to;

for(i=0;i<2\*n;i++)

{

for(j=0;j<sz(mp[i]);j++)

{

to=mp[i][j];

if(belong[i]!=belong[to])

{

degree[belong[i]]++;

dag[belong[to]].pb(belong[i]);

}

}

}

}

void twosat(int n)

{

int i;

for(i=0;i<2\*n;i++)

{

if(!dfn[i]) tarjan(i);

}

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

{

if(belong[2\*i]==belong[2\*i+1])//无解

{

puts("NO");

return;

}

pos[belong[2\*i]]=belong[2\*i+1];

pos[belong[2\*i+1]]=belong[2\*i];

}

builddag(n);

topsort(n);

cnt=0;

for(i=0;i<2\*n;i++)

{

if(outflag[belong[i]]==1) ans[cnt++]=i+1;

}

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

{

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

}

}

# **4.**数论

## **4.1 素数筛与分解质因数**

/\*

prime[x]表示x的最小质因数(x>=2)

prime[x]==x(x>=2)，表示x是素数。

\*/

4.1.1 埃氏筛

int p[MAX],tot,prime[MAX];

void init(int n)

{

int i,j;

tot=0;

mem(prime,0);

prime[1]=1;

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

{

if(prime[i]) continue;

p[tot++]=i;

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

{

if(!prime[j]) prime[j]=i;

}

}

}

4.1.2 线性筛

int p[MAX],tot,prime[MAX];

void init(int n)

{

int i,j;

tot=0;

mem(prime,0);

prime[1]=1;

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

{

if(!prime[i]) prime[i]=p[tot++]=i;

for(j=0;j<tot&&p[j]\*i<=n;j++)

{

prime[i\*p[j]]=p[j];

if(i%p[j]==0) break;

}

}

}

4.1.3 区间筛

ll p[MAX],tot;

bool flag[MAX],prime[MAX];

void init(ll l,ll r)

{

ll i,j,sq=sqrt(r+0.5);

tot=0;

for(i=0;i<=sq;i++) flag[i]=1;

for(i=l;i<=r;i++) prime[i-l]=1;

if(l==0) prime[0]=prime[1]=0;

if(l==1) prime[0]=0;

for(i=2;i<=sq;i++)

{

if(!flag[i]) continue;

for(j=i+i;j<=sq;j+=i) flag[j]=0;

for(j=max(2LL,(l+i-1)/i)\*i;j<=r;j+=i) prime[j-l]=0;

}

for(i=l;i<=r;i++)

{

if(prime[i-l]) p[tot++]=i;

}

}

4.1.4 分解质因数

vector<int> res;

void work(int x)

{

int t;

res.clear();

while(x>1)

{

t=prime[x];

while(x%t==0&&x>1) x/=t;

res.pb(t);

}

}

## **4.2 Miller\_Rabin + Pollard\_rho**

const int S=20;

mt19937 rd(time(0));

ll mul2(ll a,ll b,ll p)

{

ll res=0;

while(b)

{

if(b&1) res=(res+a)%p;

a=(a+a)%p;

b>>=1;

}

return res;

}

ll pow2(ll a,ll b,ll p)

{

ll res=1;

while(b)

{

if(b&1) res=mul2(res,a,p);

a=mul2(a,a,p);

b>>=1;

}

return res;

}

int check(ll a,ll n,ll x,ll t)//一定是合数返回1,不一定返回0

{

ll now,nex,i;

now=nex=pow2(a,x,n);

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

{

now=mul2(now,now,n);

if(now==1&&nex!=1&&nex!=n-1) return 1;

nex=now;

}

if(now!=1) return 1;

return 0;

}

int Miller\_Rabin(ll n)

{

if(n<2) return 0;

if(n==2) return 1;

if((n&1)==0) return 0;

ll x,t,i;

x=n-1;

t=0;

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

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

{

if(check(rd()%(n-1)+1,n,x,t)) return 0;

}

return 1;

}

ll Pollard\_rho(ll x,ll c)

{

ll i,k,g,t,y;

i=1;

k=2;

y=t=rd()%x;

while(1)

{

i++;

t=(mul2(t,t,x)+c)%x;

g=\_\_gcd(y-t+x,x);

if(g!=1&&g!=x) return g;

if(y==t) return x;

if(i==k)

{

y=t;

k+=k;

}

}

}

vector<ll> fac;

void findfac(ll n)

{

if(Miller\_Rabin(n))

{

fac.pb(n);

return;

}

ll t=n;

while(t>=n) t=Pollard\_rho(t,rd()%(n-1)+1);

findfac(t);

findfac(n/t);

}

void work(ll x)

{

fac.clear();

findfac(x);

}

## **4.3 exgcd**

4.3.1 exgcd

/\*

解xa+yb=gcd(a,b)

返回值为gcd(a,b)

其中一组解为x y

通解:

x1=x+b/gcd(a,b)\*t

y1=y-a/gcd(a,b)\*t

(t为任意整数)

\*/

ll exgcd(ll a,ll b,ll &x,ll &y)

{

if(b==0)

{

x=1;

y=0;

return a;

}

ll g,t;

g=exgcd(b,a%b,x,y);

t=x;

x=y;

y=t-a/b\*y;

return g;

}

4.3.2 xa+yb=c

//xa+yb=c 有解条件 c%gcd(a,b)==0

ll linear\_equation(ll a,ll b,ll c,ll &x,ll &y)

{

ll g,t;

g=exgcd(a,b,x,y);

if(!c) x=y=0;

else if((!a&&!b&&c)||c%g) return -1;//no solution

else if(!a&&b) x=1,y=c/b;

else if(a&&!b) x=c/a,y=-c/a;

else

{

a/=g,b/=g,c/=g;

x\*=c,y\*=c;

t=x;

x%=b;

if(x<=0) x+=b;//or x<0

ll k=(t-x)/b;

y+=k\*a;

}

return g;

}

## **4.4 逆元**

4.4.1 费马小定理

//条件:mod为素数

ll inv(ll x,ll p){return pow2(x,p-2);}

4.4.2 扩展欧几里得

/\*

条件:gcd(a,mod)==1

如果gcd(a,mod)!=1 返回-1

\*/

ll inv(ll a,ll p)

{

ll g,x,y;

g=exgcd(a,p,x,y);

return g==1?(x+p)%p:-1;

}

4.4.3 公式

/\*

a/b%mod=c

->a%(b\*mod)/b=c

\*/

4.4.4 逆元打表

/\*

p是模

p要求是奇素数

\*/

ll inv[MAX];

void getinv(int n,ll p)

{

ll i;

inv[1]=1;

for(i=2;i<=n;i++) inv[i]=(p-p/i)\*inv[p%i]%p;

}

## **4.5 中国剩余定理**

//m是除数 r是余数 p是除数的LCM(也就是答案的循环节)

int CRT(int \*m,int \*r,int n)

{

int p=m[0],res=r[0],x,y,g;

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

{

g=exgcd(p,m[i],x,y);

if((r[i]-res)%g) return -1;//无解

x=(r[i]-res)/g\*x%(m[i]/g);

res+=x\*p;

p=p/g\*m[i];

res%=p;

}

return res>0?res:res+p;

}

## **4.6 欧拉函数**

//应用： <=n且与n互质的数的和：n\*phi[n]/2

4.6.1 直接求某个数的欧拉函数 O(sqrt(n))

int Euler(int n)

{

int ans,i;

ans=n;

for(i=2;i\*i<=n;i++)

{

if(n%i==0)

{

ans=ans-ans/i;

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

}

}

if(n>1) ans=ans-ans/n;

return ans;

}

4.6.2 线性筛 O(n)

int prime[MAX],phi[MAX],cnt;

bool flag[MAX];

void Euler(int n)

{

int i,j,k;

cnt=0;

mem(flag,0);

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

{

if(!flag[i])

{

prime[cnt++]=i;

phi[i]=i-1;

}

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

{

k=i\*prime[j];

flag[k]=1;

if(i%prime[j]==0)

{

phi[k]=phi[i]\*prime[j];

break;

}

else phi[k]=phi[i]\*(prime[j]-1);

}

}

}

## **4.7 莫比乌斯函数**

int flag[MAX],mo[MAX],prime[MAX],cnt;

void init()

{

int i,j,cnt;

mem(flag,0);

mem(mo,0);

cnt=0;

for(i=2;i<=MAX;i++)

{

if(!flag[i])

{

prime[cnt++]=i;

mo[i]=-1;

for(j=i+i;j<=MAX;j+=i)

{

flag[j]++;

}

}

}

mo[1]=1;

for(i=2;2\*i<=MAX;i++)

{

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

{

if(i%prime[j]==0)

{

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

break;

}

mo[prime[j]\*i]=-mo[i];

}

}

}

## **4.8 组合数**

4.8.1 小范围

ll C[1010][1010];

void init(int n)

{

int i,j;

for(i=(C[0][0]=1);i<=n;i++)

{

for(j=(C[i][0]=1);j<=n;j++)

{

C[i][j]=(C[i-1][j]+C[i-1][j-1])%mod;

}

}

}

4.8.2 大范围

ll pow2(ll a,ll b)

{

ll res=1;

while(b)

{

if(b&1) res=res\*a%mod;

a=a\*a%mod;

b>>=1;

}

return res;

}

ll inv(ll x){return pow2(x,mod-2);}

ll fac[MAX],invfac[MAX];

void init(int n)

{

ll i;

fac[0]=1;

for(i=1;i<=n;i++) fac[i]=fac[i-1]\*i%mod;

invfac[n]=inv(fac[n]);

for(i=n-1;~i;i--) invfac[i]=invfac[i+1]\*(i+1)%mod;

}

ll C(int n,int m)

{

if(m>n||m<0||n<0) return 0;

return fac[n]\*invfac[m]%mod\*invfac[n-m]%mod;

}

## **4.9 Lucas定理**

//C(n,m) n,m<=1e18 p<=1e5

//p must be a prime number

ll Lucas(ll n,ll m,ll p)

{

if(m==0) return 1;

return C(n%p,m%p)\*Lucas(n/p,m/p,p)%p;

}

## **4.10 第二类Stirling数**

//dp[i][j]表示i个元素划分到k个不可区分的非空盒子里的方案数。

ll dp[MAX][MAX];

void init()

{

ll i,j;

mem(dp,0);

dp[1][1]=1;

for(i=2;i<MAX;i++)

{

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

{

dp[i][j]=(dp[i-1][j-1]+j\*dp[i-1][j])%mod;

}

}

}

## **4.11 线性基**

struct Base

{

#define type ll

#define mx 60

type d[mx+3],p[mx+3],cnt;

Base()

{

memset(d,0,sizeof(d));

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

cnt=0;

}

bool insert(type x)

{

int i;

for(i=mx;~i;i--)

{

if(!(x&(1LL<<i))) continue;

if(!d[i])

{

cnt++;

d[i]=x;

break;

}

x^=d[i];

}

return x>0;

}

type query\_max()

{

int i;

type res=0;

for(i=mx;~i;i--)

{

if((res^d[i])>res) res^=d[i];

}

return res;

}

type query\_min()

{

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

{

if(d[i]) return d[i];

}

return 0;

}

void rebuild()

{

int i,j;

cnt=0;

for(i=mx;~i;i--)

{

for(j=i-1;~j;j--)

{

if(d[i]&(1LL<<j)) d[i]^=d[j];

}

}

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

{

if(d[i]) p[cnt++]=d[i];

}

}

type kth(type k)//能异或出的第k小的数 使用前先调用rebuild

{

type res=0;

if(k>=(1LL<<cnt)) return -1;

for(int i=mx;~i;i--)

{

if(k&(1LL<<i)) res^=p[i];

}

return res;

}

void merge(Base x)

{

for(int i=mx;~i;i--)

{

if(x.d[i]) insert(x.d[i]);

}

}

};

## **4.12** Berlekamp-Massey

typedef vector<int> VI;

namespace linear\_seq

{

#define rep(i,a,n) for (int i=a;i<n;i++)

#define SZ(x) ((int)(x).size())

const ll mod=1e9+7;

ll powmod(ll a,ll b){ll res=1;a%=mod; assert(b>=0); for(;b;b>>=1){if(b&1)res=res\*a%mod;a=a\*a%mod;}return res;}

const int N=10010;

ll res[N],base[N],\_c[N],\_md[N];

vector<int> Md;

void mul(ll \*a,ll \*b,int k)

{

rep(i,0,k+k) \_c[i]=0;

rep(i,0,k) if (a[i]) rep(j,0,k) \_c[i+j]=(\_c[i+j]+a[i]\*b[j])%mod;

for (int i=k+k-1;i>=k;i--) if (\_c[i])

rep(j,0,SZ(Md)) \_c[i-k+Md[j]]=(\_c[i-k+Md[j]]-\_c[i]\*\_md[Md[j]])%mod;

rep(i,0,k) a[i]=\_c[i];

}

int solve(ll n,VI a,VI b){

ll ans=0,pnt=0;

int k=SZ(a);

assert(SZ(a)==SZ(b));

rep(i,0,k) \_md[k-1-i]=-a[i];\_md[k]=1;

Md.clear();

rep(i,0,k) if (\_md[i]!=0) Md.push\_back(i);

rep(i,0,k) res[i]=base[i]=0;

res[0]=1;

while ((1ll<<pnt)<=n) pnt++;

for (int p=pnt;p>=0;p--) {

mul(res,res,k);

if ((n>>p)&1) {

for (int i=k-1;i>=0;i--) res[i+1]=res[i];res[0]=0;

rep(j,0,SZ(Md)) res[Md[j]]=(res[Md[j]]-res[k]\*\_md[Md[j]])%mod;

}

}

rep(i,0,k) ans=(ans+res[i]\*b[i])%mod;

if (ans<0) ans+=mod;

return ans;

}

VI BM(VI s){

VI C(1,1),B(1,1);

int L=0,m=1,b=1;

rep(n,0,SZ(s)){

ll d=0;

rep(i,0,L+1) d=(d+(ll)C[i]\*s[n-i])%mod;

if(d==0) ++m;

else if(2\*L<=n){

VI T=C;

ll c=mod-d\*powmod(b,mod-2)%mod;//逆元

while (SZ(C)<SZ(B)+m) C.pb(0);

rep(i,0,SZ(B)) C[i+m]=(C[i+m]+c\*B[i])%mod;

L=n+1-L; B=T; b=d; m=1;

} else {

ll c=mod-d\*powmod(b,mod-2)%mod;//逆元

while (SZ(C)<SZ(B)+m) C.pb(0);

rep(i,0,SZ(B)) C[i+m]=(C[i+m]+c\*B[i])%mod;

++m;

}

}

return C;

}

int gao(VI a,ll n)

{

VI c=BM(a);

c.erase(c.begin());

rep(i,0,SZ(c)) c[i]=(mod-c[i])%mod;

return solve(n,c,VI(a.begin(),a.begin()+SZ(c)));

}

};//linear\_seq::gao(VI{},n-1)

## **4.13 原根**

4.13.1 原根性质

1.一个数m如果有原根，则其原根个数为phi[phi[m]]。若m为素数，则其原根个数为phi[phi[m]]=phi[m-1]。

2.有原根的数只有2,4,p^n,2\*p^n (p为质数,n为正整数)

3.一个数的最小原根的大小是O(n^0.25)的

4.如果g为n的原根，则g^d为n的原根的充要条件是gcd(d,phi[n])=1

4.13.2 指标法则

1. I(a\*b)≡I(a)+I(b) (mod p-1)

2. I(a^k)≡k\*I(a) (mod p-1)

注：I(a)表示a的指标。

4.13.3 求素数原根与指标表

int p[MAX],tot,prime[MAX];

void init(int n)

{

int i,j;

tot=0;

mem(prime,0);

prime[1]=1;

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

{

if(!prime[i]) prime[i]=p[tot++]=i;

for(j=0;j<tot&&p[j]\*i<=n;j++)

{

prime[i\*p[j]]=p[j];

if(i%p[j]==0) break;

}

}

}

ll pow2(ll a,ll b,ll p)

{

ll res=1;

while(b)

{

if(b&1) res=res\*a%p;

a=a\*a%p;

b>>=1;

}

return res;

}

int tp[MAX];

int find\_root(int x)//求素数原根

{

if(x==2) return 1;

int f,phi=x-1;

tp[0]=0;

for(int i=0;phi&&i<tot;i++)

{

if(phi%p[i]==0)

{

tp[++tp[0]]=p[i];

while(phi%p[i]==0) phi/=p[i];

}

}

if(phi!=1) tp[++tp[0]]=phi;

phi=x-1;

for(int g=2;g<=x-1;g++)

{

f=1;

for(int i=1;i<=tp[0];i++)

{

if(pow2(g,phi/tp[i],x)==1)

{

f=0;

break;

}

}

if(f) return g;

}

return 0;

}

int I[MAX];

void get\_I(int p)//求指标表

{

int g,now;

g=find\_root(p);

now=1;

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

{

now=now\*g%p;

I[now]=i;

}

}

## **4.14 ex Baby-Step-Giant-Step a^x≡b (mod c)**

ll exBSGS(ll a,ll b,ll c)

{

ll i,g,d,num,now,sq,t,x,y;

if(c==1) return b?-1:(a!=1);

if(b==1) return a?0:-1;

if(a%c==0) return b?-1:1;

num=0;

d=1;

while((g=\_\_gcd(a,c))>1)

{

if(b%g) return -1;

num++;

b/=g;

c/=g;

d=(d\*a/g)%c;

if(d==b) return num;

}

mp.clear();

sq=ceil(sqrt(c));

t=1;

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

{

if(!mp.count(t)) mp[t]=i;

else mp[t]=min(mp[t],i);

t=t\*a%c;

}

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

{

exgcd(d,c,x,y);

x=(x\*b%c+c)%c;

if(mp.count(x)) return i\*sq+mp[x]+num;

d=d\*t%c;

}

return -1;

}

# **5.多项式**

## **5.1 FFT**

namespace FFT

{

#define rep(i,a,b) for(int i=(a);i<=(b);i++)

const double pi=acos(-1);

const int maxn=(1<<19)+10;

struct cp

{

double a,b;

cp(){}

cp(double \_x,double \_y){a=\_x,b=\_y;}

cp operator +(const cp &o)const{return (cp){a+o.a,b+o.b};}

cp operator -(const cp &o)const{return (cp){a-o.a,b-o.b};}

cp operator \*(const cp &o)const{return (cp){a\*o.a-b\*o.b,b\*o.a+a\*o.b};}

cp operator \*(const double &o)const{return (cp){a\*o,b\*o};}

cp operator !()const{return (cp){a,-b};}

}x[maxn],y[maxn],z[maxn],w[maxn];

void fft(cp x[],int k,int v)

{

int i,j,l;

for(i=0,j=0;i<k;i++)

{

if(i>j)swap(x[i],x[j]);

for(l=k>>1;(j^=l)<l;l>>=1);

}

w[0]=(cp){1,0};

for(i=2;i<=k;i<<=1)

{

cp g=(cp){cos(2\*pi/i),(v?-1:1)\*sin(2\*pi/i)};

for(j=(i>>1);j>=0;j-=2)w[j]=w[j>>1];

for(j=1;j<i>>1;j+=2)w[j]=w[j-1]\*g;

for(j=0;j<k;j+=i)

{

cp \*a=x+j,\*b=a+(i>>1);

for(l=0;l<i>>1;l++)

{

cp o=b[l]\*w[l];

b[l]=a[l]-o;

a[l]=a[l]+o;

}

}

}

if(v)for(i=0;i<k;i++)x[i]=(cp){x[i].a/k,x[i].b/k};

}

//多项式b与多项式c相乘 结果存在a

//l1为b的长度-1 l2为c的长度-1

void mul(int \*a,int \*b,int \*c,int l1,int l2)

{

if(l1<128&&l2<128)

{

rep(i,0,l1+l2)a[i]=0;

rep(i,0,l1)rep(j,0,l2)a[i+j]+=b[i]\*c[j];

return;

}

int K;

for(K=1;K<=l1+l2;K<<=1);

rep(i,0,l1)x[i]=cp(b[i],0);

rep(i,0,l2)y[i]=cp(c[i],0);

rep(i,l1+1,K)x[i]=cp(0,0);

rep(i,l2+1,K)y[i]=cp(0,0);

fft(x,K,0);fft(y,K,0);

rep(i,0,K)z[i]=x[i]\*y[i];

fft(z,K,1);

rep(i,0,l1+l2)a[i]=(ll)(z[i].a+0.5);

}

};

## **5.2 NTT**

namespace NTT

{

const ll g=3;

const ll p=998244353;

ll wn[35];

ll pow2(ll a,ll b)

{

ll res=1;

while(b)

{

if(b&1) res=res\*a%p;

a=a\*a%p;

b>>=1;

}

return res;

}

void getwn()

{

for(int i=0;i<25;i++) wn[i]=pow2(g,(p-1)/(1LL<<i));

}

void ntt(VL &a,int len,int f)

{

ll i,j=0,t,k,w,id;

for(i=1;i<len-1;i++)

{

for(t=len;j^=t>>=1,~j&t;);

if(i<j) swap(a[i],a[j]);

}

for(i=1,id=1;i<len;i<<=1,id++)

{

t=i<<1;

for(j=0;j<len;j+=t)

{

for(k=0,w=1;k<i;k++,w=w\*wn[id]%p)

{

ll x=a[j+k],y=w\*a[j+k+i]%p;

a[j+k]=(x+y)%p;

a[j+k+i]=(x-y+p)%p;

}

}

}

if(f)

{

for(i=1,j=len-1;i<j;i++,j--) swap(a[i],a[j]);

ll inv=pow2(len,p-2);

for(i=0;i<len;i++) a[i]=a[i]\*inv%p;

}

}

//结果存在a

void mul(ll \*a,ll \*b,int l1,int l2)

{

int len,i;

for(len=1;len<=l1+l2;len<<=1);

for(i=l1+1;i<=len;i++) a[i]=0;

for(i=l2+1;i<=len;i++) b[i]=0;

ntt(a,len,0);ntt(b,len,0);

for(i=0;i<len;i++) a[i]=a[i]\*b[i]%p;

ntt(a,len,1);

}

};//NTT::getwn();

## **5.3 FWT**

namespace FWT

{

ll inv2;//2对p的逆元

const ll p=1e9+7;

ll pow2(ll a,ll b)

{

ll res=1;

while(b)

{

if(b&1) res=res\*a%p;

a=a\*a%p;

b>>=1;

}

return res;

}

void fwt(ll \*a,int n,int f,int v)

{

for(int d=1;d<n;d<<=1)

{

for(int m=d<<1,i=0;i<n;i+=m)

{

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

{

ll x=a[i+j],y=a[i+j+d];

if(!v)

{

if(f==1) a[i+j]=(x+y)%p,a[i+j+d]=(x-y+p)%p;//xor

else if(f==2) a[i+j]=(x+y)%p;//and

else if(f==3) a[i+j+d]=(x+y)%p;//or

}

else

{

if(f==1) a[i+j]=(x+y)\*inv2%p,a[i+j+d]=(x-y+p)%p\*inv2%p;//xor

else if(f==2) a[i+j]=(x-y+p)%p;//and

else if(f==3) a[i+j+d]=(y-x+p)%p;//or

}

}

}

}

}

//结果存在a

void XOR(ll \*a,ll \*b,int n)

{

int len;

for(len=1;len<=n;len<<=1);

fwt(a,len,1,0);

fwt(b,len,1,0);

for(int i=0;i<len;i++) a[i]=a[i]\*b[i]%p;

inv2=pow2(2,p-2);

fwt(a,len,1,1);

}

void AND(ll \*a,ll \*b,int n)

{

int len;

for(len=1;len<=n;len<<=1);

fwt(a,len,2,0);

fwt(b,len,2,0);

for(int i=0;i<len;i++) a[i]=a[i]\*b[i]%p;

fwt(a,len,2,1);

}

void OR(ll \*a,ll \*b,int n)

{

int len;

for(len=1;len<=n;len<<=1);

fwt(a,len,3,0);

fwt(b,len,3,0);

for(int i=0;i<len;i++) a[i]=a[i]\*b[i]%p;

fwt(a,len,3,1);

}

};

## **5.4 拉格朗日插值**

namespace polysum {

#define rep(i,a,n) for (int i=a;i<n;i++)

#define per(i,a,n) for (int i=n-1;i>=a;i--)

const int D=101000;

ll a[D],tmp[D],f[D],g[D],p[D],p1[D],p2[D],b[D],h[D][2],C[D];

ll powmod(ll a,ll b){ll res=1;a%=mod;assert(b>=0);for(;b;b>>=1){if(b&1)res=res\*a%mod;a=a\*a%mod;}return res;}

ll calcn(int d,ll \*a,ll n) { // a[0].. a[d] a[n]

if (n<=d) return a[n];

p1[0]=p2[0]=1;

rep(i,0,d+1) {

ll t=(n-i+mod)%mod;

p1[i+1]=p1[i]\*t%mod;

}

rep(i,0,d+1) {

ll t=(n-d+i+mod)%mod;

p2[i+1]=p2[i]\*t%mod;

}

ll ans=0;

rep(i,0,d+1) {

ll t=g[i]\*g[d-i]%mod\*p1[i]%mod\*p2[d-i]%mod\*a[i]%mod;

if ((d-i)&1) ans=(ans-t+mod)%mod;

else ans=(ans+t)%mod;

}

return ans;

}

void init(int M) {

f[0]=f[1]=g[0]=g[1]=1;

rep(i,2,M+5) f[i]=f[i-1]\*i%mod;

g[M+4]=powmod(f[M+4],mod-2);

per(i,1,M+4) g[i]=g[i+1]\*(i+1)%mod;

}

ll polysum(ll n,ll \*a,ll m) { // a[0].. a[m] \sum\_{i=0}^{n-1} a[i]

rep(i,0,m+1) tmp[i]=a[i];

tmp[m+1]=calcn(m,tmp,m+1);

rep(i,1,m+2) tmp[i]=(tmp[i-1]+tmp[i])%mod;

return calcn(m+1,tmp,n-1);

}

ll qpolysum(ll R,ll n,ll \*a,ll m) { // a[0].. a[m] \sum\_{i=0}^{n-1} a[i]\*R^i

if (R==1) return polysum(n,a,m);

a[m+1]=calcn(m,a,m+1);

ll r=powmod(R,mod-2),p3=0,p4=0,c,ans;

h[0][0]=0;h[0][1]=1;

rep(i,1,m+2) {

h[i][0]=(h[i-1][0]+a[i-1])\*r%mod;

h[i][1]=h[i-1][1]\*r%mod;

}

rep(i,0,m+2) {

ll t=g[i]\*g[m+1-i]%mod;

if (i&1) p3=((p3-h[i][0]\*t)%mod+mod)%mod,p4=((p4-h[i][1]\*t)%mod+mod)%mod;

else p3=(p3+h[i][0]\*t)%mod,p4=(p4+h[i][1]\*t)%mod;

}

c=powmod(p4,mod-2)\*(mod-p3)%mod;

rep(i,0,m+2) h[i][0]=(h[i][0]+h[i][1]\*c)%mod;

rep(i,0,m+2) C[i]=h[i][0];

ans=(calcn(m,C,n)\*powmod(R,n)-c)%mod;

if (ans<0) ans+=mod;

return ans;

}

} // polysum::init();

# **6.矩阵**

## **6.1 矩阵基本操作**

const ll mod=1e9+7;

const int sz=10;

struct Matrix

{

ll c[sz][sz];

Matrix(){}

friend Matrix operator \*(Matrix a,Matrix b)

{

Matrix res;

mem(res.c,0);

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

{

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

{

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

{

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

}

}

}

return res;

}

};

## **6.2 矩阵快速幂**

Matrix matpow2(Matrix a,ll b)

{

Matrix res;

mem(res.c,0);

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

{

res.c[i][i]=1;

}

while(b)

{

if(b&1) res=res\*a;

a=a\*a;

b>>=1;

}

return res;

}

## **6.3 高斯消元**

6.3.1 同余方程 mod=2时 异或加速

int GE(Matrix a,int n,int m)

{

int i,j;

for(i=1,j=1;i<=n&&j<=m;j++)

{

int k=i;

while(k<=n&&!a.c[k][j]) k++;

if(a.c[k][j])

{

for(int r=1;r<=m+1;r++)

{

swap(a.c[i][r],a.c[k][r]);

}

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

{

if(r!=i&&a.c[r][j])

{

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

{

a.c[r][k]^=a.c[i][k];

}

}

}

i++;

}

}

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

{

if(a.c[j][m+1]) return -1;

}

return m-i+1;//返回解的个数

}

# **7.博弈论**

## **7.1 威佐夫博弈**

有两堆各若干个物品，两个人轮流从任一堆取至少一个或同时从两堆中取同样多的物品，规定每次至少取一个，多者不限，最后取光者得胜。

结论：

若两堆物品的初始值为（x，y），且x < y，则另z=y-x；

记w=（int）[（（sqrt（5）+1）/2）\*z ]；

若w=x，则先手必败，否则先手必胜

## **7.2 SG函数**

//sg函数

//f[m]:可改变当前状态的方式，N为方式的种类，要先从小到大sort

//sg[]:0~n的sg函数值

//flag[m]:为x后继状态的集合

7.2.1 sg表

int f[111],sg[MAX];

void SG(int n,int m)

{

int i,j,flag[111];

mem(sg,0);

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

{

mem(flag,0);

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

{

flag[sg[i-f[j]]]=1;

}

for(j=0;;j++)

{

if(!flag[j])

{

sg[i]=j;

break;

}

}

}

}

7.2.2 记忆化搜索求sg表

int f[105],sg[MAX],m;

int dfs(int x)

{

int i,j,flag[105];

if(sg[x]!=-1) return sg[x];

mem(flag,0);

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

{

if(x>=f[i])

{

dfs(x-f[i]);

flag[sg[x-f[i]]]=1;

}

}

for(i=0;;i++)

{

if(!flag[i])

{

j=i;

break;

}

}

return sg[x]=j;

}

## **7.3 阶梯博弈**

0层为终点的阶梯博弈，等价于奇数层的nim，偶数层的移动不影响结果

## **7.4 SJ定理**

SJ定理：

对于任意一个Anti-SG游戏，如果我们规定当局面中所有的单一游戏的SG值为0时，游戏结束。

先手必胜当且仅当：

(1)游戏的SG函数不为0且游戏中某个单一游戏的SG函数大于1；

(2)游戏的SG函数为0且游戏中没有单一游戏的SG函数大于1。

# **8.dp**

## **8.1 LIS**

//最长上升子序列(>)nlogn 返回长度

//最长下降子序列(<) 把原数组取负数

int a[MAX],b[MAX],n;

int LIS()

{

int i;

mem(b,0x3f);

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

{

\*lower\_bound(b,b+n,a[i])=a[i];//最长不下降子序列(>=)改为upper\_bound

}

return lower\_bound(b,b+n,INF)-b;

}

## **8.2 LPS**

/\*

最长回文子序列

dp[i][j]表示子串[i,j]中最长回文子序列的长度

复杂度O(n^2)

\*/

int dp[2222][2222];

void LPS(char \*s,int n)

{

int i,j,len;

for(i=1;i<=n;i++) dp[i][i]=1;

for(len=2;len<=n;len++)

{

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

{

j=i+len-1;

if(s[i]==s[j]) dp[i][j]=dp[i+1][j-1]+2;

else dp[i][j]=max({dp[i+1][j],dp[i][j-1],dp[i+1][j-1]});

}

}

}

## **8.3 数位dp**

const int DIG=20+2;

ll dp[DIG][2];

ll gao(ll x)

{

const int base=10;

int p[DIG],tot=0;

if(x==-1) return 0;

while(1)

{

p[tot++]=x%base;

x/=base;

if(!x) break;

}

function<ll(int,int,int,int)> dfs=[&](int pos,int lead,int sta,int limt)->ll

{

if(pos==-1) return ;

if(!limt&&!lead&&dp[pos][sta]!=-1) return dp[pos][sta];

ll res=0;

for(int i=(limt?p[pos]:base-1);~i;i--)

{

res+=dfs(pos-1,lead&&i==0&&pos,,limt&&i==p[pos]);

}

if(!limt&&!lead) dp[pos][sta]=res;

return res;

};

return dfs(tot-1,1,0,1);

}

# **9.Other**

## **9.1 FastIO**

9.1.1 fast input

struct FastIO

{

static const int S=200;

int wpos;

char wbuf[S];

FastIO():wpos(0){}

inline int xchar()

{

static char buf[S];

static int len=0,pos=0;

if(pos==len) pos=0,len=fread(buf,1,S,stdin);

if(pos==len) exit(0);

return buf[pos++];

}

inline int read()

{

int s=1,c=xchar(),x=0;

while(c<=32) c=xchar();

if(c=='-') s=-1,c=xchar();

for(;'0'<=c&&c<='9';c=xchar()) x=x\*10+c-'0';

return x\*s;

}

~FastIO()

{

if(wpos) fwrite(wbuf,1,wpos,stdout),wpos=0;

}

}io;

9.1.2 FastIO

namespace fastIO{

#define BUF\_SIZE 100000

#define OUT\_SIZE 100000

#define ll long long

//fread->read

bool IOerror=0;

// inline char nc(){char ch=getchar();if(ch==-1)IOerror=1;return ch;}

inline char nc(){

static char buf[BUF\_SIZE],\*p1=buf+BUF\_SIZE,\*pend=buf+BUF\_SIZE;

if(p1==pend){

p1=buf;pend=buf+fread(buf,1,BUF\_SIZE,stdin);

if(pend==p1){IOerror=1;return -1;}

}

return \*p1++;

}

inline bool blank(char ch){return ch==' '||ch=='\n'||ch=='\r'||ch=='\t';}

template<class T> inline bool read(T &x){

bool sign=0;char ch=nc();x=0;

for(;blank(ch);ch=nc());

if(IOerror)return false;

if(ch=='-')sign=1,ch=nc();

for(;ch>='0'&&ch<='9';ch=nc())x=x\*10+ch-'0';

if(sign)x=-x;

return true;

}

inline bool read(double &x){

bool sign=0;char ch=nc();x=0;

for(;blank(ch);ch=nc());

if(IOerror)return false;

if(ch=='-')sign=1,ch=nc();

for(;ch>='0'&&ch<='9';ch=nc())x=x\*10+ch-'0';

if(ch=='.'){

double tmp=1; ch=nc();

for(;ch>='0'&&ch<='9';ch=nc())tmp/=10.0,x+=tmp\*(ch-'0');

}

if(sign)x=-x;

return true;

}

inline bool read(char \*s){

char ch=nc();

for(;blank(ch);ch=nc());

if(IOerror)return false;

for(;!blank(ch)&&!IOerror;ch=nc())\*s++=ch;

\*s=0;

return true;

}

inline bool read(char &c){

for(c=nc();blank(c);c=nc());

if(IOerror){c=-1;return false;}

return true;

}

template<class T,class... U>bool read(T& h,U&... t){return read(h)&&read(t...);}

//fwrite->print

struct Ostream\_fwrite{

char \*buf,\*p1,\*pend;

Ostream\_fwrite(){buf=new char[BUF\_SIZE];p1=buf;pend=buf+BUF\_SIZE;}

// void out(char ch){putchar(ch);}

void out(char ch){if(p1==pend){fwrite(buf,1,BUF\_SIZE,stdout);p1=buf;}\*p1++=ch;}

template<class T>void print(T x){

static char s[33],\*s1;s1=s;

if(!x)\*s1++='0';if(x<0)out('-'),x=-x;

while(x)\*s1++=x%10+'0',x/=10;

while(s1--!=s)out(\*s1);

}

void print(double x,int y){

static ll mul[]=

{1,10,100,1000,10000,100000,1000000,10000000,100000000,1000000000,

10000000000LL,100000000000LL,1000000000000LL,10000000000000LL,

100000000000000LL,1000000000000000LL,10000000000000000LL,100000000000000000LL};

if(x<-1e-12)out('-'),x=-x;

ll x2=(ll)floor(x);if(!y&&x-x2>=0.5)++x2;x-=x2;x\*=mul[y];

ll x3=(ll)floor(x);if(y&&x-x3>=0.5)++x3;print(x2);

if(y>0){out('.');for(size\_t i=1;i<y&&x3\*mul[i]<mul[y];out('0'),++i);print(x3);}

}

void print(char \*s){while(\*s)out(\*s++);}

void print(const char \*s){while(\*s)out(\*s++);}

void flush(){if(p1!=buf){fwrite(buf,1,p1-buf,stdout);p1=buf;}}

~Ostream\_fwrite(){flush();}

}Ostream;

template<class T>void print(T x){Ostream.print(x);}

inline void print(char x){Ostream.out(x);}

inline void print(char \*s){Ostream.print(s);}

inline void print(string s){Ostream.print(s.c\_str());}

inline void print(const char \*s){Ostream.print(s);}

inline void print(double x,int y){Ostream.print(x,y);}

template<class T,class... U>void print(const T& h,const U&... t){print(h);print(t...);}

void println(){print('\n');}

template<class T,class... U>void println(const T& h,const U&... t){print(h);println(t...);}

inline void flush(){Ostream.flush();}

#undef ll

#undef OUT\_SIZE

#undef BUF\_SIZE

};

using namespace fastIO;

## **9.2 网格整数点共有多少个正方形**

struct node

{

int x,y;

void input()

{

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

}

}p[511];

int main()

{

int n,i,j,ans;

while(~scanf("%d",&n))

{

map<pair<int,int>,int> mp;

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

{

p[i].input();

mp[MP(p[i].x,p[i].y)]=1;

}

ans=0;

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

{

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

{

int a,b,c,d,e,f,g,h;

a=p[i].x;

b=p[i].y;

c=p[j].x;

d=p[j].y;

e=a+b+c-d;

f=-a+b+c+d;

g=a-b+c+d;

h=a+b-c+d;

if(abs(e%2)+abs(f%2)+abs(g%2)+abs(h%2)==0)

{

if(mp[MP(e/2,f/2)]&&mp[MP(g/2,h/2)]) ans++;

}

}

}

printf("%d\n",ans/2);

}

return 0;

}

## **9.3 模拟退火**

9.3.1 简单版 ->模拟退火求费马点

9.3.2 复杂版

//求矩形区域内一点到各点距离之和最短

//时间复杂度 cnt\*c1\*c2\*n

int sgn(double x)

{

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

else return x>0?1:-1;

}

struct Point

{

double x,y;

Point(){}

Point(double a,double b)

{

x=a;

y=b;

}

void input()

{

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

}

};

typedef Point Vector;

Vector operator -(Vector a,Vector b){return Vector(a.x-b.x,a.y-b.y);}

double dot(Vector a,Vector b){return a.x\*b.x+a.y\*b.y;}

double dist(Point a,Point b){return sqrt(dot(a-b,a-b));}

double lx,ly;//矩形区域(0,0)-(lx,ly)

int check(double x,double y)

{

if(sgn(x)<0||sgn(y)<0||sgn(x-lx)>0||sgn(y-ly)>0) return 1;

return 0;

}

double Rand(double r,double l)

{

return(rand()%((int)(l-r)\*1000))/(1000.0+r);

}

double getres(Point t,Point \*p,int n)//求距离之和

{

double res=0;

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

{

res+=dist(t,p[i]);

}

return res;

}

pair<Point,double> SA(Point \*p,int n)//模拟退火

{

srand(time(0));//重置随机种子

const double k=0.85;//退火常数

const int c1=30;//随机取点的个数

const int c2=50;//退火次数

Point q[c1+10];//随机取点

double dis[c1+10];//每个点的计算结果

int i,j;

for(i=1;i<=c1;i++)

{

q[i]=Point(Rand(0,lx),Rand(0,ly));

dis[i]=getres(q[i],p,n);

}

double tmax=max(lx,ly);

double tmin=1e-3;

// int cnt=0;//计算外层循环次数

while(tmax>tmin)

{

for(i=1;i<=c1;i++)

{

for(j=1;j<=c2;j++)

{

double ang=Rand(0,2\*PI);

Point z;

z.x=q[i].x+cos(ang)\*tmax;

z.y=q[i].y+sin(ang)\*tmax;

if(check(z.x,z.y)) continue;

double temp=getres(z,p,n);

if(temp<dis[i])

{

dis[i]=temp;

q[i]=z;

}

}

}

// cnt++;

tmax\*=k;

}

// cout<<cnt\*c1\*c2\*n<<endl;//时间复杂度

int pos=1;

for(i=2;i<=c1;i++)

{

if(dis[i]<dis[pos])

{

pos=i;

}

}

pair<Point,double> res;

res=make\_pair(q[pos],dis[pos]);

return res;

}

## **9.4 矩形面积并**

struct node

{

ll l,r,h;

int tag;

friend bool operator <(node a,node b)

{

return a.h<b.h;

}

}seg[MAX<<1];//线段

ll x[MAX<<1];//横坐标离散化

struct Segment\_Tree

{

#define ls (id<<1)

#define rs (id<<1|1)

ll n,ql,qr,qv;

ll cover[MAX<<3],len[MAX<<3];//注意这里要开8倍

void build(ll \_n)

{

mem(cover,0);

mem(len,0);

n=\_n;

}

void callen(int id,int l,int r)

{

if(cover[id]) len[id]=x[r+1]-x[l];//被整段覆盖

else if(l==r) len[id]=0;//不是一条线段

else len[id]=len[ls]+len[rs];//是一条线段但又没有被整段覆盖

}

void update(int l,int r,int id)

{

if(l>=ql&&r<=qr)

{

cover[id]+=qv;//覆盖情况

callen(id,l,r);

return;

}

int mid=(l+r)>>1;

if(ql<=mid) update(l,mid,ls);

if(qr>mid) update(mid+1,r,rs);

callen(id,l,r);

}

}tree;

int main()

{

int n,i,tot,l,r,cnt;

ll x1,y1,x2,y2,ans;

while(~scanf("%d",&n)&&n)

{

tot=0;

mem(x,0);

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

{

scanf("%lld%lld%lld%lld",&x1,&y1,&x2,&y2);

//矩形的左下和右上坐标

x[tot]=x1;

seg[tot].tag=-1;

seg[tot].l=x1;

seg[tot].r=x2;

seg[tot++].h=y1;

//上边界

x[tot]=x2;

seg[tot].tag=1;

seg[tot].l=x1;

seg[tot].r=x2;

seg[tot++].h=y2;

//下边界

}

sort(seg,seg+tot);//线段按纵坐标升序

sort(x,x+tot);//横坐标升序

cnt=unique(x,x+tot)-x;

tree.build(cnt-1);

ans=0;

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

{

if(i) ans+=(seg[i].h-seg[i-1].h)\*tree.len[1];

tree.ql=lower\_bound(x,x+cnt-1,seg[i].l)-x;

tree.qr=lower\_bound(x,x+cnt-1,seg[i].r)-x-1;

tree.qv=seg[i].tag;

tree.update(0,cnt-1,1);

}

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

}

return 0;

}

## **9.5 判断星期几**

int judge(int y,int m,int d)

{

int res;

if(m==1||m==2) m+=12,y--;//1月2月当作前一年的13,14月

if((y<1752)||(y==1752&&m<9)||(y==1752&&m==9&&d<3)) res=(d+2\*m+3\*(m+1)/5+y+y/4+5)%7;

else res=(d+2\*m+3\*(m+1)/5+y+y/4-y/100+y/400)%7;

return res+1;

}

## **9.6 hash\_map**

**//放在using namespace std;的下面**

#define \_GLIBCXX\_PERMIT\_BACKWARD\_HASH

#include <ext/hash\_map>

using namespace \_\_gnu\_cxx;

struct str\_hash{size\_t operator()(const string& str)const{return \_\_stl\_hash\_string(str.c\_str());}};

## **9.7 O(1)快速乘**

ll mul2(ll x,ll y,ll p)

{

ll res=(x\*y-ll((long double)x/p\*y+1.0e-8)\*p);

return res<0?res+p:res;

}

## **9.8 快速模**

typedef long long i64;

typedef unsigned long long u64;

typedef \_\_uint128\_t u128;

const int word\_bits=sizeof(u64)\*8;

struct FastMod

{

static u64 mod,inv,r2;

u64 x;

FastMod():x(0){}

FastMod(u64 n):x(init(n)){}

static u64 modulus(){return mod;}

static u64 init(u64 w){return reduce(u128(w)\*r2);}

static void set\_mod(u64 m)

{

mod=m;

assert(mod&1);

inv=m;

for(int i=0;i<5;i++) inv\*=2-inv\*m;

r2=-u128(m)%m;

}

static u64 reduce(u128 x)

{

u64 y=u64(x>>word\_bits)-u64((u128(u64(x)\*inv)\*mod)>>word\_bits);

return i64(y)<0?y+mod:y;

}

FastMod& operator+=(FastMod rhs)

{

x+=rhs.x-mod;

if(i64(x)<0) x+=mod;

return \*this;

}

FastMod operator+(FastMod rhs)const {return FastMod(\*this)+=rhs;}

FastMod& operator\*=(FastMod rhs)

{

x=reduce(u128(x)\*rhs.x);

return \*this;

}

FastMod operator\*(FastMod rhs)const {return FastMod(\*this)\*=rhs;}

u64 get()const {return reduce(x);}

}a[MAX];

u64 FastMod::mod,FastMod::inv,FastMod::r2;

// FastMod::set\_mod(p);

## **9.9 离散化**

struct Discretization

{

#define type ll

vector<type> a;

void init(){a.clear();}

void add(type x){a.pb(x);}

void work(){sort(all(a));a.resize(unique(all(a))-a.begin());}

int get(type x){return lower\_bound(all(a),x)-a.begin()+1;}

int size(){return a.size();}

#undef type

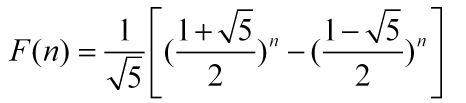
}di;

**NTT常用mod**

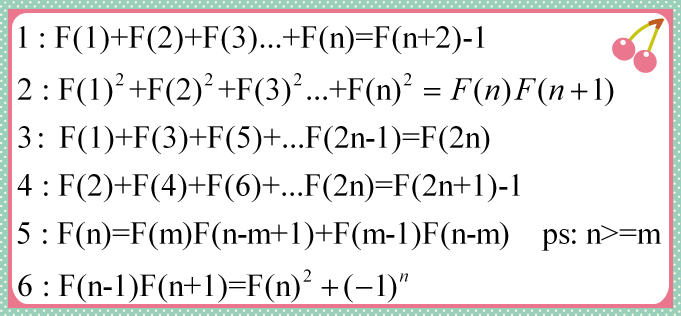
|  |  |  |  |
| --- | --- | --- | --- |
| r\*2^k+1 | r | k | g |
| 3 | 1 | 1 | 2 |
| 5 | 1 | 2 | 2 |
| 17 | 1 | 4 | 3 |
| 97 | 3 | 5 | 5 |
| 193 | 3 | 6 | 5 |
| 257 | 1 | 8 | 3 |
| 7681 | 15 | 9 | 17 |
| 12289 | 3 | 12 | 11 |
| 40961 | 5 | 13 | 3 |
| 65537 | 1 | 16 | 3 |
| 786433 | 3 | 18 | 10 |
| 5767169 | 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 |

**斐波那契数列性质**

**斐波那契通项公式：**

****

**关于斐波那契的一些恒等式：**

****

**斐波那契的数论相关：**

**性质1：IMG_268**

**证明：先证明斐波那契数列相邻两项是互素的。**

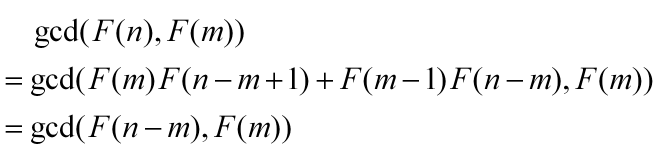
**反证法:假设不互素。那么有a=gcd(F(n),F(n-1)),a>1.**

**那么对于F(n)=F(n-1)+F(n-2).因为a|F(n),a|F(n-1),所以a|F(n-2).**

**由于a|F(n-1),a|F(n-2).又可以获得a|F(n-3)...可以知道a|F(1)其中。F(1)=1.**

**如果a|F(1)->a|1那么与a>1不符。相邻互素得证.(其实 a|F(2)就已经不行了.)**

**那么再由上面斐波那契恒等式5.可以推理。**

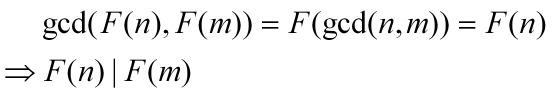
****

**中间推导依靠一小点数论知识.观察开始式子和结果。**

**一直将上式递推下去。结合gcd(n,m)=gcd(n-m,m).结果会是gcd(a,b) = gcd(0,gcd(a,b))**

**那么就可以证明上述式子成立。**

**性质2：IMG_270**

**证明:当n|m时。  
**