|  |
| --- |
| HUST |
| 算法必会模板 |
| 必会！ |

|  |
| --- |
| 佚名  2021-11-26 |

目录

[1数学知识 0](#_Toc499469552)

[//组合数 0](#_Toc499469553)

[//欧几里得算法 0](#_Toc499469554)

[//拓展欧几里得算法 1](#_Toc499469555)

[//逆元 1](#_Toc499469556)

[//lucas定理 1](#_Toc499469557)

[//快速幂 2](#_Toc499469558)

[//中国剩余定理 3](#_Toc499469559)

[//大步小步算法 4](#_Toc499469560)

[//快速筛法求素数表 5](#_Toc499469561)

[//欧拉函数 5](#_Toc499469562)

[//莫比乌斯函数 7](#_Toc499469563)

[//高斯消元 7](#_Toc499469564)

[//高精度 9](#_Toc499469565)

[//矩阵乘法 15](#_Toc499469566)

[2．数据结构 16](#_Toc499469567)

[//树状数组 16](#_Toc499469568)

[//线段树 17](#_Toc499469569)

[//Treap 21](#_Toc499469570)

[//splay自上而下 25](#_Toc499469571)

[//主席树 27](#_Toc499469572)

[//Link-Cut-Tree 29](#_Toc499469573)

[3图 33](#_Toc499469574)

[//2-sat 33](#_Toc499469575)

[//有向图的强联通分量 35](#_Toc499469576)

[//无向图的边的双连通分量 37](#_Toc499469577)

[//最短路 40](#_Toc499469578)

[//最小生成树 43](#_Toc499469579)

[//最大流 44](#_Toc499469580)

[//最小费用最大流 47](#_Toc499469581)

[//KM算法 50](#_Toc499469582)

[4树 53](#_Toc499469583)

[//LCA 53](#_Toc499469584)

[//树链剖分 57](#_Toc499469585)

[//点分治 59](#_Toc499469586)

[5字符串 62](#_Toc499469587)

[//KMP 62](#_Toc499469588)

[//AC自动机 63](#_Toc499469589)

[//后缀自动机SAM 64](#_Toc499469590)

[//后缀数组 66](#_Toc499469591)

[//Manacher算法 68](#_Toc499469592)

[6计算几何 69](#_Toc499469593)

[//计算几何基础 69](#_Toc499469594)

[//凸包 74](#_Toc499469595)

[//半平面交 76](#_Toc499469596)

**省 选 模 板**

## 1数学知识

### //组合数

//C(n,m) 在n个数中选m个的方案数

ll C[N][N];

void get\_C(int n)

{

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

{

C[i][i]=C[i][0]=1;

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

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

}

}

### //欧几里得算法

//(a,b)

ll gcd(ll a,ll b)

{

return b==0? a:gcd(b,a%b);

}

### //拓展欧几里得算法

//解同余方程 a\*x+b\*y = (a,b)

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

{

if(!b) { d=a; x=1; y=0; }

else { exgcd(b,a%b,d,y,x); y-=x\*(a/b); }

}

### //逆元

//a\*inv(a,n) = 1 mod n

ll inv(ll a,ll n)

{

ll d,x,y;

exgcd(a,n,d,x,y);

return d==1? (x+n)%n:-1;

}

### //lucas定理

//计算较大的组合数

ll fac[N];

void get\_pre(int n)

{

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

fac[i]=(fac[i-1]\*i)%mod;

}

ll C(ll n,ll m,ll mod)

{

if(n<m) return 0;

if(n<mod&&m<mod)

return fac[n]\*inv(fac[m],mod)%mod\*inv(fac[n-m],mod)%mod;

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

}

### //快速幂

//a^p % mod

ll pow(ll a,ll p,ll mod)

{

ll ans=1;

while(p)

{

if(p&1) ans=(ans\*a)%mod;

a=(a\*a)%mod; p>>=1;

}

return ans;

}

### //中国剩余定理

//解线性同余方程组

//sigma{ ai\*(1-ai\*mi) } % M , ai\*mi+wi\*y=1

ll a[N],m[N];

ll china(int n)

{

ll M=1,d,x=0,y;

for(int i=1;i<=n;i++) M\*=m[i];

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

{

ll w=M/m[i];

exgcd(m[i],w,d,d,y);

x=(x+y\*w\*a[i])%M;

}

return (x+M)%M;

}

### //大步小步算法

//计算a^x=b mod n中的最小x

map<int,int> mp;

int BSGS(int a,int b,int n)

{

int m=sqrt(n)+1,e=1,i;

int v=inv(pow(a,m,n),n);

mp[e]=0;

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

{

e=(e\*m)%n;

if(!mp.count(e)) mp[e]=i;

}

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

{

if(mp.count(b)) return i\*m+mp[b];

b=(b\*v)%mod;

}

return -1;

}

### //快速筛法求素数表

int su[N],vis[N];

void get\_su(int n)

{

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

{

if(!vis[i]) su[++su[0]]=i;

for(int j=1;j<=su[0]&&i\*su[j]<=n;j++)

{

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

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

}

}

}

### //欧拉函数

//phi(n)小于n的数中与n互素的数的个数

ll get\_phi(int n)

{

int m=sqrt(n)+1;

ll ans=n;

for(int i=2;i<=m;i++) if(n%i==0)

{

ans=ans/i\*(i-1);

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

}

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

return ans;

}

ll phi[N];

void get\_phi\_table(int n)

{

phi[1]=1;

for(int i=2;i<=n;i++) if(!phi[i])

{

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

{

if(!phi[j]) phi[j]=j;

phi[j]=phi[j]/i\*(i-1);

}

}

}

### //莫比乌斯函数

int mu[N],su[N],vis[N];

void get\_mu(int n)

{

mu[1]=1;

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

{

if(!vis[i]) mu[i]=-1,su[++su[0]]=i;

for(int j=1;j<=su[0]&&i\*su[j]<=n;j++)

{

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

if(i%su[j]==0) mu[i\*su[j]]=0;

else mu[i\*su[j]]=-mu[i];

}

}

}

### //高斯消元

//解线性方程组

double a[N][N];

void gause(int n)

{

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

{

int r=i;

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

if(fabs(a[j][i])>fabs(a[r][i])) r=i;

for(int j=1;j<=n+1;j++) swap(a[i][j],a[r][j]);

for(int j=n+1;j>=i;j--)

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

a[k][j]-=a[k][i]/a[i][i]\*a[i][j];

}

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

{

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

a[i][n+1]-=a[j][n+1]\*a[i][j];

a[i][n+1]/=a[i][i];

}

}

### //高精度

//高精度

const int maxn = 1e3+10;

const int base = 1e8;

const int wlen = 8;

int trans(char\* s,int st,int ed)

{

int x=0;

for(int i=st;i<ed;i++) x=x\*10+s[i]-‘0‘;

return x;

}

struct Bign

{

int len; ll N[maxn];

Bign() { len=1; memset(N,0,sizeof(N)); }

Bign(ll num) { \*this=num; }

Bign(const char\* s) { \*this=s; }

void print()

{

printf("%d",N[len-1]);

for(int i=len-2;i>=0;i--)

printf("%08d",N[i]);

puts("");

}

Bign operator = (const ll x)

{

ll num=x;

while(num>base)

{

N[len++]=num%base;

num/=base;

}

N[len++]=num;

return \*this;

}

Bign operator = (char\* s)

{

int L=strlen(s);

len=(L-1)/wlen+1;

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

{

int ed=L-i\*wlen;

int st=max(0,ed-wlen);

N[i]=trans(s,st,ed);

}

return \*this;

}

bool operator < (const Bign& B) const

{

if(len!=B.len) return len<B.len;

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

if(N[i]!=B.N[i]) return N[i]<B.N[i];

return 0;

}

bool operator <= (const Bign& B) const

{

return !(B<(\*this));

}

void clear()

{

while(len>1&&N[len-1]==0) len--;

}

Bign operator + (const Bign& B) const

{

Bign C;

C.len=max(len,B.len)+10;

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

{

C.N[i]+=N[i]+B.N[i];

C.N[i+1]+=C.N[i]/base;

C.N[i]%=base;

}

C.clear();

return C;

}

Bign operator - (Bign B)

{

Bign C;

C.len=max(len,B.len);

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

{

if(N[i]<B.N[i]) {

N[i+1]--;

N[i]+=base;

}

C.N[i]=N[i]-B.N[i];

}

C.clear();

return C;

}

Bign operator \* (const Bign& B) const

{

Bign C;

C.len=len+B.len;

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

for(int j=0;j<B.len;j++)

C.N[i+j]+=N[i]\*B.N[j];

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

{

C.N[i+1]+=C.N[i]/base;

C.N[i]%=base;

}

C.clear();

return C;

}

Bign operator / (const Bign& B)

{

Bign C,F;

C.len=len;

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

{

F=F\*base;

F.N[0]=N[i];

while(B<=F)

{

F=F-B;

C.N[i]++;

}

}

C.clear();

return C;

}

Bign operator % (const Bign& B)

{

Bign r=\*this/B;

return \*this-r\*B;

}

}A,B;

### //矩阵乘法

//矩阵乘法

struct Mat

{

int r,c; ll N[maxn][maxn];

Mat(int r=0,int c=0)

{

this->r=r,this->c=c;

memset(N,0,sizeof(N));

}

Mat operator \* (const Mat& B) const

{

Mat C(r,B.c);

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

for(int j=0;j<B.c;j++)

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

C.N[i][j]=(C.N[i][j]+N[i][k]\*B.N[k][j])%mod;

return C;

}

Mat operator ^ (int p)

{

Mat ans(r,r),tmp=\*this;

for(int i=0;i<r;i++) ans.N[i][i]=1;

while(p)

{

if(p&1) ans=ans\*tmp;

tmp=tmp\*tmp; p>>=1;

}

return ans;

}

};

## 2．数据结构

### //树状数组

int C[N],mx;

void Add(int x,int v)

{

for(int i=x;i<=mx;i+=i&-i) C[i]+=v;

}

int query(int x)

{

int ans=0;

for(int i=x;i;i-=i&-i) ans+=C[i];

return ans;

}

### //线段树

//区间加,区间乘,区间求和

int mod;

struct Tnode

{

int u,l,r;

ll sum,add,mul;

void mulv(ll x) {

sum=(sum\*x)%mod;

mul=(mul\*x)%mod;

add=(add\*x)%mod;

}

void addv(ll x) {

sum=(sum+(r-l+1)\*x%mod)%mod;

add=(add+x)%mod;

}

void pushdown() ;

void maintain() ;

}T[N];

void Tnode::pushdown() {

if(mul^1) {

T[u<<1].mulv(mul);

T[u<<1|1].mulv(mul);

mul=1;

}

if(add) {

T[u<<1].addv(add);

T[u<<1|1].addv(add);

add=0;

}

}

void Tnode::maintain() {

sum=(T[u<<1].sum+T[u<<1|1].sum)%mod;

}

void update(int u,int L,int R,int x,int f)

{

T[u].pushdown();

if(L<=T[u].l&&T[u].r<=R) {

if(!f) T[u].addv(x);

else T[u].mulv(x);

} else {

int mid=T[u].l+T[u].r>>1;

if(L<=mid) update(u<<1,L,R,x,f);

if(mid<R) update(u<<1|1,L,R,x,f);

T[u].maintain();

}

}

ll query(int u,int L,int R)

{

T[u].pushdown();

if(L<=T[u].l&&T[u].r<=R)

return T[u].sum;

else {

int mid=T[u].l+T[u].r>>1;

ll ans=0;

if(L<=mid) ans=(ans+query(u<<1,L,R))%mod;

if(mid<R) ans=(ans+query(u<<1|1,L,R))%mod;

return ans;

}

}

ll a[N];

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

{

T[u]=(Tnode){ u,l,r,0,0,1 };

if(l==r) {

T[u].sum=a[l];

} else {

int mid=l+r>>1;

build(u<<1,l,mid);

build(u<<1|1,mid+1,r);

T[u].maintain();

}

}

### //Treap

struct Node

{

Node \*ch[2];

int v,r,m,w,s;

Node(int v):v(v) { ch[0]=ch[1]=NULL; r=rand(); s=w=1; }

int cmp(int x) {

if(v==x) return -1;

return x<v? 0:1;

}

void maintain() {

s=w;

if(ch[0]!=NULL) s+=ch[0]->s;

if(ch[1]!=NULL) s+=ch[1]->s;

}

};

void rotate(Node\* &o,int d)

{

Node\* k=o->ch[d^1];o->ch[d^1]=k->ch[d];k->ch[d]=o;

o->maintain(); k->maintain(); o=k;

}

void insert(Node \*&o,int x)

{

if(o==NULL) o=new Node(x);

int d=o->cmp(x);

if(d==-1) o->w++;

else {

insert(o->ch[d],x);

if(o->ch[d]->r > o->r) rotate(o,d^1);

}

o->maintain();

}

void remove(Node \*&o,int x)

{

int d=o->cmp(x);

if(d==-1)

{

if(o->s>1) { o->w--; o->maintain(); return ; }

else {

if(o->ch[0]!=NULL&&o->ch[1]!=NULL) {

int d2=o->ch[0]->r > o->ch[1]->r ? 1:0;

rotate(o,d2); remove(o->ch[d2],x);

} else {

if(o->ch[0]!=NULL) o=o->ch[0]; else o=o->ch[1];

delete o;

}

}

} else

remove(o->ch[d],x);

if(o!=NULL) o->maintain();

}

int kth(Node\* o,int rk)

{

if(o==NULL) return 0;

int s=o->ch[0]==NULL? 0:o->ch[0]->s;

if(rk==s+1) return o->v;

else if(rk<=s) return kth(o->ch[0],rk);

else return kth(o->ch[1],rk-s-o->w);

}

int rank(Node\* o,int x)

{

if(o==NULL) return 0;

int s=o->ch[0]==NULL? 0:o->ch[0]->s;

int d=o->cmp(x);

if(d==-1) return 1;

else if(d==0) return rank(o->ch[0],x);

else return s+o->w+rank(o->ch[1],x);

}

int tmp;

void before(Node\* o,int x)

{

if(o==NULL) return ;

if(o->v<x) { tmp=max(tmp,o->v); before(o->ch[1],x); }

else before(o->ch[0],x);

}

void after(Node\* o,int x)

{

if(o==NULL) return ;

if(o->v>x) { tmp=min(tmp,o->v); after(o->ch[0],x); }

else after(o->ch[1],x);

}

### //splay自上而下

struct Node

{

Node \*ch[2];

int s;

int cmp(int x)

{

int d=x-ch[0]->s;

if(d==1) return -1;

return d<=0? 0:1;

}

void maintain()

{

s=ch[0]->s+ch[1]->s;

}

void pushdown() {}

}mempool[N],\*G=mempool;

Node\* null=new Node();

void rotate(Node\* &o,int d)

{

Node\* k=o->ch[d^1]; o->ch[d^1]=k->ch[d],k->ch[d]=o;

o->maintain(); k->maintain(); o=k;

}

void splay(Node\* &o,int k)

{

o->pushdown();

int d=o->cmp(k);

if(d==1) k-=o->ch[0]->s+1;

if(d!=-1) {

Node\* p=o->ch[d];

p->pushdown();

int d2=p->cmp(k),k2=d2==0? k:k-p->ch[d]->s-1;

if(d2!=-1) {

splay(p->ch[d2],k2);

if(d==d2) rotate(o,d^1); else rotate(o->ch[d],d);

}

rotate(o,d^1);

}

}

Node\* merge(Node\* left,Node\* right)

{

splay(left,left->s);

left->ch[1]=right,left->maintain();

return left;

}

void split(Node\* o,int k,Node\*&left,Node\*&right)

{

splay(o,k);

left=o,right=left->ch[1],left->ch[1]=NULL;

left->maintain();

}

Node\* build(int l,int r)

{

if(r<l) return null;

int mid=l+r>>1;

G->s=1;

G->ch[0]=build(l,mid-1);

G->ch[1]=build(mid+1,r);

G->maintain();

return G++;

}

### //主席树

struct Tnode

{

Tnode \*ls,\*rs;

int sum;

} \*T[N\*50],mempool[N\*50],\*G=mempool;

Tnode\* Nw(Tnode\* l,Tnode\* r,int x)

{

G->ls=l,G->rs=r,G->sum=x;

return G++;

}

Tnode\* build(Tnode\* p,int l,int r,int pos)

{

if(l==r)

return Nw(T[0],T[0],p->sum+1);

else {

int mid=l+r>>1;

if(pos<=mid) Nw(build(p->ls,l,mid,pos),p->rs,p->sum+1);

else return Nw(p->ls,build(p->rs,mid+1,r,pos),p->sum+1);

}

}

int query(Tnode\* x,int l,int r,int pos)

{

if(l==r) return x->sum;

else {

int mid=l+r>>1;

if(pos<=mid) return query(x->ls,l,mid,pos);

else return query(x->rs,mid+1,r,pos);

}

}

### //Link-Cut-Tree

namespace LCT

{

struct Node {

Node \*ch[2],\*fa;

int rev;

//others v

Node() {};

Node(int x) ;

void reverse() {

swap(ch[0],ch[1]);

rev^=1;

}

void up\_push() {

if(fa->ch[0]==this||fa->ch[1]==this)

fa->up\_push();

if(rev) {

ch[0]->reverse();

ch[1]->reverse();

rev=0;

}

}

void maintain() { }

} T[N<<1],\*null=&T[0];

Node::Node(int x) {

ch[0]=ch[1]=fa=null;

rev=0; //v=x;

}

void rot(Node\* o,int d) {

Node\* p=o->fa;

p->ch[d]=o->ch[d^1];

o->ch[d^1]->fa=p;

o->ch[d^1]=p;

o->fa=p->fa;

if(p==p->fa->ch[0])

p->fa->ch[0]=o;

else if(p==p->fa->ch[1])

p->fa->ch[1]=o;

p->fa=o;

p->maintain();

}

void splay(Node\* o) {

o->up\_push();

Node \*nf,\*nff;

while(o->fa->ch[0]==o||o->fa->ch[1]==o) {

nf=o->fa,nff=nf->fa;

if(o==nf->ch[0]) {

if(nf==nff->ch[0]) rot(nf,0);

rot(o,0);

} else {

if(nf==nff->ch[1]) rot(nf,1);

rot(o,1);

}

}

o->maintain();

}

void Access(Node\* o) {

Node \*son=null;

while(o!=null) {

splay(o);

o->ch[1]=son;

o->maintain();

son=o; o=o->fa;

}

}

void evert(Node\* o) {

Access(o);

splay(o);

o->reverse();

}

void Link(Node\* u,Node\* v) {

evert(u);

u->fa=v;

}

void Cut(Node\* u,Node\* v) {

evert(u);

Access(v),splay(v);

v->ch[0]=u->fa=null;

v->maintain();

}

Node\* find(Node\* o) {

while(o->fa!=null) o=o->fa;

return o;

}

}

using namespace LCT ;

## 3图

### //2-sat

struct TwoSAT {

int n;

vector<int> g[N<<1];

int st[N<<1],mark[N<<1],top;

bool dfs(int x) {

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

if(mark[x]) return 1;

mark[x]=1;

st[++top]=x;

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

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

return 1;

}

void init(int n) {

this->n=n;

for(int i=0;i<2\*n;i++) g[i].clear();

memset(mark,0,sizeof(mark));

}

void addc(int x,int xval,int y,int yval) {

x=x\*2+xval;

y=y\*2+yval;

g[x^1].push\_back(y);

g[y^1].push\_back(x);

}

bool solve() {

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

if(!mark[i]&&!mark[i+1]) {

top=0;

if(!dfs(i)) {

while(top) mark[st[top--]]=0;

if(!dfs(i+1)) return 0;

}

}

}

return 1;

}

} s;

### //有向图的强联通分量

//tarjan求SCC

struct Edge {

int v,nxt;

}e[M];

int en=1,front[N];

void adde(int u,int v)

{

e[++en]=(Edge){v,front[u]}; front[u]=en;

}

int n,top,dfn;

int st[N],sccno[N],scc\_cnt,pre[N],lowlink[N];

void tarjan(int u)

{

pre[u]=lowlink[u]=++dfn;

st[++top]=u;

trav(u,i) {

int v=e[i].v;

if(!pre[v]) {

tarjan(v);

lowlink[u]=min(lowlink[u],lowlink[v]);

} else

if(!sccno[v])

lowlink[u]=min(lowlink[u],pre[v]);

}

if(lowlink[u]==pre[u]) {

scc\_cnt++;

for(;;) {

int x=st[top--];

sccno[x]=scc\_cnt;

if(x==u) break;

}

}

}

### //无向图的边的双连通分量

//BCC

struct Edge {

int u,v,nxt;

}e[M];

int en=1,front[N];

void adde(int u,int v)

{

e[++en]=(Edge){u,v,front[u]}; front[u]=en;

}

Edge st[N];

vector<int> bcc[N];

int pre[N],iscut[N],bccno[N],top,dfn,bcc\_cnt;

int dfs(int u,int fa)

{

int lowu=pre[u]=++dfn;

int child=0;

trav(u,i) {

int v=e[i].v;

Edge E=e[i];

if(!pre[v]) {

st[++top]=E;

child++;

int lowv=dfs(v,u);

lowu=min(lowu,lowv);

if(lowv>=pre[u]) {

iscut[u]=1;

bcc\_cnt++;

for(;;) {

Edge x=st[top--];

if(bccno[x.u]!=bcc\_cnt) {

bccno[x.u]=bcc\_cnt;

bcc[bcc\_cnt].push\_back(x.u);

}

if(bccno[x.v]!=bcc\_cnt) {

bccno[x.v]=bcc\_cnt;

bcc[bcc\_cnt].push\_back(x.v);

}

if(x.u==u&&x.v==v) break;

}

}

} else

if(pre[v]<pre[u] && v!=fa) {

st[++top]=E;

lowu=min(lowu,pre[v]);

}

}

if(fa<0&&child==1) iscut[u]=0;

return lowu;

}

### //最短路

#### //spfa

struct Edge {

int v,w,nxt;

}e[M];

int en=1,front[N];

void adde(int u,int v,int w)

{

e[++en]=(Edge){v,w,front[u]}; front[u]=en;

}

queue<int> q;

int inq[N],dis[N];

void spfa(int s)

{

dis[s]=0; inq[s]=1;

q.push(s);

while(!q.empty()) {

int u=q.front(); q.pop();

inq[u]=0;

trav(u,i) {

int v=e[i].v;

if(dis[v]>dis[u]+e[i].w) {

dis[v]=dis[u]+e[i].w;

if(!inq[v]) {

inq[v]=1;

q.push(v);

}

}

}

}

}

#### //dijkstra

struct Node {

int id,dis;

bool operator < (const Node& rhs) const

{

return dis>rhs.dis;

}

};

priority\_queue<Node> q;

int vis[N],dis[N];

void dijkstra(int s)

{

q.push((Node){s,0});

while(!q.empty()) {

int u=q.top().id;

if(vis[u]) continue;

vis[u]=1;

trav(u,i) {

int v=e[i].v;

if(dis[v]>dis[u]+e[i].w) {

dis[v]=dis[u]+e[i].w;

q.push((Node){v,dis[v]});

}

}

}

}

### //最小生成树

#### //Kruskal

int fa[N];

int find(int u)

{

if(!fa[u] || u==fa[u]) return fa[u]=u;

return fa[u]=find(fa[u]);

}

struct Edge {

int u,v,w;

bool operator < (const Edge& rhs) const

{

return w<rhs.w;

}

}e[M];

int tot;

void Kruskal()

{

sort(e+1,e+tot+1);

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

int u=e[i].u,v=e[i].v;

int x=find(u),y=find(v);

if(x!=y) {

fa[x]=y;

//加入树边(u,v)

}

}

}

### //最大流

#### //Dinic算法求最大流

struct Edge {

int u,v,cap,flow;

};

struct Dinic {

int d[N],cur[N],vis[N];

vector<Edge> es;

vector<int> g[N];

queue<int> q;

void AddEdge (int u,int v,int w) {

es.push\_back((Edge){u,v,w,0});

es.push\_back((Edge){v,u,0,0});

int m=es.size();

g[u].push\_back(m-2);

g[v].push\_back(m-1);

}

bool bfs(int s,int t) {

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

d[s]=0; vis[s]=1;

q.push(s);

while(!q.empty()) {

int u=q.front(); q.pop();

FOR(i,0,(int)g[u].size()-1) {

Edge& e=es[g[u][i]];

int v=e.v;

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

vis[v]=1;

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

q.push(v);

}

}

}

return vis[t];

}

int dfs(int u,int a,int t) {

if(u==t||a==0) return a;

int flow=0,f;

for(int& i=cur[u];i<g[u].size();i++) {

Edge& e=es[g[u][i]];

int v=e.v;

if(d[v]==d[u]+1&&(f=dfs(v,min(a,e.cap-e.flow),t))>0) {

e.flow+=f;

es[g[u][i]^1].flow-=f;

flow+=f,a-=f;

if(!a) break;

}

}

return flow;

}

int maxflow(int s,int t) {

int flow=0;

while(bfs(s,t)) {

memset(cur,0,sizeof(cur));

flow+=dfs(s,inf,t);

}

return flow;

}

} dc;

### //最小费用最大流

#### //最短路算法求最小费用最大流

struct Edge {

int u,v,cap,flow,cost;

Edge(int \_,int \_\_,int \_\_\_,int \_\_\_\_,int \_\_\_\_\_)

{ u=\_,v=\_\_,cap=\_\_\_,flow=\_\_\_\_,cost=\_\_\_\_\_; }

};

struct MCMF {

int n,m,s,t;

int d[N],p[N],a[N],inq[N];

vector<Edge> es;

vector<int> g[N];

queue<int> q;

void init(int n) {

this->n=n;

es.clear();

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

}

void AddEdge(int u,int v,int w,int c) {

es.push\_back(Edge(u,v,w,0,c));

es.push\_back(Edge(v,u,0,0,-c));

int m=es.size();

g[u].push\_back(m-2);

g[v].push\_back(m-1);

}

bool spfa(int s,int t,ll& flow,ll& cost) {

memset(inq,0,sizeof(inq));

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

inq[s]=1; d[s]=p[s]=0; a[s]=inf;

q.push(s);

while(!q.empty()) {

int u=q.front(); q.pop();

inq[u]=0;

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

Edge& e=es[g[u][i]];

int v=e.v;

if(d[v]>d[u]+e.cost && e.cap>e.flow) {

d[v]=d[u]+e.cost;

a[v]=min(a[u],e.cap-e.flow);

p[v]=g[u][i];

if(!inq[v])

inq[v]=1 , q.push(v);

}

}

}

if(d[t]==inf) return 0;

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

for(int x=t;x!=s;x=es[p[x]].u) {

es[p[x]].flow+=a[t];

es[p[x]^1].flow-=a[t];

}

return 1;

}

void mcmf(int s,int t,ll& cost,ll& flow) {

flow=cost=0;

while(spfa(s,t,cost,flow)) ;

}

} mc;

### //KM算法

#### //KM算法求二分图的最佳完美匹配

struct KM {

int slack[N],res[N];

int l[N],r[N],lx[N],rx[N],g[N][N];

void clear(int n) {

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

res[i]=0;

for(int j=1;j<=n;j++) g[i][j]=-1;

}

}

bool find(int x,int n) {

lx[x]=1;

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

if(!rx[i]&&g[x][i]!=-1) {

int tmp=g[x][i]-l[x]-r[i];

if(!tmp) {

rx[i]=1;

if(!res[i]||find(res[i],n)) {

res[i]=x;

return 1;

}

} else

slack[i]=min(slack[i],tmp);

}

return 0;

}

int solve(int n) {

if(!n) return 0;

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

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

l[i]=INF;

for(int j=1;j<=n;j++) if(g[i][j]!=-1)

l[i]=min(l[i],g[i][j]);

}

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

for(int j=1;j<=n;j++) slack[j]=INF;

for(;;) {

for(int j=1;j<=n;j++) lx[j]=rx[j]=0;

if(find(i,n)) break;

int mini=INF;

for(int i=1;i<=n;i++) if(!rx[i])

mini=min(mini,slack[i]);

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

if(lx[i]) l[i]+=mini;

if(rx[i]) r[i]-=mini;

else slack[i]-=mini;

}

}

}

int ans=0;

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

ans+=l[i]+r[i];

return ans;

}

} km;

## 4树

### //LCA

#### //倍增法求LCA

//倍增法可以在线构造 比较灵活

struct Edge {

int v,nxt;

}e[M];

int en=1,front[N];

void adde(int u,int v)

{

e[++en]=(Edge){v,front[u]}; front[u]=en;

}

int fa[N][D],dep[N];

void dfs(int u)

{

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

fa[u][i]=fa[fa[u][i-1]][i-1];

trav(u,i) {

int v=e[i].v;

if(v!=fa[u][0]) {

fa[v][0]=u;

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

dfs(v);

}

}

}

int lca(int u,int v)

{

if(dep[u]<dep[v]) swap(u,v);

int t=dep[u]-dep[v];

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

if(t&(1<<i)) u=fa[u][i];

if(u==v) return u;

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

if(fa[u][i]!=fa[v][i])

u=fa[u][i],v=fa[v][i];

return fa[u][0];

}

#### //树链剖分求LCA

//比较快

struct Edge {

int v,nxt;

}e[M];

int en=1,front[N];

void adde(int u,int v)

{

e[++en]=(Edge){v,front[u]}; front[u]=en;

}

int fa[N],top[N],siz[N],dep[N],son[N];

void dfs1(int u)

{

siz[u]=1; son[u]=0;

trav(u,i) {

int v=e[i].v;

if(v!=fa[u]) {

fa[v]=u;

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

dfs1(v);

siz[u]+=siz[v];

if(siz[v]>siz[son[u]]) son[u]=v;

}

}

}

void dfs2(int u,int tp)

{

top[u]=tp;

if(son[u]) dfs2(son[u],tp);

trav(u,i)

if(e[i].v!=fa[u]&&e[i].v!=son[u])

dfs2(e[i].v,e[i].v);

}

int lca(int u,int v)

{

while(top[u]!=top[v]) {

if(dep[top[u]]<dep[top[v]]) swap(u,v);

u=fa[top[u]];

}

return dep[u]<dep[v]? u:v;

}

### //树链剖分

#### //树链剖分

struct Edge {

int v,nxt;

}e[M];

int en=1,front[N];

void adde(int u,int v)

{

e[++en]=(Edge){v,front[u]}; front[u]=en;

}

int fa[N],top[N],siz[N],dep[N],son[N],bl[N],dfn;

void dfs1(int u)

{

siz[u]=1; son[u]=0;

trav(u,i) {

int v=e[i].v;

if(v!=fa[u]) {

fa[v]=u;

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

dfs1(v);

siz[u]+=siz[v];

if(siz[v]>siz[son[u]]) son[u]=v;

}

}

}

void dfs2(int u,int tp)

{

top[u]=tp; bl[u]=++dfn;

if(son[u]) dfs2(son[u],tp);

trav(u,i)

if(e[i].v!=fa[u]&&e[i].v!=son[u])

dfs2(e[i].v,e[i].v);

}

//以合适的数据结构T维护重链

int ans;

int query(int u,int v)

{

while(top[u]!=top[v]) {

if(dep[top[u]]<dep[top[v]]) swap(u,v);

ans<-query(T,bl[top[u]],bl[u]);

u=fa[top[u]];

}

if(u==v) return ;

if(dep[u]>dep[v]) swap(u,v);

ans<-query(T,bl[u],bl[v]);

<-ans

}

//类似-查询树上任意两节点的方法

void modify() {}

### //点分治

#### //点分治

struct Edge {

int v,nxt;

}e[M];

int en=1,front[N];

void adde(int u,int v)

{

e[++en]=(Edge){v,front[u]}; front[u]=en;

}

int rt,n,size,vis[N],siz[N],f[N],dep[N];

void get\_root(int u,int fa)

{

siz[u]=1; f[u]=0;

trav(u,i) {

int v=e[i].v;

if(v!=fa) {

get\_root(v,u);

siz[u]+=siz[v];

if(siz[v]>f[u]) f[u]=siz[v];

}

}

f[u]=max(f[u],size-siz[u]);

if(f[u]<f[rt]) rt=u;

}

void solve(int u)

{

vis[u]=1;

//计算经过根u的信息

trav(u,i) if(!vis[e[i].v])

{

//统计当前子树信息

//与前i-1个子树信息结合计算贡献

//将当前子树信息加入前i-1个子树信息

}

trav(u,i) if(!vis[e[i].v]) {

int v=e[i].v;

size=siz[v]; rt=0;

get\_root(v,-1);

solve(rt);

}

}

int main()

{

//blabla

size=f[0]=n;

rt=0; get\_root(rt,-1);

solve(rt);

}

## 5字符串

### //KMP

//KMP算法

//s[s]==s[fail[i]] , fail[i]!=0

int f[N]; char s[N];

void get\_fail()

{

int j=0;

int n=strlen(s);

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

while(j&&s[j+1]!=s[i]) j=f[j];

if(s[j+1]==s[i]) j++;

f[i]=j;

}

}

### //AC自动机

//AC自动机

struct AC\_auto {

int sz,ch[N][26],f[N],val[N];

AC\_auto() {

sz=1;

memset(ch,0,sizeof(ch));

}

void insert(char\* s) {

int u=0;

for(int i=0;s[i];i++) {

int c=s[i]-‘a‘;

if(!ch[u][c]) ch[u][c]=++sz;

u=ch[u][c];

}

val[u]=1;

}

void get\_fail() {

queue<int> q;

f[0]=0;

for(int c=0;c<26;c++)

if(ch[0][c]) f[ch[0][c]]=0,q.push(ch[0][c]);

while(!q.empty()) {

int qr=q.front(); q.pop();

for(int c=0;c<26;c++) {

int u=ch[qr][c];

if(!u) continue;

q.push(u);

int v=f[qr];

while(v&&!ch[v][c]) v=f[v];

if(val[ch[v][c]]) val[u]=1;

f[u]=ch[v][c];

}

}

}

};

### //后缀自动机SAM

struct SAM {

int sz,last,fa[N],ch[N][26],l[N];

SAM() {

sz=0; last=++sz;

memset(l,0,sizeof(l));

memset(fa,0,sizeof(fa));

}

void Add(int c) {

int np=++sz,p=last; last=np;

l[np]=l[p]+1;

for(;p&&!ch[p][c];p=fa[p]) ch[p][c]=np;

if(!p) fa[np]=1;

else {

int q=ch[p][c];

if(l[q]==l[p]+1) fa[np]=q;

else {

int nq=++sz; l[nq]=l[p]+1;

memcpy(ch[nq],ch[q],sizeof(ch[q]));

fa[nq]=fa[q];

fa[q]=fa[np]=nq;

for(;q==ch[p][c];p=fa[p]) ch[p][c]=nq;

}

}

}

//do some other things

} sam;

### //后缀数组

#define rep(a,b,c) for(int a=(b);a>=(c);a--)

#define FOR(a,b,c) for(int a=(b);a<=(c);a++)

char s[N];

int c[N],t[N],t2[N],height[N],rank[N],sa[N];

void build\_sa(int m,int n) {

int \*x=t,\*y=t2,p,k;

FOR(i,0,m-1) c[i]=0;

FOR(i,0,n-1) c[x[i]=s[i]]++;

FOR(i,0,m-1) c[i]+=c[i-1];

rep(i,n-1,0) sa[--c[x[i]]]=i;

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

p=0;

FOR(i,n-k,n-1) y[p++]=i;

FOR(i,0,n-1) if(sa[i]>=k) y[p++]=sa[i]-k;

FOR(i,0,m-1) c[i]=0;

FOR(i,0,n-1) c[x[y[i]]]++;

FOR(i,0,m-1) c[i]+=c[i-1];

rep(i,n-1,0) sa[--c[x[y[i]]]]=y[i];

swap(x,y);

p=1; x[sa[0]]=0;

FOR(i,1,n-1)

x[sa[i]]=y[sa[i]]==y[sa[i-1]]&&y[sa[i]+k]==y[sa[i-1]+k]? p-1:p++;

if(p>=n) break;

m=p;

}

}

void get\_height(int n)

{

FOR(i,0,n-1) rank[sa[i]]=i;

int k=0;

FOR(i,0,n-1) {

if(k) k--;

int j=sa[rank[i]-1];

while(s[i+k]==s[j+k]) k++;

height[rank[i]]=k;

}

}

Manacher：

### //Manacher算法

char s[N],a[N];

int p[N];

void Add(int l,int r)

{

l=l/2,r=r/2-1;

if(l>r) return ;

//q[++tot]=(Seg){l,r};

//[l,r]为一个回文串

}

void Manacher()

{

int n=strlen(s+1);

int m=n\*2+1;

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

a[i<<1]=s[i];

a[i<<1|1]=‘#‘;

}

a[0]=‘+‘,a[1]=‘#‘,a[m+1]=‘-‘;

int mx=0,id;

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

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

else p[i]=1;

while(a[i-p[i]]==a[i+p[i]]) p[i]++;

Add(i-p[i],i+p[i]);

if(p[i]+i>mx) mx=i+p[i],id=i;

}

}

## 6计算几何

计算几何基础知识：

### //计算几何基础

const double eps = 1e-10;

int dcmp(double x) {

if(fabs(x)<eps) return 0; else return x<0? -1:1;

}

struct Pt {

double x,y;

Pt(double x=0,double y=0):x(x),y(y) {}

};

typedef Pt vec;

vec operator - (Pt A,Pt B) { return vec(A.x-B.x,A.y-B.y); }

vec operator + (vec A,vec B) { return vec(A.x+B.x,A.y+B.y); }

vec operator \* (vec A,double p) { return vec(A.x\*p , A.y\*p); }

bool operator < (const Pt& a,const Pt& b) {

return a.x<b.x || (a.x==b.x && a.y<b.y);

}

bool operator == (const Pt& a,const Pt& b) {

return dcmp(a.x-b.x)==0 && dcmp(a.y-b.y)==0;

}

double cross(vec A,vec B) { return A.x\*B.y-A.y\*B.x; }

double Dot(vec A,vec B) { return A.x\*B.x+A.y\*B.y; }

double Len(vec A) { return sqrt(Dot(A,A)); }

double Angle(vec A,vec B) { return acos(Dot(A,B)/Len(A)/Len(B)); }

#### //逆时针旋转rad角度

vec rotate(vec A,double rad) {

return vec(A.x\*cos(rad)-A.y\*sin(rad) , A.x\*sin(rad)+A.y\*cos(rad));

}

#### //法向量 左转90度 长度归1

vec Normal(vec A)

{

double L=Len(A);

return vec(-A.y/L,A.x/L);

}

#### //判断点在线段上

bool OnSeg(Pt P,Pt a1,Pt a2) {

return dcmp(cross(a1-P,a2-P))==0 && dcmp(Dot(a1-P,a2-P))<0;

}

#### //直线交点

Pt LineIntersection(Pt P,vec v,Pt Q,vec w) {

vec u=P-Q;

double t=cross(w,u)/cross(v,w);

return P+v\*t;

}

double DistoLine(Pt P,Pt A,Pt B) {

vec v1=B-A,v2=P-A;

return fabs(cross(v1,v2))/Len(v1);

}

#### //线段不含端点 判断相交

bool SegIntersection(Pt a1,Pt a2,Pt b1,Pt b2) {

double c1=cross(a2-a1,b1-a1) , c2=cross(a2-a1,b2-a1) ,

c3=cross(b2-b1,a1-b1) , c4=cross(b2-b1,a2-b1);

return dcmp(c1)\*dcmp(c2)<0 && dcmp(c3)\*dcmp(c4)<0;

// b1 b2在线段a1a2的两侧 a1 a2在线段b1b2的两侧 规范相交

}

#### //线段含端点 判断线段严格相交

bool SegInter(Pt s1, Pt e1, Pt s2, Pt e2) {

if( min(s1.x, e1.x) <= max(s2.x, e2.x) &&

min(s1.y, e1.y) <= max(s2.y, e2.y) &&

min(s2.x, e2.x) <= max(s1.x, e1.x) &&

min(s2.y, e2.y) <= max(s1.y, e1.y) &&

cross(e1-s1,s2-s1) \* cross(e1-s1,e2-s1) <= 0 &&

cross(e2-s2,s1-s2) \* cross(e2-s2,e1-s2) <= 0

) return true;

return false;

}

#### //点到线段的距离

double DistoSeg(Pt P,Pt A,Pt B) {

if(A==B) return Len(P-A);

vec v1=B-A , v2=P-A , v3=P-B;

if(dcmp(Dot(v1,v2))<0) return Len(v2);

else if(dcmp(Dot(v1,v3))>0) return Len(v3);

else return fabs(cross(v1,v2))/Len(v1);

}

#### //多边形面积

double PolygonArea(Pt\* p,int n)

{

double S=0;

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

S+=cross(p[i]-p[0],p[i+1]-p[0]);

return S/2;

}

### //凸包

const int N = 400000+10;

const double PI = acos(-1.0);

const double eps = 1e-12;

int dcmp(double x) {

if(fabs(x)<eps) return 0; else return x<0? -1:1;

}

struct Pt {

double x,y;

Pt(double x=0,double y=0) :x(x),y(y) {};

};

typedef Pt vec;

vec operator - (Pt a,Pt b) { return vec(a.x-b.x,a.y-b.y); }

vec operator + (vec a,vec b) { return vec(a.x+b.x,a.y+b.y); }

bool operator == (Pt a,Pt b) {

return dcmp(a.x-b.x)==0 && dcmp(a.y-b.y)==0;

}

bool operator < (const Pt& a,const Pt& b) {

return a.x<b.x || (a.x==b.x && a.y<b.y);

}

vec rotate(vec a,double x) {

return vec(a.x\*cos(x)-a.y\*sin(x),a.x\*sin(x)+a.y\*cos(x));

}

double cross(vec a,vec b) { return a.x\*b.y-a.y\*b.x; }

double dist(Pt a,Pt b) {

return sqrt((a.x-b.x)\*(a.x-b.x)+(a.y-b.y)\*(a.y-b.y));

}

vector<Pt> ConvexHull(vector<Pt> p) {

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

p.erase(unique(p.begin(),p.end()),p.end());

int n=p.size() , m=0;

vector<Pt> ch(n+1);

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

while(m>1 && cross(ch[m-1]-ch[m-2],p[i]-ch[m-2])<=0) m--;

ch[m++]=p[i];

}

int k=m;

for(int i=n-2;i>=0;i--) {

while(m>k && cross(ch[m-1]-ch[m-2],p[i]-ch[m-2])<=0) m--;

ch[m++]=p[i];

}

if(n>1) m--;

ch.resize(m); return ch;

}

### //半平面交

const int N = 305;

const double bond = 100001;

const double eps = 1e-10;

struct Pt {

double x,y;

Pt (double x=0,double y=0):x(x),y(y){}

};

typedef Pt vec;

vec operator + (Pt a,Pt b) { return vec(a.x+b.x,a.y+b.y); }

vec operator - (Pt a,Pt b) { return vec(a.x-b.x,a.y-b.y); }

vec operator \* (Pt a,double p) { return vec(a.x\*p,a.y\*p); }

double cross(Pt a,Pt b) { return a.x\*b.y-a.y\*b.x; }

struct Line {

Pt p; vec v; double ang;

Line () {}

Line (Pt p,vec v) :p(p),v(v){ ang=atan2(v.y,v.x); }

bool operator < (const Line& rhs) const {

return ang<rhs.ang;

}

};

bool onleft(Line L,Pt p) { return cross(L.v,p-L.p)>0; }

Pt LineInter(Line a,Line b)

{

vec u=a.p-b.p;

double t=cross(b.v,u)/cross(a.v,b.v);

return a.p+a.v\*t;

}

vector<Pt> HPI(vector<Line> L)

{

int n=L.size();

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

int f,r;

vector<Pt> p(n) , ans;

vector<Line> q(n);

q[f=r=0]=L[0];

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

while(f<r&&!onleft(L[i],p[r-1])) r--;

while(f<r&&!onleft(L[i],p[f])) f++;

q[++r]=L[i];

if(fabs(cross(q[r].v,q[r-1].v))<eps) {

r--;

if(onleft(q[r],L[i].p)) q[r]=L[i];

}

if(f<r) p[r-1]=LineInter(q[r-1],q[r]);

}

while(f<r&&!onleft(q[f],p[r-1])) r--;

if(r-f<=1) return ans;

p[r]=LineInter(q[r],q[f]);

for(int i=f;i<=r;i++) ans.push\_back(p[i]);

return ans;

}