# 模板整理

目录

[模板整理 1](#_Toc415866304)

[【数论】 1](#_Toc415866305)

[1.扩展的欧几里德定理 1](#_Toc415866306)

[2.求模乘法的逆 2](#_Toc415866307)

[3.快速幂求解a^b 2](#_Toc415866308)

[4.求解模方程a^x=b(mod n)。 2](#_Toc415866309)

[5.中国剩余定理 5](#_Toc415866310)

[6.素数 6](#_Toc415866311)

[7.欧拉phi函数 7](#_Toc415866312)

[【数据结构】 8](#_Toc415866313)

[1.点更新线段树 8](#_Toc415866314)

[2.区间更新线段树 9](#_Toc415866315)

[【图论】 11](#_Toc415866316)

[1.LCA-tarjan 11](#_Toc415866317)

[2.tarjan(用于缩点) 13](#_Toc415866318)

[3.最大流-Dinic(模拟栈) 15](#_Toc415866319)

[4.最小费用流 17](#_Toc415866320)

[5.二分图匹配 19](#_Toc415866321)

[【其他】 20](#_Toc415866322)

[Set: 20](#_Toc415866323)

[最长回文子串(Manacher) 21](#_Toc415866324)

[位运算 23](#_Toc415866325)

[旋转卡壳 23](#_Toc415866326)

## 【数论】

### 1.扩展的欧几里德定理

//拓展欧几里得定理，求ax+by=gcd(a,b)的一组解(x,y),d=gcd(a,b)

void gcd(int a,int b,int &d,int &x,int &y)

{

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

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

}

用法1：

求ax+by=c的整数解。ax+by=gcd(a,b)=g的一组解为(x0,y0),则ax+by=c的一组解为(x0\*c/g,y0\*c/g)。当c不是g的倍数时无整数解

若(x1,y1)是ax+by=c的一组解，则其任意整数解为(x1+k\*bb,y1-k\*aa)，其中aa=a/gcd(a,b),bb=b/gcd(bb)，k为任意整数

### 2.求模乘法的逆

//求得a在模n条件下的逆

int inv(int a,int n)

{

int d,x,y;

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

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

}

或则：

v=pow\_mod(a,n-m-1,n);//n为素数，pow\_mod(a,n-1,n)=1,费马小定理。所以a^m\*a^(n-m-1)=a^(n-1)=1(mod n).

### 3.快速幂求解a^b

//快速幂求a^b

int pow\_mod(int a,int b)

{

int s=1;

while(b)

{

if(b&1)

s=(s\*a)%mod;

a=(a\*a)%mod;

b=b>>1;

}

return s;

}

### 4.求解模方程a^x=b(mod n)。

1.n为素数。无解返回-1

//求解模方程a^x=b(mod n)。n为素数，无解返回-1

int log\_mod (int a,int b,int n)

{

int m,v,e=1,i;

m=ceil(sqrt(n+0.5));

v=inv(pow\_mod(a,m),n);

map<int,int>x;

x[1]=0;

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

{

e=(e\*a)%n;

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

}

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

{

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

b=(b\*v)%n;

}

return -1;

}

2.n不是素数。

//hdu 2815 Mod Tree

#include<bits/stdc++.h>

using namespace std;

#define LL \_\_int64

LL gcd(LL a,LL b)

{

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

}

//拓展欧几里得定理，求ax+by=gcd(a,b)的一组解(x,y),d=gcd(a,b)

void gcd\_mod(LL a,LL b,LL &d,LL &x,LL &y)

{

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

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

}

//求解模方程d\*a^(x-c)=b(mod n)。d,a和n互质，无解返回-1

LL log\_mod (LL a,LL b,LL n,LL c,LL d)

{

LL m,v,e=1,i,x,y,dd;

m=ceil(sqrt(n+0.5)); //x=i\*m+j

map<LL,LL>f;

f[1]=m;

for(i=1;i<m;i++) //建哈希表，保存a^0,a^1,...,a^m-1

{

e=(e\*a)%n;

if(!f[e])f[e]=i;

}

e=(e\*a)%n;//e=a^m

for(i=0;i<m;i++)//每次增加m次方，遍历所有1<=f<=n

{

gcd\_mod(d,n,dd,x,y);//d\*x+n\*y=1-->(d\*x)%n=1-->d\*(x\*b)%n==b

x=(x\*b%n+n)%n;

if(f[x])

{

LL num=f[x];

f.clear();//需要清空，不然会爆内存

return c+i\*m+(num==m?0:num);

}

d=(d\*e)%n;

}

return -1;

}

int main()

{

LL a,b,n;

while(scanf("%I64d%I64d%I64d",&a,&n,&b)!=EOF)

{

if(b>=n)

{

printf("Orz,I can’t find D!\n");

continue;

}

if(b==0)

{

printf("0\n");

continue;

}

LL ans=0,c=0,d=1,t;

while((t=gcd(a,n))!=1)

{

if(b%t){ans=-1;break;}

c++;

n=n/t;

b=b/t;

d=d\*a/t%n;

if(d==b){ans=c;break;}//特判下是否成立。

}

if(ans!=0)

{

if(ans==-1){printf("Orz,I can’t find D!\n");}

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

}

else

{

ans=log\_mod(a,b,n,c,d);

if(ans==-1)printf("Orz,I can’t find D!\n");

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

}

}

return 0;

}

/\*

求解模方程a^x=b(mod n)，n不为素数。拓展Baby Step Giant Step

方法：

初始d=1,c=0,i=0;

1.令g=gcd(a,n),若g==1则执行下一步。否则由于a^x=k\*n+b;(k为某一整数),则(a/g)\*a^k=k\*(n/g)+b/g,(b/g为整除，若不成立则无解)令n=n/g，d=d\*a/g，b=b/g,c++则d\*a^(x-c)=b(mod n),接着重复1步骤。

2.通过1步骤后，保证了a和d都与n互质，方程转换为d\*a^(x-c)=b(mod n)。由于a和n互质，所以由欧拉定理a^phi(n)==1(mod n),(a,n互质)可知，phi(n)<=n,a^0==1(mod n),所以构成循环，且循环节不大于n。从而推出如果存在解，则必定1<=x<n。(在此基础上我们就可以用Baby Step Giant Step方法了)

3.令m=ceil(sqrt(n)),则m\*m>=n。用哈希表存储a^0,a^1,..,a^(m-1)，接着判断1~m\*m-1中是否存在解。

4.为了减少时间，所以用哈希表缩减复杂度。分成m次遍历，每次遍历a^m长度。由于a和d都与n互质，所以gcd(d,n)=1，所以用拓展的欧几里德定理求得d\*x+n\*y=gcd(d,n)=1,的一组整数解(x,y)。则d\*x+n\*y=1-->d\*x%n=(d\*x+n\*y)%n=1-->d\*(x\*b)%n=((d\*x)%n\*b%n)%n=b。

所以若x\*b在哈希表中存在，值为k，则a^k\*d=b(mod n),答案就是ans=k+c+i\*m。如果不存在，则令d=d\*a^m,i++后遍历下一个a^m，直到遍历a^0到a^(m-1)还未找到，则说明不解并退出。

### 5.中国剩余定理

互质情况

//中国剩余定理，求得M%A=a,M%B=b,...中的M，其中A,B,C...互质

int china(int a[])

{

int i,j,k,d,ans=0,x,y,M=1;

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

M=M\*x[i];

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

{

m=M/x[i];

gcd(x[i],m,d,x,y);

ans=(ans+y\*m\*a[i])%M;

}

return (ans+M)%M;

}

不互质的情况

int China(int n)

{

int m1,r1,m2,r2,flag=0,i,d,x,y,c,t;

scanf("%d%d",&m1,&r1);

flag=0;

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

{

scanf("%d%d",&m2,&r2);

if(flag)continue;

gcd(m1,m2,d,x,y);//d=gcd(m1,m2);x\*m1+y\*m2=d;

c=r2-r1;

if(c%d)//对于方程m1\*x+m2\*y=c，如果c不是d的倍数就无整数解

{

flag=1;

continue;

}

t=m2/d;//对于方程m1x+m2y=c=r2-r1,若(x0,y0)是一组整数解,那么(x0+k\*m2/d,y0-k\*m1/d)也是一组整数解(k为任意整数)

//其中x0=x\*c/d,y0=x\*c/d;

x=(c/d\*x%t+t)%t;//保证x0是正数，因为x+k\*t是解，(x%t+t)%t也必定是正数解(必定存在某个k使得(x%t+t)%t=x+k\*t)

r1=m1\*x+r1;//新求的r1就是前i组的解，Mi=m1\*x+M(i-1)=r2-m2\*y(m1为前i个m的最小公倍数);对m2取余时，余数为r2；

//对以前的m取余时，Mi%m=m1\*x%m+M(i-1)%m=M(i-1)%m=r

m1=m1\*m2/d;

}

if(flag)return -1;

else return r1;

}

### 6.素数

线性筛法

在O（n）时间复杂度内找出所有的素数

void init()//预处理，找出所有1e7以内的素数，以减少查找1e14范围数的因子的时间

{ //现行筛素数的方法，时间复杂度为O(n)

memset(check,false,sizeof(check));

int i,j;

tot=0;

for(i=2;i<=1e7;i++)

{

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

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

{

if(i\*prime[j]>1e7)break;

check[i\*prime[j]]=true;

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

}

}

//printf("%d\n",tot);

//for(i=0;i<20;i++)

// printf("prime[%d]:%d\n",i,prime[i]);

}

### 7.欧拉phi函数

求不超过n且与n互质的正整数个数

int euler\_phi(int n)//求单个欧拉函数

{

int m=(int)sqrt(n+0.5);

int i,ans=n;

for(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;

}

int phi[maxn];

void euler\_phi()

{

int i,j,k;

//欧拉函数，phi[i]表示不超过i的与i互质的整数个数

for(i=2;i<maxn;i++)phi[i]=0;

phi[1]=1;

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

if(!phi[i])

for(j=i;j<=maxn;j+=i){

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

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

}

}

## 【数据结构】

### 1.点更新线段树

struct tree

{

int l,r;

int s;

}t[400001];

int a[100001];

int p,b,l,r;

int calc(int x,int y)//加和运算，如果求最大最小值，更改这里就可以了。

{

return x+y;

}

void buildtree(int id,int l,int r,int \*a)

{

if (l==r)

{

t[id].l=l;

t[id].r=r;

t[id].s=a[l];

} else

{

int mid=(l+r)/2;

buildtree(id\*2,l,mid,a);

buildtree(id\*2+1,mid+1,r,a);

t[id].l=l;

t[id].r=r;

t[id].s=calc(t[id\*2].s,t[id\*2+1].s);

}

}

void update(int id,int s,int w)//s点加上w，可以根据需要改成赋值号

{

if (t[id].l==t[id].r)

{

t[id].s+=w;

} else

{

int mid=(t[id].l+t[id].r)/2;

if (s<=mid) update(id\*2,s,w);

else update(id\*2+1,s,w);

t[id].s=calc(t[id\*2].s,t[id\*2+1].s);

}

}

int sum(int id,int l,int r)

{

if ((t[id].l==l)&&(t[id].r==r))

{

return t[id].s;

} else

{

int mid=(t[id].l+t[id].r)/2;

if (r<=mid) return sum(id\*2,l,r);

if (mid<l) return sum(id\*2+1,l,r);

return calc(sum(id\*2,l,mid),sum(id\*2+1,mid+1,r));

}

}

### 2.区间更新线段树

struct tree

{

int l,r;

long long sum,lazy;

};

tree t[4000004];

int a[1000001];

int temp,n,m,i,l,r,s;

void pushdown(int id)

{

long long temp=t[id].lazy;

int l=id\*2,r=id\*2+1;

t[l].lazy+=temp;

t[r].lazy+=temp;

t[l].sum+=(long long)(t[l].r-t[l].l+1)\*temp;

t[r].sum+=(long long)(t[r].r-t[r].l+1)\*temp;

t[id].lazy=0;

}

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

{

if (l==r)

{

t[id].l=l;

t[id].r=r;

t[id].sum=a[l];

t[id].lazy=0;

} else

{

int mid=(l+r)/2;

t[id].l=l;

t[id].r=r;

t[id].lazy=0;

buildtree(id\*2,l,mid);

buildtree(id\*2+1,mid+1,r);

t[id].sum=t[id\*2].sum+t[id\*2+1].sum;

}

}

void add(int id,int l,int r,int s)

{

if ((t[id].l==l)&&(t[id].r==r))

{

t[id].lazy+=(long long)s;

t[id].sum+=(long long)(r-l+1)\*(long long)s;

} else

{

int mid=(t[id].l+t[id].r)/2;

if (t[id].lazy!=0) pushdown(id);

if (r<=mid) add(id\*2,l,r,s);

else if (mid<l) add(id\*2+1,l,r,s);

else

{

add(id\*2,l,mid,s);

add(id\*2+1,mid+1,r,s);

}

t[id].sum=t[id\*2].sum+t[id\*2+1].sum;

}

}

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

{

if ((t[id].l==l)&&(t[id].r==r))

{

return t[id].sum;

} else

{

int mid=(t[id].l+t[id].r)/2;

if (t[id].lazy!=0) pushdown(id);

if (mid<l) return query(id\*2+1,l,r);

if (r<=mid) return query(id\*2,l,r);

return query(id\*2,l,mid)+query(id\*2+1,mid+1,r);

}

}

## 【图论】

### 1.LCA-tarjan

int i,j,k,n,m,x,y,T,ans,big,cas,num,len;

bool flag;

int u,v,w,d;

int edge,head[MAXN];

int fa[MAXN],query[MAXN][3],dist[MAXN];

bool vis[MAXN];

struct node

{

int v,id;

node (int \_v,int \_id):v(\_v),id(\_id){}

};

int find(int x)

{

if (x==fa[x]) return fa[x];

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

}

vector <vector<node> > mp;

struct edgenode

{

int to,next,w;

} G[MAXM];

void add\_edge(int x,int y,int w)

{

G[edge].to=y;

G[edge].w=w;

G[edge].next=head[x];

head[x]=edge++;

}

void tarjan(int u)

{

vis[u]=true;

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

{

int v=mp[u][i].v,id=mp[u][i].id;

if (vis[v]) query[id][2]=find(v);

}

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

{

int v=G[i].to,w=G[i].w;

if (!vis[v])

{

dist[v]=dist[u]+w;

tarjan(v);

fa[v]=u;

}

}

}

int main()

{

scanf("%d",&T);

while (T--)

{

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

edge=0;

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

for (i=0;i<n-1;i++)

{

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

add\_edge(u,v,w);

add\_edge(v,u,w);

}

mp.clear();

mp.resize(n+4);

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

{

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

query[i][0]=u;

query[i][1]=v;

mp[u].push\_back(node(v,i));

mp[v].push\_back(node(u,i));

}

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

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

dist[1]=0;

tarjan(1);

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

{

u=query[i][0];

v=query[i][1];

d=query[i][2];

printf("%d\n",dist[u]+dist[v]-2\*dist[d]);

}

}

return 0;

}

### 2.tarjan(用于缩点)

int i,j,k,n,T,ans,K,m,x,y,indexs,nn,mm;

int ru[P],head[P],head2[P],dfn[P],low[P],instack[P],belong[P];

bool b[E];

stack <int> tar;

struct node

{

int from;

int to;

int next;

}G[E],G2[E];

void addedge(int num,int x,int y)

{

G[num].from=x;

G[num].to=y;

G[num].next=head[x];

head[x]=num;

}

void addedge2(int num,int x,int y)

{

//printf("NewEdge %d %d\n",x,y);

ru[y]++;

G2[num].from=x;

G2[num].to=y;

G2[num].next=head2[x];

head2[x]=num;

}

void tarjan(int k)

{

int p;

tar.push(k);

instack[k]=1;

dfn[k]=low[k]=++indexs;

for(int j=head[k];j!=-1;j=G[j].next)

{

p=G[j].to;

if (instack[p])

{

low[k]=min(low[k],dfn[p]);

}

else

if(dfn[p]==-1)

{

tarjan(p);

low[k]=min(low[k],low[p]);

}

}

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

{

nn++;

do

{

j=tar.top();

tar.pop();

instack[j]=0;

belong[j]=nn;

}while(j!=k);

}

}

void build\_new\_map()

{

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

{

if(belong[G[i].from]==belong[G[i].to])

continue;

addedge2(++mm,belong[G[i].from],belong[G[i].to]);

}

}

void build\_map()

{

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

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

{

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

addedge(i,x,y);

}

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

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

memset(instack,0,sizeof(instack));

indexs=0;nn=0;

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

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

{

if (dfn[i]==-1)

tarjan(i);

}

build\_new\_map();

}

### 3.最大流-Dinic(模拟栈)

Edge为前向星的边数，所以需要初始化Edge和head数组，其中head数组应初始化为-1

int dinic(int n,int s,int t);  
n表示有n个点，这个版无所谓点从0开始还是从1开始，s表示源点，t表示汇点  
很好的一个是，这个版的DFS使用的是模拟栈，防止爆栈

int i,j,k,n,m,x,y,T,num,w;

const int inf = 0x3f3f3f3f;

struct edgenode

{

int from,to,next;

int cap;

}edge[MAXM];

int Edge,head[MAXN],ps[MAXN],dep[MAXN];

void addedge(int x,int y,int c)

{

edge[Edge].from=x;

edge[Edge].to=y;

edge[Edge].cap=c;

edge[Edge].next=head[x];

head[x]=Edge++;

edge[Edge].from=y;

edge[Edge].to=x;

edge[Edge].cap=0;

edge[Edge].next=head[y];

head[y]=Edge++;

}

int dinic(int n,int s,int t)

{

int tr,flow=0;

int i,j,k,l,r,top;

while(1){

memset(dep,-1,(n+1)\*sizeof(int));

for(l=dep[ps[0]=s]=0,r=1;l!=r;)//BFS部分，将给定图分层

{

for(i=ps[l++],j=head[i];j!=-1;j=edge[j].next)

{

if (edge[j].cap&&-1==dep[k=edge[j].to])

{

dep[k]=dep[i]+1;ps[r++]=k;

if(k==t)

{

l=r;

break;

}

}

}

}

if(dep[t]==-1)break;

for(i=s,top=0;;)//DFS部分

{

if(i==t)//当前点就是汇点时

{

for(k=0,tr=inf;k<top;++k)

if(edge[ps[k]].cap<tr)tr=edge[ps[l=k]].cap;

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

edge[ps[k]].cap-=tr,edge[ps[k]^1].cap+=tr;

flow+=tr;

i=edge[ps[top=l]].from;

}

for(j=head[i];j!=-1;j=edge[j].next)//找当前点所指向的点

if(edge[j].cap&&dep[i]+1==dep[edge[j].to]) break;

if(j!=-1)

{

ps[top++]=j;//当前点有所指向的点，把这个点加入栈中

i=edge[j].to;

}

else

{

if (!top) break;//当前点没有指向的点，回溯

dep[i]=-1;

i=edge[ps[--top]].from;

}

}

}

return flow;

}

### 4.最小费用流

/\*size表示网络中的结点数，编号从0开始，如果是从1开始则size=n+1\*/

/\*初始化：head全设为-1，Edge为边数预先设为0\*/

int head[MAXN],vis[MAXN],dis[MAXN],pos[MAXN],Edge,size;

char s[305][305];

struct edgenode

{

int to,next,w,cost;

} edge[MAXM];

void add\_edge(int x,int y,int w,int cost)

{

edge[Edge].to=y;

edge[Edge].w=w;

edge[Edge].cost=cost;

edge[Edge].next=head[x];

head[x]=Edge;

Edge++;

edge[Edge].to=x;

edge[Edge].w=0;

edge[Edge].cost=-cost;

edge[Edge].next=head[y];

head[y]=Edge;

Edge++;

}

bool SPFA(int s, int t)

{

int u,v,i;

queue <int> q;

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

for(i=0;i<size;i++) dis[i]=INF;

dis[s]=0;

vis[s]=1;

q.push(s);

while(!q.empty())

{

u=q.front(); q.pop(); vis[u]=0;

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

{

v=edge[i].to;

if(edge[i].w>0&&dis[u]+edge[i].cost<dis[v])

{

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

pos[v]=i;

if(!vis[v])

{

vis[v]=1;

q.push(v);

}

}

}

}

return dis[t]!=INF;

}

int MinCostFlow(int s,int t)//源点与汇点

{

int i,cost=0,flow=0;

while(SPFA(s,t))

{

int d=INF;

for (i=t;i!=s;i=edge[pos[i]^1].to)

{

d=min(d,edge[pos[i]].w);

}

for(i=t;i!=s;i=edge[pos[i]^1].to)

{

edge[pos[i]].w-=d;

edge[pos[i]^1].w+=d;

}

flow+=d;

cost+=dis[t]\*d;

}

return cost; // flow是最大流值

}

### 5.二分图匹配

邻接表法：

class edge{

public:

int v,nex;

};edge e[100005];

int n,m,k,head[100005];

int map[1501][1501],link[1501];

bool vis[1501];

void addedge(int b,int a){//向图中加边的算法，注意加上的是有向边//b为a的后续节点既是a---->b

e[k].v=a;

e[k].nex=head[b];

head[b]=k;k++;

}

bool dfs(int u){

for(int i = head[u]; i != 0; i = e[i].nex){

int v = e[i].v;

if(!vis[v]){

vis[v] = true;

if(link[v] == -1 || dfs(link[v])){

link[v] = u;

return true;

}

}

}

return false;

}

int main(){

…………

int ans=0;

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

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

if(dfs(i)) ans++;

}

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

…………

return 0;

}

邻接矩阵：

int map[1000][1000],n,ans,link[1000];

bool vis[1000];

int dfs(int s){

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

if(!vis[i]&&map[s][i]){

vis[i]=1;

if(link[i]==-1||dfs(link[i])){

link[i]=s;

return 1;

}

}

}

return 0;

}

int main(){

………………

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

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

if(dfs(i))ans++;

}

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

………………

}

return 0;

}

## 【其他】

### Set:

(1)元素不是结构体：

例：

//自定义比较函数myComp,重载“（）”操作符

struct myComp

{

bool operator()(const your\_type &a,const your\_type &b)

{

return a.data-b.data>0;

}

}

set<int,myComp>s;

......

set<int,myComp>::iterator it;

(2)如果元素是结构体，可以直接将比较函数写在结构体内。

例：

struct Info

{

string name;

float score;

//重载“<”操作符，自定义排序规则

bool operator < (const Info &a) const

{

//按score从大到小排列

return a.score<score;

}

}

set<Info> s;

......

set<Info>::iterator it;

### 最长回文子串(Manacher)

#include<vector>

#include<iostream>

using namespace std;

const int N=300010;

int n, p[N];

char s[N], str[N];

#define \_min(x, y) ((x)<(y)?(x):(y))

void kp()

{

int i;

int mx = 0;

int id;

for(i=n; str[i]!=0; i++)//清除n后边多余的部分

str[i] = 0; //没有这一句有问题。。就过不了ural1297，比如数据：ababa aba

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

{

if( mx > i )

p[i] = \_min( p[2\*id-i], p[id]+id-i );

//因为是从左往右扫描的这里i>id, 2\*id-i是i关于id的对称点，该对称点在id的左端

//p[id]+id是描述中的mx，即id向右延伸的端点位置

//显然向右延伸是可能超出mx的，所以要有下边的for循环

else

p[i] = 1;

for(; str[i+p[i]] == str[i-p[i]]; p[i]++);

if( p[i] + i > mx )//更新mx与id，因为mx是向右延伸的最大长度，所以实时更新

{

mx = p[i] + i;

id = i;

}

}

}

void init()//处理字符串

{

int i, j, k;

str[0] = '$';

str[1] = '#';

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

{

str[i\*2+2] = s[i];

str[i\*2+3] = '#';

}

n = n\*2+2;

s[n] = 0;

}

int main()

{

int i, ans;

while(scanf("%s", s)!=EOF)

{

n = strlen(s);//n为给定字符串s的长度

init();

kp();

ans = 0;

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

if(p[i]>ans)//寻找最大的长度ans

ans = p[i];

printf("%d\n", ans-1);

}

return 0;

}

### 位运算

枚举子集的方法

for (j=x;j;j=(j-1)&x)

其中x为原集合

取反操作 (~i)&((1<<n)-1)

其中n为i的二进制位数，注意由于优先级的关系1<<n必须加括号

取第k位x&(1<<k)

将第i位赋值为1 s|1<<i

将第i位赋值为0 s&~(1<<i)

### 旋转卡壳

int diameter2(vector<Point>& points)

{

vector<Point> p = ConvexHull(points);

int n = p.size();

//for(int i = 0; i < n; ++i) printf("%d %d\n", p[i].x, p[i].y);

if(n == 1) return 0;

if(n == 2) return Dist2(p[0], p[1]);

p.push\_back(p[0]);

int ans = 0;

for(int u = 0, v = 1; u < n; ++u)

{// 一条直线贴住边p[u]-p[u+1]

while(true)

{

// 当Area(p[u], p[u+1], p[v+1]) <= Area(p[u], p[u+1], p[v])时停止旋转

//因为两个三角形有一公共边，所以面积大的那个点到直线距离大

// 即Cross(p[u+1]-p[u], p[v+1]-p[u]) - Cross(p[u+1]-p[u], p[v]-p[u]) <= 0

// 根据Cross(A,B) - Cross(A,C) = Cross(A,B-C)

// 化简得Cross(p[u+1]-p[u], p[v+1]-p[v]) <= 0

int diff = Cross(p[u+1]-p[u], p[v+1]-p[v]);

if(diff <= 0)

{

ans = max(ans, Dist2(p[u], p[v]));

if(diff == 0) ans = max(ans, Dist2(p[u], p[v+1]));

break;

}

v = (v+1)%n;

}

}

return ans;

}