ACM模板Beta0.55

StandNotAlone

February 4th，2021

一、来一场ACM吧
1.头文件————————————————————P4
2.整数读入挂———————————————————P5
3.实数gcd——————————————————P6
4.支持O(1)翻转的双端队列类——————————P6
二、数论
1.素数线性筛法——————————————————P7
2.欧拉函数———————————————————P8
3.矩阵快速幂——————————————————P8
4.扩展欧几里得(exgcd)————————————P9
5.中国剩余定理——————————————————P10
6.高次同余方程——————————————————P11
7.高斯消元———————————————————P12
8.卢卡斯定理——————————————————P13
9.高精度求组合数—————————————————P14
10.卡特兰数———————————————————-P15
11.SG函数————————————————————P16
12.“龟速”快速幂——————————————————P17
三、数据结构
1.并查集————————————————————P18
2.树状数组———————————————————P19
3.线段树————————————————————P21
四、图论
1.链式前向星——————————————————P22
2.堆优化Dijkstra——————————————————P23
3.二分图匹配——————————————————P24
五、计算几何
1.基本结构体——————————————————P27
2.凸包—————————————————————P29
六、字符串
1.KMP——————————————————————P29
2.Trie———————————————————————P30

七、其他

1.离散化————————————————————P31
2.实数三分———————————————————P31
3.整数三分———————————————————P31
4.高精度————————————————————P32

一、来一场ACM吧

1.1头文件

```
#include<map>
#include<set>
#include<cmath>
#include<queue>
#include<stack>
#include<cstdio>
#include<vector>
#include<bitset>
#include<string>
#include<cstdlib>
#include<cstring>
#include<fstream>
#include<sstream>
#include<iomanip>
#include<iostream>
#include<algorithm>
#include<functional>
#include<unordered_map>
#include<unordered_set>
//#include<bits/stdc++.h>
#define INF 0x7f7f7f7f //2139062143
#define INF1 0x3f3f3f3f //1061109567
#define INF2 2147483647
#define llINF 9223372036854775807
#define pi 3.141592653589793//23846264338327950254
#define ft first
#define sd second
#define endl "\n"
#define mp make_pair
#define pb push_back
#define ll long long
#define int long long
#define vec vector<ll>
#define mat vector<vector<ll>>
#define rep(i,n) for(ll i=0;i<(ll)(n);i++)
#define _rep(i,n) for(ll i=n-1;i>=0;i--)
#define REP(i,n) for(ll i=1;i<=(ll)(n);i++)
```

```cpp
#define _REP(i,n) for(ll i=n;i>0;i--)
#define at(x,n) for(auto &x:n)
//cout<<fixed<<setprecision(6)<<
//freopen(".in","r",stdin);
//freopen(".out","w",stdout);
//ifstream f1("/Users/wangzichao/Documents/wzc.in");
//ofstream f2("/Users/wangzichao/Documents/wzc.out");
#define IOS ios::sync_with_stdio(0); cin.tie(0);
cout.tie(0);
using namespace std;
typedef pair<ll,ll> PLL;
#define local
#ifdef local
#endif
const ll maxn=1e3+7;
const double eps=1e-10;
const ll mod=1e9+7;

int32_t main()
{
    IOS;
}
```

1.2整数读入挂

```cpp
ll read()
{
    ll x=0,f=1;
    char c=getchar();
    while(c<'0'||c>'9')
    {
        if(c=='-') f=-1;
        c=getchar();
    }
    while(c>='0'&&c<='9')
    {
        x=x*10+c-'0';
        c=getchar();
    }
    return x*f;
}
```

1.3实数gcd

```
double gcd(double x,double y)
{
    while(fabs(x)>eps&&fabs(y)>eps)
    {
        if(x>y) x-=floor(x/y)*y;
        else y-=floor(y/x)*x;
    }
    return x+y;
}
```

1.4支持O(1)翻转的双端队列类

```
template<class T>
struct Deque//支持O(1)翻转操作的deque
{
    T data[maxn*2];
    int head=maxn,tail=maxn-1;
    bool f=0;//翻转次数奇偶性标记
    bool empty(){return tail<head;}
    int size(){return tail-head+1;}
    T front(){return f?data[tail]:data[head];}
    T back(){return f?data[head]:data[tail];}
    void push_front(T x){f?data[++tail]=x:data[--head]=x;}
    void push_back(T x){f?data[--head]=x:data[++tail]=x;}
    void pop_back(){f?head++:tail--;}
    void pop_front(){f?tail--:head++;}
    void reverse(){f^=1;}//翻转
};
```

二、数论

2.1素数线性筛法O(n)

```
vector<int>prime;
bool v[maxn];
void primes(int n)
{
    for(int i=2;i<=n;i++)
    {
        if(!v[i]) prime.push_back(i);//v[i]为0代表i为质数
        for(int j=0;prime[j]<=n/i;j++)
        {
            //如果当前找寻的质数大于i的最小质因数或者相乘后超出n的范围则停止
            v[prime[j]*i]=1;
            if(i%prime[j]==0) break;
        }
    }
}
```

//素数Eratosthenes筛法O(nloglogn)

```
void primes(ll n)
{
    memset(v,0,sizeof(v));
    for(ll i=2;i<=n;i++)
    {
        if(v[i]) continue;
        cout<<i<<endl;
        for(int j=i;j<=n/i;j++) v[i*j]=1;
    }
}
```

2.2欧拉函数（利用线性筛在O（n）时间算出2-n的欧拉函数）

```
int prime[maxn],phi[maxn],tot=0;
bool v[maxn];
void euler(int n)
{
    phi[1]=1;
    for(int i=2;i<=n;i++)
    {
        if(!v[i])
        {
            prime[tot++]=i;
            phi[i]=i-1;
        }
        for(int j=0;prime[j]<=n/i;j++)
        {
            v[i*prime[j]]=1;
            phi[i*prime[j]]=phi[i]*(i%prime[j]?
prime[j]-1:prime[j]);
            if(i%prime[j]==0) break;
        }
    }
}
```

2.3矩阵快速幂

```
//矩阵快速幂，以对一个 3 * 3 的矩阵求快速幂为例
struct Mat
{
    ll a[3][3];
    void init()
    {
        memset(a,0,sizeof(a));
    }
};
```

```
Mat operator * (Mat a, Mat b)
{
    Mat ans; ans.init();
    for(int i = 0 ; i < 3; i ++)
        for(int j = 0; j < 3; j ++)
            for(int k = 0 ; k < 3; k ++)
                ans.a[i][j] = (ans.a[i][j] + a.a[i]
[k] * b.a[k][j]) % mod;
    return ans;
}
```

2.4扩展欧几里得（exgcd)

```
ll exgcd(ll a,ll b,ll &x,ll &y)
{
    if(!a&&!b) return -1;
    if(!b)
    {
        x=1;y=0;
        return a;
    }
    ll ret=exgcd(b,a%b,y,x);
    y-=a/b*x;
    return ret;
}
```

得出一组特解x0,y0后，可以得到通解。d=gcd(a,b)

通解为

x=x0-b/d*k

y=y0+a/d*k

线性同余方程xa+yb=c有解当且仅当c是d的倍数。对应的求出exgcd的一个特解后，乘以c除以d即为线性同余方程的一个特解。通解的形式与上面相同。

//一个同余的公式，当a和p互质时，a^x%p=1的最小解，必然是p的欧拉函数的约数（算竞P150）

2.5中国剩余定理

```
//求n个方程x%M[i]=A[i]，满足条件的最小正整数解x
//1.如果所有的M[i]之间互质
typedef pair<ll,ll> PLL;
ll china()
{
	ll ret=0,m=1;
	ll x,y;
	for(int i=1;i<=n;i++) m*=M[i];
	for(int i=1;i<=n;i++)
	{
		ll w=m/M[i];
		exgcd(w,M[i],x,y);
		ret=(ret+w*A[i]*x)%m;
	}
	return (ret+m)%m;
}

//2.M[i]之间不全部互质时
PLL exchina()
//求n个方程x%M[i]=A[i],满足条件的最小正整数解x
{
    ll ret=0,m=1;
    ll x,y;
    for(int i=1;i<=n;i++)
    {
        ll c=A[i]-ret,d=exgcd(m,M[i],x,y);
        if(c%d) return PLL(0,-1);//用lcm值=-1标记无答案
        ll t=x*c/d;
        ret=ret+t%M[i]*m;
        m*=M[i]/d;
    }
    ret=(ret%m+m)%m;
    return PLL(ret,m);//ret为答案，m是最后的lcm
}


PLL exchina()
```

```
//求n个方程A[i]x%M[i]=B[i],满足条件的最小正整数解x
{
    ll ret=0,m=1;
    ll x,y;
    for(int i=1;i<=n;i++)
    {
        ll a=A[i]*m,c=B[i]-A[i]*ret,d=gcd(M[i],a);
        if(c%d) return PLL(0,-1);//用lcm值=-1标记无答案
        exgcd(a/d,M[i]/d,x,y);
        ll t=x*c/d%(M[i]/d);
        ret=ret+t%M[i]*m;
        m*=M[i]/d;
    }
    ret=(ret%m+m)%m;
    return PLL(ret,m);//ret为答案，m是最后的lcm
}
```

2.6高次同余方程O（sqrt(mod))

BSGS算法

```
ll baby_step_giant_step(ll a,ll b,ll c)//求解高次同余方程c*a^x%p=b（a与p互质）
{
    unordered_map<ll,ll>hash;
    b%=mod;
    ll t=sqrt(mod)+1;
    for(ll j=0;j<t;j++)
    {
        ll val=b*qpow(a,j)%mod;
        hash[val]=j;
    }
    a=qpow(a,t);
    if(a==0) return b==0?1:-1;
    for(ll i=0;i<=t;i++)
    {
        ll val=qpow(a,i)*c%mod;
        ll j=hash.find(val)==hash.end()?-1:hash[val];
        if(j>=0&&i*t-j>0) return i*t-j;
    }
    return -1;
}
```

2.7高斯消元

```cpp
const double eps=1e-5;
const int maxn=1e2+7;
int n;
double a[maxn][maxn];
int gauss()//a[n+1][n+1]的增广矩阵，方程数量小于n的时候直接无解
//方程数量等于n时最后一行全部补0即可
{
    int r,c;
    for(r=c=0;c<n;c++)
    {
        int tar=r;
        for(int i=r+1;i<=n;i++)//找绝对值最大的行并替换到当前第一行
            if(fabs(a[i][c])>fabs(a[tar][c])) tar=i;
        if(fabs(a[tar][c])<eps) continue;
        for(int i=c;i<=n;i++) swap(a[tar][i],a[r][i]);
        for(int i=n;i>=c;i--) a[r][i]/=a[r][c];
        for(int i=r+1;i<=n;i++)//利用当前第一行将当前第一列下方全部消为0
            if(fabs(a[i][c])>eps)
                for(int j=n;j>=c;j--)
                    a[i][j]-=a[r][j]*a[i][c];
        r++;
    }
    if(r<n)
    {
        for(int i=r;i<=n;i++)
            if(fabs(a[i][n])>eps) return 2;//无解
        return 1;//有无穷组解
    }
    for(int i=n;i>=0;i--)
        for(int j=i+1;j<=n;j++)
            a[i][n]-=a[i][j]*a[j][n];
    return 0;//有唯一解
}
```

```
int32_t main()
{
    IOS;
    memset(a,0,sizeof(a));
    cin>>n;
    for(int i=0;i<n;i++)
        for(int j=0;j<=n;j++)
            cin>>a[i][j];
    int ans=gauss();
    if(ans==1) cout<<"Infinite group 
solutions"<<endl;
    else if(ans==2) cout<<"No solution"<<endl;
    else for(int i=0;i<n;i++) cout<<a[i][n]<<endl;
}
```

2.8卢卡斯定理

```
//计算C(a,b)%p，a>=b，其中a和b很大，p很小，且p为质数。
//卢卡斯定理:C(a,b)%p=C(a/p,b/p)*C(a%p,b%p)%P
ll jiechen[maxn];

ll zuhe(ll a,ll b,ll p)
{
    if(a<b) return 0;
    return jiechen[a]*qpow(jiechen[b]*jiechen[a-b]
%p,p-2,p)%p;
}

ll lucas(ll a,ll b,ll p)
{
    return b?lucas(a/p,b/p,p)*zuhe(a%p,b%p,p)%p:1;
}

int32_t main()
{
    IOS;
    jiechen[0]=jiechen[1]=1;
    int n;
    cin>>n;
```

```
    while(n--)
    {
        ll a,b,p;
        cin>>a>>b>>p;
        for(int i=2;i<p;i++)
jiechen[i]=jiechen[i-1]*i%p;
        cout<<lucas(a,b,p)<<endl;
    }
}
```

2.9高精度求取组合数

```
//当需要高精度求取组合数的时候，可以采取质因数分解的方式求
//get函数为求取n!中质因数p出现的次数
void mul_big(vector<int>&a,int b)
{
    int rest=0;
    for(int i=0;i<a.size();i++)
    {
        rest=rest+b*a[i];
        a[i]=rest%10;
        rest/=10;
    }
    while(rest)
    {
        a.push_back(rest%10);
        rest/=10;
    }
}
int get(int n,int p)
{
    int ret=0;
    while(n)
    {
        ret+=n/p;
        n/=p;
    }
    return ret;
}
```

```
int32_t main()
{
    IOS;
    primes(5000);
    int a,b;
    cin>>a>>b;
    ans.push_back(1);
    for(int i=0;i<prime.size();i++)
    {
        int num=get(a,prime[i])-get(b,prime[i])-
get(a-b,prime[i]);
        for(int j=0;j<num;j++)
            mul_big(ans,prime[i]);
    }
    for(int i=(int)ans.size()-1;i>=0;i--)
cout<<ans[i];
    cout<<endl;
}
```

2.10卡特兰数

f(n)=C(2n,n)/(n+1)=C(2n,n)-C(2n,n-1)

通过证明不符合情况可以与另一种容易计算的情况（即一个互相的一一映射）得到该公式

//常见的卡特兰数

1.n个0和n个1构成的长度2n的序列，任意的前缀子序列中0的个数不少于1的数列数量

2.n个左括号和n个右括号组成的合法序列数量

3.n个结点构成的不同二叉树数量

//增加n+1个叶子结点变为2n+1个结点的满二叉树（一一对应）

//通过先序遍历的过程类比出卡特兰数

4.在平面坐标系上，每一步只能向上或向右走，从（0，0）走到（n，n）且过程中不能在直线y=x上方，路径的数量

//对应的每种不符合的情况，根据它最早的出现在直线y=x上方的点沿着y=x方向翻折

//可以转化成从(0,0)到（n-1,n+1）的路径

卡特兰数的前30项:

[1,1,2,5,14,42,132,429,1430,4862,16796,58786,

```
208012,742900,2674440,9694845,35357670,129644790,
477638700,1767263190,6564120420,24466267020,
91482563640,343059613650,1289904147324,
4861946401452,18367353072152,69533550916004,
263747951750360,1002242216651368,3814986502092304]
```

2.11 SG函数

必败态：当前状态不管怎么选择，只要对手采取最优策略就败

必胜态：当前状态只要你采取最优策略就胜

1. 当某个局面导向的所有状态都是必胜态时候，当前状态必败
2. 当某个局面导向有一个状态是必败的时候，当前状态必胜
3. SG(x) = mex{SG(y1),SG(y2),SG(y3)...SG(yk)} 其中 y1..yk 是 x 导向的所有状态

当 SG(x) > 0 时说明当前状态必胜，SG(x) < 0 时说明当前状态必败

4. 多个有向图游戏的和 SG 函数值等于他包含的各个子游戏的异或和

SG(G) = SG(G1) ^ SG(G2) ^ SG(G3)..... ^ SG(Gm)

SG 可以通过递推或者记忆化搜索等方式求出

```
//模板:共n堆石子，每个人只能抓取位于num数组集合里的数，轮流抓取，不能抓的人输
int n,k;
int num[107];
int sg[maxn];
int getsg(int x)
{
    if(sg[x]!=-1) return sg[x];
    unordered_set<int>S;//用于计算当前状态对应的mex值
    for(int i=0;i<k;i++)
        if(x>=num[i]) S.insert(getsg(x-num[i]));//递归计算
    for(int i=0;;i++)
        if(!S.count(i))
            return sg[x]=i;
}
```

```cpp
int32_t main()
{
    IOS;
    memset(sg,-1,sizeof(sg));
    cin>>k;
    for(int i=0;i<k;i++) cin>>num[i];
    int ans=0;
    cin>>n;
    for(int i=0;i<n;i++)
    {
        int x;
        cin>>x;
        ans^=getsg(x);
    }
    if(ans) cout<<"Yes"<<endl;
    else cout<<"No"<<endl;
}
```

2.12“龟速”快速幂

```cpp
//当数据范围较大时mod值超过3e9时，快速幂的乘法运算会爆longlong范围。此时我们将快速幂的乘法操作用lpow运算替代，使用快速加法来替代运算
//注意这里qmul的p只能为正数，如果是负数要加个判定再最后取负
ll lmul(ll a,ll p,ll mod)
{
    ll ret=0;
    while(p)
    {
        if(p&1) ret=(ret+a)%mod;
        a=(a+a)%mod;
        p>>=1;
    }
    return ret;
}
```

```
ll qpow(ll a,ll p,ll mod)
{
    ll ret=1;
    while(p)
    {
        if(p&1) ret=lmul(ret,a,mod);
        a=lmul(a,a,mod);
        p>>=1;
    }
    return ret;
}
```

三、数据结构

3.1并查集

```
vector<int>fa;//并查集数组

void init()//并查集数组初始化
{
    for(int i=0;i<fa.size();i++) fa[i]=i;
}

int get(int x)//访问x所在的根节点(代表元素)
{
    return x==fa[x]?x:fa[x]=get(fa[x]);//路径压缩，让当前位置指示的父亲直接为根节点（代表元素）
}

void merge(int x,int y)//合并x和y所在的集合，即让x的根节点作为y的树根的子节点
{
    fa[get(x)]=get(y);
}
```

//边带权版本并查集

```
vector<int>fa;
vector<ll>dis;//dis[i]记录i距离自己所在集合根节点的之间的距
```

```
void init(int n)
{
    fa.resize(n+7);
    dis.resize(n+7,0);
    for(int i=0;i<n;i++)
        fa[i]=i;
}

int get(int x)
{
    if(x==fa[x]) return x;
    int root=get(fa[x]);
    dis[x]+=dis[fa[x]];
    return fa[x]=root;
}

void merge(int x,int y,ll len)
{
    int rootx=get(x),rooty=get(y);
    fa[rootx]=rooty;
    dis[rootx]=dis[y]-dis[x]+len;    //注意推导这两个结论怎么来的，我们以下标从小的到大的为正方向，可以讨论发现，当rootx>rooty的时候，得到的距离恰好是反向也就是取相反数的值
}                                    //在重新参与计算的时候仍然是满足该等式的，此处不要将距离看成一个标量，看成一个有方向的向量
```

3.2树状数组

```
int tree[maxn];
int sp=0;

void add(int x,int v)//单点修改，增加v个x进入数组
{
    for(;x<=n;x+=x&-x) tree[x]+=v;
}
```

```
int sum(int x)    //查询下标小于等于x的元素的前缀和
{
    int ret=0;
    for(;x>0;x-=x&-x) ret+=tree[x];
    return ret;
}


void getsp()
{
    int temp=n;
    while(temp)
    {
        temp>>=1;
        sp++;
    }
}

int kth(int k)        //查询数组中从小到大第K小的数字
{
    int ret=0;
    int sum=0;
    for(int i=sp-1;i>=0;i--)     //sp为N的二进制最高位
    {
        ret+=(1<<i);
        if(ret>=n||sum+tree[ret]>=k) ret-=(1<<i);
        else sum+=tree[ret];
    }
    return ret+1;
}
```

3.3线段树

```
int A[maxn];
struct Node
{
    int l,r,ope;
    ll data;
};
Node st[maxn<<2];
void build(int l,int r,int loca)
{
    st[loca].l=l;st[loca].r=r;st[loca].ope=0;
    if(l==r) {st[loca].data=1;return;}
    int mid=(l+r)>>1;
    build(l,mid,loca<<1);
    build(mid+1,r,loca<<1|1);
    st[loca].data=l-r+1;
}
void spread(int loca)
{
    if(st[loca].ope)
    {
        st[loca<<1].data=st[loca].ope*(st[loca<<1].r-
st[loca<<1].l+1);
        st[loca<<1|1].data=st[loca].ope*(st[loca<<1|
1].r-st[loca<<1|1].l+1);
        st[loca<<1].ope=st[loca].ope;
        st[loca<<1|1].ope=st[loca].ope;
        st[loca].ope=0;
    }
}

void change(int l,int r,int loca,int ope)
{
    if(st[loca].l>=l&&st[loca].r<=r)
    {
        st[loca].data=ope*(st[loca].r-st[loca].l+1);
        st[loca].ope=ope;
        return;
    }
    spread(loca);
```

```cpp
    int mid=(st[loca].r+st[loca].l)>>1;
    if(l<=mid) change(l,r,loca<<1,ope);
    if(r>mid) change(l,r,loca<<1|1,ope);
    st[loca].data=st[loca<<1].data+st[loca<<1|1].data;
}

ll ask(int l,int r,int loca)
{
    if(st[loca].l>=l&&st[loca].r<=r) return st[loca].data;
    spread(loca);
    int mid=(st[loca].l+st[loca].r)>>1;
    ll temp=0;
    if(l<=mid) temp+=ask(l,r,loca<<1);
    if(r>mid) temp+=ask(l,r,loca<<1|1);
    return temp;
}
```

四、图论

4.1链式前向星

```cpp
struct Edge
{
    int to,next;ll dis;
}edge[maxn];
int head[maxn],tot;

void init()
{
    for(int i=1;i<=n;i++) head[i]=-1;
    tot=0;
}
void add(int u,int v,ll w)
{
    edge[tot].to=v;
    edge[tot].next=head[u];
    edge[tot].dis=w;
    head[u]=tot++;
}
```

4.2堆优化Dijkstra

```
ll dis[maxn];
struct Node
{
    int pos;ll val;
    Node(int pos,ll val):pos(pos),val(val){}
    friend bool operator < (Node a,Node b)
    {
        return a.val>b.val;
    }
};
ll Dijkstra(int start,int target)
{
    for(int i=1;i<=n;i++) dis[i]=llINF;
    dis[start]=0;
    priority_queue<Node>Q;
    Q.push(Node(start,0));
    while(Q.size())
    {
        Node now=Q.top();
        Q.pop();
        if(now.val>dis[now.pos]) continue;
        for(int i=head[now.pos];i!=-1;i=edge[i].next)
        {
            int to=edge[i].to;
            if(dis[to]>edge[i].dis+now.val)
            {
                dis[to]=edge[i].dis+now.val;
                Q.push(Node(to,dis[to]));
            }
        }
    }
    return dis[target];
}
```

4.3二分图匹配

匈牙利算法O（NM）左边 N 个点匹配右边 M 个点，总共 K 条边的最大匹配数

```
int N,M,K,love[maxn],vis[maxn];
bool ntr(int u)
{
    for(int i=head[u];i!=-1;i=edge[i].next)
    {
        int v=edge[i].to;
        if(!vis[v])
        {
            vis[v]=1;
            if(love[v]==-1||ntr(love[v]))
            {
                love[v]=u;
                return 1;
            }
        }
    }
    return 0;
}

int startntr()
{
    int ans=0;
    for(int i=1;i<=M;i++) linker[i]=-1; //注意是右边点的 linker
    for(int i=1;i<=N;i++)
    {
        for(int j=1;j<=M;j++) vis[j]=0;
        if(ntr(i)) ans++;
    }
    return ans;
}

int main()
{
    cin>>N>>M>>K;
    init();
```

```
    for(int i=1;i<=K;i++)
    {
        int u,v;
        scanf("%d%d",&u,&v);
        if(u>N||v>M) continue;
        add(u,v);
    }
    Pri(startntr());
    return 0;
}
```

HK算法O（sqrt(n)*m）

```
int boylove[maxn],girllove[maxn];
int boydeep[maxn],girldeep[maxn];
int vis[maxn],dis;
bool bfs()  //对匈牙利算法优化，检查当前状态下是否仍然存在可被ntr的匹配
{            //并利用bfs计算和标记最短的那一条增广路
    queue<int>Q;dis=INF;
    memset(boydeep,-1,sizeof(boydeep));
    memset(girldeep,-1,sizeof(girldeep));
    for(int i=1;i<=n;i++)
    {
        if(boylove[i]==-1)
        {
            Q.push(i);
            boydeep[i]=0;
        }
    }
    while(Q.size())
    {
        int now=Q.front();Q.pop();
        if(boydeep[now]>dis) break;
        for(int i=head[now];i!=-1;i=edge[i].next)
        {
            int to=edge[i].to;
            if(girldeep[to]==-1)
            {
                girldeep[to]=boydeep[now]+1;
```

```
                if(girllove[to]==-1)
dis=girldeep[to];
                else
                {

boydeep[girllove[to]]=girldeep[to]+1;
                    Q.push(girllove[to]);
                }
            }
        }
    }
    return dis!=INF;
}

bool ntr(int now)
{
    for(int i=head[now];i!=-1;i=edge[i].next)
    {
        int to=edge[i].to;
        if(!vis[to]&&girldeep[to]==boydeep[now]+1)
        {
            vis[to]=1;
            if(girllove[to]!=-1&&dis==girldeep[to])
continue;
            if(girllove[to]==-1||ntr(girllove[to]))
            {
                girllove[to]=now;
                boylove[now]=to;
                return 1;
            }
        }
    }
    return 0;
}

int startntr()
{
    int ret=0;
    memset(boylove,-1,sizeof(boylove));
    memset(girllove,-1,sizeof(girllove));
```

```
    while(bfs())
    {
        memset(vis,0,sizeof(vis));
        for(int i=1;i<=n;i++)
            if(boylove[i]==-1&&ntr(i)) ret++;
    }
    return ret;
}
```

五、计算几何

5.1基本结构体

```
double add(double a,double b)
{
    if(fabs(a+b)<eps*(fabs(a)+fabs(b))) return 0;
    return a+b;
}

struct Point
{
    double x,y;
    Point(){}
    Point(double x,double y):x(x),y(y){}
    Point operator + (Point P)
    {
        return Point(add(x,P.x),add(y,P.y));
    }
    Point operator - (Point P)
    {
        return Point(add(x,-P.x),add(y,-P.y));
    }
    Point operator * (double d)
    {
        return Point(x*d,y*d);
    }
    Point operator / (double d)
    {
        return Point(x/d,y/d);
    }
```

```
    bool operator == (Point P)
    {
        return fabs(x-P.x)<eps&&fabs(y-P.y)<eps;
    }
    double dist(Point P)
    {
        return sqrt((x-P.x)*(x-P.x)+(y-P.y)*(y-P.y));
    }
    double dot(Point P)//内积
    {
        return add(x*P.x,y*P.y);
    }
    double det(Point P)//外积
    {
        return add(x*P.y,-y*P.x);
    }
};

bool on_seg(Point P1,Point P2,Point P3)//判断点P3是否在
线段P1-P2上
{
    return (P1-P3).det(P2-P3)==0&&(P1-P3).dot(P2-
P3)<=0;
}
//计算直线P1-P2与直线Q1-Q2的交点位置，注意这必须建立在两直线
不平行的条件下
Point intersection(Point P1,Point P2,Point Q1,Point
Q2)
{
    return P1+(P2-P1)*((Q2-Q1).det(Q1-P1)/(Q2-
Q1).det(P2-P1));
}
//判断线段P1-P2与线段Q1-Q2是否规范相交
bool SegmentProperIntersection(Point P1,Point
P2,Point Q1,Point Q2)
{
    return (P2-P1).det(Q1-P1)*(P2-P1).det(Q2-
P1)<0&&(Q2-Q1).det(P1-Q1)*(Q2-Q1).det(P2-Q1)<0;
}
```

5.2凸包

```
vector<Point>point,tubao;
int n;
bool cmp(Point a,Point b)
{
    if(a.x==b.x) return a.y<b.y;
    else return a.x<b.x;
}

sort(point.begin(),point.end(),cmp);
for(int i=0;i<n;i++)
{
    while(tubao.size()>1&&(tubao[tubao.size()-1]-
tubao[tubao.size()-2]).det(point[i]-
tubao[tubao.size()-1])<=0) tubao.pop_back();
    tubao.push_back(point[i]);
}
int temp=(int)tubao.size();
for(int i=n-2;i>=0;i--)
{
    while(tubao.size()>temp&&(tubao[tubao.size()-1]-
tubao[tubao.size()-2]).det(point[i]-
tubao[tubao.size()-1])<=0) tubao.pop_back();
    tubao.push_back(point[i]);
}
tubao.pop_back();
```

六、字符串

6.1KMP

```
//这里下标是从0开始的
int net[maxn];
int cas[maxn];
void getnet(string s)
{
    int len=s.size();
    net[0]=-1;
    for(int i=1,j=-1;i<len;i++)
    {
```

```
        while(j>-1&&s[i]!=s[j+1]) j=net[j];
        if(s[i]==s[j+1]) j++;
        net[i]=j;
    }
}
void KMP(string a,string b)//计算字符串b[1-i]的部分的后缀与字符串a的前缀最大匹配长度
{
    getnet(b);
    int lena=a.size(),lenb=b.size();
    for(int i=0,j=-1;i<lena;i++)
    {
        while(j>-1&&(j==lenb-1||a[i]!=b[j+1]))
j=net[j];
        if(a[i]==b[j+1]) j++;
        cas[i]=j;
    }
}
```

6.2Trie (字典树)

```
//可以改用结构体指针来实现，用时间来换取空间
int trie[maxn][26],ed[maxn],tot=0;
void insert(string s)
{
    int len=s.size(),tar=0;
    for(int i=0;i<len;i++)
    {
        int net=s[i]-'a';
        if(trie[tar][net]==0) trie[tar][net]=++tot;
        tar=trie[tar][net];
    }
    ed[tar]++;
}
int search(string s)
{
    int len=s.size(),tar=0;
    for(int i=0;i<len;i++)
    {
        tar=trie[tar][s[i]-'a'];
```

```
        if(tar==0) return 0;
    }
    return ed[tar];
}
```

七、其他

7.1离散化

```
vector<ll>origin;
sort(origin.begin(),origin.end());
origin.erase(unique(origin.begin(),origin.end()),origin.end());//去重
ll find(ll x)
{
    return
lower_bound(origin.begin(),origin.end(),x)-origin.begin();
}
```

7.2实数三分

```
while(r-l>eps)
{
    double lmid=l+(r-l)/3;
    double rmid=r-(r-l)/3;
    double lans=cal(lmid),rans=cal(rmid);
    //凹函数最小值
    if(lans<=rans) r=rmid;
    else l=lmid;
    //凸函数最大值
    if(lans>=rans) l=lmid;
    else r=rmid;
}
```

7.3整数三分

```
while(l<r-1)
{
    ll m=(l+r)>>1;
    ll mm=(r+m)>>1;
    if(cal(m)>cal(mm)) l=m;
    else r=mm;
}
min(cal(l),cal(r));
```

7.4高精度

```
void read_big(vector<int>&A)//-123在vector中储存是1321
{
    A.clear();
    char s[100000];
    scanf("%s",s);
    if(s[0]=='-') A.push_back(1);//A[0]=0表示A是正数，
A[0]=1表示A是负数
    else A.push_back(0);
    for(int i=(int)strlen(s)-1;i>0;i--)
A.push_back(s[i]-'0');
    if(A[0]==0) A.push_back(s[0]-'0');
    if(A.size()==2&&A[1]==0) A[0]=0;//-0变成+0
}

void out_big(vector<int>&A)//输出函数
{
    if(A[0]) printf("-");
    for(int i=(int)A.size()-1;i>0;i--)
printf("%d",A[i]);
    printf("\n");
}

vector<int> add_big(vector<int>&A,vector<int> &B)
{
    vector<int>C;
    if(A[0]==B[0])//加法
    {
        C.push_back(A[0]);
        int rest=0;
        for(int i=1;i<A.size()||i<B.size();i++)
        {
            if(i<A.size()) rest+=A[i];
            if(i<B.size()) rest+=B[i];
            C.push_back(rest%10);
            rest/=10;
        }
        if(rest) C.push_back(1);
    }
```

```
    else//减法，如果只需要进行非负整数间的加法就不要写这个部分
    {
        int f;
        if(A.size()>B.size()) f=1;
        else if(A.size()<B.size()) f=-1;
        else
        {
            f=0;
            for(int i=(int)A.size()-1;i>0;i--)
            {
                if(A[i]>B[i]) {f=1;break;}
                else if(A[i]<B[i]) {f=-1;break;}
            }
        }
        if(f==0) C.push_back(0),C.push_back(0);
        else
        {
            vector<int>D;
            if(f==1) C=A,D=B;
            else C=B,D=A;
            for(int i=1;i<C.size();i++)
            {
                if(i<D.size()) C[i]-=D[i];
                if(C[i]<0) C[i]+=10,C[i+1]--;
            }
            while(C.size()>2&&C[(int)C.size()-1]==0)
C.pop_back();
        }
    }
    if(C.size()==2&&C[1]==0) C[0]=0;
    return C;
}

vector<int> sub_big(vector<int>&A,vector<int>&B)//减法
本质写在了加法函数里
{
    B[0]=!B[0];
    return add_big(A,B);
}
```

```
vector<int> mul_big(vector<int>&A,vector<int>&B)
{
    vector<int>C;
    C.resize((int)A.size()+(int)B.size()-1,0);
    if(A[0]!=B[0]) C[0]=1;
    for(int i=1;i<A.size();i++)
    {
        int rest=0;
        for(int j=1;j<B.size();j++)
        {
            C[i+j-1]+=A[i]*B[j]+rest;
            rest=C[i+j-1]/10;
            C[i+j-1]%=10;
        }
        C[i+(int)B.size()-1]=rest;
    }
    while(C.size()>2&&C[(int)C.size()-1]==0)
C.pop_back();
    if(C.size()==2&&C[1]==0) C[0]=0;
    return C;
}

void div_big(vector<int>&A,ll &B,vector<int>&C,ll
&D)//计算A%B=C余D
{
    if(!B) exit(-1);
    C.clear();
    C.resize(A.size(),0);
    D=0;
    if(A[0]&&B>0||A[0]==0&&B<0) C[0]=1;
    B=abs(B);
    for(int i=(int)A.size()-1;i>0;i--)
    {
        C[i]=(D*10+A[i])/B;
        D=(D*10+A[i])%B;
    }
    while(C.size()>2&&C[(int)C.size()-1]==0)
C.pop_back();
    if(C.size()==2&&C[1]==0) C[0]=0;
```

```
    if(A[0]==1) D=-D;
}
```