[字符串处理 2](#_bookmark0)

[1、KMP 算法 2](#_bookmark1)

[2、扩展 KMP 5](#_bookmark2)

[3、Manacher 最长回文子串 6](#_bookmark3)

[4、AC 自动机 7](#_bookmark4)

[5、后缀数组 9](#_bookmark5)

[6、后缀自动机 13](#_bookmark6)

[7、字符串 HASH 15](#_bookmark7)

[数学 17](#_bookmark8)

[1、素数 17](#_bookmark9)

[2、素数筛选和合数分解 19](#_bookmark10)

[3、扩展欧几里得算法（求 ax+by=gcd 的解以及逆元素） 20](#_bookmark11)

[4、求逆元 20](#_bookmark12)

[5、模线性方程组 20](#_bookmark13)

[6、随机素数测试和大数分解(POJ 1811) 21](#_bookmark14)

[7、欧拉函数 24](#_bookmark15)

[8、高斯消元（浮点数） 25](#_bookmark16)

[9、FFT 26](#_bookmark17)

[10、高斯消元法求方程组的解 29](#_bookmark18)

[11、 整数拆分 34](#_bookmark19)

[12、求 A^B 的约数之和对 MOD 取模 36](#_bookmark20)

[13、莫比乌斯反演 37](#_bookmark21)

[14、Baby-Step Giant-Step 39](#_bookmark22)

[15、自适应 simpson 积分 40](#_bookmark23)

[相关公式 40](#_bookmark24)

[数据结构 42](#_bookmark25)

[1、划分树 42](#_bookmark26)

[2、RMQ 43](#_bookmark27)

[3、树链剖分 45](#_bookmark28)

[4、伸展树（splay tree） 50](#_bookmark29)

[5、动态树 55](#_bookmark30)

[6、主席树 60](#_bookmark31)

[7、Treap 70](#_bookmark32)

[图论 75](#_bookmark33)

[1、最短路 75](#_bookmark34)

[2、最小生成树 79](#_bookmark35)

[3、次小生成树 80](#_bookmark36)

[4、有向图的强连通分量 81](#_bookmark37)

[5、图的割点、桥和双连通分支的基本概念 84](#_bookmark38)

[6、割点与桥 85](#_bookmark39)

[7、边双连通分支 88](#_bookmark40)

[8、点双连通分支 90](#_bookmark41)

[9、最小树形图 93](#_bookmark42)

[10、二分图匹配 95](#_bookmark43)

[11、生成树计数 98](#_bookmark44)

[11、二分图多重匹配 101](#_bookmark45)

[12、KM 算法（二分图最大权匹配） 102](#_bookmark46)

[13、最大流 103](#_bookmark47)

[14、最小费用最大流 109](#_bookmark48)

[15、2-SAT 110](#_bookmark49)

[16、曼哈顿最小生成树 114](#_bookmark50)

[17、一般图匹配带花树 117](#_bookmark51)

[18、LCA 120](#_bookmark52)

[19、欧拉路 126](#_bookmark53)

[计算几何 133](#_bookmark54)

[1、基本函数 133](#_bookmark55)

[2、凸包 138](#_bookmark56)

[3、平面最近点对（HDU 1007） 139](#_bookmark57)

[4、旋转卡壳 140](#_bookmark58)

[5、半平面交 146](#_bookmark59)

[6、三点求圆心坐标（三角形外心） 149](#_bookmark60)

[7、求两圆相交的面积 150](#_bookmark61)

[8、Pick 公式 150](#_bookmark62)

[动态规划 150](#_bookmark63)

[1、最长上升子序列 O(nlogn) 150](#_bookmark64)

[搜索 151](#_bookmark65)

[1、Dancing Links 151](#_bookmark66)

[其他 156](#_bookmark67)

[1、高精度 156](#_bookmark68)

[2、完全高精度 158](#_bookmark69)

[3、strtok 和 sscanf 结合输入 163](#_bookmark70)

[4、解决爆栈，手动加栈 163](#_bookmark71)

[5、STL 163](#_bookmark72)

[6、输入输出外挂 165](#_bookmark73)

[7、莫队算法 166](#_bookmark74)

[8、VIM 配置 172](#_bookmark75)

# 字符串处理

## 1、KMP 算法

/\*

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

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

\*/

**void kmp\_pre**(**char** x[],**int** m,**int** next[])

{

**int** i,j; j=next[0]=-1; i=0;

**while**(i<m)

{

**while**(-1!=j && x[i]!=x[j])j=next[j]; next[++i]=++j;

}

}

/\*

\* kmpNext[]的意思：next'[i]=next[next[...[next[i]]]] (直到next'[i]<0或者

x[next'[i]]!=x[i])

\* 这样的预处理可以快一些

\*/

**void preKMP**(**char** x[],**int** m,**int** kmpNext[])

{

**int** i,j; j=kmpNext[0]=-1; i=0;

**while**(i<m)

{

**while**(-1!=j && x[i]!=x[j])j=kmpNext[j]; **if**(x[++i]==x[++j])kmpNext[i]=kmpNext[j]; **else** kmpNext[i]=j;

}

}

/\*

\* 返回x在y中出现的次数，可以重叠

\*/

**int** next[10010];

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

{//x是模式串，y是主串

**int** i,j;

**int** ans=0;

//preKMP(x,m,next); kmp\_pre(x,m,next); i=j=0;

**while**(i<n)

{

**while**(-1!=j && y[i]!=x[j])j=next[j]; i++;j++;

**if**(j>=m)

{

ans++; j=next[j];

}

}

**return** ans;

}

经典题目：POJ 3167

/\*

\* POJ 3167 Cow Patterns

\* 模式串可以浮动的模式匹配问题

\* 给出模式串的相对大小，需要找出模式串匹配次数和位置

\* 比如说模式串：1，4，4，2，3，1 而主串：5,6,2,10,10,7,3,2,9

\* 那么2,10,10,7,3,2就是匹配的

\*

\* 统计比当前数小，和于当前数相等的，然后进行kmp

\*/

**#include** <iostream>

**#include** <stdio.h>

**#include** <string.h>

**#include** <algorithm>

**#include** <vector> **using namespace** std; **const int** MAXN=100010; **const int** MAXM=25010; **int** a[MAXN];

**int** b[MAXN];

**int** n,m,s;

**int** as[MAXN][30];

**int** bs[MAXM][30];

void init()

{

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

**if**(i==0)

{

**for**(**int** j=1;j<=25;j++)as[i][j]=0;

}

else

{

**for**(**int** j=1;j<=25;j++)as[i][j]=as[i-1][j];

}

as[i][a[i]]++;

}

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

{

**if**(i==0)

{

**for**(**int** j=1;j<=25;j++)bs[i][j]=0;

}

else

{

**for**(**int** j=1;j<=25;j++)bs[i][j]=bs[i-1][j];

}

bs[i][b[i]]++;

}

}

**int** next[MAXM];

void kmp\_pre()

{

**int** i,j; j=next[0]=-1; i=0;

**while**(i<m)

{

**int** t11=0,t12=0,t21=0,t22=0;

**for**(**int** k=1;k<b[i];k++)

{

**if**(i-j>0)t11+=bs[i][k]-bs[i-j-1][k]; **else** t11+=bs[i][k];

}

**if**(i-j>0)t12=bs[i][b[i]]-bs[i-j-1][b[i]]; **else** t12=bs[i][b[i]];

**for**(**int** k=1;k<b[j];k++)

{

t21+=bs[j][k];

}

t22=bs[j][b[j]];

**if**(j==-1 || (t11==t21&&t12==t22))

{

next[++i]=++j;

}

**else** j=next[j];

}

}

vector<**int**>ans;

void kmp()

{

ans.clear(); **int** i,j; kmp\_pre();

i=j=0;

**while**(i<n)

{

**int** t11=0,t12=0,t21=0,t22=0;

**for**(**int** k=1;k<a[i];k++)

{

**if**(i-j>0)t11+=as[i][k]-as[i-j-1][k]; **else** t11+=as[i][k];

}

**if**(i-j>0)t12=as[i][a[i]]-as[i-j-1][a[i]]; **else** t12=as[i][a[i]];

**for**(**int** k=1;k<b[j];k++)

{

t21+=bs[j][k];

}

t22=bs[j][b[j]];

**if**(j==-1 || (t11==t21&&t12==t22))

{

i++;j++;

**if**(j>=m)

{

ans.push\_back(i-m+1); j=next[j];

}

}

**else** j=next[j];

}

}

int main()

{

**while**(**scanf**("%d%d%d",&n,&m,&s)==3)

{

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

{

**scanf**("%d",&a[i]);

}

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

{

**scanf**("%d",&b[i]);

}

init();

kmp(); **printf**("%d\n",ans.size()); **for**(**int** i=0;i<ans.size();i++)

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

}

**return** 0;

}

## 2、扩展 KMP

/\*

\* 扩展KMP算法

\*/

//next[i]:x[i...m-1]与x[0...m-1]的最长公共前缀

//extend[i]:y[i...n-1]与x[0...m-1]的最长公共前缀

**void pre\_EKMP**(**char** x[],**int** m,**int** next[])

{

next[0]=m;

**int** j=0;

**while**(j+1<m && x[j]==x[j+1])j++; next[1]=j;

**int** k=1;

**for**(**int** i=2;i<m;i++)

{

**int** p=next[k]+k-1; **int** L=next[i-k]; **if**(i+L<p+1)next[i]=L; **else**

{

j=max(0,p-i+1);

**while**(i+j<m && x[i+j]==x[j])j++; next[i]=j;

k=i;

}

}

}

**void EKMP**(**char** x[],**int** m,**char** y[],**int** n,**int** next[],**int** extend[])

{

pre\_EKMP(x,m,next);

**int** j=0;

**while**(j<n && j<m && x[j]==y[j])j++; extend[0]=j;

**int** k=0;

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

{

**int** p=extend[k]+k-1; **int** L=next[i-k]; **if**(i+L<p+1)extend[i]=L; **else**

{

j=max(0,p-i+1);

**while**(i+j<n && j<m && y[i+j]==x[j])j++; extend[i]=j;

k=i;

}

}

}

## 3、Manacher 最长回文子串

/\*

\* 求最长回文子串

\*/

**const int** MAXN=110010;

**char** Ma[MAXN\*2];

**int** Mp[MAXN\*2];

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

{

**int** l=0; Ma[l++]='$';

Ma[l++]='#';

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

{

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

Ma[l++]='#';

}

Ma[l]=0;

**int** mx=0,id=0;

**for**(**int** i=0;i<l;i++)

{

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

**while**(Ma[i+Mp[i]]==Ma[i-Mp[i]])Mp[i]++; **if**(i+Mp[i]>mx)

{

mx=i+Mp[i]; id=i;

}

}

}

/\*

\* abaaba

\* i: 0 1 2 3 4 5 6 7 8 9 10 11 12 13

\* Ma[i]: $ # a # b # a # a $ b # a #

\* Mp[i]: 1 1 2 1 4 1 2 7 2 1 4 1 2 1

\*/

**char** s[MAXN];

int main()

{

**while**(**scanf**("%s",s)==1)

{

**int** len=**strlen**(s); Manacher(s,len); **int** ans=0;

**for**(**int** i=0;i<2\*len+2;i++) ans=max(ans,Mp[i]-1);

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

}

**return** 0;

}

## 4、AC 自动机

//======================

// HDU 2222

// 求目标串中出现了几个模式串

//====================

**#include** <stdio.h>

**#include** <algorithm>

**#include** <iostream>

**#include** <string.h>

**#include** <queue>

using namespace std;

**struct** Trie

{

**int** next[500010][26],fail[500010],end[500010];

**int** root,L;

int newnode()

{

**for**(**int** i = 0;i < 26;i++) next[L][i] = -1;

end[L++] = 0;

**return** L-1;

}

void init()

{

L = 0;

root = newnode();

}

**void insert**(**char** buf[])

{

**int** len = **strlen**(buf); **int** now = root;

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

{

**if**(next[now][buf[i]-'a'] == -1)

next[now][buf[i]-'a'] = newnode();

now = next[now][buf[i]-'a'];

}

end[now]++;

}

void build()

{

queue<**int**>Q; fail[root] = root;

**for**(**int** i = 0;i < 26;i++)

**if**(next[root][i] == -1) next[root][i] = root;

else

{

fail[next[root][i]] = root; Q.push(next[root][i]);

}

**while**( !Q.empty() )

{

**int** now = Q.front(); Q.pop();

**for**(**int** i = 0;i < 26;i++)

**if**(next[now][i] == -1)

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

else

{

fail[next[now][i]]=next[fail[now]][i]; Q.push(next[now][i]);

}

}

}

**int query**(**char** buf[])

{

**int** len = **strlen**(buf); **int** now = root;

**int** res = 0;

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

{

now = next[now][buf[i]-'a'];

**int** temp = now;

**while**( temp != root )

{

res += end[temp]; end[temp] = 0; temp = fail[temp];

}

}

**return** res;

}

void debug()

{

**for**(**int** i = 0;i < L;i++)

{

**printf**("id = %3d,fail = %3d,end = %3d,chi = [",i,fail[i],end[i]);

**for**(**int** j = 0;j < 26;j++)

**printf**("%2d",next[i][j]); **printf**("]\n");

}

}

};

**char** buf[1000010]; Trie ac;

int main()

{

**int** T;

**int** n; **scanf**("%d",&T); **while**( T-- )

{

**scanf**("%d",&n); ac.init();

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

{

**scanf**("%s",buf); ac.insert(buf);

}

ac.build();

**scanf**("%s",buf); **printf**("%d\n",ac.query(buf));

}

**return** 0;

}

## 5、后缀数组

5.1 DA 算法

/\*

\*suffix array

\*倍增算法 O(n\*logn)

\*待排序数组长度为n,放在0~n-1中，在最后面补一个0

\*da(str ,n+1,sa,rank,height, , );//注意是n+1;

\*例如：

\*n = 8;

\*num[] = { 1, 1, 2, 1, 1, 1, 1, 2, $ };注意num最后一位为0，其他大于0

\*rank[] = { 4, 6, 8, 1, 2, 3, 5, 7, 0 };rank[0~n-1]为有效值，rank[n]必定为0无效 值

\*sa[] = { 8, 3, 4, 5, 0, 6, 1, 7, 2 };sa[1~n]为有效值，sa[0]必定为n是无效值

\*height[]= { 0, 0, 3, 2, 3, 1, 2, 0, 1 };height[2~n]为有效值

\*

\*/

**const int** MAXN=20010;

**int** t1[MAXN],t2[MAXN],c[MAXN];//求SA数组需要的中间变量，不需要赋值

//待排序的字符串放在s数组中，从s[0]到s[n-1],长度为n,且最大值小于m,

//除s[n-1]外的所有s[i]都大于0，r[n-1]=0

//函数结束以后结果放在sa数组中

**bool cmp**(**int** \*r,**int** a,**int** b,**int** l)

{

**return** r[a] == r[b] && r[a+l] == r[b+l];

}

**void da**(**int** str[],**int** sa[],**int** rank[],**int** height[],**int** n,**int** m)

{

str[n++]=0;

**int** i, j, p, \*x = t1, \*y = t2;

//第一轮基数排序，如果s的最大值很大，可改为快速排序

**for**(i = 0;i < m;i++)c[i] = 0;

**for**(i = 0;i < n;i++)c[x[i] = str[i]]++;

**for**(i = 1;i < m;i++)c[i] += c[i-1];

**for**(i = n-1;i >= 0;i--)sa[--c[x[i]]] = i;

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

{

p = 0;

//直接利用sa数组排序第二关键字

**for**(i = n-j; i < n; i++)y[p++] = i;//后面的j个数第二关键字为空的最小

**for**(i = 0; i < n; i++)**if**(sa[i] >= j)y[p++] = sa[i] - j;

//这样数组y保存的就是按照第二关键字排序的结果

//基数排序第一关键字

**for**(i = 0; i < m; i++)c[i] = 0;

**for**(i = 0; i < n; i++)c[x[y[i]]]++;

**for**(i = 1; i < m;i++)c[i] += c[i-1];

**for**(i = n-1; i >= 0;i--)sa[--c[x[y[i]]]] = y[i];

//根据sa和x数组计算新的x数组

swap(x,y);

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

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

x[sa[i]] = cmp(y,sa[i-1],sa[i],j)?p-1:p++;

**if**(p >= n)**break**;

m = p;//下次基数排序的最大值

}

**int** k = 0; n--;

**for**(i = 0;i <= n;i++)rank[sa[i]] = i;

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

{

**if**(k)k--;

j = sa[rank[i]-1]; **while**(str[i+k] == str[j+k])k++; height[rank[i]] = k;

}

}

**int** rank[MAXN],height[MAXN];

**int** RMQ[MAXN];**int** mm[MAXN];

**int** best[20][MAXN];

void initRMQ(int n)

{

mm[0]=-1;

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

mm[i]=((i&(i-1))==0)?mm[i-1]+1:mm[i-1];

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

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

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

{

**int** a=best[i-1][j];

**int** b=best[i-1][j+(1<<(i-1))];

**if**(RMQ[a]<RMQ[b])best[i][j]=a; **else** best[i][j]=b;

}

}

**int askRMQ**(**int** a,**int** b)

{

**int** t; t=mm[b-a+1];

b-=(1<<t)-1;

a=best[t][a];b=best[t][b];

**return** RMQ[a]<RMQ[b]?a:b;

}

**int lcp**(**int** a,**int** b)

{

a=rank[a];b=rank[b];

**if**(a>b)swap(a,b);

**return** height[askRMQ(a+1,b)];

}

**char** str[MAXN]; **int** r[MAXN]; **int** sa[MAXN]; **int main**()

{

**while**(**scanf**("%s",str) == 1)

{

**int** len = **strlen**(str); **int** n = 2\*len + 1;

**for**(**int** i = 0;i < len;i++)r[i] = str[i];

**for**(**int** i = 0;i < len;i++)r[len + 1 + i] = str[len - 1 - i]; r[len] = 1;

r[n] = 0;

da(r,sa,rank,height,n,128);

**for**(**int** i=1;i<=n;i++)RMQ[i]=height[i]; initRMQ(n);

**int** ans=0,st;

**int** tmp;

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

{

tmp=lcp(i,n-i);//偶对称

**if**(2\*tmp>ans)

{

ans=2\*tmp; st=i-tmp;

}

tmp=lcp(i,n-i-1);//奇数对称

**if**(2\*tmp-1>ans)

{

ans=2\*tmp-1; st=i-tmp+1;

}

}

str[st+ans]=0;

**printf**("%s\n",str+st);

}

**return** 0;

}

5.2 DC3 算法

da[]和str[]数组要开大三倍，相关数组也是三倍

/\*

\* 后缀数组

\* DC3算法，复杂度O(n)

\* 所有的相关数组都要开三倍

\*/

**const int** MAXN = 2010;

**#define** F(x) ((x)/3+((x)%3==1?0:tb))

**#define** G(x) ((x)<tb?(x)\*3+1:((x)-tb)\*3+2)

**int** wa[MAXN\*3],wb[MAXN\*3],wv[MAXN\*3],wss[MAXN\*3];

**int c0**(**int** \*r,**int** a,**int** b)

{

**return** r[a] == r[b] && r[a+1] == r[b+1] && r[a+2] == r[b+2];

}

**int c12**(**int** k,**int** \*r,**int** a,**int** b)

{

**if**(k == 2)

**return** r[a] < r[b] || ( r[a] == r[b] && c12(1,r,a+1,b+1) );

**else return** r[a] < r[b] || ( r[a] == r[b] && wv[a+1] < wv[b+1] );

}

**void sort**(**int** \*r,**int** \*a,**int** \*b,**int** n,**int** m)

{

**int** i;

**for**(i = 0;i < n;i++)wv[i] = r[a[i]]; **for**(i = 0;i < m;i++)wss[i] = 0; **for**(i = 0;i < n;i++)wss[wv[i]]++;

**for**(i = 1;i < m;i++)wss[i] += wss[i-1];

**for**(i = n-1;i >= 0;i--)

b[--wss[wv[i]]] = a[i];

}

**void dc3**(**int** \*r,**int** \*sa,**int** n,**int** m)

{

**int** i, j, \*rn = r + n;

**int** \*san = sa + n, ta = 0, tb = (n+1)/3, tbc = 0, p; r[n] = r[n+1] = 0;

**for**(i = 0;i < n;i++)**if**(i %3 != 0)wa[tbc++] = i; sort(r + 2, wa, wb, tbc, m);

sort(r + 1, wb, wa, tbc, m);

sort(r, wa, wb, tbc, m);

**for**(p = 1, rn[F(wb[0])] = 0, i = 1;i < tbc;i++) rn[F(wb[i])] = c0(r, wb[i-1], wb[i]) ? p - 1 : p++;

**if**(p < tbc)dc3(rn,san,tbc,p);

**else for**(i = 0;i < tbc;i++)san[rn[i]] = i;

**for**(i = 0;i < tbc;i++) **if**(san[i] < tb)wb[ta++] = san[i] \* 3;

**if**(n % 3 == 1)wb[ta++] = n - 1; sort(r, wb, wa, ta, m);

**for**(i = 0;i < tbc;i++)wv[wb[i] = G(san[i])] = i;

**for**(i = 0, j = 0, p = 0;i < ta && j < tbc;p++)

sa[p] = c12(wb[j] % 3, r, wa[i], wb[j]) ? wa[i++] : wb[j++];

**for**(;i < ta;p++)sa[p] = wa[i++];

**for**(;j < tbc;p++)sa[p] = wb[j++];

}

//str和sa也要三倍

**void da**(**int** str[],**int** sa[],**int** rank[],**int** height[],**int** n,**int** m)

{

**for**(**int** i = n;i < n\*3;i++) str[i] = 0;

dc3(str, sa, n+1, m);

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

**for**(i = 0;i <= n;i++)rank[sa[i]] = i;

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

{

**if**(k) k--;

j = sa[rank[i]-1]; **while**(str[i+k] == str[j+k]) k++; height[rank[i]] = k;

}

}

## 6、后缀自动机

**const int** CHAR = 26; **const int** MAXN = 250010; **struct** SAM\_Node

{

SAM\_Node \*fa,\*next[CHAR];

**int** len; **int** id,pos; **SAM\_Node**(){}

**SAM\_Node**(**int** \_len)

{

fa = 0;

len = \_len;

**memset**(next,0,**sizeof**(next));

}

};

SAM\_Node SAM\_node[MAXN\*2], \*SAM\_root, \*SAM\_last;

**int** SAM\_size;

SAM\_Node \***newSAM\_Node**(**int** len)

{

SAM\_node[SAM\_size] = SAM\_Node(len); SAM\_node[SAM\_size].id = SAM\_size; **return** &SAM\_node[SAM\_size++];

}

SAM\_Node \***newSAM\_Node**(SAM\_Node \*p)

{

SAM\_node[SAM\_size] = \*p; SAM\_node[SAM\_size].id = SAM\_size; **return** &SAM\_node[SAM\_size++];

}

void SAM\_init()

{

SAM\_size = 0;

SAM\_root = SAM\_last = newSAM\_Node(0); SAM\_node[0].pos = 0;

}

**void SAM\_add**(**int** x,**int** len)

{

SAM\_Node \*p = SAM\_last, \*np = newSAM\_Node(p->len+1); np->pos = len;

SAM\_last = np;

**for**(;p && !p->next[x];p = p->fa) p->next[x] = np;

**if**(!p)

{

np->fa = SAM\_root;

return;

}

SAM\_Node \*q = p->next[x];

**if**(q->len == p->len + 1)

{

np->fa = q;

return;

}

SAM\_Node \*nq = newSAM\_Node(q); nq->len = p->len + 1;

q->fa = nq; np->fa = nq;

**for**(;p && p->next[x] == q;p = p->fa) p->next[x] = nq;

}

void SAM\_build(char \*s)

{

SAM\_init();

**int** len = **strlen**(s); **for**(**int** i = 0;i < len;i++)

SAM\_add(s[i] - 'a',i+1);

}

//加入串后进行拓扑排序。

**char** str[MAXN];

**int** topocnt[MAXN]; SAM\_Node \*topsam[MAXN\*2];

**int** n = **strlen**(str); SAM\_build(str); **memset**(topocnt,0,**sizeof**(topocnt)); **for**(**int** i = 0;i < SAM\_size;i++)

topocnt[SAM\_node[i].len]++;

**for**(**int** i = 1;i <= n;i++) topocnt[i] += topocnt[i-1];

**for**(**int** i = 0;i < SAM\_size;i++)

topsam[--topocnt[SAM\_node[i].len]] = &SAM\_node[i];

多串的建立：

//多串的建立,注意SAM\_init()的调用

void SAM\_build(char \*s)

{

**int** len = **strlen**(s); SAM\_last = SAM\_root; **for**(**int** i = 0;i < len;i++)

{ i+1) )

}

**if**( !SAM\_last->next[s[i] - '0'] || !(SAM\_last->next[s[i] - '0']->len == SAM\_add(s[i] - '0',i+1);

**else** SAM\_last = SAM\_last->next[s[i] - '0'];

}

## 7、字符串 HASH

HDU4622 求区间不相同子串个数 **const int** HASH = 10007; **const int** MAXN = 2010; **struct** HASHMAP

{

**int** head[HASH],next[MAXN],size; **unsigned long long** state[MAXN]; **int** f[MAXN];

void init()

{

size = 0;

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

}

**int insert**(**unsigned long long** val,**int** \_id)

{

**int** h = val%HASH;

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

**if**(val == state[i])

{

**int** tmp = f[i]; f[i] = \_id; **return** tmp;

}

}

} H;

f[size] = \_id; state[size] = val; next[size] = head[h]; head[h] = size++; **return** 0;

**const int** SEED = 13331; **unsigned long long** P[MAXN]; **unsigned long long** S[MAXN]; **char** str[MAXN];

**int** ans[MAXN][MAXN];

int main()

{

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

//freopen("out.txt","w",stdout); P[0] = 1;

**for**(**int** i = 1;i < MAXN;i++) P[i] = P[i-1] \* SEED;

**int** T; **scanf**("%d",&T); **while**(T--)

{

**scanf**("%s",str);

**int** n = **strlen**(str); S[0] = 0;

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

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

**memset**(ans,0,**sizeof**(ans)); **for**(**int** L = 1; L <= n;L++)

{

H.init();

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

{

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

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

}

}

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

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

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

**int** m,u,v; **scanf**("%d",&m); **while**(m--)

{

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

}

}

**return** 0;

}

# 数学

## 1、素数

1.1 素数筛选（判断<MAXN 的数是否素数）

/\*

\* 素数筛选，判断小于MAXN的数是不是素数。

\* notprime是一张表，为false表示是素数，true表示不是素数

\*/

**const int** MAXN=1000010;

**bool** notprime[MAXN];//值为false表示素数，值为true表示非素数

void init()

{

**memset**(notprime,**false**,**sizeof**(notprime)); notprime[0]=notprime[1]=**true**;

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

**if**(!notprime[i])

{

**if**(i>MAXN/i)**continue**;//防止后面i\*i溢出(或者i,j用long long)

//直接从i\*i开始就可以，小于i倍的已经筛选过了,注意是j+=i

**for**(**int** j=i\*i;j<MAXN;j+=i) notprime[j]=**true**;

}

}

1.2 素数筛选（筛选出小于等于 MAXN 的素数）

/\*

\* 素数筛选，存在小于等于MAXN的素数

\* prime[0] 存的是素数的个数

\*/

**const int** MAXN=10000; **int** prime[MAXN+1]; **void getPrime**()

{

**memset**(prime,0,**sizeof**(prime)); **for**(**int** i=2;i<=MAXN;i++)

{

**if**(!prime[i])prime[++prime[0]]=i;

**for**(**int** j=1;j<=prime[0]&&prime[j]<=MAXN/i;j++)

{

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

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

}

}

}

1.3 大区间素数筛选（POJ 2689）

/\*

\* POJ 2689 Prime Distance

\* 给出一个区间[L,U]，找出区间内容、相邻的距离最近的两个素数和

\* 距离最远的两个素数。

\* 1<=L<U<=2,147,483,647 区间长度不超过1,000,000

\* 就是要筛选出[L,U]之间的素数

\*/

**#include** <stdio.h>

**#include** <algorithm>

**#include** <iostream>

**#include** <string.h>

using namespace std;

**const int** MAXN=100010; **int** prime[MAXN+1]; **void getPrime**()

{

**memset**(prime,0,**sizeof**(prime)); **for**(**int** i=2;i<=MAXN;i++)

{

**if**(!prime[i])prime[++prime[0]]=i;

**for**(**int** j=1;j<=prime[0]&&prime[j]<=MAXN/i;j++)

{

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

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

}

}

}

**bool** notprime[1000010];

**int** prime2[1000010];

void getPrime2(int L,int R)

{

**memset**(notprime,**false**,**sizeof**(notprime)); **if**(L<2)L=2;

**for**(**int** i=1;i<=prime[0]&&(**long long**)prime[i]\*prime[i]<=R;i++)

{

**int** s=L/prime[i]+(L%prime[i]>0);

**if**(s==1)s=2;

**for**(**int** j=s;(**long long**)j\*prime[i]<=R;j++) **if**((**long long**)j\*prime[i]>=L)

notprime[j\*prime[i]-L]=**true**;

}

prime2[0]=0;

**for**(**int** i=0;i<=R-L;i++)

**if**(!notprime[i]) prime2[++prime2[0]]=i+L;

}

int main()

{

getPrime();

**int** L,U;

**while**(**scanf**("%d%d",&L,&U)==2)

{

getPrime2(L,U);

**if**(prime2[0]<2)**printf**("There are no adjacent primes.\n");

else

{

**int** x1=0,x2=100000000,y1=0,y2=0;

**for**(**int** i=1;i<prime2[0];i++)

{

**if**(prime2[i+1]-prime2[i]<x2-x1)

{

x1=prime2[i]; x2=prime2[i+1];

}

**if**(prime2[i+1]-prime2[i]>y2-y1)

{

y1=prime2[i]; y2=prime2[i+1];

}

}

**printf**("%d,%d are closest, %d,%d are most distant.\n",x1,x2,y1,y2);

}

}

}

## 2、素数筛选和合数分解

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

//素数筛选和合数分解 **const int** MAXN=10000; **int** prime[MAXN+1]; **void getPrime**()

{

**memset**(prime,0,**sizeof**(prime)); **for**(**int** i=2;i<=MAXN;i++)

{

**if**(!prime[i])prime[++prime[0]]=i;

**for**(**int** j=1;j<=prime[0]&&prime[j]<=MAXN/i;j++)

{

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

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

}

}

}

**long long** factor[100][2];

**int** fatCnt;

int getFactors(long long x)

{

fatCnt=0;

**long long** tmp=x;

**for**(**int** i=1;prime[i]<=tmp/prime[i];i++)

{

factor[fatCnt][1]=0;

**if**(tmp%prime[i]==0)

{

factor[fatCnt][0]=prime[i];

**while**(tmp%prime[i]==0)

{

factor[fatCnt][1]++; tmp/=prime[i];

}

fatCnt++;

}

}

**if**(tmp!=1)

{

factor[fatCnt][0]=tmp;

factor[fatCnt++][1]=1;

}

**return** fatCnt;

}

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

## 3、扩展欧几里得算法（求 ax+by=gcd 的解以及逆元素）

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

//返回d=gcd(a,b);和对应于等式ax+by=d中的x,y

long long extend\_gcd(long long a,long long b,long long &x,long long &y)

{

**if**(a==0&&b==0) **return** -1;//无最大公约数

**if**(b==0){x=1;y=0;**return** a;}

**long long** d=extend\_gcd(b,a%b,y,x); y-=a/b\*x;

**return** d;

}

//\*\*\*\*\*\*\*\*\*求逆元素\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//ax = 1(mod n)

long long mod\_reverse(long long a,long long n)

{

**long long** x,y;

**long long** d=extend\_gcd(a,n,x,y);

**if**(d==1) **return** (x%n+n)%n;

else return -1;

}

## 4、求逆元

4.1 扩展欧几里德法（见上面）

4.2 简洁写法

注意：这个只能求a < m的情况，而且必须保证a和m互质

//求ax = 1( mod m) 的x值，就是逆元(0<a<m)

long long inv(long long a,long long m)

{

**if**(a == 1)**return** 1;

**return** inv(m%a,m)\*(m-m/a)%m;

}

4.3 利用欧拉函数

mod为素数,而且a和m互质

**long long inv**(**long long** a,**long long** mod)//mod为素数

{

**return** pow\_m(a,mod-2,mod);

}

## 5、模线性方程组

long long extend\_gcd(long long a,long long b,long long &x,long long &y)

{

**if**(a == 0 && b == 0)**return** -1;

**if**(b ==0 ){x = 1; y = 0;**return** a;} **long long** d = extend\_gcd(b,a%b,y,x); y -= a/b\*x;

**return** d;

}

**int** m[10],a[10];//模数为m,余数为a, X % m = a

**bool solve**(**int** &m0,**int** &a0,**int** m,**int** a)

{

**long long** y,x;

**int** g = extend\_gcd(m0,m,x,y); **if**( **abs**(a - a0)%g )**return false**; x \*= (a - a0)/g;

x %= m/g;

a0 = (x\*m0 + a0); m0 \*= m/g;

a0 %= m0;

**if**( a0 < 0 )a0 += m0;

return true;

}

/\*

\* 无解返回false,有解返回true;

\* 解的形式最后为 a0 + m0 \* t (0<=a0<m0)

\*/

**bool MLES**(**int** &m0 ,**int** &a0,**int** n)//解为 X = a0 + m0 \* k

{

**bool** flag = **true**; m0 = 1;

a0 = 0;

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

**if**( !solve(m0,a0,m[i],a[i]) )

{

flag = **false**; **break**;

}

**return** flag;

}

## 6、随机素数测试和大数分解(POJ 1811)

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

\* Miller\_Rabin 算法进行素数测试

\* 速度快，可以判断一个 < 2^63 的数是不是素数

\*

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

**const int** S = 8; //随机算法判定次数，一般8~10就够了

// 计算ret = (a\*b)%c a,b,c < 2^63

long long mult\_mod(long long a,long long b,long long c)

{

a %= c; b %= c;

**long long** ret = 0;

**long long** tmp = a;

**while**(b)

{

**if**(b & 1)

{

ret += tmp;

**if**(ret > c)ret -= c;//直接取模慢很多

}

tmp <<= 1;

**if**(tmp > c)tmp -= c; b >>= 1;

}

**return** ret;

}

// 计算 ret = (a^n)%mod

long long pow\_mod(long long a,long long n,long long mod)

{

**long long** ret = 1;

**long long** temp = a%mod;

**while**(n)

{

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

n >>= 1;

}

**return** ret;

}

// 通过 a^(n-1)=1(mod n)来判断n是不是素数

// n-1 = x\*2^t 中间使用二次判断

// 是合数返回true, 不一定是合数返回false

bool check(long long a,long long n,long long x,long long t)

{

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

**long long** last = ret;

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

{

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

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

last = ret;

}

**if**(ret != 1)**return true**; **else return false**;

}

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

// Miller\_Rabin算法

// 是素数返回true,(可能是伪素数)

// 不是素数返回false

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

bool Miller\_Rabin(long long n)

{

**if**( n < 2)**return false**; **if**( n == 2)**return true**;

**if**( (n&1) == 0)**return false**;//偶数

**long long** x = n - 1;

**long long** t = 0;

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

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

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

{

**long long** a = **rand**()%(n-1) + 1;

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

return false;

}

return true;

}

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

// pollard\_rho 算法进行质因素分解

//

//

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

**long long** factor[100];//质因素分解结果（刚返回时时无序的）

**int** tol;//质因素的个数，编号0~tol-1

long long gcd(long long a,long long b)

{

long long t;

**while**(b)

{

t = a; a = b;

b = t%b;

}

**if**(a >= 0)**return** a;

else return -a;

}

//找出一个因子

long long pollard\_rho(long long x,long long c)

{

**long long** i = 1, k = 2;

**srand**(**time**(NULL));

**long long** x0 = **rand**()%(x-1) + 1;

**long long** y = x0;

**while**(1)

{

i ++;

x0 = (mult\_mod(x0,x0,x) + c)%x; **long long** d = gcd(y - x0,x); **if**( d != 1 && d != x)**return** d; **if**(y == x0)**return** x;

**if**(i == k){y = x0; k += k;}

}

}

//对 n进行素因子分解，存入factor. k设置为107左右即可

void findfac(long long n,int k)

{

**if**(n == 1)**return**;

**if**(Miller\_Rabin(n))

{

factor[tol++] = n;

return;

}

**long long** p = n; **int** c = k; **while**( p >= n)

p = pollard\_rho(p,c--);//值变化，防止死循环k

findfac(p,k); findfac(n/p,k);

}

//POJ 1811

//给出一个N(2 <= N < 2^54),如果是素数，输出"Prime",否则输出最小的素因子

int main()

{

**int** T;

**long long** n; **scanf**("%d",&T); **while**(T--)

{

**scanf**("%I64d",&n); **if**(Miller\_Rabin(n))**printf**("Prime\n"); **else**

{

tol = 0; findfac(n,107);

**long long** ans = factor[0];

**for**(**int** i = 1;i < tol;i++) ans = min(ans,factor[i]);

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

}

}

**return** 0;

}

## 7、欧拉函数

6.1 分解质因素求欧拉函数

getFactors(n);

**int** ret = n;

**for**(**int** i = 0;i < fatCnt;i++)

{

ret = ret/factor[i][0]\*(factor[i][0]-1);

}

6.2 筛法欧拉函数

**int** euler[3000001];

void getEuler()

{

**memset**(euler,0,**sizeof**(euler)); euler[1] = 1;

**for**(**int** i = 2;i <= 3000000;i++)

**if**(!euler[i])

**for**(**int** j = i;j <= 3000000; j += i)

{

**if**(!euler[j]) euler[j] = j;

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

}

}

6.2 求单个数的欧拉函数

long long eular(long long n)

{

**long long** ans = n;

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

{

**if**(n % i == 0)

{

ans -= ans/i;

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

}

}

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

**return** ans;

}

6.3 线性筛（同时得到欧拉函数和素数表）

**const int** MAXN = 10000000;

**bool** check[MAXN+10]; **int** phi[MAXN+10]; **int** prime[MAXN+10];

**int** tot;//素数的个数

void phi\_and\_prime\_table(int N)

{

**memset**(check,**false**,**sizeof**(check)); phi[1] = 1;

tot = 0;

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

{

**if**( !check[i] )

{

prime[tot++] = i; phi[i] = i-1;

}

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

{

**if**(i \* prime[j] > N)**break**; check[i \* prime[j]] = **true**; **if**( i % prime[j] == 0)

{

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

break;

}

else

{

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

}

}

}

}

## 8、高斯消元（浮点数）

**#define** eps 1e-9

**const int** MAXN=220;

**double** a[MAXN][MAXN],x[MAXN];//方程的左边的矩阵和等式右边的值，求解之后x存的就是结果

**int** equ,var;//方程数和未知数个数

/\*

\*返回0表示无解，1表示有解

\*/

int Gauss()

{

**int** i,j,k,col,max\_r;

**for**(k=0,col=0;k<equ&&col<var;k++,col++)

{

max\_r=k;

**for**(i=k+1;i<equ;i++) **if**(**fabs**(a[i][col])>**fabs**(a[max\_r][col]))

max\_r=i; **if**(**fabs**(a[max\_r][col])<eps)**return** 0; **if**(k!=max\_r)

{

**for**(j=col;j<var;j++) swap(a[k][j],a[max\_r][j]);

swap(x[k],x[max\_r]);

}

x[k]/=a[k][col]; **for**(j=col+1;j<var;j++)a[k][j]/=a[k][col]; a[k][col]=1;

**for**(i=0;i<equ;i++) **if**(i!=k)

{

x[i]-=x[k]\*a[i][k]; **for**(j=col+1;j<var;j++)a[i][j]-=a[k][j]\*a[i][col]; a[i][col]=0;

}

}

**return** 1;

}

## 9、FFT

//HDU 1402 求高精度乘法

**#include** <stdio.h>

**#include** <string.h>

**#include** <iostream>

**#include** <algorithm>

**#include** <math.h>

using namespace std;

**const double** PI = **acos**(-1.0);

//复数结构体

**struct** Complex

{

**double** x,y;//实部和虚部 x+yi

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

{

x = \_x; y = \_y;

}

Complex **operator -**(**const** Complex &b)**const**

{

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

}

Complex **operator +**(**const** Complex &b)**const**

{

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

}

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

{

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

}

};

/\*

\* 进行FFT和IFFT前的反转变换。

\* 位置i和 （i二进制反转后位置）互换

\* len必须去2的幂

\*/

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

{

**int** i,j,k;

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

{

**if**(i < j)swap(y[i],y[j]);

//交换互为小标反转的元素，i<j保证交换一次

//i做正常的+1，j左反转类型的+1,始终保持i和j是反转的 k = len/2;

**while**(j >= k)

{

j -= k; k /= 2;

}

**if**(j < k)j += k;

}

}

/\*

\* 做FFT

\* len必须为2^k形式，

\* on==1时是DFT，on==-1时是IDFT

\*/

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

{

change(y,len);

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

{

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

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

{

Complex w(1,0);

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

{

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

y[k+h/2] = u-t; w = w\*wn;

}

}

}

**if**(on == -1)

**for**(**int** i = 0;i < len;i++) y[i].x /= len;

}

**const int** MAXN = 200010; Complex x1[MAXN],x2[MAXN];

**char** str1[MAXN/2],str2[MAXN/2];

**int** sum[MAXN];

int main()

{

**while**(**scanf**("%s%s",str1,str2)==2)

{

**int** len1 = **strlen**(str1); **int** len2 = **strlen**(str2); **int** len = 1;

**while**(len < len1\*2 || len < len2\*2)len<<=1;

**for**(**int** i = 0;i < len1;i++)

x1[i] = Complex(str1[len1-1-i]-'0',0);

**for**(**int** i = len1;i < len;i++) x1[i] = Complex(0,0);

**for**(**int** i = 0;i < len2;i++)

x2[i] = Complex(str2[len2-1-i]-'0',0);

**for**(**int** i = len2;i < len;i++) x2[i] = Complex(0,0);

//求DFT

fft(x1,len,1);

fft(x2,len,1);

**for**(**int** i = 0;i < len;i++) x1[i] = x1[i]\*x2[i];

fft(x1,len,-1);

**for**(**int** i = 0;i < len;i++) sum[i] = (**int**)(x1[i].x+0.5);

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

{

sum[i+1]+=sum[i]/10; sum[i]%=10;

}

len = len1+len2-1;

**while**(sum[len] <= 0 && len > 0)len--;

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

**printf**("%c",sum[i]+'0'); **printf**("\n");

}

**return** 0;

}

//HDU 4609

//给出 n 条线段长度，问任取 3 根，组成三角形的概率。

//n<=10^5 用 FFT 求可以组成三角形的取法有几种

**const int** MAXN = 400040; Complex x1[MAXN];

**int** a[MAXN/4];

**long long** num[MAXN];//100000\*100000会超int

**long long** sum[MAXN];

int main()

{

**int** T;

**int** n; **scanf**("%d",&T); **while**(T--)

{

**scanf**("%d",&n); **memset**(num,0,**sizeof**(num)); **for**(**int** i = 0;i < n;i++)

{

**scanf**("%d",&a[i]); num[a[i]]++;

}

sort(a,a+n);

**int** len1 = a[n-1]+1;

**int** len = 1;

**while**( len < 2\*len1 )len <<= 1;

**for**(**int** i = 0;i < len1;i++) x1[i] = Complex(num[i],0);

**for**(**int** i = len1;i < len;i++) x1[i] = Complex(0,0);

fft(x1,len,1);

**for**(**int** i = 0;i < len;i++) x1[i] = x1[i]\*x1[i];

fft(x1,len,-1);

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

num[i] = (**long long**)(x1[i].x+0.5); len = 2\*a[n-1];

//减掉取两个相同的组合

**for**(**int** i = 0;i < n;i++) num[a[i]+a[i]]--;

**for**(**int** i = 1;i <= len;i++)num[i]/=2; sum[0] = 0;

**for**(**int** i = 1;i <= len;i++) sum[i] = sum[i-1]+num[i];

**long long** cnt = 0;

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

{

cnt += sum[len]-sum[a[i]];

//减掉一个取大，一个取小的

cnt -= (**long long**)(n-1-i)\*i;

//减掉一个取本身，另外一个取其它 cnt -= (n-1);

cnt -= (**long long**)(n-1-i)\*(n-i-2)/2;

}

**long long** tot = (**long long**)n\*(n-1)\*(n-2)/6; **printf**("%.7lf\n",(**double**)cnt/tot);

}

**return** 0;

}

## 10、高斯消元法求方程组的解

10.1 一类开关问题，对 2 取模的 01 方程组

POJ 1681 需要枚举自由变元，找解中 1 个数最少的

//对2取模的01方程组

**const int** MAXN = 300;

//有equ个方程，var个变元。增广矩阵行数为equ,列数为var+1,分别为0到var

**int** equ,var;

**int** a[MAXN][MAXN]; //增广矩阵

**int** x[MAXN]; //解集

**int** free\_x[MAXN];//用来存储自由变元（多解枚举自由变元可以使用）

**int** free\_num;//自由变元的个数

//返回值为-1表示无解，为0是唯一解，否则返回自由变元个数

int Gauss()

{

**int** max\_r,col,k; free\_num = 0;

**for**(k = 0, col = 0 ; k < equ && col < var ; k++, col++)

{

max\_r = k;

**for**(**int** i = k+1;i < equ;i++)

{

**if**(**abs**(a[i][col]) > **abs**(a[max\_r][col])) max\_r = i;

}

**if**(a[max\_r][col] == 0)

{

k--;

free\_x[free\_num++] = col;//这个是自由变元

continue;

}

**if**(max\_r != k)

{

**for**(**int** j = col; j < var+1; j++) swap(a[k][j],a[max\_r][j]);

}

**for**(**int** i = k+1;i < equ;i++)

{

**if**(a[i][col] != 0)

{

**for**(**int** j = col;j < var+1;j++) a[i][j] ^= a[k][j];

}

}

}

**for**(**int** i = k;i < equ;i++)

**if**(a[i][col] != 0)

**return** -1;//无解

**if**(k < var) **return** var-k;//自由变元个数

//唯一解，回代

**for**(**int** i = var-1; i >= 0;i--)

{

x[i] = a[i][var];

**for**(**int** j = i+1;j < var;j++) x[i] ^= (a[i][j] && x[j]);

}

**return** 0;

}

**int** n;

void init()

{

**memset**(a,0,**sizeof**(a));

**memset**(x,0,**sizeof**(x)); equ = n\*n;

var = n\*n;

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

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

{

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

**if**(i > 0)a[(i-1)\*n+j][t] = 1;

**if**(i < n-1)a[(i+1)\*n+j][t] = 1;

**if**(j > 0)a[i\*n+j-1][t] = 1;

**if**(j < n-1)a[i\*n+j+1][t] = 1;

}

}

void solve()

{

**int** t = Gauss();

**if**(t == -1)

{

**printf**("inf\n"); **return**;

}

**else if**(t == 0)

{

**int** ans = 0;

**for**(**int** i = 0;i < n\*n;i++) ans += x[i];

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

}

else

{

//枚举自由变元

**int** ans = 0x3f3f3f3f;

**int** tot = (1<<t);

**for**(**int** i = 0;i < tot;i++)

{

**int** cnt = 0;

**for**(**int** j = 0;j < t;j++)

{

**if**(i&(1<<j))

{

x[free\_x[j]] = 1; cnt++;

}

**else** x[free\_x[j]] = 0;

}

**for**(**int** j = var-t-1;j >= 0;j--)

{

**int** idx;

**for**(idx = j;idx < var;idx++)

**if**(a[j][idx]) **break**;

x[idx] = a[j][var];

**for**(**int** l = idx+1;l < var;l++)

**if**(a[j][l])

x[idx] ^= x[l];

cnt += x[idx];

}

ans = min(ans,cnt);

}

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

}

}

**char** str[30][30];

int main()

{

**int** T; **scanf**("%d",&T); **while**(T--)

{

**scanf**("%d",&n); init();

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

{

**scanf**("%s",str[i]); **for**(**int** j = 0;j < n;j++)

{

**if**(str[i][j] == 'y') a[i\*n+j][n\*n] = 0;

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

}

}

solve();

}

**return** 0;

}

10.2 解同余方程组

POJ 2947 **Widget Factory**

//求解对MOD取模的方程组 **const int** MOD = 7; **const int** MAXN = 400;

**int** a[MAXN][MAXN];//增广矩阵

**int** x[MAXN];//最后得到的解集

inline int gcd(int a,int b)

{

**while**(b != 0)

{

**int** t = b; b = a%b;

a = t;

}

**return** a;

}

inline int lcm(int a,int b)

{

**return** a/gcd(a,b)\*b;

}

long long inv(long long a,long long m)

{

**if**(a == 1)**return** 1;

**return** inv(m%a,m)\*(m-m/a)%m;

}

**int Gauss**(**int** equ,**int** var)

{

**int** max\_r,col,k;

**for**(k = 0, col = 0; k < equ && col < var; k++,col++)

{

max\_r = k;

**for**(**int** i = k+1; i < equ;i++)

**if**(**abs**(a[i][col]) > **abs**(a[max\_r][col])) max\_r = i;

**if**(a[max\_r][col] == 0)

{

k--;

continue;

}

**if**(max\_r != k)

**for**(**int** j = col; j < var+1;j++) swap(a[k][j],a[max\_r][j]);

**for**(**int** i = k+1;i < equ;i++)

{

**if**(a[i][col] != 0)

{

**int** LCM = lcm(**abs**(a[i][col]),**abs**(a[k][col]));

**int** ta = LCM/**abs**(a[i][col]);

**int** tb = LCM/**abs**(a[k][col]); **if**(a[i][col]\*a[k][col] < 0)tb = -tb; **for**(**int** j = col;j < var+1;j++)

a[i][j] = ((a[i][j]\*ta - a[k][j]\*tb)%MOD + MOD)%MOD;

}

}

}

**for**(**int** i = k;i < equ;i++)

**if**(a[i][col] != 0)

**return** -1;//无解

**if**(k < var) **return** var-k;//多解

**for**(**int** i = var-1;i >= 0;i--)

{

**int** temp = a[i][var];

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

{

**if**(a[i][j] != 0)

{

temp -= a[i][j]\*x[j];

temp = (temp%MOD + MOD)%MOD;

}

}

x[i] = (temp\*inv(a[i][i],MOD))%MOD;

}

**return** 0;

}

**int change**(**char** s[])

{

**if**(**strcmp**(s,"MON") == 0) **return** 1; **else if**(**strcmp**(s,"TUE")==0) **return** 2; **else if**(**strcmp**(s,"WED")==0) **return** 3; **else if**(**strcmp**(s,"THU")==0) **return** 4; **else if**(**strcmp**(s,"FRI")==0) **return** 5; **else if**(**strcmp**(s,"SAT")==0) **return** 6;

else return 7;

}

int main()

{

**int** n,m;

**while**(**scanf**("%d%d",&n,&m) == 2)

{

**if**(n == 0 && m == 0)**break**; **memset**(a,0,**sizeof**(a)); **char** str1[10],str2[10]; **int** k;

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

{

**scanf**("%d%s%s",&k,str1,str2);

a[i][n] = ((change(str2) - change(str1) + 1)%MOD + MOD)%MOD;

**int** t;

**while**(k--)

{

**scanf**("%d",&t); t--;

a[i][t] ++;

a[i][t]%=MOD;

}

}

**int** ans = Gauss(m,n);

**if**(ans == 0)

{

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

**if**(x[i] <= 2)

x[i] += 7;

**for**(**int** i = 0;i < n-1;i++)**printf**("%d ",x[i]);

**printf**("%d\n",x[n-1]);

}

**else if**(ans == -1)**printf**("Inconsistent data.\n");

**else printf**("Multiple solutions.\n");

}

**return** 0;

}

## 11、 整数拆分

HDU 4651

**const int** MOD = 1e9+7;

**int** dp[100010];

void init()

{

**memset**(dp,0,**sizeof**(dp)); dp[0] = 1;

**for**(**int** i = 1;i <= 100000;i++)

{

**for**(**int** j = 1, r = 1; i - (3 \* j \* j - j) / 2 >= 0; j++, r \*= -1)

{

dp[i] += dp[i -(3 \* j \* j - j) / 2] \* r; dp[i] %= MOD;

dp[i] = (dp[i]+MOD)%MOD;

**if**( i - (3 \* j \* j + j) / 2 >= 0 )

{

dp[i] += dp[i - (3 \* j \* j + j) / 2] \* r; dp[i] %= MOD;

dp[i] = (dp[i]+MOD)%MOD;

}

}

}

}

int main()

{

**int** T; **int** n; init();

**scanf**("%d",&T); **while**(T--)

{

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

**printf**("%d\n",dp[n]);

}

**return** 0;

}

HDU 4658

数 n(<=10^5)的划分，相同的数重复不能超过 k 个。

**const int** MOD = 1e9+7;

**int** dp[100010];

void init()

{

**memset**(dp,0,**sizeof**(dp)); dp[0] = 1;

**for**(**int** i = 1;i <= 100000;i++)

{

**for**(**int** j = 1, r = 1; i - (3 \* j \* j - j) / 2 >= 0; j++, r \*= -1)

{

dp[i] += dp[i -(3 \* j \* j - j) / 2] \* r; dp[i] %= MOD;

dp[i] = (dp[i]+MOD)%MOD;

**if**( i - (3 \* j \* j + j) / 2 >= 0 )

{

dp[i] += dp[i - (3 \* j \* j + j) / 2] \* r; dp[i] %= MOD;

dp[i] = (dp[i]+MOD)%MOD;

}

}

}

}

**int solve**(**int** n,**int** k)

{

**int** ans = dp[n];

**for**(**int** j = 1, r = -1; n - k\*(3 \* j \* j - j) / 2 >= 0; j++, r \*= -1)

{

ans += dp[n -k\*(3 \* j \* j - j) / 2] \* r; ans %= MOD;

ans = (ans+MOD)%MOD;

**if**( n - k\*(3 \* j \* j + j) / 2 >= 0 )

{

ans += dp[n - k\*(3 \* j \* j + j) / 2] \* r;

ans %= MOD;

ans = (ans+MOD)%MOD;

}

}

**return** ans;

}

int main()

{

init(); **int** T; **int** n,k;

**scanf**("%d",&T); **while**(T--)

{

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

**printf**("%d\n",solve(n,k));

}

**return** 0;

}

## 12、求 A^B 的约数之和对 MOD 取模

参考 POJ 1845

里面有一种求 1+p+p^2+p^3+…p^n 的方法。 需要素数筛选和合数分解的程序，需要先调用 getPrime(); **long long pow\_m**(**long long** a,**long long** n)

{

**long long** ret = 1; **long long** tmp = a%MOD; **while**(n)

{

**if**(n&1)ret = (ret\*tmp)%MOD; tmp = tmp\*tmp%MOD;

n >>= 1;

}

**return** ret;

}

//计算1+p+p^2+...+p^n

long long sum(long long p,long long n)

{

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

{

**return** ((1+pow\_m(p,n/2+1))%MOD\*sum(p,n/2)%MOD)%MOD;

}

**else return** ((1+pow\_m(p,n/2+1))%MOD\*sum(p,n/2-1)+pow\_m(p,n/2)%MOD)%MOD;

}

//返回A^B的约数之和 % MOD

long long solve(long long A,long long B)

{

getFactors(A);

**long long** ans = 1;

**for**(**int** i = 0;i < fatCnt;i++)

{

ans \*= sum(factor[i][0],B\*factor[i][1])%MOD; ans %= MOD;

}

**return** ans;

}

## 13、莫比乌斯反演

莫比乌斯反演公式：

 *n* 

*F*(*n*)  *d* |*n f*(*d* )

则 *f*(*n* )  *d* |*n **d* *F* *d* 

莫比乌斯函数 **

 1



 

*n*  1

***n*  

 1*k*



*n*  *p*1*p*2  *pk*

 0 其余情况

另外一种更常用的形式：

在某一范围内： *F*(*n*) 

*n*|*d f*(*d* )

则 *f*(*n* ) 

*d* 

*n*|*d*  *n* *F*

 ** *d* 

####  

线性筛法求解积性函数（莫比乌斯函数）

**const int** MAXN = 1000000;

**bool** check[MAXN+10]; **int** prime[MAXN+10]; **int** mu[MAXN+10]; **void Moblus**()

{

**memset**(check,**false**,**sizeof**(check)); mu[1] = 1;

**int** tot = 0;

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

{

**if**( !check[i] )

{

prime[tot++] = i; mu[i] = -1;

}

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

{

**if**(i \* prime[j] > MAXN) **break**; check[i \* prime[j]] = **true**; **if**( i % prime[j] == 0)

{

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

break;

}

else

{

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

}

}

}

}

例题：

BZOJ 2301

对于给出的 n 个询问，每次求有多少个数对(x,y)，满足 a≤x≤b，c≤y≤d，且 gcd(x,y) = k， gcd(x,y)函数为 x 和 y 的最大公约数。 1≤n≤50000，1≤a≤b≤50000，1≤c≤d≤50000，1≤k≤50000

**const int** MAXN = 100000;

**bool** check[MAXN+10]; **int** prime[MAXN+10]; **int** mu[MAXN+10]; **void Moblus**()

{

**memset**(check,**false**,**sizeof**(check)); mu[1] = 1;

**int** tot = 0;

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

{

**if**( !check[i] )

{

prime[tot++] = i; mu[i] = -1;

}

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

{

**if**( i \* prime[j] > MAXN) **break**; check[i \* prime[j]] = **true**; **if**( i % prime[j] == 0)

{

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

break;

}

else

{

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

}

}

}

}

**int** sum[MAXN+10];

//找[1,n],[1,m]内互质的数的对数

long long solve(int n,int m)

{

**long long** ans = 0;

**if**(n > m)swap(n,m);

**for**(**int** i = 1, la = 0; i <= n; i = la+1)

{

la = min(n/(n/i),m/(m/i));

ans += (**long long**)(sum[la] - sum[i-1])\*(n/i)\*(m/i);

}

**return** ans;

}

int main()

{

Moblus();

sum[0] = 0;

**for**(**int** i = 1;i <= MAXN;i++) sum[i] = sum[i-1] + mu[i];

**int** a,b,c,d,k; **int** T; **scanf**("%d",&T); **while**(T--)

{

**scanf**("%d%d%d%d%d",&a,&b,&c,&d,&k);

**long long** ans = solve(b/k,d/k) - solve((a-1)/k,d/k) - solve(b/k,(c-1)/k)

+ solve((a-1)/k,(c-1)/k);

**printf**("%lld\n",ans);

}

**return** 0;

}

## 14、Baby-Step Giant-Step

(POJ 2417,3243)

//baby\_step giant\_step

// a^x = b (mod n) n是素数和不是素数都可以

// 求解上式 0<=x < n的解

**#define** MOD 76543

**int** hs[MOD],head[MOD],next[MOD],id[MOD],top;

**void insert**(**int** x,**int** y)

{

**int** k = x%MOD;

hs[top] = x, id[top] = y, next[top] = head[k], head[k] = top++;

}

int find(int x)

{

**int** k = x%MOD;

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

**if**(hs[i] == x)

**return** id[i];

**return** -1;

}

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

{

**memset**(head,-1,**sizeof**(head)); top = 1;

**if**(b == 1)**return** 0; **int** m = **sqrt**(n\*1.0), j; **long long** x = 1, p = 1;

**for**(**int** i = 0; i < m; ++i, p = p\*a%n)insert(p\*b%n,i);

**for**(**long long** i = m; ;i += m)

{

**if**( (j = find(x = x\*p%n)) != -1 )**return** i-j;

**if**(i > n)**break**;

}

**return** -1;

}

## 15、自适应 simpson 积分

double simpson(double a,double b)

{

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

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

}

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

{

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

**double** L = simpson(a,c), R = simpson(c,b);

**if**(**fabs**(L + R - A) <= 15\*eps)**return** L + R + (L + R - A)/15.0;

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

}

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

{

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

}

## 相关公式

1、欧拉定理

对于互质的整数 a 和 n，有 *a*(*n* )

#####  （1

##### mod *n*）

费马定理：a 是不能被质数 p 整除的正整数，有 *ap* 1

2、Polya 定理

 1(mod *p* )

设 G 是 p 个对象的一个置换群，用 k 种颜色去染这 p 个对象，若一种染色方案在群 G 的作 用下变为一种方案，则这两个方案当作是同一种方案，这样的不同染色方案数为：

*L*  1

*G*

  *k C* (*f* ),*f*  *G*

*C*(*f* )为循环节， *G* 表示群的置换方法数 。

对于 n 个位置的手镯，有 n 种旋转置换和 n 种翻转置换 对于旋转置换：

*C*(*fi* ) 

gcd(*n*,*i*),i 表示旋转 i 颗宝石以后。i=0 时 gcd(n,0)=n

对于翻转置换：

如果 n 为偶数：则有 n/2 个置换*C*(*f* ) 

### n n

,有 n/2 个置换*C*(*f* )   1

#### 2 2

如果 n 为奇数：则有 n 个置换*C*(*f* )  *n*  1

#### 2

3、欧拉函数***n* 

***n* 

积 性 函 数 ， 对 于 一 个 质 数 *p* 和 正 整 数 *k* ， 有

***pk*   *p k*

* *p k* 1

 (*p*

 1)*p k* 1

 *p k*(1  1 )

### p

*d* |*n*

***d*   *n*

当 *n*  1时，1  *n* 中与 *n* 互质的整数和为 *n* *n* 

#### 2

# 数据结构

## 1、划分树

/\*

\* 划分树（查询区间第k大）

\*/

**const int** MAXN = 100010;

**int** tree[20][MAXN];//表示每层每个位置的值

**int** sorted[MAXN];//已经排序好的数

**int** toleft[20][MAXN];//toleft[p][i]表示第i层从1到i有数分入左边

**void build**(**int** l,**int** r,**int** dep)

{

**if**(l == r)**return**;

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

**int** same = mid - l + 1;//表示等于中间值而且被分入左边的个数

**for**(**int** i = l; i <= r; i++) //注意是l,不是one

**if**(tree[dep][i] < sorted[mid]) same--;

**int** lpos = l;

**int** rpos = mid+1;

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

{

**if**(tree[dep][i] < sorted[mid]) tree[dep+1][lpos++] = tree[dep][i];

**else if**(tree[dep][i] == sorted[mid] && same > 0)

{

tree[dep+1][lpos++] = tree[dep][i]; same--;

}

else

tree[dep+1][rpos++] = tree[dep][i]; toleft[dep][i] = toleft[dep][l-1] + lpos - l;

}

build(l,mid,dep+1); build(mid+1,r,dep+1);

}

//查询区间第k大的数,[L,R]是大区间，[l,r]是要查询的小区间

**int query**(**int** L,**int** R,**int** l,**int** r,**int** dep,**int** k)

{

**if**(l == r)**return** tree[dep][l];

**int** mid = (L+R)>>1;

**int** cnt = toleft[dep][r] - toleft[dep][l-1];

**if**(cnt >= k)

{

**int** newl = L + toleft[dep][l-1] - toleft[dep][L-1];

**int** newr = newl + cnt - 1;

**return** query(L,mid,newl,newr,dep+1,k);

}

else

{

**int** newr = r + toleft[dep][R] - toleft[dep][r];

**int** newl = newr - (r-l-cnt);

**return** query(mid+1,R,newl,newr,dep+1,k-cnt);

}

}

int main()

{

**int** n,m;

**while**(**scanf**("%d%d",&n,&m)==2)

{

**memset**(tree,0,**sizeof**(tree)); **for**(**int** i = 1;i <= n;i++)

{

**scanf**("%d",&tree[0][i]); sorted[i] = tree[0][i];

}

sort(sorted+1,sorted+n+1); build(1,n,0);

**int** s,t,k;

**while**(m--)

{

**scanf**("%d%d%d",&s,&t,&k); **printf**("%d\n",query(1,n,s,t,0,k));

}

}

**return** 0;

}

## 2、RMQ

2.1 一维

求最大值，数组下标从 1 开始。

求最小值，或者最大最小值下标，或者数组从 0 开始对应修改即可。

**const int** MAXN = 50010;

**int** dp[MAXN][20];

**int** mm[MAXN];

//初始化RMQ, b数组下标从1开始，从0开始简单修改

**void initRMQ**(**int** n,**int** b[])

{

mm[0] = -1;

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

{

mm[i] = ((i&(i-1)) == 0)?mm[i-1]+1:mm[i-1]; dp[i][0] = b[i];

}

**for**(**int** j = 1; j <= mm[n];j++)

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

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

}

//查询最大值

**int rmq**(**int** x,**int** y)

{

**int** k = mm[y-x+1];

**return** max(dp[x][k],dp[y-(1<<k)+1][k]);

}

2.2 二维

/\*

\* 二维RMQ，预处理复杂度 n\*m\*log\*(n)\*log(m)

\* 数组下标从1开始

\*/

**int** val[310][310];

**int** dp[310][310][9][9];//最大值

**int** mm[310];//二进制位数减一，使用前初始化

void initRMQ(int n,int m)

{

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

**for**(**int** j = 1;j <= m;j++) dp[i][j][0][0] = val[i][j];

**for**(**int** ii = 0; ii <= mm[n]; ii++)

**for**(**int** jj = 0; jj <= mm[m]; jj++)

**if**(ii+jj)

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

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

{

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

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

}

}

//查询矩形内的最大值(x1<=x2,y1<=y2)

**int rmq**(**int** x1,**int** y1,**int** x2,**int** y2)

{

**int** k1 = mm[x2-x1+1]; **int** k2 = mm[y2-y1+1]; x2 = x2 - (1<<k1) + 1; y2 = y2 - (1<<k2) + 1;

**return** max(max(dp[x1][y1][k1][k2],dp[x1][y2][k1][k2]),max(dp[x2][y1][k1][k2],dp[x2] [y2][k1][k2]));

}

int main()

{

//在外面对mm数组进行初始化

mm[0] = -1;

**for**(**int** i = 1;i <= 305;i++)

mm[i] = ((i&(i-1))==0)?mm[i-1]+1:mm[i-1];

**int** n,m;

**int** Q;

**int** r1,c1,r2,c2;

**while**(**scanf**("%d%d",&n,&m) == 2)

{

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

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

**scanf**("%d",&val[i][j]); initRMQ(n,m);

**scanf**("%d",&Q); **while**(Q--)

{

**scanf**("%d%d%d%d",&r1,&c1,&r2,&c2);

**if**(r1 > r2)swap(r1,r2);

**if**(c1 > c2)swap(c1,c2);

**int** tmp = rmq(r1,c1,r2,c2);

**printf**("%d ",tmp);

**if**(tmp == val[r1][c1] || tmp == val[r1][c2] || tmp == val[r2][c1] || tmp == val[r2][c2])

**printf**("yes\n"); **else printf**("no\n");

}

}

**return** 0;

}

## 3、树链剖分

3.1 点权

基于点权，查询单点值，修改路径的上的点权（HDU 3966 树链剖分+树状数组）

**const int** MAXN = 50010;

**struct** Edge

{

**int** to,next;

}edge[MAXN\*2];

**int** head[MAXN],tot;

**int** top[MAXN];//top[v] 表示v所在的重链的顶端节点

**int** fa[MAXN];//父亲节点

**int** deep[MAXN];//深度

**int** num[MAXN];//num[v] 表示以v为根的子树的节点数

**int** p[MAXN];//p[v]表示v对应的位置

**int** fp[MAXN];//fp和p数组相反

**int** son[MAXN];//重儿子

**int** pos;

void init()

{

tot = 0;

**memset**(head,-1,**sizeof**(head)); pos = 1;//使用树状数组，编号从头1开始

**memset**(son,-1,**sizeof**(son));

}

void addedge(int u,int v)

{

edge[tot].to = v; edge[tot].next = head[u]; head[u] = tot++;

}

**void dfs1**(**int** u,**int** pre,**int** d)

{

deep[u] = d; fa[u] = pre; num[u] = 1;

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

{

**int** v = edge[i].to;

**if**(v != pre)

{

dfs1(v,u,d+1);

num[u] += num[v];

**if**(son[u] == -1 || num[v] > num[son[u]])

son[u] = v;

}

}

}

**void getpos**(**int** u,**int** sp)

{

top[u] = sp; p[u] = pos++; fp[p[u]] = u;

**if**(son[u] == -1) **return**; getpos(son[u],sp);

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

{

**int** v = edge[i].to;

**if**( v != son[u] && v != fa[u]) getpos(v,v);

}

}

//树状数组

int lowbit(int x)

{

**return** x&(-x);

}

**int** c[MAXN];

**int** n;

**int sum**(**int** i)

{

**int** s = 0;

**while**(i > 0)

{

s += c[i];

i -= lowbit(i);

}

**return** s;

}

**void add**(**int** i,**int** val)

{

**while**(i <= n)

{

c[i] += val;

i += lowbit(i);

}

}

**void Change**(**int** u,**int** v,**int** val)//u->v的路径上点的值改变val

{

**int** f1 = top[u], f2 = top[v];

**int** tmp = 0;

**while**(f1 != f2)

{

**if**(deep[f1] < deep[f2])

{

swap(f1,f2);

swap(u,v);

}

add(p[f1],val);

add(p[u]+1,-val); u = fa[f1];

f1 = top[u];

}

**if**(deep[u] > deep[v]) swap(u,v); add(p[u],val);

add(p[v]+1,-val);

}

**int** a[MAXN];

int main()

{

**int** M,P;

**while**(**scanf**("%d%d%d",&n,&M,&P) == 3)

{

**int** u,v;

**int** C1,C2,K;

**char** op[10]; init();

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

{

**scanf**("%d",&a[i]);

}

**while**(M--)

{

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

addedge(v,u);

}

dfs1(1,0,0);

getpos(1,1); **memset**(c,0,**sizeof**(c)); **for**(**int** i = 1;i <= n;i++)

{

add(p[i],a[i]);

add(p[i]+1,-a[i]);

}

**while**(P--)

{

**scanf**("%s",op);

**if**(op[0] == 'Q')

{

**scanf**("%d",&u); **printf**("%d\n",sum(p[u]));

}

else

{

**scanf**("%d%d%d",&C1,&C2,&K); **if**(op[0] == 'D')

K = -K;

Change(C1,C2,K);

}

}

}

**return** 0;

}

3.2 边权

基于边权，修改单条边权，查询路径边权最大值（SPOJ QTREE 树链剖分+线段树 ）

**const int** MAXN = 10010;

**struct** Edge

{

**int** to,next;

}edge[MAXN\*2];

**int** head[MAXN],tot;

**int** top[MAXN];//top[v]表示v所在的重链的顶端节点

**int** fa[MAXN]; //父亲节点

**int** deep[MAXN];//深度

**int** num[MAXN];//num[v]表示以v为根的子树的节点数

**int** p[MAXN];//p[v]表示v与其父亲节点的连边在线段树中的位置

**int** fp[MAXN];//和p数组相反 **int** son[MAXN];//重儿子 **int** pos;

void init()

{

tot = 0;

**memset**(head,-1,**sizeof**(head)); pos = 0;

**memset**(son,-1,**sizeof**(son));

}

void addedge(int u,int v)

{

edge[tot].to = v;edge[tot].next = head[u];head[u] = tot++;

}

**void dfs1**(**int** u,**int** pre,**int** d) //第一遍dfs求出fa,deep,num,son

{

deep[u] = d; fa[u] = pre; num[u] = 1;

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

{

**int** v = edge[i].to;

**if**(v != pre)

{

dfs1(v,u,d+1);

num[u] += num[v];

**if**(son[u] == -1 || num[v] > num[son[u]]) son[u] = v;

}

}

}

**void getpos**(**int** u,**int** sp) //第二遍dfs求出top和p

{

top[u] = sp; p[u] = pos++; fp[p[u]] = u;

**if**(son[u] == -1) **return**; getpos(son[u],sp);

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

{

**int** v = edge[i].to;

**if**(v != son[u] && v != fa[u]) getpos(v,v);

}

}

//线段树

**struct** Node

{

**int** l,r;

**int** Max;

}segTree[MAXN\*3];

**void build**(**int** i,**int** l,**int** r)

{

segTree[i].l = l; segTree[i].r = r; segTree[i].Max = 0; **if**(l == r)**return**; **int** mid = (l+r)/2; build(i<<1,l,mid);

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

}

void push\_up(int i)

{

segTree[i].Max = max(segTree[i<<1].Max,segTree[(i<<1)|1].Max);

}

**void update**(**int** i,**int** k,**int** val) // 更新线段树的第k个值为val

{

**if**(segTree[i].l == k && segTree[i].r == k)

{

segTree[i].Max = val;

return;

}

**int** mid = (segTree[i].l + segTree[i].r)/2;

**if**(k <= mid)update(i<<1,k,val); **else** update((i<<1)|1,k,val); push\_up(i);

}

**int query**(**int** i,**int** l,**int** r) //查询线段树中[l,r] 的最大值

{

**if**(segTree[i].l == l && segTree[i].r == r)

**return** segTree[i].Max;

**int** mid = (segTree[i].l + segTree[i].r)/2;

**if**(r <= mid)**return** query(i<<1,l,r);

**else if**(l > mid)**return** query((i<<1)|1,l,r);

**else return** max(query(i<<1,l,mid),query((i<<1)|1,mid+1,r));

}

**int find**(**int** u,**int** v)//查询u->v边的最大值

{

**int** f1 = top[u], f2 = top[v];

**int** tmp = 0;

**while**(f1 != f2)

{

**if**(deep[f1] < deep[f2])

{

swap(f1,f2);

swap(u,v);

}

tmp = max(tmp,query(1,p[f1],p[u])); u = fa[f1]; f1 = top[u];

}

**if**(u == v)**return** tmp;

**if**(deep[u] > deep[v]) swap(u,v);

**return** max(tmp,query(1,p[son[u]],p[v]));

}

**int** e[MAXN][3];

int main()

{

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

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

**int** T;

**int** n; **scanf**("%d",&T); **while**(T--)

{

init();

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

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

{

**scanf**("%d%d%d",&e[i][0],&e[i][1],&e[i][2]); addedge(e[i][0],e[i][1]);

addedge(e[i][1],e[i][0]);

}

dfs1(1,0,0);

getpos(1,1); build(1,0,pos-1);

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

{

**if**(deep[e[i][0]] > deep[e[i][1]])

swap(e[i][0],e[i][1]);

update(1,p[e[i][1]],e[i][2]);

}

**char** op[10];

**int** u,v;

**while**(**scanf**("%s",op) == 1)

{

**if**(op[0] == 'D')**break**;

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

**if**(op[0] == 'Q')

**printf**("%d\n",find(u,v));//查询u->v路径上边权的最大值 **else** update(1,p[e[u-1][1]],v);//修改第u条边的长度为v

}

}

**return** 0;

}

## 4、伸展树（splay tree）

题目：维修数列。 经典题，插入、删除、修改、翻转、求和、求和最大的子序列

**#define** Key\_value ch[ch[root][1]][0]

**const int** MAXN = 500010;

**const int** INF = 0x3f3f3f3f;

**int** pre[MAXN],ch[MAXN][2],key[MAXN],size[MAXN];

**int** root,tot1;

**int** sum[MAXN],rev[MAXN],same[MAXN];

**int** lx[MAXN],rx[MAXN],mx[MAXN];

**int** s[MAXN],tot2;//内存池和容量

**int** a[MAXN];

**int** n,q;

//debug部分\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void Treavel(int x)

{

**if**(x)

{

Treavel(ch[x][0]);

**printf**("结点：%2d: 左儿子 %2d 右儿子 %2d 父结点 %2d size

= %2d\n",x,ch[x][0],ch[x][1],pre[x],size[x]); Treavel(ch[x][1]);

}

}

void debug()

{

**printf**("root:%d\n",root); Treavel(root);

}

//以上是debug部分\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**void NewNode**(**int** &r,**int** father,**int** k)

{

**if**(tot2) r = s[tot2--];//取的时候是tot2--,存的时候就是++tot2

**else** r = ++tot1; pre[r] = father;

ch[r][0] = ch[r][1] = 0; key[r] = k;

sum[r] = k;

rev[r] = same[r] = 0; lx[r] = rx[r] = mx[r] = k; size[r] = 1;

}

void Update\_Rev(int r)

{

**if**(!r)**return**; swap(ch[r][0],ch[r][1]); swap(lx[r],rx[r]); rev[r] ^= 1;

}

void Update\_Same(int r,int v)

{

**if**(!r)**return**;

key[r] = v;

sum[r] = v\*size[r];

lx[r] = rx[r] = mx[r] = max(v,v\*size[r]); same[r] = 1;

}

void push\_up(int r)

{

**int** lson = ch[r][0], rson = ch[r][1]; size[r] = size[lson] + size[rson] + 1; sum[r] = sum[lson] + sum[rson] + key[r];

lx[r] = max(lx[lson],sum[lson] + key[r] + max(0,lx[rson]));

rx[r] = max(rx[rson],sum[rson] + key[r] + max(0,rx[lson]));

mx[r] = max(0,rx[lson]) + key[r] + max(0,lx[rson]);

mx[r] = max(mx[r],max(mx[lson],mx[rson]));

}

void push\_down(int r)

{

**if**(same[r])

{

Update\_Same(ch[r][0],key[r]);

Update\_Same(ch[r][1],key[r]); same[r] = 0;

}

**if**(rev[r])

{

Update\_Rev(ch[r][0]);

Update\_Rev(ch[r][1]); rev[r] = 0;

}

}

**void Build**(**int** &x,**int** l,**int** r,**int** father)

{

**if**(l > r)**return**;

**int** mid = (l+r)/2; NewNode(x,father,a[mid]); Build(ch[x][0],l,mid-1,x);

Build(ch[x][1],mid+1,r,x); push\_up(x);

}

void Init()

{

root = tot1 = tot2 = 0;

ch[root][0] = ch[root][1] = size[root] = pre[root] = 0; same[root] = rev[root] = sum[root] = key[root] = 0; lx[root] = rx[root] = mx[root] = -INF;

NewNode(root,0,-1); NewNode(ch[root][1],root,-1); **for**(**int** i = 0;i < n;i++)

**scanf**("%d",&a[i]);

Build(Key\_value,0,n-1,ch[root][1]); push\_up(ch[root][1]); push\_up(root);

}

//旋转,0为左旋，1为右旋

**void Rotate**(**int** x,**int** kind)

{

**int** y = pre[x]; push\_down(y); push\_down(x);

ch[y][!kind] = ch[x][kind]; pre[ch[x][kind]] = y; **if**(pre[y])

ch[pre[y]][ch[pre[y]][1]==y] = x; pre[x] = pre[y];

ch[x][kind] = y; pre[y] = x; push\_up(y);

}

//Splay调整，将r结点调整到goal下面

**void Splay**(**int** r,**int** goal)

{

push\_down(r);

**while**(pre[r] != goal)

{

**if**(pre[pre[r]] == goal)

{

push\_down(pre[r]); push\_down(r); Rotate(r,ch[pre[r]][0] == r);

}

else

{

push\_down(pre[pre[r]]); push\_down(pre[r]); push\_down(r);

**int** y = pre[r];

**int** kind = ch[pre[y]][0]==y;

**if**(ch[y][kind] == r)

{

Rotate(r,!kind); Rotate(r,kind);

}

else

{

Rotate(y,kind); Rotate(r,kind);

}

}

}

push\_up(r);

**if**(goal == 0) root = r;

}

**int Get\_kth**(**int** r,**int** k)

{

push\_down(r);

**int** t = size[ch[r][0]] + 1;

**if**(t == k)**return** r;

**if**(t > k)**return** Get\_kth(ch[r][0],k);

**else return** Get\_kth(ch[r][1],k-t);

}

//在第pos个数后面插入tot个数

**void Insert**(**int** pos,**int** tot)

{

**for**(**int** i = 0;i < tot;i++)**scanf**("%d",&a[i]); Splay(Get\_kth(root,pos+1),0); Splay(Get\_kth(root,pos+2),root); Build(Key\_value,0,tot-1,ch[root][1]); push\_up(ch[root][1]);

push\_up(root);

}

//删除子树

void erase(int r)

{

**if**(!r)**return**; s[++tot2] = r; erase(ch[r][0]);

erase(ch[r][1]);

}

//从第pos个数开始连续删除tot个数

**void Delete**(**int** pos,**int** tot)

{

Splay(Get\_kth(root,pos),0); Splay(Get\_kth(root,pos+tot+1),root); erase(Key\_value);

pre[Key\_value] = 0;

Key\_value = 0; push\_up(ch[root][1]); push\_up(root);

}

//将从第pos个数开始的连续的tot个数修改为c

**void Make\_Same**(**int** pos,**int** tot,**int** c)

{

Splay(Get\_kth(root,pos),0); Splay(Get\_kth(root,pos+tot+1),root); Update\_Same(Key\_value,c); push\_up(ch[root][1]);

push\_up(root);

}

//将第pos个数开始的连续tot个数进行反转

**void Reverse**(**int** pos,**int** tot)

{

Splay(Get\_kth(root,pos),0); Splay(Get\_kth(root,pos+tot+1),root); Update\_Rev(Key\_value); push\_up(ch[root][1]);

push\_up(root);

}

//得到第pos个数开始的tot个数的和

**int Get\_Sum**(**int** pos,**int** tot)

{

Splay(Get\_kth(root,pos),0); Splay(Get\_kth(root,pos+tot+1),root); **return** sum[Key\_value];

}

//得到第pos个数开始的tot个数中最大的子段和

**int Get\_MaxSum**(**int** pos,**int** tot)

{

Splay(Get\_kth(root,pos),0); Splay(Get\_kth(root,pos+tot+1),root); **return** mx[Key\_value];

}

void InOrder(int r)

{

**if**(!r)**return**; push\_down(r); InOrder(ch[r][0]);

**printf**("%d ",key[r]);

InOrder(ch[r][1]);

}

int main()

{

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

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

**while**(**scanf**("%d%d",&n,&q) == 2)

{

Init();

**char** op[20];

**int** x,y,z;

**while**(q--)

{

**scanf**("%s",op); **if**(**strcmp**(op,"INSERT") == 0)

{

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

}

**else if**(**strcmp**(op,"DELETE") == 0)

{

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

}

**else if**(**strcmp**(op,"MAKE-SAME") == 0)

{

**scanf**("%d%d%d",&x,&y,&z); Make\_Same(x,y,z);

}

**else if**(**strcmp**(op,"REVERSE") == 0)

{

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

}

**else if**(**strcmp**(op,"GET-SUM") == 0)

{

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

**printf**("%d\n",Get\_Sum(x,y));

}

**else if**(**strcmp**(op,"MAX-SUM") == 0)

**printf**("%d\n",Get\_MaxSum(1,size[root]-2));

}

}

**return** 0;

}

## 5、动态树

5.1 HDU 4010(切割、合并子树，路径上所有点的点权增加一个值，查询路径上点权的最 大值)

//动态维护一组森林，要求支持一下操作:

//link(a,b) : 如果a,b不在同一颗子树中，则通过在a,b之间连边的方式，连接这两颗子树

//cut(a,b) : 如果a,b在同一颗子树中，且a!=b,则将a视为这颗子树的根以后，切断b与其父亲结点 的连接

//ADD(a,b,w): 如果a,b在同一颗子树中，则将a,b之间路径上所有点的点权增加w

//query(a,b): 如果a,b在同一颗子树中，返回a,b之间路径上点权的最大值

**const int** MAXN = 300010;

**int** ch[MAXN][2],pre[MAXN],key[MAXN]; **int** add[MAXN],rev[MAXN],Max[MAXN]; **bool** rt[MAXN];

void Update\_Add(int r,int d)

{

**if**(!r)**return**; key[r] += d;

add[r] += d; Max[r] += d;

}

void Update\_Rev(int r)

{

**if**(!r)**return**; swap(ch[r][0],ch[r][1]); rev[r] ^= 1;

}

void push\_down(int r)

{

**if**(add[r])

{

Update\_Add(ch[r][0],add[r]);

Update\_Add(ch[r][1],add[r]); add[r] = 0;

}

**if**(rev[r])

{

Update\_Rev(ch[r][0]);

Update\_Rev(ch[r][1]); rev[r] = 0;

}

}

void push\_up(int r)

{

Max[r] = max(max(Max[ch[r][0]],Max[ch[r][1]]),key[r]);

}

void Rotate(int x)

{

**int** y = pre[x], kind = ch[y][1]==x; ch[y][kind] = ch[x][!kind]; pre[ch[y][kind]] = y;

pre[x] = pre[y]; pre[y] = x; ch[x][!kind] = y; **if**(rt[y])

rt[y] = **false**, rt[x] = **true**; **else**

ch[pre[x]][ch[pre[x]][1]==y] = x; push\_up(y);

}

//P函数先将根结点到r的路径上所有的结点的标记逐级下放

**void P**(**int** r)

{

**if**(!rt[r])P(pre[r]); push\_down(r);

}

void Splay(int r)

{

P(r);

**while**( !rt[r] )

{

**int** f = pre[r], ff = pre[f];

**if**(rt[f])

Rotate(r);

**else if**( (ch[ff][1]==f)==(ch[f][1]==r) ) Rotate(f), Rotate(r);

else

Rotate(r), Rotate(r);

}

push\_up(r);

}

int Access(int x)

{

**int** y = 0;

**for**( ; x ; x = pre[y=x])

{

Splay(x);

rt[ch[x][1]] = **true**, rt[ch[x][1]=y] = **false**; push\_up(x);

}

**return** y;

}

//判断是否是同根(真实的树，非splay)

**bool judge**(**int** u,**int** v)

{

**while**(pre[u]) u = pre[u];

**while**(pre[v]) v = pre[v];

**return** u == v;

}

//使r成为它所在的树的根

void mroot(int r)

{

Access(r); Splay(r); Update\_Rev(r);

}

//调用后u是原来u和v的lca,v和ch[u][1]分别存着lca的2个儿子

//(原来u和v所在的2颗子树)

**void lca**(**int** &u,**int** &v)

{

Access(v), v = 0;

**while**(u)

{

Splay(u);

**if**(!pre[u])**return**;

rt[ch[u][1]] = **true**;

rt[ch[u][1]=v] = **false**; push\_up(u);

u = pre[v = u];

}

}

**void link**(**int** u,**int** v)

{

**if**(judge(u,v))

{

**puts**("-1"); **return**;

}

mroot(u);

pre[u] = v;

}

//使u成为u所在树的根，并且v和它父亲的边断开

**void cut**(**int** u,**int** v)

{

**if**(u == v || !judge(u,v))

{

**puts**("-1"); **return**;

}

mroot(u);

Splay(v);

pre[ch[v][0]] = pre[v]; pre[v] = 0; rt[ch[v][0]] = **true**;

ch[v][0] = 0;

push\_up(v);

}

**void ADD**(**int** u,**int** v,**int** w)

{

**if**(!judge(u,v))

{

**puts**("-1"); **return**;

}

lca(u,v); Update\_Add(ch[u][1],w); Update\_Add(v,w);

key[u] += w; push\_up(u);

}

**void query**(**int** u,**int** v)

{

**if**(!judge(u,v))

{

**puts**("-1"); **return**;

}

lca(u,v);

**printf**("%d\n",max(max(Max[v],Max[ch[u][1]]),key[u]));

}

**struct** Edge

{

**int** to,next;

}edge[MAXN\*2];

**int** head[MAXN],tot;

void addedge(int u,int v)

{

edge[tot].to = v; edge[tot].next = head[u]; head[u] = tot++;

}

void dfs(int u)

{

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

{

**int** v = edge[i].to; **if**(pre[v] != 0)**continue**; pre[v] = u;

dfs(v);

}

}

int main()

{

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

//freopen("out.txt","w",stdout); **int** n,q,u,v; **while**(**scanf**("%d",&n) == 1)

{

tot = 0;

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

{

head[i] = -1;

pre[i] = 0;

ch[i][0] = ch[i][1] = 0; rev[i] = 0;

add[i] = 0; rt[i] = **true**;

}

Max[0] = -2000000000;

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

{

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

addedge(v,u);

}

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

{

**scanf**("%d",&key[i]); Max[i] = key[i];

}

**scanf**("%d",&q); pre[1] = -1;

dfs(1);

pre[1] = 0; **int** op; **while**(q--)

{

**scanf**("%d",&op); **if**(op == 1)

{

**int** x,y; **scanf**("%d%d",&x,&y); link(x,y);

}

**else if**(op == 2)

{

**int** x,y; **scanf**("%d%d",&x,&y); cut(x,y);

}

**else if**(op == 3)

{

**int** w,x,y; **scanf**("%d%d%d",&w,&x,&y); ADD(x,y,w);

}

else

{

**int** x,y; **scanf**("%d%d",&x,&y); query(x,y);

}

}

**printf**("\n");

}

**return** 0;

}

## 6、主席树

6.1 查询区间有多少个不同的数（SPOJ DQUERY）

/\*

\* 给出一个序列，查询区间内有多少个不相同的数

\*/

**const int** MAXN = 30010; **const int** M = MAXN \* 100; **int** n,q,tot;

**int** a[MAXN];

**int** T[MAXN],lson[M],rson[M],c[M];

**int build**(**int** l,**int** r)

{

**int** root = tot++; c[root] = 0;

**if**(l != r)

{

**int** mid = (l+r)>>1; lson[root] = build(l,mid); rson[root] = build(mid+1,r);

}

**return** root;

}

**int update**(**int** root,**int** pos,**int** val)

{

**int** newroot = tot++, tmp = newroot; c[newroot] = c[root] + val;

**int** l = 1, r = n;

**while**(l < r)

{

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

**if**(pos <= mid)

{

lson[newroot] = tot++; rson[newroot] = rson[root]; newroot = lson[newroot]; root = lson[root];

r = mid;

}

else

{

rson[newroot] = tot++; lson[newroot] = lson[root]; newroot = rson[newroot]; root = rson[root];

l = mid+1;

}

c[newroot] = c[root] + val;

}

**return** tmp;

}

**int query**(**int** root,**int** pos)

{

**int** ret = 0;

**int** l = 1, r = n;

**while**(pos < r)

{

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

**if**(pos <= mid)

{

r = mid;

root = lson[root];

}

else

{

ret += c[lson[root]]; root = rson[root];

l = mid+1;

}

}

**return** ret + c[root];

}

int main()

{

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

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

**while**(**scanf**("%d",&n) == 1)

{

tot = 0;

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

**scanf**("%d",&a[i]);

T[n+1] = build(1,n); map<**int**,**int**>mp;

**for**(**int** i = n;i>= 1;i--)

{

**if**(mp.find(a[i]) == mp.end())

{

T[i] = update(T[i+1],i,1);

}

else

{

**int** tmp = update(T[i+1],mp[a[i]],-1); T[i] = update(tmp,i,1);

}

mp[a[i]] = i;

}

**scanf**("%d",&q); **while**(q--)

{

**int** l,r;

**scanf**("%d%d",&l,&r); **printf**("%d\n",query(T[l],r));

}

}

**return** 0;

}

6.2 静态区间第 k 大（POJ 2104）

**const int** MAXN = 100010; **const int** M = MAXN \* 30; **int** n,q,m,tot;

**int** a[MAXN], t[MAXN];

**int** T[MAXN], lson[M], rson[M], c[M];

void Init\_hash()

{

**for**(**int** i = 1; i <= n;i++) t[i] = a[i];

sort(t+1,t+1+n);

m = unique(t+1,t+1+n)-t-1;

}

**int build**(**int** l,**int** r)

{

**int** root = tot++; c[root] = 0;

**if**(l != r)

{

**int** mid = (l+r)>>1; lson[root] = build(l,mid); rson[root] = build(mid+1,r);

}

**return** root;

}

int hash(int x)

{

**return** lower\_bound(t+1,t+1+m,x) - t;

}

**int update**(**int** root,**int** pos,**int** val)

{

**int** newroot = tot++, tmp = newroot; c[newroot] = c[root] + val;

**int** l = 1, r = m;

**while**(l < r)

{

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

**if**(pos <= mid)

{

lson[newroot] = tot++; rson[newroot] = rson[root]; newroot = lson[newroot]; root = lson[root];

r = mid;

}

else

{

rson[newroot] = tot++; lson[newroot] = lson[root]; newroot = rson[newroot]; root = rson[root];

l = mid+1;

}

c[newroot] = c[root] + val;

}

**return** tmp;

}

**int query**(**int** left\_root,**int** right\_root,**int** k)

{

**int** l = 1, r = m;

**while**( l < r)

{

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

**if**(c[lson[left\_root]]-c[lson[right\_root]] >= k )

{

r = mid;

left\_root = lson[left\_root]; right\_root = lson[right\_root];

}

else

{

l = mid + 1;

k -= c[lson[left\_root]] - c[lson[right\_root]]; left\_root = rson[left\_root];

right\_root = rson[right\_root];

}

}

**return** l;

}

int main()

{

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

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

**while**(**scanf**("%d%d",&n,&q) == 2)

{

tot = 0;

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

**scanf**("%d",&a[i]); Init\_hash();

T[n+1] = build(1,m);

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

{

**int** pos = hash(a[i]);

T[i] = update(T[i+1],pos,1);

}

**while**(q--)

{

**int** l,r,k;

**scanf**("%d%d%d",&l,&r,&k); **printf**("%d\n",t[query(T[l],T[r+1],k)]);

}

}

**return** 0;

}

6.3 树上路径点权第 k 大（SPOJ COT）

LCA + 主席树

//主席树部分 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*8 **const int** MAXN = 200010; **const int** M = MAXN \* 40;

**int** n,q,m,TOT;

**int** a[MAXN], t[MAXN];

**int** T[MAXN], lson[M], rson[M], c[M];

void Init\_hash()

{

**for**(**int** i = 1; i <= n;i++) t[i] = a[i];

sort(t+1,t+1+n);

m = unique(t+1,t+n+1)-t-1;

}

**int build**(**int** l,**int** r)

{

**int** root = TOT++; c[root] = 0;

**if**(l != r)

{

**int** mid = (l+r)>>1; lson[root] = build(l,mid); rson[root] = build(mid+1,r);

}

**return** root;

}

int hash(int x)

{

**return** lower\_bound(t+1,t+1+m,x) - t;

}

**int update**(**int** root,**int** pos,**int** val)

{

**int** newroot = TOT++, tmp = newroot; c[newroot] = c[root] + val;

**int** l = 1, r = m;

**while**( l < r)

{

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

**if**(pos <= mid)

{

lson[newroot] = TOT++; rson[newroot] = rson[root]; newroot = lson[newroot]; root = lson[root];

r = mid;

}

else

{

rson[newroot] = TOT++; lson[newroot] = lson[root]; newroot = rson[newroot]; root = rson[root];

l = mid+1;

}

c[newroot] = c[root] + val;

}

**return** tmp;

}

**int query**(**int** left\_root,**int** right\_root,**int** LCA,**int** k)

{

**int** lca\_root = T[LCA]; **int** pos = hash(a[LCA]); **int** l = 1, r = m; **while**(l < r)

{

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

**int** tmp = c[lson[left\_root]] + c[lson[right\_root]] - 2\*c[lson[lca\_root]]

+ (pos >= l && pos <= mid);

**if**(tmp >= k)

{

left\_root = lson[left\_root]; right\_root = lson[right\_root]; lca\_root = lson[lca\_root];

r = mid;

}

else

{

k -= tmp;

left\_root = rson[left\_root]; right\_root = rson[right\_root]; lca\_root = rson[lca\_root];

l = mid + 1;

}

}

**return** l;

}

//LCA部分

**int** rmq[2\*MAXN];//rmq数组，就是欧拉序列对应的深度序列

**struct** ST

{

**int** mm[2\*MAXN];

**int** dp[2\*MAXN][20];//最小值对应的下标

void init(int n)

{

mm[0] = -1;

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

{

mm[i] = ((i&(i-1)) == 0)?mm[i-1]+1:mm[i-1]; dp[i][0] = i;

}

**for**(**int** j = 1; j <= mm[n];j++)

**for**(**int** i = 1; i + (1<<j) - 1 <= n; i++) dp[i][j] = rmq[dp[i][j-1]] <

rmq[dp[i+(1<<(j-1))][j-1]]?dp[i][j-1]:dp[i+(1<<(j-1))][j-1];

}

**int query**(**int** a,**int** b)//查询[a,b]之间最小值的下标

{

**if**(a > b)swap(a,b);

**int** k = mm[b-a+1];

**return** rmq[dp[a][k]] <=

rmq[dp[b-(1<<k)+1][k]]?dp[a][k]:dp[b-(1<<k)+1][k];

}

};

//边的结构体定义

**struct** Edge

{

**int** to,next;

};

Edge edge[MAXN\*2];

**int** tot,head[MAXN];

**int** F[MAXN\*2];//欧拉序列，就是dfs遍历的顺序，长度为2\*n-1,下标从1开始

**int** P[MAXN];//P[i]表示点i在F中第一次出现的位置

**int** cnt;

ST st;

void init()

{

tot = 0;

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

}

**void addedge**(**int** u,**int** v)//加边，无向边需要加两次

{

edge[tot].to = v; edge[tot].next = head[u]; head[u] = tot++;

}

**void dfs**(**int** u,**int** pre,**int** dep)

{

F[++cnt] = u; rmq[cnt] = dep; P[u] = cnt;

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

{

**int** v = edge[i].to; **if**(v == pre)**continue**; dfs(v,u,dep+1); F[++cnt] = u; rmq[cnt] = dep;

}

}

**void LCA\_init**(**int** root,**int** node\_num)//查询LCA前的初始化

{

cnt = 0; dfs(root,root,0); st.init(2\*node\_num-1);

}

**int query\_lca**(**int** u,**int** v)//查询u,v的lca编号

{

**return** F[st.query(P[u],P[v])];

}

**void dfs\_build**(**int** u,**int** pre)

{

**int** pos = hash(a[u]);

T[u] = update(T[pre],pos,1);

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

{

**int** v = edge[i].to; **if**(v == pre)**continue**; dfs\_build(v,u);

}

}

int main()

{

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

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

**while**(**scanf**("%d%d",&n,&q) == 2)

{

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

**scanf**("%d",&a[i]); Init\_hash();

init(); TOT = 0;

**int** u,v;

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

{

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

addedge(v,u);

}

LCA\_init(1,n); T[n+1] = build(1,m); dfs\_build(1,n+1); **int** k;

**while**(q--)

{

**scanf**("%d%d%d",&u,&v,&k); **printf**("%d\n",t[query(T[u],T[v],query\_lca(u,v),k)]);

}

**return** 0;

}

**return** 0;

}

6.4 动态第 k 大（ZOJ 2112）

树状数组套主席树

**const int** MAXN = 60010; **const int** M = 2500010; **int** n,q,m,tot;

**int** a[MAXN], t[MAXN];

**int** T[MAXN], lson[M], rson[M],c[M];

**int** S[MAXN];

**struct** Query

{

**int** kind;

**int** l,r,k;

}query[10010];

void Init\_hash(int k)

{

sort(t,t+k);

m = unique(t,t+k) - t;

}

int hash(int x)

{

**return** lower\_bound(t,t+m,x)-t;

}

**int build**(**int** l,**int** r)

{

**int** root = tot++; c[root] = 0;

**if**(l != r)

{

**int** mid = (l+r)/2; lson[root] = build(l,mid); rson[root] = build(mid+1,r);

}

**return** root;

}

**int Insert**(**int** root,**int** pos,**int** val)

{

**int** newroot = tot++, tmp = newroot;

**int** l = 0, r = m-1; c[newroot] = c[root] + val; **while**(l < r)

{

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

**if**(pos <= mid)

{

lson[newroot] = tot++; rson[newroot] = rson[root]; newroot = lson[newroot]; root = lson[root];

r = mid;

}

else

{

rson[newroot] = tot++; lson[newroot] = lson[root]; newroot = rson[newroot]; root = rson[root];

l = mid+1;

}

c[newroot] = c[root] + val;

}

**return** tmp;

}

int lowbit(int x)

{

**return** x&(-x);

}

**int** use[MAXN];

**void add**(**int** x,**int** pos,**int** val)

{

**while**(x <= n)

{

S[x] = Insert(S[x],pos,val); x += lowbit(x);

}

}

**int sum**(**int** x)

{

**int** ret = 0;

**while**(x > 0)

{

ret += c[lson[use[x]]]; x -= lowbit(x);

}

**return** ret;

}

**int Query**(**int** left,**int** right,**int** k)

{

**int** left\_root = T[left-1]; **int** right\_root = T[right]; **int** l = 0, r = m-1;

**for**(**int** i = left-1;i;i -= lowbit(i)) use[i] = S[i]; **for**(**int** i = right;i ;i -= lowbit(i)) use[i] = S[i]; **while**(l < r)

{

**int** mid = (l+r)/2;

**int** tmp = sum(right) - sum(left-1) + c[lson[right\_root]] - c[lson[left\_root]];

**if**(tmp >= k)

{

r = mid;

**for**(**int** i = left-1; i ;i -= lowbit(i))

use[i] = lson[use[i]];

**for**(**int** i = right; i; i -= lowbit(i)) use[i] = lson[use[i]];

left\_root = lson[left\_root]; right\_root = lson[right\_root];

}

else

{

l = mid+1; k -= tmp;

**for**(**int** i = left-1; i;i -= lowbit(i)) use[i] = rson[use[i]];

**for**(**int** i = right;i ;i -= lowbit(i)) use[i] = rson[use[i]];

left\_root = rson[left\_root]; right\_root = rson[right\_root];

}

}

**return** l;

}

**void Modify**(**int** x,**int** p,**int** d)

{

**while**(x <= n)

{

S[x] = Insert(S[x],p,d); x += lowbit(x);

}

}

int main()

{

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

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

**int** Tcase; **scanf**("%d",&Tcase); **while**(Tcase--)

{

**scanf**("%d%d",&n,&q); tot = 0;

m = 0;

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

{

**scanf**("%d",&a[i]); t[m++] = a[i];

}

**char** op[10];

**for**(**int** i = 0;i < q;i++)

{

**scanf**("%s",op);

**if**(op[0] == 'Q')

{

query[i].kind = 0;

**scanf**("%d%d%d",&query[i].l,&query[i].r,&query[i].k);

}

else

{

query[i].kind = 1;

**scanf**("%d%d",&query[i].l,&query[i].r);

t[m++] = query[i].r;

}

}

Init\_hash(m);

T[0] = build(0,m-1);

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

T[i] = Insert(T[i-1],hash(a[i]),1);

**for**(**int** i = 1;i <= n;i++) S[i] = T[0];

**for**(**int** i = 0;i < q;i++)

{

**if**(query[i].kind == 0)

**printf**("%d\n",t[Query(query[i].l,query[i].r,query[i].k)]); **else**

{

Modify(query[i].l,hash(a[query[i].l]),-1);

Modify(query[i].l,hash(query[i].r),1); a[query[i].l] = query[i].r;

}

}

}

**return** 0;

}

## 7、Treap

ZOJ3765

long long gcd(long long a,long long b)

{

**if**(b == 0)**return** a;

**else return** gcd(b,a%b);

}

**const int** MAXN = 300010;

**int** num[MAXN],st[MAXN];

**struct** Treap

{

**int** tot1;

**int** s[MAXN],tot2;//内存池和容量

**int** ch[MAXN][2];

**int** key[MAXN],size[MAXN]; **int** sum0[MAXN],sum1[MAXN]; **int** status[MAXN];

void Init()

{

tot1 = tot2 = 0; size[0] = 0;

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

sum0[0] = sum1[0] = 0;

}

bool random(double p)

{

**return** (**double**)**rand**() / RAND\_MAX < p;

}

**int newnode**(**int** val,**int** \_status)

{

**int** r;

**if**(tot2)r = s[tot2--]; **else** r = ++tot1; size[r] = 1;

key[r] = val; status[r] = \_status;

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

sum0[r] = sum1[r] = 0;//需要push\_up

**return** r;

}

void del(int r)

{

**if**(!r)**return**; s[++tot2] = r; del(ch[r][0]);

del(ch[r][1]);

}

void push\_up(int r)

{

**int** lson = ch[r][0], rson = ch[r][1]; size[r] = size[lson] + size[rson] + 1; sum0[r] = gcd(sum0[lson],sum0[rson]); sum1[r] = gcd(sum1[lson],sum1[rson]); **if**(status[r] == 0)

sum0[r] = gcd(sum0[r],key[r]);

**else** sum1[r] = gcd(sum1[r],key[r]);

}

**void merge**(**int** &p,**int** x,**int** y)

{

**if**(!x || !y)

p = x|y;

**else if**(random((**double**)size[x]/(size[x]+size[y])))

{

merge(ch[x][1],ch[x][1],y); push\_up(p=x);

}

else

{

merge(ch[y][0],x,ch[y][0]); push\_up(p=y);

}

}

**void split**(**int** p,**int** &x,**int** &y,**int** k)

{

**if**(!k)

{

x = 0; y = p;

return;

}

**if**(size[ch[p][0]] >= k)

{

y = p; split(ch[p][0],x,ch[y][0],k); push\_up(y);

}

else

{

x = p;

split(ch[p][1],ch[x][1],y,k - size[ch[p][0]] - 1); push\_up(x);

}

}

**void build**(**int** &p,**int** l,**int** r)

{

**if**(l > r)**return**;

**int** mid = (l + r)/2;

p = newnode(num[mid],st[mid]); build(ch[p][0],l,mid-1);

build(ch[p][1],mid+1,r); push\_up(p);

}

**void debug**(**int** root)

{

**if**(root == 0)**return**;

**printf**("%d 左儿子：%d 右儿子: %d size = %d key

= %d\n",root,ch[root][0],ch[root][1],size[root],key[root]); debug(ch[root][0]);

debug(ch[root][1]);

}

};

Treap T;

**char** op[10];

int main()

{

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

//freopen("out.txt","w",stdout); **int** n,q; **while**(**scanf**("%d%d",&n,&q) == 2)

{

**int** root = 0; T.Init();

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

**scanf**("%d%d",&num[i],&st[i]); T.build(root,1,n);

**while**(q--)

{

**scanf**("%s",op);

**if**(op[0] == 'Q')

{

**int** l,r,s; **scanf**("%d%d%d",&l,&r,&s); **int** x,y,z; T.split(root,x,z,r);

T.split(x,x,y,l-1);

**if**(s == 0)

**printf**("%d\n",T.sum0[y] == 0? -1:T.sum0[y]);

else

**printf**("%d\n",T.sum1[y] == 0?-1:T.sum1[y]); T.merge(x,x,y);

T.merge(root,x,z);

}

**else if**(op[0] == 'I')

{

**int** v,s,loc; **scanf**("%d%d%d",&loc,&v,&s); **int** x,y; T.split(root,x,y,loc); T.merge(x,x,T.newnode(v,s)); T.merge(root,x,y);

}

**else if**(op[0] == 'D')

{

**int** loc;

**scanf**("%d",&loc); **int** x,y,z;

T.split(root,x,z,loc);

T.split(x,x,y,loc-1); T.del(y); T.merge(root,x,z);

}

**else if**(op[0] == 'R')

{

**int** loc; **scanf**("%d",&loc); **int** x,y,z;

T.split(root,x,z,loc);

T.split(x,x,y,loc-1);

T.status[y] = 1-T.status[y]; T.push\_up(y); T.merge(x,x,y);

T.merge(root,x,z);

}

else

{

**int** loc,v; **scanf**("%d%d",&loc,&v); **int** x,y,z; T.split(root,x,z,loc);

T.split(x,x,y,loc-1); T.key[y] = v;

T.push\_up(y); T.merge(x,x,y);

T.merge(root,x,z);

}

}

}

**return** 0;

}

# 图论

## 1、最短路

1.1 Dijkstra 单源最短路，邻接矩阵形式 权值必须是非负

/\*

\* 单源最短路径，Dijkstra算法，邻接矩阵形式，复杂度为O(n^2)

\* 求出源beg到所有点的最短路径，传入图的顶点数，和邻接矩阵cost[][]

\* 返回各点的最短路径lowcost[], 路径pre[].pre[i]记录beg到i路径上的父结点，pre[beg]=-1

\* 可更改路径权类型，但是权值必须为非负

\*

\*/

**const int** MAXN=1010;

**#define** typec **int**

**const** typec INF=0x3f3f3f3f;//防止后面溢出，这个不能太大

**bool** vis[MAXN];

**int** pre[MAXN];

**void Dijkstra**(typec cost[][MAXN],typec lowcost[],**int** n,**int** beg)

{

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

{

lowcost[i]=INF;vis[i]=**false**;pre[i]=-1;

}

lowcost[beg]=0;

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

{

**int** k=-1;

**int** Min=INF;

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

**if**(!vis[i]&&lowcost[i]<Min)

{

Min=lowcost[i]; k=i;

}

**if**(k==-1)**break**; vis[k]=**true**;

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

**if**(!vis[i]&&lowcost[k]+cost[k][i]<lowcost[i])

{

lowcost[i]=lowcost[k]+cost[k][i];

pre[i]=k;

}

}

}

* 1. Dijkstar 算法+堆优化 使用优先队列优化，复杂度 O (E log E)

/\*

\* 使用优先队列优化Dijkstra算法

* + 复杂度O(ElogE)
  + 注意对vector<Edge>E[MAXN]进行初始化后加边

\*/

**const int** INF=0x3f3f3f3f; **const int** MAXN=1000010; **struct** qnode

{

**int** v;

**int** c;

**qnode**(**int** \_v=0,**int** \_c=0):v(\_v),c(\_c){}

**bool operator <**(**const** qnode &r)**const**

{

**return** c>r.c;

}

};

**struct** Edge

{

**int** v,cost;

**Edge**(**int** \_v=0,**int** \_cost=0):v(\_v),cost(\_cost){}

};

vector<Edge>E[MAXN];

**bool** vis[MAXN];

**int** dist[MAXN];

**void Dijkstra**(**int** n,**int** start)//点的编号从1开始

{

**memset**(vis,**false**,**sizeof**(vis)); **for**(**int** i=1;i<=n;i++)dist[i]=INF; priority\_queue<qnode>que; **while**(!que.empty())que.pop(); dist[start]=0; que.push(qnode(start,0));

qnode tmp;

**while**(!que.empty())

{

tmp=que.top(); que.pop();

**int** u=tmp.v; **if**(vis[u])**continue**; vis[u]=**true**;

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

{

**int** v=E[tmp.v][i].v;

**int** cost=E[u][i].cost;

**if**(!vis[v]&&dist[v]>dist[u]+cost)

{

dist[v]=dist[u]+cost; que.push(qnode(v,dist[v]));

}

}

}

}

**void addedge**(**int** u,**int** v,**int** w)

{

E[u].push\_back(Edge(v,w));

}

* 1. 单源最短路 bellman\_ford 算法

/\*

* + 单源最短路bellman\_ford算法，复杂度O(VE)

\* 可以处理负边权图。

\* 可以判断是否存在负环回路。返回true,当且仅当图中不包含从源点可达的负权回路

\* vector<Edge>E;先E.clear()初始化，然后加入所有边

\* 点的编号从1开始(从0开始简单修改就可以了)

\*/

**const int** INF=0x3f3f3f3f;

**const int** MAXN=550; **int** dist[MAXN]; **struct** Edge

{

**int** u,v;

**int** cost;

**Edge**(**int** \_u=0,**int** \_v=0,**int** \_cost=0):u(\_u),v(\_v),cost(\_cost){}

};

vector<Edge>E;

**bool bellman\_ford**(**int** start,**int** n)//点的编号从1开始

{

**for**(**int** i=1;i<=n;i++)dist[i]=INF; dist[start]=0;

**for**(**int** i=1;i<n;i++)//最多做n-1次

{

**bool** flag=**false**;

**for**(**int** j=0;j<E.size();j++)

{

**int** u=E[j].u;

**int** v=E[j].v;

**int** cost=E[j].cost;

**if**(dist[v]>dist[u]+cost)

{

dist[v]=dist[u]+cost; flag=**true**;

}

}

**if**(!flag)**return true**;//没有负环回路

}

**for**(**int** j=0;j<E.size();j++)

**if**(dist[E[j].v]>dist[E[j].u]+E[j].cost) **return false**;//有负环回路

**return true**;//没有负环回路

}

1.4 单源最短路 SPFA

/\*

\* 单源最短路SPFA

\* 时间复杂度 0(kE)

\* 这个是队列实现，有时候改成栈实现会更加快，很容易修改

\* 这个复杂度是不定的

\*/

**const int** MAXN=1010; **const int** INF=0x3f3f3f3f; **struct** Edge

{

**int** v;

**int** cost;

**Edge**(**int** \_v=0,**int** \_cost=0):v(\_v),cost(\_cost){}

};

vector<Edge>E[MAXN];

**void addedge**(**int** u,**int** v,**int** w)

{

E[u].push\_back(Edge(v,w));

}

**bool** vis[MAXN];//在队列标志

**int** cnt[MAXN];//每个点的入队列次数

**int** dist[MAXN];

**bool SPFA**(**int** start,**int** n)

{

**memset**(vis,**false**,**sizeof**(vis)); **for**(**int** i=1;i<=n;i++)dist[i]=INF; vis[start]=**true**;

dist[start]=0; queue<**int**>que;

**while**(!que.empty())que.pop(); que.push(start); **memset**(cnt,0,**sizeof**(cnt)); cnt[start]=1; **while**(!que.empty())

{

**int** u=que.front(); que.pop(); vis[u]=**false**;

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

{

**int** v=E[u][i].v;

**if**(dist[v]>dist[u]+E[u][i].cost)

{

dist[v]=dist[u]+E[u][i].cost;

**if**(!vis[v])

{

vis[v]=**true**; que.push(v);

**if**(++cnt[v]>n)**return false**;

//cnt[i]为入队列次数，用来判定是否存在负环回路

}

}

}

}

return true;

}

## 2、最小生成树

* 1. Prim 算法

/\*

* + Prim求MST

\* 耗费矩阵cost[][]，标号从0开始，0~n-1

\* 返回最小生成树的权值，返回-1表示原图不连通

\*/

**const int** INF=0x3f3f3f3f;

**const int** MAXN=110;

**bool** vis[MAXN];

**int** lowc[MAXN];

**int Prim**(**int** cost[][MAXN],**int** n)//点是0~n-1

{

**int** ans=0; **memset**(vis,**false**,**sizeof**(vis)); vis[0]=**true**;

**for**(**int** i=1;i<n;i++)lowc[i]=cost[0][i];

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

{

**int** minc=INF;

**int** p=-1;

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

**if**(!vis[j]&&minc>lowc[j])

{

minc=lowc[j]; p=j;

}

**if**(minc==INF)**return** -1;//原图不连通

ans+=minc; vis[p]=**true**; **for**(**int** j=0;j<n;j++)

**if**(!vis[j]&&lowc[j]>cost[p][j]) lowc[j]=cost[p][j];

}

**return** ans;

}

* 1. Kruskal 算法

/\*

* + Kruskal算法求MST

\*/

**const int** MAXN=110;//最大点数 **const int** MAXM=10000;//最大边数 **int** F[MAXN];//并查集使用

**struct** Edge

{

**int** u,v,w;

}edge[MAXM];//存储边的信息，包括起点/终点/权值

**int** tol;//边数，加边前赋值为0

**void addedge**(**int** u,**int** v,**int** w)

{

edge[tol].u=u; edge[tol].v=v; edge[tol++].w=w;

}

**bool cmp**(Edge a,Edge b)

{//排序函数，讲边按照权值从小到大排序

**return** a.w<b.w;

}

int find(int x)

{

**if**(F[x]==-1)**return** x;

**else return** F[x]=find(F[x]);

}

**int Kruskal**(**int** n)//传入点数，返回最小生成树的权值，如果不连通返回-1

{

**memset**(F,-1,**sizeof**(F)); sort(edge,edge+tol,cmp);

**int** cnt=0;//计算加入的边数

**int** ans=0;

**for**(**int** i=0;i<tol;i++)

{

**int** u=edge[i].u; **int** v=edge[i].v; **int** w=edge[i].w; **int** t1=find(u); **int** t2=find(v); **if**(t1!=t2)

{

ans+=w; F[t1]=t2;

cnt++;

}

**if**(cnt==n-1)**break**;

}

**if**(cnt<n-1)**return** -1;//不连通

**else return** ans;

}

## 3、次小生成树

/\*

\* 次小生成树

\* 求最小生成树时，用数组Max[i][j]来表示MST中i到j最大边权

\* 求完后，直接枚举所有不在MST中的边，替换掉最大边权的边，更新答案

\* 点的编号从0开始

\*/

**const int** MAXN=110;

**const int** INF=0x3f3f3f3f;

**bool** vis[MAXN]; **int** lowc[MAXN]; **int** pre[MAXN];

**int** Max[MAXN][MAXN];//Max[i][j]表示在最小生成树中从i到j的路径中的最大边权

**bool** used[MAXN][MAXN];

**int Prim**(**int** cost[][MAXN],**int** n)

{

**int** ans=0; **memset**(vis,**false**,**sizeof**(vis)); **memset**(Max,0,**sizeof**(Max)); **memset**(used,**false**,**sizeof**(used)); vis[0]=**true**;

pre[0]=-1;

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

{

lowc[i]=cost[0][i]; pre[i]=0;

}

lowc[0]=0;

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

{

**int** minc=INF;

**int** p=-1;

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

**if**(!vis[j]&&minc>lowc[j])

{

minc=lowc[j]; p=j;

}

**if**(minc==INF)**return** -1; ans+=minc;

vis[p]=**true**; used[p][pre[p]]=used[pre[p]][p]=**true**; **for**(**int** j=0;j<n;j++)

{

**if**(vis[j]&&j!=p)Max[j][p]=Max[p][j]=max(Max[j][pre[p]],lowc[p]); **if**(!vis[j]&&lowc[j]>cost[p][j])

{

lowc[j]=cost[p][j]; pre[j]=p;

}

}

}

**return** ans;

}

## 4、有向图的强连通分量

* 1. Tarjan

/\*

* + Tarjan算法

\* 复杂度O(N+M)

\*/

**const int** MAXN = 20010;//点数

**const int** MAXM = 50010;//边数

**struct** Edge

{

**int** to,next;

}edge[MAXM];

**int** head[MAXN],tot;

**int** Low[MAXN],DFN[MAXN],Stack[MAXN],Belong[MAXN];//Belong数组的值是1~scc

**int** Index,top;

**int** scc;//强连通分量的个数

**bool** Instack[MAXN];

**int** num[MAXN];//各个强连通分量包含点的个数，数组编号1~scc

//num数组不一定需要，结合实际情况

void addedge(int u,int v)

{

edge[tot].to = v;edge[tot].next = head[u];head[u] = tot++;

}

void Tarjan(int u)

{

**int** v;

Low[u] = DFN[u] = ++Index; Stack[top++] = u; Instack[u] = **true**;

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

{

v = edge[i].to;

**if**( !DFN[v] )

{

Tarjan(v);

**if**( Low[u] > Low[v] )Low[u] = Low[v];

}

**else if**(Instack[v] && Low[u] > DFN[v]) Low[u] = DFN[v];

}

**if**(Low[u] == DFN[u])

{

scc++;

do

{

v = Stack[--top]; Instack[v] = **false**; Belong[v] = scc; num[scc]++;

}

**while**( v != u);

}

}

void solve(int N)

{

**memset**(DFN,0,**sizeof**(DFN)); **memset**(Instack,**false**,**sizeof**(Instack)); **memset**(num,0,**sizeof**(num));

Index = scc = top = 0;

**for**(**int** i = 1;i <= N;i++)

**if**(!DFN[i])

Tarjan(i);

}

void init()

{

tot = 0;

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

}

* 1. Kosaraju

/\*

* + Kosaraju算法，复杂度O(N+M)

\*/

**const int** MAXN = 20010; **const int** MAXM = 50010; **struct** Edge

{

**int** to,next;

}edge1[MAXM],edge2[MAXM];

//edge1是原图G，edge2是逆图GT **int** head1[MAXN],head2[MAXN]; **bool** mark1[MAXN],mark2[MAXN]; **int** tot1,tot2;

**int** cnt1,cnt2;

**int** st[MAXN];//对原图进行dfs，点的结束时间从小到大排序 **int** Belong[MAXN];//每个点属于哪个连通分量(0~cnt2-1) **int** num;//中间变量，用来数某个连通分量中点的个数

**int** setNum[MAXN];//强连通分量中点的个数，编号0~cnt2-1

void addedge(int u,int v)

{

edge1[tot1].to = v;edge1[tot1].next = head1[u];head1[u] = tot1++; edge2[tot2].to = u;edge2[tot2].next = head2[v];head2[v] = tot2++;

}

void DFS1(int u)

{

mark1[u] = **true**;

**for**(**int** i = head1[u];i != -1;i = edge1[i].next)

**if**(!mark1[edge1[i].to])

DFS1(edge1[i].to); st[cnt1++] = u;

}

void DFS2(int u)

{

mark2[u] = **true**; num++;

Belong[u] = cnt2;

**for**(**int** i = head2[u];i != -1;i = edge2[i].next)

**if**(!mark2[edge2[i].to])

DFS2(edge2[i].to);

}

**void solve**(**int** n)//点的编号从1开始

{

**memset**(mark1,**false**,**sizeof**(mark1)); **memset**(mark2,**false**,**sizeof**(mark2)); cnt1 = cnt2 = 0;

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

**if**(!mark1[i]) DFS1(i);

**for**(**int** i = cnt1-1;i >= 0; i--)

**if**(!mark2[st[i]])

{

num = 0; DFS2(st[i]);

setNum[cnt2++] = num;

}

}

## 5、图的割点、桥和双连通分支的基本概念

[点连通度与边连通度]

在一个无向连通图中，如果有一个顶点集合，删除这个顶点集合，以及这个集合中所有顶点相关联的边以 后，原图变成多个连通块，就称这个点集为**割点集合**。一个图的**点连通度**的定义为，最小割点集合中的顶 点数。 类似的，如果有一个边集合，删除这个边集合以后，原图变成多个连通块，就称这个点集为**割边集合**。一

个图的**边连通度**的定义为，最小割边集合中的边数。

[双连通图、割点与桥]

如果一个无向连通图的点连通度大于 1，则称该图是**点双连通的(point biconnected)**，简称**双连通**或**重连通**。 一个图有割点，当且仅当这个图的点连通度为 1，则割点集合的唯一元素被称为**割点(cut point)**，又叫**关节 点(articulation point)**。

如果一个无向连通图的边连通度大于 1，则称该图是**边双连通的(edge biconnected)**，简称双连通或重连通。 一个图有桥，当且仅当这个图的边连通度为 1，则割边集合的唯一元素被称为**桥(bridge)**，又叫**关节边 (articulation edge)。**

可以看出，点双连通与边双连通都可以简称为双连通，它们之间是有着某种联系的，下文中提到的双连通， 均既可指点双连通，又可指边双连通。

[双连通分支]

在图 G 的所有子图 G'中，如果 G'是双连通的，则称 G'为**双连通子图**。如果一个双连通子图 G'它不是任何一 个双连通子图的真子集，则 G'为**极大双连通子图**。**双连通分支(biconnected component)**，或**重连通分支**， 就是图的极大双连通子图。特殊的，点双连通分支又叫做**块**。

[求割点与桥]

该算法是 R.Tarjan 发明的。对图深度优先搜索，定义 DFS(u)为 u 在搜索树（以下简称为树）中被遍历到的次 序号。定义 Low(u)为 u 或 u 的子树中能通过非父子边追溯到的最早的节点，即 DFS 序号最小的节点。根据 定义，则有：

Low(u)=Min { DFS(u) DFS(v) (u,v)为后向边(返祖边) 等价于 DFS(v)<DFS(u)且 v 不为 u 的父亲节点 Low(v) (u,v)

为树枝边(父子边) }

一个顶点 u 是割点，当且仅当满足(1)或(2) (1) u 为树根，且 u 有多于一个子树。 (2) u 不为树根，且满足存 在(u,v)为树枝边(或称父子边，即 u 为 v 在搜索树中的父亲)，使得 DFS(u)<=Low(v)。 一条无向边(u,v)是桥，当且仅当(u,v)为树枝边，且满足 DFS(u)<Low(v)。

[求双连通分支]

下面要分开讨论点双连通分支与边双连通分支的求法。 对于点双连通分支，实际上在求割点的过程中就能顺便把每个点双连通分支求出。建立一个栈，存储当前 双连通分支，在搜索图时，每找到一条树枝边或后向边(非横叉边)，就把这条边加入栈中。如果遇到某时满 足 DFS(u)<=Low(v)，说明 u 是一个割点，同时把边从栈顶一个个取出，直到遇到了边(u,v)，取出的这些边与 其关联的点，组成一个点双连通分支。割点可以属于多个点双连通分支，其余点和每条边只属于且属于一 个点双连通分支。 对于边双连通分支，求法更为简单。只需在求出所有的桥以后，把桥边删除，原图变成了多个连通块，则 每个连通块就是一个边双连通分支。桥不属于任何一个边双连通分支，其余的边和每个顶点都属于且只属 于一个边双连通分支。

[构造双连通图] 一个有桥的连通图，如何把它通过加边变成边双连通图？方法为首先求出所有的桥，然后删除这些桥边， 剩下的每个连通块都是一个双连通子图。把每个双连通子图收缩为一个顶点，再把桥边加回来，最后的这 个图一定是一棵树，边连通度为 1。

统计出树中度为 1 的节点的个数，即为叶节点的个数，记为 leaf。则至少在树上添加(leaf+1)/2 条边，就能 使树达到边二连通，所以至少添加的边数就是(leaf+1)/2。具体方法为，首先把两个最近公共祖先最远的两

个叶节点之间连接一条边，这样可以把这两个点到祖先的路径上所有点收缩到一起，因为一个形成的环一 定是双连通的。然后再找两个最近公共祖先最远的两个叶节点，这样一对一对找完，恰好是(leaf+1)/2 次， 把所有点收缩到了一起。

## 6、割点与桥

模板：

/\*

\* 求 无向图的割点和桥

\* 可以找出割点和桥，求删掉每个点后增加的连通块。

\* 需要注意重边的处理，可以先用矩阵存，再转邻接表，或者进行判重

\* 找割点调用必须是Tarjan(i,i)

\* 若要找桥，需要保证正边、反边addedge的时候是相邻的

\*/

**const int** MAXN = 10010; **const int** MAXM = 100010; **struct** Edge

{

**int** to,next;

**bool** cut;//是否为桥的标记

}edge[MAXM];

**int** head[MAXN],tot;

**int** Low[MAXN],DFN[MAXN],Stack[MAXN];

**int** Index,top;

**bool** Instack[MAXN];

**bool** cut[MAXN];

**int** add\_block[MAXN];//删除一个点后增加的连通块

**int** bridge;

void addedge(int u,int v)

{

edge[tot].to = v;edge[tot].next = head[u];edge[tot].cut = **false**; head[u] = tot++;

}

**void Tarjan**(**int** u,**int** pre)

{

**int** v;

Low[u] = DFN[u] = ++Index; Stack[top++] = u; Instack[u] = **true**;

**int** son = 0;

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

{

v = edge[i].to;

**if**(v == pre)**continue**;

**if**( !DFN[v] )

{

son++; Tarjan(v,u);

**if**(Low[u] > Low[v])Low[u] = Low[v];

//桥

//一条无向边(u,v)是桥，当且仅当(u,v)为树枝边，且满足DFS(u)<Low(v)。

**if**(Low[v] > DFN[u])

{

bridge++; edge[i].cut = **true**; edge[i^1].cut = **true**;

}

//割点

//一个顶点u是割点，当且仅当满足(1)或(2) (1) u为树根，且u有多于一个子树。

//(2) u不为树根，且满足存在(u,v)为树枝边(或称父子边，

//即u为v在搜索树中的父亲)，使得DFS(u)<=Low(v)

**if**(u != pre && Low[v] >= DFN[u])//不是树根

{

cut[u] = **true**; add\_block[u]++;

}

}

**else if**( Low[u] > DFN[v]) Low[u] = DFN[v];

}

//树根，分支数大于1

**if**(u == pre && son > 1)cut[u] = **true**; **if**(u == pre)add\_block[u] = son - 1; Instack[u] = **false**;

top--;

}

调用：

1）UVA 796 Critical Links 给出一个无向图，按顺序输出桥

void solve(int N)

{

**memset**(DFN,0,**sizeof**(DFN)); **memset**(Instack,**false**,**sizeof**(Instack)); **memset**(add\_block,0,**sizeof**(add\_block)); **memset**(cut,**false**,**sizeof**(cut));

Index = top = 0; bridge = 0;

**for**(**int** i = 1;i <= N;i++)

**if**( !DFN[i] )

Tarjan(i,i);

**printf**("%d critical links\n",bridge); vector<pair<**int**,**int**> >ans;

**for**(**int** u = 1;u <= N;u++)

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

**if**(edge[i].cut && edge[i].to > u)

{

ans.push\_back(make\_pair(u,edge[i].to));

}

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

//按顺序输出桥

**for**(**int** i = 0;i < ans.size();i++)

**printf**("%d - %d\n",ans[i].first-1,ans[i].second-1);

**printf**("\n");

}

void init()

{

tot = 0;

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

}

//处理重边

map<**int**,**int**>mapit;

inline bool isHash(int u,int v)

{

**if**(mapit[u\*MAXN+v])**return true**; **if**(mapit[v\*MAXN+u])**return true**; mapit[u\*MAXN+v] = mapit[v\*MAXN+u] = 1; **return false**;

}

int main()

{

**int** n;

**while**(**scanf**("%d",&n) == 1)

{

init(); **int** u; **int** k; **int** v;

//mapit.clear();

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

{

**scanf**("%d (%d)",&u,&k); u++;

//这样加边，要保证正边和反边是相邻的，建无向图

**while**(k--)

{

**scanf**("%d",&v); v++;

**if**(v <= u)**continue**;

//if(isHash(u,v))continue; addedge(u,v);

addedge(v,u);

}

}

solve(n);

}

**return** 0;

}

2）POJ 2117 求删除一个点后，图中最多有多少个连通块

void solve(int N)

{

**memset**(DFN,0,**sizeof**(DFN)); **memset**(Instack,0,**sizeof**(Instack)); **memset**(add\_block,0,**sizeof**(add\_block)); **memset**(cut,**false**,**sizeof**(cut));

Index = top = 0;

**int** cnt = 0;//原来的连通块数

**for**(**int** i = 1;i <= N;i++)

**if**( !DFN[i] )

{

Tarjan(i,i);//找割点调用必须是Tarjan(i,i)

cnt++;

}

**int** ans = 0;

**for**(**int** i = 1;i <= N;i++)

ans = max(ans,cnt+add\_block[i]);

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

}

void init()

{

tot = 0;

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

}

int main()

{

**int** n,m;

**int** u,v;

**while**(**scanf**("%d%d",&n,&m)==2)

{

**if**(n==0 && m == 0)**break**; init();

**while**(m--)

{

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

addedge(u,v);

addedge(v,u);

}

solve(n);

}

**return** 0;

}

## 7、边双连通分支

去掉桥，其余的连通分支就是边双连通分支了。一个有桥的连通图要变成边双连通图的话，把双连通子图 收缩为一个点，形成一颗树。需要加的边为(leaf+1)/2 (leaf 为叶子结点个数)

POJ 3177 **给定一个连通的无向图 G，至少要添加几条边，才能使其变为双连通图。**

**#include** <stdio.h>

**#include** <string.h>

**#include** <iostream>

**#include** <algorithm>

**#include** <map>

using namespace std;

**const int** MAXN = 5010;//点数

**const int** MAXM = 20010;//边数，因为是无向图，所以这个值要\*2

**struct** Edge

{

**int** to,next;

**bool** cut;//是否是桥标记

}edge[MAXM];

**int** head[MAXN],tot;

**int** Low[MAXN],DFN[MAXN],Stack[MAXN],Belong[MAXN];//Belong数组的值是1~block

**int** Index,top;

**int** block;//边双连通块数 **bool** Instack[MAXN]; **int** bridge;//桥的数目

void addedge(int u,int v)

{

edge[tot].to = v;edge[tot].next = head[u];edge[tot].cut=**false**; head[u] = tot++;

}

**void Tarjan**(**int** u,**int** pre)

{

**int** v;

Low[u] = DFN[u] = ++Index; Stack[top++] = u; Instack[u] = **true**;

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

{

v = edge[i].to;

**if**(v == pre)**continue**;

**if**( !DFN[v] )

{

Tarjan(v,u);

**if**( Low[u] > Low[v] )Low[u] = Low[v];

**if**(Low[v] > DFN[u])

{

bridge++; edge[i].cut = **true**;

edge[i^1].cut = **true**;

}

}

**else if**( Instack[v] && Low[u] > DFN[v] ) Low[u] = DFN[v];

}

**if**(Low[u] == DFN[u])

{

block++;

do

{

v = Stack[--top]; Instack[v] = **false**; Belong[v] = block;

}

**while**( v!=u );

}

}

void init()

{

tot = 0;

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

}

**int** du[MAXN];//缩点后形成树，每个点的度数

void solve(int n)

{

**memset**(DFN,0,**sizeof**(DFN)); **memset**(Instack,**false**,**sizeof**(Instack)); Index = top = block = 0;

Tarjan(1,0);

**int** ans = 0; **memset**(du,0,**sizeof**(du)); **for**(**int** i = 1;i <= n;i++)

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

**if**(edge[j].cut)

du[Belong[i]]++;

**for**(**int** i = 1;i <= block;i++)

**if**(du[i]==1) ans++;

//找叶子结点的个数ans,构造边双连通图需要加边(ans+1)/2

**printf**("%d\n",(ans+1)/2);

}

int main()

{

**int** n,m;

**int** u,v;

**while**(**scanf**("%d%d",&n,&m)==2)

{

init();

**while**(m--)

{

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

addedge(v,u);

}

solve(n);

}

**return** 0;

}

## 8、点双连通分支

对于点双连通分支，实际上在求割点的过程中就能顺便把每个点双连通分支求出。建立一个栈，存储 当前双连通分支，在搜索图时，每找到一条树枝边或后向边(非横叉边)，就把这条边加入栈中。如果遇到 某时满足 DFS(u)<=Low(v)，说明 u 是一个割点，同时把边从栈顶一个个取出，直到遇到了边(u,v)， 取出的这些边与其关联的点，组成一个点双连通分支。割点可以属于多个点双连通分支，其余点和每条边 只属于且属于一个点双连通分支。

POJ 2942

奇圈，二分图判断的染色法，求点双连通分支

/\*

POJ 2942 Knights of the Round Table 亚瑟王要在圆桌上召开骑士会议，为了不引发骑士之间的冲突， 并且能够让会议的议题有令人满意的结果，每次开会前都必须对出席会议的骑士有如下要求： 1、 相互憎恨的两个骑士不能坐在直接相邻的2个位置；

2、 出席会议的骑士数必须是奇数，这是为了让投票表决议题时都能有结果。

注意：1、所给出的憎恨关系一定是双向的，不存在单向憎恨关系。 2、由于是圆桌会议，则每个出席的骑士身边必定刚好有2个骑士。 即每个骑士的座位两边都必定各有一个骑士。 3、一个骑士无法开会，就是说至少有3个骑士才可能开会。

首先根据给出的互相憎恨的图中得到补图。 然后就相当于找出不能形成奇圈的点。 利用下面两个定理：

（1）如果一个双连通分量内的某些顶点在一个奇圈中（即双连通分量含有奇圈）， 那么这个双连通分量的其他顶点也在某个奇圈中；

（2）如果一个双连通分量含有奇圈，则他必定不是一个二分图。反过来也成立，这是一个充要条件。

所以本题的做法，就是对补图求点双连通分量。 然后对于求得的点双连通分量，使用染色法判断是不是二分图，不是二分图，这个双连通分量的点是可以 存在的

\*/

**const int** MAXN = 1010;

**const int** MAXM = 2000010;

**struct** Edge

{

**int** to,next;

}edge[MAXM];

**int** head[MAXN],tot;

**int** Low[MAXN],DFN[MAXN],Stack[MAXN],Belong[MAXN];

**int** Index,top;

**int** block;//点双连通分量的个数

**bool** Instack[MAXN];

**bool** can[MAXN];

**bool** ok[MAXN];//标记

**int** tmp[MAXN];//暂时存储双连通分量中的点

**int** cc;//tmp的计数

**int** color[MAXN];//染色

void addedge(int u,int v)

{

edge[tot].to = v;edge[tot].next = head[u];head[u] = tot++;

}

**bool dfs**(**int** u,**int** col)//染色判断二分图

{

color[u] = col;

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

{

**int** v = edge[i].to;

**if**( !ok[v] )**continue**;

**if**(color[v] != -1)

{

**if**(color[v]==col)**return false**; **continue**;

}

**if**(!dfs(v,!col))**return false**;

}

return true;

}

**void Tarjan**(**int** u,**int** pre)

{

**int** v;

Low[u] = DFN[u] = ++Index; Stack[top++] = u; Instack[u] = **true**;

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

{

v = edge[i].to;

**if**(v == pre)**continue**;

**if**( !DFN[v] )

{

Tarjan(v,u);

**if**(Low[u] > Low[v])Low[u] = Low[v];

**if**( Low[v] >= DFN[u])

{

block++; **int** vn; cc = 0;

**memset**(ok,**false**,**sizeof**(ok)); **do**

{

vn = Stack[--top]; Belong[vn] = block; Instack[vn] = **false**; ok[vn] = **true**; tmp[cc++] = vn;

}

**while**( vn!=v );

ok[u] = 1;

**memset**(color,-1,**sizeof**(color)); **if**( !dfs(u,0) )

{

can[u] = **true**;

**while**(cc--)can[tmp[cc]]=**true**;

}

}

}

**else if**(Instack[v] && Low[u] > DFN[v]) Low[u] = DFN[v];

}

}

void solve(int n)

{

**memset**(DFN,0,**sizeof**(DFN)); **memset**(Instack,**false**,**sizeof**(Instack)); Index = block = top = 0; **memset**(can,**false**,**sizeof**(can));

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

**if**(!DFN[i])

Tarjan(i,-1);

**int** ans = n;

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

**if**(can[i])

ans--;

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

}

void init()

{

tot = 0;

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

}

**int** g[MAXN][MAXN];

int main()

{

**int** n,m;

**int** u,v;

**while**(**scanf**("%d%d",&n,&m)==2)

{

**if**(n==0 && m==0)**break**; init(); **memset**(g,0,**sizeof**(g)); **while**(m--)

{

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

g[u][v]=g[v][u]=1;

}

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

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

**if**(i != j && g[i][j]==0) addedge(i,j);

solve(n);

}

**return** 0;

}

## 9、最小树形图

**#include** <stdio.h>

**#include** <string.h>

**#include** <iostream>

**#include** <algorithm>

using namespace std;

/\*

\* 最小树形图

\* int型

\* 复杂度O(NM)

\* 点从0开始

\*/

**const int** INF = 0x3f3f3f3f; **const int** MAXN = 1010; **const int** MAXM = 40010;

**struct** Edge

{

**int** u,v,cost;

};

Edge edge[MAXM];

**int** pre[MAXN],id[MAXN],visit[MAXN],in[MAXN];

**int zhuliu**(**int** root,**int** n,**int** m,Edge edge[])

{

**int** res = 0,u,v;

**while**(1)

{

**for**(**int** i = 0;i < n;i++) in[i] = INF;

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

**if**(edge[i].u != edge[i].v && edge[i].cost < in[edge[i].v])

{

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

in[edge[i].v] = edge[i].cost;

}

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

**if**(i != root && in[i] == INF)

**return** -1;//不存在最小树形图

**int** tn = 0;

**memset**(id,-1,**sizeof**(id)); **memset**(visit,-1,**sizeof**(visit)); in[root] = 0;

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

{

res += in[i]; v = i;

**while**( visit[v] != i && id[v] == -1 && v != root)

{

visit[v] = i; v = pre[v];

}

**if**( v != root && id[v] == -1 )

{

**for**(**int** u = pre[v]; u != v ;u = pre[u]) id[u] = tn;

id[v] = tn++;

}

}

**if**(tn == 0)**break**;//没有有向环

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

**if**(id[i] == -1) id[i] = tn++;

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

{

v = edge[i].v;

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

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

**if**(edge[i].u != edge[i].v)

edge[i].cost -= in[v];

}

n = tn;

root = id[root];

}

**return** res;

}

**int** g[MAXN][MAXN];

int main()

{

**int** n,m;

**int** iCase = 0; **int** T; **scanf**("%d",&T); **while**( T-- )

{

iCase ++; **scanf**("%d%d",&n,&m); **for**(**int** i = 0;i < n;i++)

**for**(**int** j = 0;j < n;j++) g[i][j] = INF;

**int** u,v,cost;

**while**(m--)

{

**scanf**("%d%d%d",&u,&v,&cost); **if**(u == v)**continue**;

g[u][v] = min(g[u][v],cost);

}

**int** L = 0;

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

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

**if**(g[i][j] < INF)

{

edge[L].u = i;

edge[L].v = j; edge[L++].cost = g[i][j];

}

**int** ans = zhuliu(0,n,L,edge);

**printf**("Case #%d: ",iCase);

**if**(ans == -1)**printf**("Possums!\n");

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

}

**return** 0;

}

## 10、二分图匹配

1）一个二分图中的最大匹配数等于这个图中的最小点覆盖数

König 定理是一个二分图中很重要的定理，它的意思是，一个二分图中的最大匹配数等于这个图中的最小 点覆盖数。如果你还不知道什么是最小点覆盖，我也在这里说一下：假如选了一个点就相当于覆盖了以它 为端点的所有边，你需要选择最少的点来覆盖所有的边。

2）最小路径覆盖＝｜G｜－最大匹配数

在一个 N\*N 的有向图中，路径覆盖就是在图中找一些路经，使之覆盖了图中的所有顶点，且任何一个顶点 有且只有一条路径与之关联；（如果把这些路径中的每条路径从它的起始点走到它的终点，那么恰好可以经 过图中的每个顶点一次且仅一次）；如果不考虑图中存在回路，那么每每条路径就是一个弱连通子集． 由上面可以得出：

1.一个单独的顶点是一条路径；

2.如果存在一路径 p1,p2,......pk，其中 p1 为起点，pk 为终点，那么在覆盖图中，顶点 p1,p2,......pk 不再 与其它的

顶点之间存在有向边． 最小路径覆盖就是找出最小的路径条数，使之成为 G 的一个路径覆盖． 路径覆盖与二分图匹配的关系：最小路径覆盖＝｜G｜－最大匹配数；

3）二分图最大独立集=顶点数-二分图最大匹配

独立集：图中任意两个顶点都不相连的顶点集合。

10.1 邻接矩阵（匈牙利算法）

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

//二分图匹配（匈牙利算法的DFS实现）(邻接矩阵形式)

//初始化：g[][]两边顶点的划分情况

//建立g[i][j]表示i->j的有向边就可以了，是左边向右边的匹配

//g没有边相连则初始化为0

//uN是匹配左边的顶点数，vN是匹配右边的顶点数

//调用：res=hungary();输出最大匹配数

//优点：适用于稠密图，DFS找增广路，实现简洁易于理解

//时间复杂度:O(VE)

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

//顶点编号从0开始的

**const int** MAXN = 510;

**int** uN,vN;//u,v的数目，使用前面必须赋值

**int** g[MAXN][MAXN];//邻接矩阵

**int** linker[MAXN]; **bool** used[MAXN]; **bool dfs**(**int** u)

{

**for**(**int** v = 0; v < vN;v++)

**if**(g[u][v] && !used[v])

{

used[v] = **true**;

**if**(linker[v] == -1 || dfs(linker[v]))

{

linker[v] = u;

return true;

}

}

return false;

}

int hungary()

{

**int** res = 0;

**memset**(linker,-1,**sizeof**(linker)); **for**(**int** u = 0;u < uN;u++)

{

**memset**(used,**false**,**sizeof**(used)); **if**(dfs(u))res++;

}

**return** res;

}

10.2 邻接表（匈牙利算法）

/\*

\* 匈牙利算法邻接表形式

\* 使用前用init()进行初始化，给uN赋值

* 加边使用函数addedge(u,v)

\*

\*/

**const int** MAXN = 5010;//点数的最大值

**const int** MAXM = 50010;//边数的最大值

**struct** Edge

{

**int** to,next;

}edge[MAXM];

**int** head[MAXN],tot;

void init()

{

tot = 0;

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

}

void addedge(int u,int v)

{

edge[tot].to = v; edge[tot].next = head[u]; head[u] = tot++;

}

**int** linker[MAXN]; **bool** used[MAXN]; **int** uN;

bool dfs(int u)

{

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

{

**int** v = edge[i].to;

**if**(!used[v])

{

used[v] = **true**;

**if**(linker[v] == -1 || dfs(linker[v]))

{

linker[v] = u;

return true;

}

}

}

return false;

}

int hungary()

{

**int** res = 0;

**memset**(linker,-1,**sizeof**(linker)); **for**(**int** u = 0; u < uN;u++)//点的编号0~uN-1

{

**memset**(used,**false**,**sizeof**(used)); **if**(dfs(u))res++;

}

**return** res;

}

10.3 Hopcroft-Carp 算法

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

* 二分图匹配（Hopcroft-Carp算法）

\* 复杂度O(sqrt(n)\*E)

\* 邻接表存图，vector实现

\* vector先初始化，然后假如边

\* uN 为左端的顶点数，使用前赋值(点编号0开始)

\*/

**const int** MAXN = 3000; **const int** INF = 0x3f3f3f3f; vector<**int**>G[MAXN];

**int** uN;

**int** Mx[MAXN],My[MAXN];

**int** dx[MAXN],dy[MAXN];

**int** dis;

**bool** used[MAXN];

bool SearchP()

{

queue<**int**>Q; dis = INF;

**memset**(dx,-1,**sizeof**(dx));

**memset**(dy,-1,**sizeof**(dy)); **for**(**int** i = 0 ; i < uN; i++)

**if**(Mx[i] == -1)

{

Q.push(i);

dx[i] = 0;

}

**while**(!Q.empty())

{

**int** u = Q.front(); Q.pop();

**if**(dx[u] > dis)**break**; **int** sz = G[u].size(); **for**(**int** i = 0;i < sz;i++)

{

**int** v = G[u][i];

**if**(dy[v] == -1)

{

dy[v] = dx[u] + 1;

**if**(My[v] == -1)dis = dy[v];

else

{

dx[My[v]] = dy[v] + 1;

Q.push(My[v]);

}

}

}

}

**return** dis != INF;

}

bool DFS(int u)

{

**int** sz = G[u].size();

**for**(**int** i = 0;i < sz;i++)

{

**int** v = G[u][i];

**if**(!used[v] && dy[v] == dx[u] + 1)

{

used[v] = **true**;

**if**(My[v] != -1 && dy[v] == dis)**continue**;

**if**(My[v] == -1 || DFS(My[v]))

{

My[v] = u;

Mx[u] = v;

return true;

}

}

}

return false;

}

int MaxMatch()

{

**int** res = 0;

**memset**(Mx,-1,**sizeof**(Mx));

**memset**(My,-1,**sizeof**(My)); **while**(SearchP())

{

**memset**(used,**false**,**sizeof**(used)); **for**(**int** i = 0;i < uN;i++)

**if**(Mx[i] == -1 && DFS(i)) res++;

}

**return** res;

}

## 11、生成树计数

Matrix-Tree 定理(Kirchhoff 矩阵-树定理)

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

2、G 的邻接矩阵 A[G]也是一个 n\*n 的矩阵， 并且满足：如果 vi、vj 之间有边直接相连，则 aij=1，否则 为 0。

我们定义 G 的 Kirchhoff 矩阵(也称为拉普拉斯算子)C[G]为 C[G]=D[G]-A[G]，则 Matrix-Tree 定理可以 描述为：G 的所有不同的生成树的个数等于其 Kirchhoff 矩阵 C[G]任何一个 n-1 阶主子式的行列式的绝对 值。所谓 n-1 阶主子式，就是对于 r(1≤r≤n)，将 C[G]的第 r 行、第 r 列同时去掉后得到的新矩阵，用 Cr[G] 表示。

// HDU 4305

// 求生成树计数部分代码，计数对 10007 取模

**const int** MOD = 10007;

**int** INV[MOD];

//求ax = 1( mod m) 的x值，就是逆元(0<a<m)

long long inv(long long a,long long m)

{

**if**(a == 1)**return** 1;

**return** inv(m%a,m)\*(m-m/a)%m;

}

**struct** Matrix

{

**int** mat[330][330];

void init()

{

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

}

**int det**(**int** n)//求行列式的值模上MOD，需要使用逆元

{

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

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

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

**int** res = 1;

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

{

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

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

{

**for**(**int** k = i;k < n;k++) swap(mat[i][k],mat[j][k]);

**if**(i != j)

res = (-res+MOD)%MOD;

break;

}

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

{

res = -1;//不存在(也就是行列式值为0)

break;

}

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

{

//int mut = (mat[j][i]\*INV[mat[i][i]])%MOD;//打表逆元

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

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

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

}

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

}

**return** res;

}

};

Matrix ret; ret.init();

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

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

**if**(i != j && g[i][j])

{

ret.mat[i][j] = -1;

ret.mat[i][i]++;

}

**printf**("%d\n",ret.det(n-1));

计算生成树个数，不取模，SPOJ 104

**#include** <stdio.h>

**#include** <string.h>

**#include** <algorithm>

**#include** <iostream>

**#include** <math.h>

using namespace std;

**const double** eps = 1e-8; **const int** MAXN = 110; **int sgn**(**double** x)

{

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

**if**(x < 0)**return** -1;

else return 1;

}

**double** b[MAXN][MAXN];

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

{

**int** i, j, k, sign = 0;

**double** ret = 1;

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

**for**(j = 0;j < n;j++) b[i][j] = a[i][j];

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

{

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

{

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

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

break;

**if**(j == n)**return** 0;

**for**(k = i;k < n;k++) swap(b[i][k],b[j][k]);

sign++;

}

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

**for**(k = i + 1;k < n;k++) b[i][k]/=b[i][i];

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

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

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

}

**if**(sign & 1)ret = -ret;

**return** ret;

}

**double** a[MAXN][MAXN];

**int** g[MAXN][MAXN];

int main()

{

**int** T; **int** n,m; **int** u,v;

**scanf**("%d",&T); **while**(T--)

{

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

**memset**(g,0,**sizeof**(g)); **while**(m--)

{

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

u--;v--;

g[u][v] = g[v][u] = 1;

}

**memset**(a,0,**sizeof**(a)); **for**(**int** i = 0;i < n;i++)

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

**if**(i != j && g[i][j])

{

a[i][i]++;

a[i][j] = -1;

}

**double** ans = det(a,n-1);

**printf**("%.0lf\n",ans);

}

**return** 0;

}

## 11、二分图多重匹配

**const int** MAXN = 1010; **const int** MAXM = 510; **int** uN,vN;

**int** g[MAXN][MAXM];

**int** linker[MAXM][MAXN];

**bool** used[MAXM];

**int** num[MAXM];//右边最大的匹配数

bool dfs(int u)

{

**for**(**int** v = 0; v < vN;v++)

**if**(g[u][v] && !used[v])

{

used[v] = **true**; **if**(linker[v][0] < num[v])

{

linker[v][++linker[v][0]] = u;

return true;

}

**for**(**int** i = 1;i <= num[v];i++)

**if**(dfs(linker[v][i]))

{

linker[v][i] = u;

return true;

}

}

return false;

}

int hungary()

{

**int** res = 0;

**for**(**int** i = 0;i < vN;i++) linker[i][0] = 0;

**for**(**int** u = 0; u < uN; u++)

{

**memset**(used,**false**,**sizeof**(used)); **if**(dfs(u))res++;

}

**return** res;

}

## 12、KM 算法（二分图最大权匹配）

/\* KM算法

* 复杂度O(nx\*nx\*ny)

\* 求最大权匹配

\* 若求最小权匹配，可将权值取相反数，结果取相反数

\* 点的编号从0开始

\*/

**const int** N = 310;

**const int** INF = 0x3f3f3f3f;

**int** nx,ny;//两边的点数

**int** g[N][N];//二分图描述

**int** linker[N],lx[N],ly[N];//y中各点匹配状态，x,y中的点标号

**int** slack[N];

**bool** visx[N],visy[N];

bool DFS(int x)

{

visx[x] = **true**;

**for**(**int** y = 0; y < ny; y++)

{

**if**(visy[y])**continue**;

**int** tmp = lx[x] + ly[y] - g[x][y];

**if**(tmp == 0)

{

visy[y] = **true**;

**if**(linker[y] == -1 || DFS(linker[y]))

{

linker[y] = x;

return true;

}

}

**else if**(slack[y] > tmp) slack[y] = tmp;

}

return false;

}

**int KM**()

{

**memset**(linker,-1,**sizeof**(linker)); **memset**(ly,0,**sizeof**(ly));

**for**(**int** i = 0;i < nx;i++)

{

lx[i] = -INF;

**for**(**int** j = 0;j < ny;j++)

**if**(g[i][j] > lx[i])

lx[i] = g[i][j];

}

**for**(**int** x = 0;x < nx;x++)

{

**for**(**int** i = 0;i < ny;i++) slack[i] = INF;

while(true)

{

**memset**(visx,**false**,**sizeof**(visx)); **memset**(visy,**false**,**sizeof**(visy)); **if**(DFS(x))**break**;

**int** d = INF;

**for**(**int** i = 0;i < ny;i++)

**if**(!visy[i] && d > slack[i]) d = slack[i];

**for**(**int** i = 0;i < nx;i++)

**if**(visx[i])

lx[i] -= d;

**for**(**int** i = 0;i < ny;i++)

{

**if**(visy[i])ly[i] += d;

**else** slack[i] -= d;

}

}

}

**int** res = 0;

**for**(**int** i = 0;i < ny;i++)

**if**(linker[i] != -1)

res += g[linker[i]][i];

**return** res;

}

//HDU 2255

int main()

{

**int** n;

**while**(**scanf**("%d",&n) == 1)

{

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

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

**scanf**("%d",&g[i][j]); nx = ny = n; **printf**("%d\n",KM());

}

**return** 0;

}

## 13、最大流

13.1 SAP 邻接矩阵形式

/\*

\* SAP算法（矩阵形式）

\* 结点编号从0开始

\*/

**const int** MAXN=1100;

**int** maze[MAXN][MAXN];

**int** gap[MAXN],dis[MAXN],pre[MAXN],cur[MAXN];

**int sap**(**int** start,**int** end,**int** nodenum)

{

**memset**(cur,0,**sizeof**(cur)); **memset**(dis,0,**sizeof**(dis)); **memset**(gap,0,**sizeof**(gap));

**int** u=pre[start]=start,maxflow=0,aug=-1; gap[0]=nodenum; **while**(dis[start]<nodenum)

{

loop:

**for**(**int** v=cur[u];v<nodenum;v++)

**if**(maze[u][v] && dis[u]==dis[v]+1)

{

**if**(aug==-1 || aug>maze[u][v])aug=maze[u][v]; pre[v]=u;

u=cur[u]=v;

**if**(v==end)

{

maxflow+=aug;

**for**(u=pre[u];v!=start;v=u,u=pre[u])

{

maze[u][v]-=aug;

maze[v][u]+=aug;

}

aug=-1;

}

**goto** loop;

}

**int** mindis=nodenum-1;

**for**(**int** v=0;v<nodenum;v++)

**if**(maze[u][v]&&mindis>dis[v])

{

cur[u]=v; mindis=dis[v];

}

**if**((--gap[dis[u]])==0)**break**; gap[dis[u]=mindis+1]++; u=pre[u];

}

**return** maxflow;

}

13.2 SAP 邻接矩阵形式 2

保留原矩阵，可用于多次使用最大流

/\*

\* SAP邻接矩阵形式

\* 点的编号从0开始

\* 增加个flow数组，保留原矩阵maze,可用于多次使用最大流

\*/

**const int** MAXN=1100;

**int** maze[MAXN][MAXN];

**int** gap[MAXN],dis[MAXN],pre[MAXN],cur[MAXN];

**int** flow[MAXN][MAXN];//存最大流的容量

**int sap**(**int** start,**int** end,**int** nodenum)

{

**memset**(cur,0,**sizeof**(cur)); **memset**(dis,0,**sizeof**(dis)); **memset**(gap,0,**sizeof**(gap));

**memset**(flow,0,**sizeof**(flow));

**int** u=pre[start]=start,maxflow=0,aug=-1; gap[0]=nodenum; **while**(dis[start]<nodenum)

{

loop:

**for**(**int** v=cur[u];v<nodenum;v++)

**if**(maze[u][v]-flow[u][v] && dis[u]==dis[v]+1)

{

**if**(aug==-1 ||

aug>maze[u][v]-flow[u][v])aug=maze[u][v]-flow[u][v]; pre[v]=u;

u=cur[u]=v;

**if**(v==end)

{

maxflow+=aug;

**for**(u=pre[u];v!=start;v=u,u=pre[u])

{

flow[u][v]+=aug;

flow[v][u]-=aug;

}

aug=-1;

}

**goto** loop;

}

**int** mindis=nodenum-1;

**for**(**int** v=0;v<nodenum;v++)

**if**(maze[u][v]-flow[u][v]&&mindis>dis[v])

{

cur[u]=v; mindis=dis[v];

}

**if**((--gap[dis[u]])==0)**break**; gap[dis[u]=mindis+1]++; u=pre[u];

}

**return** maxflow;

}

13.3 ISAP 邻接表形式

**const int** MAXN = 100010;//点数的最大值 **const int** MAXM = 400010;//边数的最大值 **const int** INF = 0x3f3f3f3f;

**struct** Edge

{

**int** to,next,cap,flow;

}edge[MAXM];//注意是MAXM

**int** tol;

**int** head[MAXN];

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

void init()

{

tol = 0;

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

}

//加边，单向图三个参数，双向图四个参数

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

{

edge[tol].to = v;edge[tol].cap = w;edge[tol].next = head[u]; edge[tol].flow = 0;head[u] = tol++;

edge[tol].to = u;edge[tol].cap = rw;edge[tol].next = head[v]; edge[tol].flow = 0;head[v]=tol++;

}

//输入参数：起点、终点、点的总数

//点的编号没有影响，只要输入点的总数

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

{

**memset**(gap,0,**sizeof**(gap)); **memset**(dep,0,**sizeof**(dep)); **memcpy**(cur,head,**sizeof**(head)); **int** u = start;

pre[u] = -1;

gap[0] = N;

**int** ans = 0;

**while**(dep[start] < N)

{

**if**(u == end)

{

**int** Min = INF;

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

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

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

{

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

}

u = start; ans += Min; **continue**;

}

**bool** flag = **false**; **int** v;

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

{

v = edge[i].to;

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

{

flag = **true**;

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

break;

}

}

**if**(flag)

{

u = v;

continue;

}

**int** Min = N;

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

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

{

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

}

gap[dep[u]]--;

**if**(!gap[dep[u]])**return** ans; dep[u] = Min+1; gap[dep[u]]++;

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

}

**return** ans;

}

13.4 ISAP+bfs 初始化+栈优化

**const int** MAXN = 100010;//点数的最大值 **const int** MAXM = 400010;//边数的最大值 **const int** INF = 0x3f3f3f3f;

**struct** Edge

{

**int** to,next,cap,flow;

}edge[MAXM];//注意是MAXM

**int** tol;

**int** head[MAXN];

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

void init()

{

tol = 0;

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

}

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

{

edge[tol].to = v; edge[tol].cap = w; edge[tol].flow = 0; edge[tol].next = head[u]; head[u] = tol++;

edge[tol].to = u; edge[tol].cap = rw; edge[tol].flow = 0; edge[tol].next = head[v]; head[v] = tol++;

}

**int** Q[MAXN];

**void BFS**(**int** start,**int** end)

{

**memset**(dep,-1,**sizeof**(dep)); **memset**(gap,0,**sizeof**(gap)); gap[0] = 1;

**int** front = 0, rear = 0; dep[end] = 0;

Q[rear++] = end;

**while**(front != rear)

{

**int** u = Q[front++];

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

{

**int** v = edge[i].to; **if**(dep[v] != -1)**continue**; Q[rear++] = v;

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

gap[dep[v]]++;

}

}

}

**int** S[MAXN];

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

{

BFS(start,end);

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

**int** top = 0; **int** u = start; **int** ans = 0;

**while**(dep[start] < N)

{

**if**(u == end)

{

**int** Min = INF;

**int** inser;

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

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

{

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

}

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

{

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

}

ans += Min; top = inser;

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

continue;

}

**bool** flag = **false**; **int** v;

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

{

v = edge[i].to;

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

{

flag = **true**; cur[u] = i; **break**;

}

}

**if**(flag)

{

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

continue;

}

**int** Min = N;

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

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

{

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

}

gap[dep[u]]--;

**if**(!gap[dep[u]])**return** ans; dep[u] = Min + 1; gap[dep[u]]++;

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

}

**return** ans;

}

## 14、最小费用最大流

最小费用最大流，求最大费用只需要取相反数，结果取相反数即可。 点的总数为 N，点的编号 0~N-1

**const int** MAXN = 10000; **const int** MAXM = 100000; **const int** INF = 0x3f3f3f3f; **struct** Edge

{

**int** to,next,cap,flow,cost;

}edge[MAXM];

**int** head[MAXN],tol;

**int** pre[MAXN],dis[MAXN];

**bool** vis[MAXN];

**int** N;//节点总个数，节点编号从0~N-1

void init(int n)

{

N = n;

tol = 0;

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

}

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

{

edge[tol].to = v; edge[tol].cap = cap; edge[tol].cost = cost; edge[tol].flow = 0; edge[tol].next = head[u]; head[u] = tol++; edge[tol].to = u; edge[tol].cap = 0; edge[tol].cost = -cost; edge[tol].flow = 0; edge[tol].next = head[v]; head[v] = tol++;

}

**bool spfa**(**int** s,**int** t)

{

queue<**int**>q;

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

{

dis[i] = INF; vis[i] = **false**; pre[i] = -1;

}

dis[s] = 0; vis[s] = **true**; q.push(s); **while**(!q.empty())

{

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

vis[u] = **false**;

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

{

**int** v = edge[i].to;

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

{

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

**if**(!vis[v])

{

vis[v] = **true**; q.push(v);

}

}

}

}

**if**(pre[t] == -1)**return false**; **else return true**;

}

//返回的是最大流，cost存的是最小费用

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

{

**int** flow = 0; cost = 0; **while**(spfa(s,t))

{

**int** Min = INF;

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

{

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

}

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

{

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

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

}

flow += Min;

}

**return** flow;

}

## 15、2-SAT

15.1 染色法（可以得到字典序最小的解）

HDU 1814

**const int** MAXN = 20020; **const int** MAXM = 100010; **struct** Edge

{

**int** to,next;

}edge[MAXM];

**int** head[MAXN],tot;

void init()

{

tot = 0;

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

}

void addedge(int u,int v)

{

edge[tot].to = v;edge[tot].next = head[u];head[u] = tot++;

}

**bool** vis[MAXN];//染色标记，为true表示选择

**int** S[MAXN],top;//栈

bool dfs(int u)

{

**if**(vis[u^1])**return false**; **if**(vis[u])**return true**; vis[u] = **true**;

S[top++] = u;

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

**if**(!dfs(edge[i].to)) **return false**;

return true;

}

bool Twosat(int n)

{

**memset**(vis,**false**,**sizeof**(vis)); **for**(**int** i = 0;i < n;i += 2)

{

**if**(vis[i] || vis[i^1])**continue**; top = 0;

**if**(!dfs(i))

{

**while**(top)vis[S[--top]] = **false**; **if**(!dfs(i^1)) **return false**;

}

}

return true;

}

int main()

{

**int** n,m;

**int** u,v;

**while**(**scanf**("%d%d",&n,&m) == 2)

{

init();

**while**(m--)

{

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

u--;v--;

addedge(u,v^1); addedge(v,u^1);

}

**if**(Twosat(2\*n))

{

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

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

}

**else printf**("NIE\n");

}

**return** 0;

}

15.2 强连通缩点法（拓扑排序只能得到任意解）

POJ 3648 Wedding

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

//2-SAT 强连通缩点

**const int** MAXN = 1010; **const int** MAXM = 100010; **struct** Edge

{

**int** to,next;

}edge[MAXM];

**int** head[MAXN],tot;

void init()

{

tot = 0;

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

}

void addedge(int u,int v)

{

edge[tot].to = v; edge[tot].next = head[u]; head[u] = tot++;

}

**int** Low[MAXN],DFN[MAXN],Stack[MAXN],Belong[MAXN];//Belong数组的值1~scc

**int** Index,top;

**int** scc;

**bool** Instack[MAXN];

**int** num[MAXN];

void Tarjan(int u)

{

**int** v;

Low[u] = DFN[u] = ++Index; Stack[top++] = u; Instack[u] = **true**;

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

{

v = edge[i].to;

**if**( !DFN[v] )

{

Tarjan(v);

**if**(Low[u] > Low[v])Low[u] = Low[v];

}

**else if**(Instack[v] && Low[u] > DFN[v]) Low[u] = DFN[v];

}

**if**(Low[u] == DFN[u])

{

scc++;

do

{

v = Stack[--top]; Instack[v] = **false**; Belong[v] = scc; num[scc]++;

}

**while**(v != u);

}

}

**bool solvable**(**int** n)//n是总个数,需要选择一半

{

**memset**(DFN,0,**sizeof**(DFN)); **memset**(Instack,**false**,**sizeof**(Instack));

**memset**(num,0,**sizeof**(num)); Index = scc = top = 0; **for**(**int** i = 0;i < n;i++)

**if**(!DFN[i])

Tarjan(i);

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

{

**if**(Belong[i] == Belong[i^1])

return false;

}

return true;

}

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

//拓扑排序求任意一组解部分 queue<**int**>q1,q2;

vector<vector<**int**> > dag;//缩点后的逆向DAG图

**char** color[MAXN];//染色，为'R'是选择的

**int** indeg[MAXN];//入度

**int** cf[MAXN];

void solve(int n)

{

dag.assign(scc+1,vector<**int**>()); **memset**(indeg,0,**sizeof**(indeg)); **memset**(color,0,**sizeof**(color)); **for**(**int** u = 0;u < n;u++)

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

{

**int** v = edge[i].to;

**if**(Belong[u] != Belong[v])

{

dag[Belong[v]].push\_back(Belong[u]); indeg[Belong[u]]++;

}

}

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

{

cf[Belong[i]] = Belong[i^1]; cf[Belong[i^1]] = Belong[i];

}

**while**(!q1.empty())q1.pop();

**while**(!q2.empty())q2.pop(); **for**(**int** i = 1;i <= scc;i++)

**if**(indeg[i] == 0) q1.push(i);

**while**(!q1.empty())

{

**int** u = q1.front(); q1.pop(); **if**(color[u] == 0)

{

color[u] = 'R';

color[cf[u]] = 'B';

}

**int** sz = dag[u].size();

**for**(**int** i = 0;i < sz;i++)

{

indeg[dag[u][i]]--;

**if**(indeg[dag[u][i]] == 0)

q1.push(dag[u][i]);

}

}

}

**int change**(**char** s[])

{

**int** ret = 0;

**int** i = 0;

**while**(s[i]>='0' && s[i]<='9')

{

ret \*= 10;

ret += s[i]-'0'; i++;

}

**if**(s[i] == 'w')**return** 2\*ret;

**else return** 2\*ret+1;

}

int main()

{

**int** n,m;

**char** s1[10],s2[10];

**while**(**scanf**("%d%d",&n,&m) == 2)

{

**if**(n == 0 && m == 0)**break**; init();

**while**(m--)

{

**scanf**("%s%s",s1,s2); **int** u = change(s1); **int** v = change(s2); addedge(u^1,v); addedge(v^1,u);

}

addedge(1,0);

**if**(solvable(2\*n))

{

solve(2\*n);

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

{

//注意这一定是判断color[Belong[

**if**(color[Belong[2\*i]] == 'R')**printf**("%dw",i);

**else printf**("%dh",i); **if**(i < n-1)**printf**(" "); **else printf**("\n");

}

}

**else printf**("bad luck\n");

}

**return** 0;

}

## 16、曼哈顿最小生成树

POJ 3241 求曼哈顿最小生成树上第 k 大的边

**const int** MAXN = 100010; **const int** INF = 0x3f3f3f3f; **struct** Point

{

**int** x,y,id;

}p[MAXN];

**bool cmp**(Point a,Point b)

{

**if**(a.x != b.x) **return** a.x < b.x;

**else return** a.y < b.y;

}

//树状数组，找y-x大于当前的，但是y+x最小的

**struct** BIT

{

**int** min\_val,pos;

void init()

{

min\_val = INF; pos = -1;

}

}bit[MAXN];

//所有有效边

**struct** Edge

{

**int** u,v,d;

}edge[MAXN<<2];

**bool cmpedge**(Edge a,Edge b)

{

**return** a.d < b.d;

}

**int** tot;

**int** n;

**int** F[MAXN];

int find(int x)

{

**if**(F[x] == -1) **return** x;

**else return** F[x] = find(F[x]);

}

**void addedge**(**int** u,**int** v,**int** d)

{

edge[tot].u = u; edge[tot].v = v; edge[tot++].d = d;

}

int lowbit(int x)

{

**return** x&(-x);

}

**void update**(**int** i,**int** val,**int** pos)

{

**while**(i > 0)

{

**if**(val < bit[i].min\_val)

{

bit[i].min\_val = val; bit[i].pos = pos;

}

i -= lowbit(i);

}

}

**int ask**(**int** i,**int** m)//查询[i,m]的最小值位置

{

**int** min\_val = INF,pos = -1;

**while**(i <= m)

{

**if**(bit[i].min\_val < min\_val)

{

min\_val = bit[i].min\_val; pos = bit[i].pos;

}

i += lowbit(i);

}

**return** pos;

}

**int dist**(Point a,Point b)

{

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

}

**void Manhattan\_minimum\_spanning\_tree**(**int** n,Point p[])

{

**int** a[MAXN],b[MAXN];

tot = 0;

**for**(**int** dir = 0; dir < 4;dir++)

{

//4种坐标变换

**if**(dir == 1 || dir == 3)

{

**for**(**int** i = 0;i < n;i++) swap(p[i].x,p[i].y);

}

**else if**(dir == 2)

{

**for**(**int** i = 0;i < n;i++) p[i].x = -p[i].x;

}

sort(p,p+n,cmp);

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

a[i] = b[i] = p[i].y - p[i].x; sort(b,b+n);

**int** m = unique(b,b+n) - b;

**for**(**int** i = 1;i <= m;i++) bit[i].init();

**for**(**int** i = n-1 ;i >= 0;i--)

{

**int** pos = lower\_bound(b,b+m,a[i]) - b + 1;

**int** ans = ask(pos,m);

**if**(ans != -1) addedge(p[i].id,p[ans].id,dist(p[i],p[ans]));

update(pos,p[i].x+p[i].y,i);

}

}

}

int solve(int k)

{

Manhattan\_minimum\_spanning\_tree(n,p);

**memset**(F,-1,**sizeof**(F));

sort(edge,edge+tot,cmpedge);

**for**(**int** i = 0;i < tot;i++)

{

**int** u = edge[i].u;

**int** v = edge[i].v;

**int** t1 = find(u), t2 = find(v);

**if**(t1 != t2)

{

F[t1] = t2;

k--;

**if**(k == 0)**return** edge[i].d;

}

}

}

int main()

{

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

//freopen("out.txt","w",stdout); **int** k; **while**(**scanf**("%d%d",&n,&k)==2 && n)

{

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

{

**scanf**("%d%d",&p[i].x,&p[i].y); p[i].id = i;

}

**printf**("%d\n",solve(n-k));

}

**return** 0;

}

## 17、一般图匹配带花树

统计match[i]>0的个数即为被匹配的“点”的个数

注意下标从1开始

URAL 1099

**const int** MAXN = 250;

**int** N; //点的个数，点的编号从1到N

**bool** Graph[MAXN][MAXN];

**int** Match[MAXN];

**bool** InQueue[MAXN],InPath[MAXN],InBlossom[MAXN];

**int** Head,Tail; **int** Queue[MAXN]; **int** Start,Finish; **int** NewBase;

**int** Father[MAXN],Base[MAXN];

**int** Count;//匹配数，匹配对数是Count/2

void CreateGraph()

{

**int** u,v; **memset**(Graph,**false**,**sizeof**(Graph)); **scanf**("%d",&N); **while**(**scanf**("%d%d",&u,&v) == 2)

{

Graph[u][v] = Graph[v][u] = **true**;

}

}

void Push(int u)

{

Queue[Tail] = u; Tail++;

InQueue[u] = **true**;

}

**int Pop**()

{

**int** res = Queue[Head]; Head++;

**return** res;

}

int FindCommonAncestor(int u,int v)

{

**memset**(InPath,**false**,**sizeof**(InPath)); **while**(**true**)

{

u = Base[u]; InPath[u] = **true**; **if**(u == Start) **break**; u = Father[Match[u]];

}

while(true)

{

v = Base[v]; **if**(InPath[v])**break**; v = Father[Match[v]];

}

**return** v;

}

void ResetTrace(int u)

{

**int** v;

**while**(Base[u] != NewBase)

{

v = Match[u];

InBlossom[Base[u]] = InBlossom[Base[v]] = **true**; u = Father[v];

**if**(Base[u] != NewBase) Father[u] = v;

}

}

void BloosomContract(int u,int v)

{

NewBase = FindCommonAncestor(u,v); **memset**(InBlossom,**false**,**sizeof**(InBlossom)); ResetTrace(u);

ResetTrace(v);

**if**(Base[u] != NewBase) Father[u] = v; **if**(Base[v] != NewBase) Father[v] = u; **for**(**int** tu = 1; tu <= N; tu++)

**if**(InBlossom[Base[tu]])

{

Base[tu] = NewBase;

**if**(!InQueue[tu]) Push(tu);

}

}

void FindAugmentingPath()

{

**memset**(InQueue,**false**,**sizeof**(InQueue)); **memset**(Father,0,**sizeof**(Father)); **for**(**int** i = 1;i <= N;i++)

Base[i] = i; Head = Tail = 1; Push(Start); Finish = 0; **while**(Head < Tail)

{

**int** u = Pop();

**for**(**int** v = 1; v <= N; v++)

**if**(Graph[u][v] && (Base[u] != Base[v]) && (Match[u] != v))

{

**if**((v == Start) || ((Match[v] > 0) && Father[Match[v]] > 0)) BloosomContract(u,v);

**else if**(Father[v] == 0)

{

Father[v] = u;

**if**(Match[v] > 0) Push(Match[v]);

else

{

Finish = v;

return;

}

}

}

}

}

void AugmentPath()

{

**int** u,v,w; u = Finish;

**while**(u > 0)

{

v = Father[u]; w = Match[v]; Match[v] = u; Match[u] = v; u = w;

}

}

void Edmonds()

{

**memset**(Match,0,**sizeof**(Match)); **for**(**int** u = 1; u <= N; u++)

**if**(Match[u] == 0)

{

Start = u; FindAugmentingPath(); **if**(Finish > 0)AugmentPath();

}

}

void PrintMatch()

{

Count = 0;

**for**(**int** u = 1; u <= N;u++)

**if**(Match[u] > 0) Count++;

**printf**("%d\n",Count); **for**(**int** u = 1; u <= N; u++)

**if**(u < Match[u])

**printf**("%d %d\n",u,Match[u]);

}

int main()

{

CreateGraph();//建图 Edmonds();//进行匹配 PrintMatch();//输出匹配数和匹配

**return** 0;

}

## 18、LCA

* 1. dfs+ST 在线算法

/\*

* LCA (POJ 1330)
* 在线算法 DFS + ST

\*/

**const int** MAXN = 10010;

**int** rmq[2\*MAXN];//rmq数组，就是欧拉序列对应的深度序列

**struct** ST

{

**int** mm[2\*MAXN];

**int** dp[2\*MAXN][20];//最小值对应的下标

void init(int n)

{

mm[0] = -1;

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

{

mm[i] = ((i&(i-1)) == 0)?mm[i-1]+1:mm[i-1]; dp[i][0] = i;

}

**for**(**int** j = 1; j <= mm[n];j++)

**for**(**int** i = 1; i + (1<<j) - 1 <= n; i++) dp[i][j] = rmq[dp[i][j-1]] <

rmq[dp[i+(1<<(j-1))][j-1]]?dp[i][j-1]:dp[i+(1<<(j-1))][j-1];

}

**int query**(**int** a,**int** b)//查询[a,b]之间最小值的下标

{

**if**(a > b)swap(a,b);

**int** k = mm[b-a+1];

**return** rmq[dp[a][k]] <=

rmq[dp[b-(1<<k)+1][k]]?dp[a][k]:dp[b-(1<<k)+1][k];

}

};

//边的结构体定义

**struct** Edge

{

**int** to,next;

};

Edge edge[MAXN\*2];

**int** tot,head[MAXN];

**int** F[MAXN\*2];//欧拉序列，就是dfs遍历的顺序，长度为2\*n-1,下标从1开始

**int** P[MAXN];//P[i]表示点i在F中第一次出现的位置

**int** cnt;

ST st;

void init()

{

tot = 0;

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

}

**void addedge**(**int** u,**int** v)//加边，无向边需要加两次

{

edge[tot].to = v; edge[tot].next = head[u]; head[u] = tot++;

}

**void dfs**(**int** u,**int** pre,**int** dep)

{

F[++cnt] = u; rmq[cnt] = dep; P[u] = cnt;

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

{

**int** v = edge[i].to; **if**(v == pre)**continue**; dfs(v,u,dep+1); F[++cnt] = u; rmq[cnt] = dep;

}

}

**void LCA\_init**(**int** root,**int** node\_num)//查询LCA前的初始化

{

cnt = 0; dfs(root,root,0); st.init(2\*node\_num-1);

}

**int query\_lca**(**int** u,**int** v)//查询u,v的lca编号

{

**return** F[st.query(P[u],P[v])];

}

**bool** flag[MAXN];

int main()

{

**int** T; **int** N; **int** u,v;

**scanf**("%d",&T); **while**(T--)

{

**scanf**("%d",&N); init();

**memset**(flag,**false**,**sizeof**(flag)); **for**(**int** i = 1; i < N;i++)

{

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

addedge(v,u); flag[v] = **true**;

}

**int** root;

**for**(**int** i = 1; i <= N;i++)

**if**(!flag[i])

{

root = i;

break;

}

LCA\_init(root,N); **scanf**("%d%d",&u,&v); **printf**("%d\n",query\_lca(u,v));

}

**return** 0;

}

* 1. 离线 Tarjan 算法

/\*

\* POJ 1470

\* 给出一颗有向树，Q个查询

\* 输出查询结果中每个点出现次数

\*/

/\*

\* LCA离线算法，Tarjan

\* 复杂度O(n+Q);

\*/

**const int** MAXN = 1010;

**const int** MAXQ = 500010;//查询数的最大值

//并查集部分

**int** F[MAXN];//需要初始化为-1

int find(int x)

{

**if**(F[x] == -1)**return** x;

**return** F[x] = find(F[x]);

}

**void bing**(**int** u,**int** v)

{

**int** t1 = find(u); **int** t2 = find(v); **if**(t1 != t2)

F[t1] = t2;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* **bool** vis[MAXN];//访问标记 **int** ancestor[MAXN];//祖先

**struct** Edge

{

**int** to,next;

}edge[MAXN\*2];

**int** head[MAXN],tot;

void addedge(int u,int v)

{

edge[tot].to = v; edge[tot].next = head[u]; head[u] = tot++;

}

**struct** Query

{

**int** q,next;

**int** index;//查询编号

}query[MAXQ\*2];

**int** answer[MAXQ];//存储最后的查询结果，下标0~Q-1

**int** h[MAXQ];

**int** tt;

**int** Q;

**void add\_query**(**int** u,**int** v,**int** index)

{

query[tt].q = v; query[tt].next = h[u]; query[tt].index = index; h[u] = tt++;

query[tt].q = u; query[tt].next = h[v]; query[tt].index = index; h[v] = tt++;

}

void init()

{

tot = 0;

**memset**(head,-1,**sizeof**(head)); tt = 0;

**memset**(h,-1,**sizeof**(h)); **memset**(vis,**false**,**sizeof**(vis)); **memset**(F,-1,**sizeof**(F)); **memset**(ancestor,0,**sizeof**(ancestor));

}

void LCA(int u)

{

ancestor[u] = u; vis[u] = **true**;

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

{

**int** v = edge[i].to; **if**(vis[v])**continue**; LCA(v);

bing(u,v); ancestor[find(u)] = u;

}

**for**(**int** i = h[u];i != -1;i = query[i].next)

{

**int** v = query[i].q;

**if**(vis[v])

{

answer[query[i].index] = ancestor[find(v)];

}

}

}

**bool** flag[MAXN];

**int** Count\_num[MAXN];

int main()

{

**int** n;

**int** u,v,k;

**while**(**scanf**("%d",&n) == 1)

{

init(); **memset**(flag,**false**,**sizeof**(flag)); **for**(**int** i = 1;i <= n;i++)

{

**scanf**("%d:(%d)",&u,&k); **while**(k--)

{

**scanf**("%d",&v); flag[v] = **true**; addedge(u,v);

addedge(v,u);

}

}

**scanf**("%d",&Q);

**for**(**int** i = 0;i < Q;i++)

{

**char** ch; cin>>ch;

**scanf**("%d %d)",&u,&v); add\_query(u,v,i);

}

**int** root;

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

**if**(!flag[i])

{

root = i;

break;

}

LCA(root); **memset**(Count\_num,0,**sizeof**(Count\_num)); **for**(**int** i = 0;i < Q;i++)

Count\_num[answer[i]]++;

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

**if**(Count\_num[i] > 0)

**printf**("%d:%d\n",i,Count\_num[i]);

}

**return** 0;

}

* 1. LCA 倍增法

/\*

\* POJ 1330

\* LCA 在线算法

\*/

**const int** MAXN = 10010;

**const int** DEG = 20;

**struct** Edge

{

**int** to,next;

}edge[MAXN\*2];

**int** head[MAXN],tot;

void addedge(int u,int v)

{

edge[tot].to = v; edge[tot].next = head[u]; head[u] = tot++;

}

void init()

{

tot = 0;

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

}

**int** fa[MAXN][DEG];//fa[i][j]表示结点i的第2^j个祖先

**int** deg[MAXN];//深度数组

**void BFS**(**int** root)

{

queue<**int**>que; deg[root] = 0; fa[root][0] = root; que.push(root); **while**(!que.empty())

{

**int** tmp = que.front(); que.pop();

**for**(**int** i = 1;i < DEG;i++)

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

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

{

**int** v = edge[i].to;

**if**(v == fa[tmp][0])**continue**; deg[v] = deg[tmp] + 1; fa[v][0] = tmp;

que.push(v);

}

}

}

**int LCA**(**int** u,**int** v)

{

**if**(deg[u] > deg[v])swap(u,v); **int** hu = deg[u], hv = deg[v]; **int** tu = u, tv = v;

**for**(**int** det = hv-hu, i = 0; det ;det>>=1, i++)

**if**(det&1)

tv = fa[tv][i];

**if**(tu == tv)**return** tu;

**for**(**int** i = DEG-1; i >= 0; i--)

{

**if**(fa[tu][i] == fa[tv][i])

**continue**; tu = fa[tu][i];

tv = fa[tv][i];

}

**return** fa[tu][0];

}

**bool** flag[MAXN];

int main()

{

**int** T; **int** n; **int** u,v;

**scanf**("%d",&T); **while**(T--)

{

**scanf**("%d",&n); init();

**memset**(flag,**false**,**sizeof**(flag)); **for**(**int** i = 1;i < n;i++)

{

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

addedge(v,u); flag[v] = **true**;

}

**int** root;

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

**if**(!flag[i])

{

root = i;

break;

}

BFS(root);

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

**printf**("%d\n",LCA(u,v));

}

**return** 0;

}

## 19、欧拉路

欧拉回路：每条边只经过一次，而且回到起点 欧拉路径：每条边只经过一次，不要求回到起点

欧拉回路判断： 无向图：连通（不考虑度为0的点），每个顶点度数都为偶数。

有向图：基图连通（把边当成无向边，同样不考虑度为0的点），每个顶点出度等于入度。 混合图（有无向边和有向边）：首先是基图连通（不考虑度为0的点），然后需要借助网络流判定。 首先给原图中的每条无向边随便指定一个方向（称为初始定向），将原图改为有向图G'，然后的任务就是 改变G'中某些边的方向（当然是无向边转化来的，原混合图中的有向边不能动）使其满足每个点的入度等 于出度。

**设D[i]为G'中(点i的出度 - 点i的入度）。**可以发现，在改变G'中边的方向的过程中，任何点的D值的

奇偶性都不会发生改变（设将边<i, j>改为<j, i>，则i入度加1出度减1，j入度减1出度加1，两者之 差加2或减2，奇偶性不变）！而最终要求的是每个点的入度等于出度，即每个点的D值都为0，是偶数，故 可得：若初始定向得到的G'中任意一个点的D值是奇数，那么原图中一定不存在欧拉环！

若初始D值都是偶数，则将G'改装成网络：设立源点S和汇点T，对于每个D[i]>0的点i，连边<S, i>， 容量为D[i]/2；对于每个D[j]<0的点j，连边<j, T>，容量为-D[j]/2；G'中的每条边在网络中仍保 留，容量为1（表示该边最多只能被改变方向一次）。求这个网络的最大流，若S引出的所有边均满流，则 原混合图是欧拉图，将网络中所有流量为1的中间边（就是不与S或T关联的边）在G'中改变方向，形成的 新图G''一定是有向欧拉图；若S引出的边中有的没有满流，则原混合图不是欧拉图。

欧拉路径的判断： 无向图：连通（不考虑度为0的点），每个顶点度数都为偶数或者仅有两个点的度数为偶数。 有向图：基图连通（把边当成无向边，同样不考虑度为0的点），每个顶点出度等于入度 或者 有且仅有

一个点的出度比入度多1，有且仅有一个点的出度比入度少1，其余出度等于入度。

混合图： 如果存在欧拉回路，一点存在欧拉路径了。否则如果有且仅有两个点的（出度-入度）是奇数， 那么给这个两个点加边，判断是否存在欧拉回路。

19.1 有向图

POJ 2337

给出n个小写字母组成的单词，要求将n个单词连接起来，使得前一个单词的最后一个字母和后一个单词的 第一个字母相同。输出字典序最小的解。

**struct** Edge

{

**int** to,next; **int** index; **bool** flag;

}edge[2010];

**int** head[30],tot;

void init()

{

tot = 0;

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

}

**void addedge**(**int** u,**int** v,**int** index)

{

edge[tot].to = v; edge[tot].next = head[u]; edge[tot].index = index; edge[tot].flag = **false**; head[u] = tot++;

}

string str[1010]; **int** in[30],out[30]; **int** cnt;

**int** ans[1010];

void dfs(int u)

{

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

**if**(!edge[i].flag)

{

edge[i].flag = **true**; dfs(edge[i].to);

ans[cnt++] = edge[i].index;

}

}

int main()

{

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

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

**int** T,n; **scanf**("%d",&T); **while**(T--)

{

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

**for**(**int** i = 0;i < n;i++) cin>>str[i];

sort(str,str+n);//要输出字典序最小的解，先按照字典序排序

init(); **memset**(in,0,**sizeof**(in)); **memset**(out,0,**sizeof**(out)); **int** start = 100;

**for**(**int** i = n-1;i >= 0;i--)//字典序大的先加入

{

**int** u = str[i][0] - 'a';

**int** v = str[i][str[i].length() - 1] - 'a'; addedge(u,v,i);

out[u]++;

in[v]++;

**if**(u < start)start = u;

**if**(v < start)start = v;

}

**int** cc1 = 0, cc2 = 0;

**for**(**int** i = 0;i < 26;i++)

{

**if**(out[i] - in[i] == 1)

{

cc1++;

start = i;//如果有一个出度比入度大1的点，就从这个点出发，否则从最小的点

出发

}

**else if**(out[i] - in[i] == -1) cc2++;

**else if**(out[i] - in[i] != 0) cc1 = 3;

}

**if**(! ( (cc1 == 0 && cc2 == 0) || (cc1 == 1 && cc2 == 1) ))

{

**printf**("\*\*\*\n"); **continue**;

}

cnt = 0; dfs(start);

**if**(cnt != n)//判断是否连通

{

**printf**("\*\*\*\n"); **continue**;

}

**for**(**int** i = cnt-1; i >= 0;i--)

{

cout<<str[ans[i]]; **if**(i > 0)**printf**("."); **else printf**("\n");

}

}

**return** 0;

}

19.2 无向图

SGU 101

**struct** Edge

{

**int** to,next; **int** index; **int** dir; **bool** flag;

}edge[220];

**int** head[10],tot;

void init()

{

**memset**(head,-1,**sizeof**(head)); tot = 0;

}

**void addedge**(**int** u,**int** v,**int** index)

{

edge[tot].to = v; edge[tot].next = head[u]; edge[tot].index = index; edge[tot].dir = 0; edge[tot].flag = **false**; head[u] = tot++; edge[tot].to = u; edge[tot].next = head[v]; edge[tot].index = index; edge[tot].dir = 1; edge[tot].flag = **false**; head[v] = tot++;

}

**int** du[10]; vector<**int**>ans; **void dfs**(**int** u)

{

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

**if**(!edge[i].flag )

{

edge[i].flag = **true**; edge[i^1].flag = **true**;

dfs(edge[i].to); ans.push\_back(i);

}

}

int main()

{

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

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

**int** n;

**while**(**scanf**("%d",&n) == 1)

{

init();

**int** u,v; **memset**(du,0,**sizeof**(du)); **for**(**int** i = 1;i <= n;i++)

{

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

addedge(u,v,i); du[u]++;

du[v]++;

}

**int** s = -1;

**int** cnt = 0;

**for**(**int** i = 0;i <= 6;i++)

{

**if**(du[i]&1) {cnt++; s = i;}

**if**(du[i] > 0 && s == -1) s = i;

}

**bool** ff = **true**;

**if**(cnt != 0 && cnt != 2)

{

**printf**("No solution\n");

continue;

}

ans.clear();

dfs(s);

**if**(ans.size() != n)

{

**printf**("No solution\n");

continue;

}

**for**(**int** i = 0;i < ans.size();i++)

{

**printf**("%d ",edge[ans[i]].index);

**if**(edge[ans[i]].dir == 0)**printf**("-\n");

**else printf**("+\n");

}

}

**return** 0;

}

19.3 混合图

POJ 1637 （本题保证了连通，故不需要判断连通，否则要判断连通）

**const int** MAXN = 210;

//最大流ISAP部分

**const int** MAXM = 20100;

**const int** INF = 0x3f3f3f3f;

**struct** Edge

{

**int** to,next,cap,flow;

}edge[MAXM];

**int** tol;

**int** head[MAXN];

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

void init()

{

tol = 0;

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

}

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

{

edge[tol].to = v; edge[tol].cap = w; edge[tol].next = head[u]; edge[tol].flow = 0; head[u] = tol++; edge[tol].to = u; edge[tol].cap = rw; edge[tol].next = head[v]; edge[tol].flow = 0; head[v] = tol++;

}

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

{

**memset**(gap,0,**sizeof**(gap)); **memset**(dep,0,**sizeof**(dep)); **memcpy**(cur,head,**sizeof**(head)); **int** u = start;

pre[u] = -1;

gap[0] = N;

**int** ans = 0;

**while**(dep[start] < N)

{

**if**(u == end)

{

**int** Min = INF;

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

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

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

{

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

}

u = start; ans += Min; **continue**;

}

**bool** flag = **false**; **int** v;

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

{

v = edge[i].to;

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

{

flag = **true**;

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

break;

}

}

**if**(flag)

{

u = v;

continue;

}

**int** Min = N;

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

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

{

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

}

gap[dep[u]]--;

**if**(!gap[dep[u]])**return** ans; dep[u] = Min+1; gap[dep[u]]++;

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

}

**return** ans;

}

//the end of 最大流部分

**int** in[MAXN],out[MAXN];//每个点的出度和入度

int main()

{

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

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

**int** T;

**int** n,m; **scanf**("%d",&T); **while**(T--)

{

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

**int** u,v,w; **memset**(in,0,**sizeof**(in)); **memset**(out,0,**sizeof**(out)); **while**(m--)

{

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

out[u]++; in[v]++;

**if**(w == 0)//双向 addedge(u,v,1);

}

**bool** flag = **true**; **for**(**int** i = 1;i <= n;i++)

{

**if**(out[i] - in[i] > 0) addedge(0,i,(out[i] - in[i])/2);

**else if**(in[i] - out[i] > 0) addedge(i,n+1,(in[i] - out[i])/2);

**if**((out[i] - in[i]) & 1)

flag = **false**;

}

**if**(!flag)

{

**printf**("impossible\n"); **continue**;

}

sap(0,n+1,n+2);

**for**(**int** i = head[0]; i != -1;i = edge[i].next)

**if**(edge[i].cap > 0 && edge[i].cap > edge[i].flow)

{

flag = **false**; **break**;

}

**if**(flag)**printf**("possible\n"); **else printf**("impossible\n");

}

**return** 0;

}

# 计算几何

## 1、基本函数

* 1. Point 定义

**const double** eps = 1e-8; **const double** PI = **acos**(-1.0); **int sgn**(**double** x)

{

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

**if**(x < 0)**return** -1;

else return 1;

}

**struct** Point

{

**double** x,y;

**Point**(){}

**Point**(**double** \_x,**double** \_y)

{

x = \_x;y = \_y;

}

Point **operator -**(**const** Point &b)**const**

{

**return** Point(x - b.x,y - b.y);

}

//叉积

**double operator ^**(**const** Point &b)**const**

{

**return** x\*b.y - y\*b.x;

}

//点积

**double operator \***(**const** Point &b)**const**

{

**return** x\*b.x + y\*b.y;

}

//绕原点旋转角度B（弧度值），后x,y的变化

void transXY(double B)

{

**double** tx = x,ty = y;

x = tx\***cos**(B) - ty\***sin**(B); y = tx\***sin**(B) + ty\***cos**(B);

}

};

* 1. Line 定义

**struct** Line

{

Point s,e;

**Line**(){}

**Line**(Point \_s,Point \_e)

{

s = \_s;e = \_e;

}

//两直线相交求交点

//第一个值为0表示直线重合，为1表示平行，为0表示相交,为2是相交

//只有第一个值为2时，交点才有意义

pair<**int**,Point> **operator &**(**const** Line &b)**const**

{

Point res = s;

**if**(sgn((s-e)^(b.s-b.e)) == 0)

{

**if**(sgn((s-b.e)^(b.s-b.e)) == 0)

**return** make\_pair(0,res);//重合

**else return** make\_pair(1,res);//平行

}

**double** t = ((s-b.s)^(b.s-b.e))/((s-e)^(b.s-b.e)); res.x += (e.x-s.x)\*t;

res.y += (e.y-s.y)\*t;

**return** make\_pair(2,res);

}

};

1.3 两点间距离

//\*两点间距离

**double dist**(Point a,Point b)

{

**return sqrt**((a-b)\*(a-b));

}

1.4 判断：线段相交

//\*判断线段相交

**bool inter**(Line l1,Line l2)

{

return

max(l1.s.x,l1.e.x) >= min(l2.s.x,l2.e.x) &&

max(l2.s.x,l2.e.x) >= min(l1.s.x,l1.e.x) &&

max(l1.s.y,l1.e.y) >= min(l2.s.y,l2.e.y) &&

max(l2.s.y,l2.e.y) >= min(l1.s.y,l1.e.y) &&

sgn((l2.s-l1.e)^(l1.s-l1.e))\*sgn((l2.e-l1.e)^(l1.s-l1.e)) <= 0 &&

sgn((l1.s-l2.e)^(l2.s-l2.e))\*sgn((l1.e-l2.e)^(l2.s-l2.e)) <= 0;

}

1.5 判断：直线和线段相交

//判断直线和线段相交

**bool Seg\_inter\_line**(Line l1,Line l2) //判断直线l1和线段l2是否相交

{

**return** sgn((l2.s-l1.e)^(l1.s-l1.e))\*sgn((l2.e-l1.e)^(l1.s-l1.e)) <= 0;

}

1.6 点到直线距离

//点到直线距离

//返回为result,是点到直线最近的点 Point **PointToLine**(Point P,Line L)

{

Point result;

**double** t = ((P-L.s)\*(L.e-L.s))/((L.e-L.s)\*(L.e-L.s)); result.x = L.s.x + (L.e.x-L.s.x)\*t;

result.y = L.s.y + (L.e.y-L.s.y)\*t;

**return** result;

}

1.7 点到线段距离

//点到线段的距离

//返回点到线段最近的点

Point **NearestPointToLineSeg**(Point P,Line L)

{

Point result;

**double** t = ((P-L.s)\*(L.e-L.s))/((L.e-L.s)\*(L.e-L.s));

**if**(t >= 0 && t <= 1)

{

result.x = L.s.x + (L.e.x - L.s.x)\*t;

result.y = L.s.y + (L.e.y - L.s.y)\*t;

}

else

{

**if**(dist(P,L.s) < dist(P,L.e)) result = L.s;

**else** result = L.e;

}

**return** result;

}

1.8 计算多边形面积

//计算多边形面积

//点的编号从0~n-1

**double CalcArea**(Point p[],**int** n)

{

**double** res = 0;

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

res += (p[i]^p[(i+1)%n])/2;

**return fabs**(res);

}

1.9 判断点在线段上

//\*判断点在线段上

**bool OnSeg**(Point P,Line L)

{

return

sgn((L.s-P)^(L.e-P)) == 0 &&

sgn((P.x - L.s.x) \* (P.x - L.e.x)) <= 0 &&

sgn((P.y - L.s.y) \* (P.y - L.e.y)) <= 0;

}

1.10 判断点在凸多边形内

//\*判断点在凸多边形内

//点形成一个凸包，而且按逆时针排序（如果是顺时针把里面的<0改为>0）

//点的编号:0~n-1

//返回值：

//-1:点在凸多边形外

//0:点在凸多边形边界上

//1:点在凸多边形内

**int inConvexPoly**(Point a,Point p[],**int** n)

{

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

{

**if**(sgn((p[i]-a)^(p[(i+1)%n]-a)) < 0)**return** -1;

**else if**(OnSeg(a,Line(p[i],p[(i+1)%n])))**return** 0;

}

**return** 1;

}

1.11 判断点在任意多边形内

//\*判断点在任意多边形内

//射线法，poly[]的顶点数要大于等于3,点的编号0~n-1

//返回值

//-1:点在凸多边形外

//0:点在凸多边形边界上

//1:点在凸多边形内

**int inPoly**(Point p,Point poly[],**int** n)

{

**int** cnt;

Line ray,side; cnt = 0;

ray.s = p; ray.e.y = p.y;

ray.e.x = -100000000000.0;//-INF,注意取值防止越界

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

{

side.s = poly[i]; side.e = poly[(i+1)%n];

**if**(OnSeg(p,side))**return** 0;

//如果平行轴则不考虑

**if**(sgn(side.s.y - side.e.y) == 0)

continue;

**if**(OnSeg(side.s,ray))

{

**if**(sgn(side.s.y - side.e.y) > 0)cnt++;

}

**else if**(OnSeg(side.e,ray))

{

**if**(sgn(side.e.y - side.s.y) > 0)cnt++;

}

**else if**(inter(ray,side)) cnt++;

}

**if**(cnt % 2 == 1)**return** 1;

else return -1;

}

1.12 判断凸多边形

//判断凸多边形

//允许共线边

//点可以是顺时针给出也可以是逆时针给出

//点的编号1~n-1

**bool isconvex**(Point poly[],**int** n)

{

**bool** s[3]; **memset**(s,**false**,**sizeof**(s)); **for**(**int** i = 0;i < n;i++)

{

s[sgn( (poly[(i+1)%n]-poly[i])^(poly[(i+2)%n]-poly[i]) )+1] = **true**; **if**(s[0] && s[2])**return false**;

}

return true;

}

2、凸包

/\*

\* 求凸包，Graham算法

\* 点的编号0~n-1

\* 返回凸包结果Stack[0~top-1]为凸包的编号

\*/

**const int** MAXN = 1010; Point list[MAXN];

**int** Stack[MAXN],top;

//相对于list[0]的极角排序

**bool \_cmp**(Point p1,Point p2)

{

**double** tmp = (p1-list[0])^(p2-list[0]);

**if**(sgn(tmp) > 0)**return true**;

**else if**(sgn(tmp) == 0 && sgn(dist(p1,list[0]) - dist(p2,list[0])) <= 0)

return true; else return false;

}

void Graham(int n)

{

Point p0;

**int** k = 0;

p0 = list[0];

//找最下边的一个点

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

{

**if**( (p0.y > list[i].y) || (p0.y == list[i].y && p0.x > list[i].x) )

{

p0 = list[i]; k = i;

}

}

swap(list[k],list[0]); sort(list+1,list+n,\_cmp); **if**(n == 1)

{

top = 1;

Stack[0] = 0;

return;

}

**if**(n == 2)

{

top = 2;

Stack[0] = 0;

Stack[1] = 1;

return ;

}

Stack[0] = 0;

Stack[1] = 1;

top = 2;

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

{

**while**(top > 1 &&

sgn((list[Stack[top-1]]-list[Stack[top-2]])^(list[i]-list[Stack[top-2]])) <= 0)

top--; Stack[top++] = i;

}

}

3、平面最近点对（HDU 1007）

**#include** <stdio.h>

**#include** <string.h>

**#include** <algorithm>

**#include** <iostream>

**#include** <math.h>

**using namespace** std; **const double** eps = 1e-6; **const int** MAXN = 100010; **const double** INF = 1e20; **struct** Point

{

**double** x,y;

};

**double dist**(Point a,Point b)

{

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

}

Point p[MAXN]; Point tmpt[MAXN];

**bool cmpxy**(Point a,Point b)

{

**if**(a.x != b.x)**return** a.x < b.x;

**else return** a.y < b.y;

}

**bool cmpy**(Point a,Point b)

{

**return** a.y < b.y;

}

**double Closest\_Pair**(**int** left,**int** right)

{

**double** d = INF;

**if**(left == right)**return** d;

**if**(left + 1 == right)

**return** dist(p[left],p[right]);

**int** mid = (left+right)/2;

**double** d1 = Closest\_Pair(left,mid); **double** d2 = Closest\_Pair(mid+1,right); d = min(d1,d2);

**int** k = 0;

**for**(**int** i = left;i <= right;i++)

{

**if**(**fabs**(p[mid].x - p[i].x) <= d) tmpt[k++] = p[i];

}

sort(tmpt,tmpt+k,cmpy);

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

{

**for**(**int** j = i+1;j < k && tmpt[j].y - tmpt[i].y < d;j++)

{

d = min(d,dist(tmpt[i],tmpt[j]));

}

}

**return** d;

}

int main()

{

**int** n;

**while**(**scanf**("%d",&n)==1 && n)

{

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

**scanf**("%lf%lf",&p[i].x,&p[i].y); sort(p,p+n,cmpxy); **printf**("%.2lf\n",Closest\_Pair(0,n-1)/2);

}

**return** 0;

}

4、旋转卡壳

4.1 求解平面最远点对（POJ 2187 Beauty Contest）

**struct** Point

{

**int** x,y;

**Point**(**int** \_x = 0,**int** \_y = 0)

{

x = \_x; y = \_y;

}

Point **operator -**(**const** Point &b)**const**

{

**return** Point(x - b.x, y - b.y);

}

**int operator ^**(**const** Point &b)**const**

{

**return** x\*b.y - y\*b.x;

}

**int operator \***(**const** Point &b)**const**

{

**return** x\*b.x + y\*b.y;

}

void input()

{

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

}

};

//距离的平方

**int dist2**(Point a,Point b)

{

**return** (a-b)\*(a-b);

}

//\*\*\*\*\*\*二维凸包，int\*\*\*\*\*\*\*\*\*\*\*

**const int** MAXN = 50010; Point list[MAXN];

**int** Stack[MAXN],top;

**bool \_cmp**(Point p1,Point p2)

{

**int** tmp = (p1-list[0])^(p2-list[0]);

**if**(tmp > 0)**return true**;

**else if**(tmp == 0 && dist2(p1,list[0]) <= dist2(p2,list[0]))

return true; else return false;

}

void Graham(int n)

{

Point p0;

**int** k = 0;

p0 = list[0];

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

**if**(p0.y > list[i].y || (p0.y == list[i].y && p0.x > list[i].x))

{

p0 = list[i]; k = i;

}

swap(list[k],list[0]); sort(list+1,list+n,\_cmp); **if**(n == 1)

{

top = 1;

Stack[0] = 0;

return;

}

**if**(n == 2)

{

top = 2;

Stack[0] = 0; Stack[1] = 1;

return;

}

Stack[0] = 0; Stack[1] = 1;

top = 2;

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

{

**while**(top > 1 &&

((list[Stack[top-1]]-list[Stack[top-2]])^(list[i]-list[Stack[top-2]])) <= 0) top--;

Stack[top++] = i;

}

}

//旋转卡壳，求两点间距离平方的最大值

**int rotating\_calipers**(Point p[],**int** n)

{

**int** ans = 0; Point v;

**int** cur = 1;

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

{

v = p[i]-p[(i+1)%n];

**while**((v^(p[(cur+1)%n]-p[cur])) < 0) cur = (cur+1)%n;

ans = max(ans,max(dist2(p[i],p[cur]),dist2(p[(i+1)%n],p[(cur+1)%n])));

}

**return** ans;

}

Point p[MAXN];

int main()

{

**int** n;

**while**(**scanf**("%d",&n) == 1)

{

**for**(**int** i = 0;i < n;i++)list[i].input(); Graham(n);

**for**(**int** i = 0;i < top;i++)p[i] = list[Stack[i]];

**printf**("%d\n",rotating\_calipers(p,top));

}

**return** 0;

}

4.2 求解平面点集最大三角形

//旋转卡壳计算平面点集最大三角形面积

**int rotating\_calipers**(Point p[],**int** n)

{

**int** ans = 0; Point v;

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

{

**int** j = (i+1)%n;

**int** k = (j+1)%n;

**while**(j != i && k != i)

{

ans = max(ans,**abs**((p[j]-p[i])^(p[k]-p[i])));

**while**( ((p[i]-p[j])^(p[(k+1)%n]-p[k])) < 0 ) k = (k+1)%n;

j = (j+1)%n;

}

}

**return** ans;

}

Point p[MAXN];

int main()

{

**int** n;

**while**(**scanf**("%d",&n) == 1)

{

**if**(n == -1)**break**;

**for**(**int** i = 0;i < n;i++)list[i].input(); Graham(n);

**for**(**int** i = 0;i < top;i++)p[i] = list[Stack[i]];

**printf**("%.2f\n",(**double**)rotating\_calipers(p,top)/2);

}

**return** 0;

}

4.3 求解两凸包最小距离（POJ 3608）

**const double** eps = 1e-8;

int sgn(double x)

{

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

**if**(x < 0)**return** -1;

else return 1;

}

**struct** Point

{

**double** x,y;

**Point**(**double** \_x = 0,**double** \_y = 0)

{

x = \_x; y = \_y;

}

Point **operator -**(**const** Point &b)**const**

{

**return** Point(x - b.x, y - b.y);

}

**double operator ^**(**const** Point &b)**const**

{

**return** x\*b.y - y\*b.x;

}

**double operator \***(**const** Point &b)**const**

{

**return** x\*b.x + y\*b.y;

}

void input()

{

**scanf**("%lf%lf",&x,&y);

}

};

**struct** Line

{

Point s,e;

**Line**(){}

**Line**(Point \_s,Point \_e)

{

s = \_s; e = \_e;

}

};

//两点间距离

**double dist**(Point a,Point b)

{

**return sqrt**((a-b)\*(a-b));

}

//点到线段的距离，返回点到线段最近的点

Point **NearestPointToLineSeg**(Point P,Line L)

{

Point result;

**double** t = ((P-L.s)\*(L.e-L.s))/((L.e-L.s)\*(L.e-L.s));

**if**(t >=0 && t <= 1)

{

result.x = L.s.x + (L.e.x - L.s.x)\*t;

result.y = L.s.y + (L.e.y - L.s.y)\*t;

}

else

{

**if**(dist(P,L.s) < dist(P,L.e)) result = L.s;

**else** result = L.e;

}

**return** result;

}

/\*

\* 求凸包，Graham算法

\* 点的编号0~n-1

\* 返回凸包结果Stack[0~top-1]为凸包的编号

\*/

**const int** MAXN = 10010; Point list[MAXN];

**int** Stack[MAXN],top;

//相对于list[0]的极角排序

**bool \_cmp**(Point p1,Point p2)

{

**double** tmp = (p1-list[0])^(p2-list[0]);

**if**(sgn(tmp) > 0)**return true**;

**else if**(sgn(tmp) == 0 && sgn(dist(p1,list[0]) - dist(p2,list[0])) <= 0)

return true; else return false;

}

void Graham(int n)

{

Point p0;

**int** k = 0;

p0 = list[0];

//找最下边的一个点

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

{

**if**( (p0.y > list[i].y) || (p0.y == list[i].y && p0.x > list[i].x) )

{

p0 = list[i]; k = i;

}

}

swap(list[k],list[0]); sort(list+1,list+n,\_cmp); **if**(n == 1)

{

top = 1;

Stack[0] = 0;

return;

}

**if**(n == 2)

{

top = 2;

Stack[0] = 0;

Stack[1] = 1;

return ;

}

Stack[0] = 0;

Stack[1] = 1;

top = 2;

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

{

**while**(top > 1 &&

sgn((list[Stack[top-1]]-list[Stack[top-2]])^(list[i]-list[Stack[top-2]])) <= 0)

top--; Stack[top++] = i;

}

}

//点p0到线段p1p2的距离

**double pointtoseg**(Point p0,Point p1,Point p2)

{

**return** dist(p0,NearestPointToLineSeg(p0,Line(p1,p2)));

}

//平行线段p0p1和p2p3的距离

**double dispallseg**(Point p0,Point p1,Point p2,Point p3)

{

**double** ans1 = min(pointtoseg(p0,p2,p3),pointtoseg(p1,p2,p3)); **double** ans2 = min(pointtoseg(p2,p0,p1),pointtoseg(p3,p0,p1)); **return** min(ans1,ans2);

}

//得到向量a1a2和b1b2的位置关系

**double Get\_angle**(Point a1,Point a2,Point b1,Point b2)

{

**return** (a2-a1)^(b1-b2);

}

**double rotating\_calipers**(Point p[],**int** np,Point q[],**int** nq)

{

**int** sp = 0, sq = 0;

**for**(**int** i = 0;i < np;i++)

**if**(sgn(p[i].y - p[sp].y) < 0) sp = i;

**for**(**int** i = 0;i < nq;i++)

**if**(sgn(q[i].y - q[sq].y) > 0) sq = i;

**double** tmp;

**double** ans = dist(p[sp],q[sq]);

**for**(**int** i = 0;i < np;i++)

{

**while**(sgn(tmp = Get\_angle(p[sp],p[(sp+1)%np],q[sq],q[(sq+1)%nq])) < 0) sq = (sq+1)%nq;

**if**(sgn(tmp) == 0)

ans = min(ans,dispallseg(p[sp],p[(sp+1)%np],q[sq],q[(sq+1)%nq]));

**else** ans = min(ans,pointtoseg(q[sq],p[sp],p[(sp+1)%np])); sp = (sp+1)%np;

}

**return** ans;

}

**double solve**(Point p[],**int** n,Point q[],**int** m)

{

**return** min(rotating\_calipers(p,n,q,m),rotating\_calipers(q,m,p,n));

}

Point p[MAXN],q[MAXN];

int main()

{

**int** n,m;

**while**(**scanf**("%d%d",&n,&m) == 2)

{

**if**(n == 0 && m == 0)**break**;

**for**(**int** i = 0;i < n;i++) list[i].input();

Graham(n);

**for**(**int** i = 0;i < top;i++) p[i] = list[i];

n = top;

**for**(**int** i = 0;i < m;i++) list[i].input();

Graham(m);

**for**(**int** i = 0;i < top;i++) q[i] = list[i];

m = top;

**printf**("%.4f\n",solve(p,n,q,m));

}

**return** 0;

}

5、半平面交

5.1 半平面交模板(from UESTC)

**const double** eps = 1e-8; **const double** PI = **acos**(-1.0); **int sgn**(**double** x)

{

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

**if**(x < 0) **return** -1;

else return 1;

}

**struct** Point

{

**double** x,y;

**Point**(){}

**Point**(**double** \_x,**double** \_y)

{

x = \_x; y = \_y;

}

Point **operator -**(**const** Point &b)**const**

{

**return** Point(x - b.x, y - b.y);

}

**double operator ^**(**const** Point &b)**const**

{

**return** x\*b.y - y\*b.x;

}

**double operator \***(**const** Point &b)**const**

{

**return** x\*b.x + y\*b.y;

}

};

**struct** Line

{

Point s,e; **double** k; **Line**(){}

**Line**(Point \_s,Point \_e)

{

s = \_s; e = \_e;

k = **atan2**(e.y - s.y,e.x - s.x);

}

Point **operator &**(**const** Line &b)**const**

{

Point res = s;

**double** t = ((s - b.s)^(b.s - b.e))/((s - e)^(b.s - b.e)); res.x += (e.x - s.x)\*t;

res.y += (e.y - s.y)\*t;

**return** res;

}

};

//半平面交，直线的左边代表有效区域

**bool HPIcmp**(Line a,Line b)

{

**if**(**fabs**(a.k - b.k) > eps)**return** a.k < b.k;

**return** ((a.s - b.s)^(b.e - b.s)) < 0;

}

Line Q[110];

**void HPI**(Line line[], **int** n, Point res[], **int** &resn)

{

**int** tot = n; sort(line,line+n,HPIcmp); tot = 1;

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

**if**(**fabs**(line[i].k - line[i-1].k) > eps) line[tot++] = line[i];

**int** head = 0, tail = 1; Q[0] = line[0];

Q[1] = line[1]; resn = 0;

**for**(**int** i = 2; i < tot; i++)

{

**if**(**fabs**((Q[tail].e-Q[tail].s)^(Q[tail-1].e-Q[tail-1].s)) < eps ||

**fabs**((Q[head].e-Q[head].s)^(Q[head+1].e-Q[head+1].s)) < eps)

return;

**while**(head < tail && (((Q[tail]&Q[tail-1]) - line[i].s)^(line[i].e-line[i].s)) > eps)

tail--;

**while**(head < tail && (((Q[head]&Q[head+1]) - line[i].s)^(line[i].e-line[i].s)) > eps)

head++; Q[++tail] = line[i];

}

**while**(head < tail && (((Q[tail]&Q[tail-1]) - Q[head].s)^(Q[head].e-Q[head].s)) > eps)

tail--;

**while**(head < tail && (((Q[head]&Q[head-1]) - Q[tail].s)^(Q[tail].e-Q[tail].e)) > eps)

head++;

**if**(tail <= head + 1)**return**;

**for**(**int** i = head; i < tail; i++) res[resn++] = Q[i]&Q[i+1];

**if**(head < tail - 1)

res[resn++] = Q[head]&Q[tail];

}

5.2 普通半平面交写法

POJ 1750

**const double** eps = 1e-18;

int sgn(double x)

{

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

**if**(x < 0)**return** -1;

else return 1;

}

**struct** Point

{

**double** x,y;

**Point**(){}

**Point**(**double** \_x,**double** \_y)

{

x = \_x; y = \_y;

}

Point **operator -**(**const** Point &b)**const**

{

**return** Point(x - b.x, y - b.y);

}

**double operator ^**(**const** Point &b)**const**

{

**return** x\*b.y - y\*b.x;

}

**double operator \***(**const** Point &b)**const**

{

**return** x\*b.x + y\*b.y;

}

};

//计算多边形面积

**double CalcArea**(Point p[],**int** n)

{

**double** res = 0;

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

res += (p[i]^p[(i+1)%n]);

**return fabs**(res/2);

}

//通过两点，确定直线方程

**void Get\_equation**(Point p1,Point p2,**double** &a,**double** &b,**double** &c)

{

a = p2.y - p1.y;

b = p1.x - p2.x;

c = p2.x\*p1.y - p1.x\*p2.y;

}

//求交点

Point **Intersection**(Point p1,Point p2,**double** a,**double** b,**double** c)

{

**double** u = **fabs**(a\*p1.x + b\*p1.y + c); **double** v = **fabs**(a\*p2.x + b\*p2.y + c); Point t;

t.x = (p1.x\*v + p2.x\*u)/(u+v);

t.y = (p1.y\*v + p2.y\*u)/(u+v);

**return** t;

}

Point tp[110];

**void Cut**(**double** a,**double** b,**double** c,Point p[],**int** &cnt)

{

**int** tmp = 0;

**for**(**int** i = 1;i <= cnt;i++)

{

//当前点在左侧，逆时针的点

**if**(a\*p[i].x + b\*p[i].y + c < eps)tp[++tmp] = p[i];

else

{

**if**(a\*p[i-1].x + b\*p[i-1].y + c < -eps)

tp[++tmp] = Intersection(p[i-1],p[i],a,b,c);

**if**(a\*p[i+1].x + b\*p[i+1].y + c < -eps) tp[++tmp] = Intersection(p[i],p[i+1],a,b,c);

}

}

**for**(**int** i = 1;i <= tmp;i++) p[i] = tp[i];

p[0] = p[tmp];

p[tmp+1] = p[1]; cnt = tmp;

}

**double** V[110],U[110],W[110];

**int** n;

**const double** INF = 100000000000.0; Point p[110];

**bool solve**(**int** id)

{

p[1] = Point(0,0);

p[2] = Point(INF,0); p[3] = Point(INF,INF); p[4] = Point(0,INF);

p[0] = p[4];

p[5] = p[1];

**int** cnt = 4;

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

**if**(i != id)

{

**double** a = (V[i] - V[id])/(V[i]\*V[id]);

**double** b = (U[i] - U[id])/(U[i]\*U[id]);

**double** c = (W[i] - W[id])/(W[i]\*W[id]);

**if**(sgn(a) == 0 && sgn(b) == 0)

{

**if**(sgn(c) >= 0)**return false**; **else continue**;

}

Cut(a,b,c,p,cnt);

}

**if**(sgn(CalcArea(p,cnt)) == 0)**return false**; **else return true**;

}

int main()

{

**while**(**scanf**("%d",&n) == 1)

{

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

**scanf**("%lf%lf%lf",&V[i],&U[i],&W[i]); **for**(**int** i = 0;i < n;i++)

{

**if**(solve(i))**printf**("Yes\n"); **else printf**("No\n");

}

}

**return** 0;

}

6、三点求圆心坐标（三角形外心）

//过三点求圆心坐标

Point **waixin**(Point a,Point b,Point c)

{

**double** a1 = b.x - a.x, b1 = b.y - a.y, c1 = (a1\*a1 + b1\*b1)/2;

**double** a2 = c.x - a.x, b2 = c.y - a.y, c2 = (a2\*a2 + b2\*b2)/2;

**double** d = a1\*b2 - a2\*b1;

**return** Point(a.x + (c1\*b2 - c2\*b1)/d, a.y + (a1\*c2 -a2\*c1)/d);

}

7、求两圆相交的面积

//两个圆的公共部分面积

**double Area\_of\_overlap**(Point c1,**double** r1,Point c2,**double** r2)

{

**double** d = dist(c1,c2);

**if**(r1 + r2 < d + eps)**return** 0;

**if**(d < **fabs**(r1 - r2) + eps)

{

**double** r = min(r1,r2);

**return** PI\*r\*r;

}

**double** x = (d\*d + r1\*r1 - r2\*r2)/(2\*d);

**double** t1 = **acos**(x / r1);

**double** t2 = **acos**((d - x)/r2);

**return** r1\*r1\*t1 + r2\*r2\*t2 - d\*r1\***sin**(t1);

}

8、Pick 公式

顶点坐标均是整点的简单多边形：面积=内部格点数目+边上格点数目/2-1

**动态规划**

1、最长上升子序列 O(nlogn)

**有性质：一个序列中，最长上升子序列的长度就是不下降子序列的个数。（hdu1257）**

**const int** MAXN=500010;

**int** a[MAXN],b[MAXN];

//用二分查找的方法找到一个位置，使得num>b[i-1] 并且num<b[i],并用num代替b[i]

**int Search**(**int** num,**int** low,**int** high)

{

**int** mid;

**while**(low<=high)

{

mid=(low+high)/2; **if**(num>=b[mid]) low=mid+1; **else** high=mid-1;

}

**return** low;

}

**int DP**(**int** n)

{

**int** i,len,pos; b[1]=a[1];

len=1;

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

{

**if**(a[i]>=b[len])//如果a[i]比b[]数组中最大还大直接插入到后面即可

{

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

}

**else**//用二分的方法在b[]数组中找出第一个比a[i]大的位置并且让a[i]替代这个位置

{

pos=Search(a[i],1,len);

b[pos]=a[i];

}

}

**return** len;

}

**搜索**

1、Dancing Links

1.1 精确覆盖

/\*

\* POJ3074

\*/

**const int** N = 9; //3\*3数独 **const int** MaxN = N\*N\*N + 10; **const int** MaxM = N\*N\*4 + 10;

**const int** maxnode = MaxN\*4 + MaxM + 10;

**char** g[MaxN];

**struct** DLX

{

**int** n,m,size;

int

U[maxnode],D[maxnode],R[maxnode],L[maxnode],Row[maxnode],Col[maxnode];

**int** H[MaxN],S[MaxM];

**int** ansd,ans[MaxN];

**void init**(**int** \_n,**int** \_m)

{

n = \_n; m = \_m;

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

{

S[i] = 0;

U[i] = D[i] = i;

L[i] = i-1;

R[i] = i+1;

}

R[m] = 0; L[0] = m;

size = m;

**for**(**int** i = 1;i <= n;i++)H[i] = -1;

}

**void Link**(**int** r,**int** c)

{

++S[Col[++size]=c]; Row[size] = r; D[size] = D[c];

U[D[c]] = size; U[size] = c; D[c] = size;

**if**(H[r] < 0)H[r] = L[size] = R[size] = size;

else

{

R[size] = R[H[r]];

L[R[H[r]]] = size;

L[size] = H[r];

R[H[r]] = size;

}

}

void remove(int c)

{

L[R[c]] = L[c]; R[L[c]] = R[c];

**for**(**int** i = D[c];i != c;i = D[i])

**for**(**int** j = R[i];j != i;j = R[j])

{

U[D[j]] = U[j];

D[U[j]] = D[j];

--S[Col[j]];

}

}

void resume(int c)

{

**for**(**int** i = U[c];i != c;i = U[i])

**for**(**int** j = L[i];j != i;j = L[j])

++S[Col[U[D[j]]=D[U[j]]=j]]; L[R[c]] = R[L[c]] = c;

}

bool Dance(int d)

{

**if**(R[0] == 0)

{

**for**(**int** i = 0;i < d;i++)g[(ans[i]-1)/9] = (ans[i]-1)%9 + '1';

**for**(**int** i = 0;i < N\*N;i++)**printf**("%c",g[i]);

**printf**("\n"); **return true**;

}

**int** c = R[0];

**for**(**int** i = R[0];i != 0;i = R[i])

**if**(S[i] < S[c]) c = i;

remove(c);

**for**(**int** i = D[c];i != c;i = D[i])

{

ans[d] = Row[i];

**for**(**int** j = R[i];j != i;j = R[j])remove(Col[j]);

**if**(Dance(d+1))**return true**;

**for**(**int** j = L[i];j != i;j = L[j])resume(Col[j]);

}

resume(c);

return false;

}

};

**void place**(**int** &r,**int** &c1,**int** &c2,**int** &c3,**int** &c4,**int** i,**int** j,**int** k)

{

r = (i\*N+j)\*N + k; c1 = i\*N+j+1; c2 = N\*N+i\*N+k; c3 = N\*N\*2+j\*N+k; c4 = N\*N\*3+((i/3)\*3+(j/3))\*N+k;

}

DLX dlx;

int main()

{

**while**(**scanf**("%s",g) == 1)

{

**if**(**strcmp**(g,"end") == 0)**break**; dlx.init(N\*N\*N,N\*N\*4);

**int** r,c1,c2,c3,c4;

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

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

**for**(**int** k = 1;k <= N;k++)

**if**(g[i\*N+j] == '.' || g[i\*N+j] == '0'+k)

{

}

dlx.Dance(0);

}

place(r,c1,c2,c3,c4,i,j,k); dlx.Link(r,c1);

dlx.Link(r,c2);

dlx.Link(r,c3);

dlx.Link(r,c4);

**return** 0;

}

1.2 可重复覆盖

/\*

\* FZU1686

\*/

**const int** MaxM = 15\*15+10;

**const int** MaxN = 15\*15+10;

**const int** maxnode = MaxN \* MaxM; **const int** INF = 0x3f3f3f3f; **struct** DLX

{

**int** n,m,size;

int

U[maxnode],D[maxnode],R[maxnode],L[maxnode],Row[maxnode],Col[maxnode];

**int** H[MaxN],S[MaxM];

**int** ansd;

**void init**(**int** \_n,**int** \_m)

{

n = \_n; m = \_m;

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

{

S[i] = 0;

U[i] = D[i] = i;

L[i] = i-1;

R[i] = i+1;

}

R[m] = 0; L[0] = m;

size = m;

**for**(**int** i = 1;i <= n;i++)H[i] = -1;

}

**void Link**(**int** r,**int** c)

{

++S[Col[++size]=c]; Row[size] = r; D[size] = D[c];

U[D[c]] = size; U[size] = c; D[c] = size;

**if**(H[r] < 0)H[r] = L[size] = R[size] = size;

else

{

R[size] = R[H[r]];

L[R[H[r]]] = size;

L[size] = H[r];

R[H[r]] = size;

}

}

void remove(int c)

{

**for**(**int** i = D[c];i != c;i = D[i]) L[R[i]] = L[i], R[L[i]] = R[i];

}

void resume(int c)

{

**for**(**int** i = U[c];i != c;i = U[i])

L[R[i]] = R[L[i]] = i;

}

**bool** v[MaxM];

**int f**()

{

**int** ret = 0;

**for**(**int** c = R[0]; c != 0;c = R[c])v[c] = **true**;

**for**(**int** c = R[0]; c != 0;c = R[c])

**if**(v[c])

{

ret++;

v[c] = **false**;

**for**(**int** i = D[c];i != c;i = D[i])

**for**(**int** j = R[i];j != i;j = R[j]) v[Col[j]] = **false**;

}

**return** ret;

}

void Dance(int d)

{

**if**(d + f() >= ansd)**return**;

**if**(R[0] == 0)

{

**if**(d < ansd)ansd = d;

return;

}

**int** c = R[0];

**for**(**int** i = R[0];i != 0;i = R[i])

**if**(S[i] < S[c]) c = i;

**for**(**int** i = D[c];i != c;i = D[i])

{

remove(i);

**for**(**int** j = R[i];j != i;j = R[j])remove(j); Dance(d+1);

**for**(**int** j = L[i];j != i;j = L[j])resume(j); resume(i);

}

}

};

DLX g;

**int** a[20][20];

**int** id[20][20];

int main()

{

**int** n,m;

**while**(**scanf**("%d%d",&n,&m) == 2)

{

**int** sz = 0; **memset**(id,0,**sizeof**(id)); **for**(**int** i = 0;i < n;i++)

**for**(**int** j = 0;j < m;j++)

{

**scanf**("%d",&a[i][j]);

**if**(a[i][j] == 1)id[i][j] = (++sz);

}

g.init(n\*m,sz); sz = 1;

**int** n1,m1; **scanf**("%d%d",&n1,&m1); **for**(**int** i = 0;i < n;i++)

**for**(**int** j = 0;j < m;j++)

{

**for**(**int** x = 0;x < n1 && i + x < n;x++)

**for**(**int** y = 0;y < m1 && j + y < m;y++)

**if**(id[i+x][j+y]) g.Link(sz,id[i+x][j+y]);

sz++;

}

g.ansd = INF; g.Dance(0); **printf**("%d\n",g.ansd);

}

**return** 0;

}

**其他**

1、高精度

/\*

\* 高精度，支持乘法和加法

\*/

**struct** BigInt

{

**const static int** *mod* = 10000; **const static int** *DLEN* = 4; **int** a[600],len;

**BigInt**()

{

**memset**(a,0,**sizeof**(a)); len = 1;

}

**BigInt**(**int** v)

{

**memset**(a,0,**sizeof**(a)); len = 0;

do

{

a[len++] = v%*mod*; v /= *mod*;

}**while**(v);

}

**BigInt**(**const char** s[])

{

**memset**(a,0,**sizeof**(a)); **int** L = **strlen**(s);

len = L/*DLEN*; **if**(L%*DLEN*)len++; **int** index = 0;

**for**(**int** i = L-1;i >= 0;i -= *DLEN*)

{

**int** t = 0;

**int** k = i - *DLEN* + 1;

**if**(k < 0)k = 0;

**for**(**int** j = k;j <= i;j++) t = t\*10 + s[j] - '0';

a[index++] = t;

}

}

BigInt **operator +**(**const** BigInt &b)**const**

{

BigInt res;

res.len = max(len,b.len);

**for**(**int** i = 0;i <= res.len;i++) res.a[i] = 0;

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

{

res.a[i] += ((i < len)?a[i]:0)+((i < b.len)?b.a[i]:0); res.a[i+1] += res.a[i]/*mod*;

res.a[i] %= *mod*;

}

**if**(res.a[res.len] > 0)res.len++;

**return** res;

}

BigInt **operator \***(**const** BigInt &b)**const**

{

BigInt res;

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

{

**int** up = 0;

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

{

**int** temp = a[i]\*b.a[j] + res.a[i+j] + up; res.a[i+j] = temp%*mod*;

up = temp/*mod*;

}

**if**(up != 0)

res.a[i + b.len] = up;

}

res.len = len + b.len;

**while**(res.a[res.len - 1] == 0 &&res.len > 1)res.len--;

**return** res;

}

void output()

{

**printf**("%d",a[len-1]); **for**(**int** i = len-2;i >=0 ;i--)

**printf**("%04d",a[i]); **printf**("\n");

}

};

2、完全高精度

HDU 1134 求卡特兰数

**#include** <iostream>

**#include** <stdio.h>

**#include** <algorithm>

**#include** <string.h>

using namespace std;

/\*

\* 完全大数模板

* 输出cin>>a
* 输出a.print();

\* 注意这个输入不能自动去掉前导0的，可以先读入到char数组，去掉前导0，再用构造函数。

\*/

**#define** MAXN 9999

**#define** MAXSIZE 1010

**#define** DLEN 4

**class** BigNum

{

private:

**int** a[500]; //可以控制大数的位数

**int** len;

public:

**BigNum**(){len=1;**memset**(a,0,**sizeof**(a));} //构造函数 **BigNum**(**const int**); //将一个int类型的变量转化成大数 **BigNum**(**const char**\*); //将一个字符串类型的变量转化为大数 **BigNum**(**const** BigNum &); //拷贝构造函数

BigNum &**operator=**(**const** BigNum &); //重载赋值运算符，大数之间进行赋值运算

**friend** istream& **operator>>**(istream&,BigNum&); //重载输入运算符

**friend** ostream& **operator<<**(ostream&,BigNum&); //重载输出运算符

BigNum **operator+**(**const** BigNum &)**const**; //重载加法运算符，两个大数之间的相加运算 BigNum **operator-**(**const** BigNum &)**const**; //重载减法运算符，两个大数之间的相减运算 BigNum **operator\***(**const** BigNum &)**const**; //重载乘法运算符，两个大数之间的相乘运算 BigNum **operator/**(**const int** &)**const**; //重载除法运算符，大数对一个整数进行相除

运算

BigNum **operator^**(**const int** &)**const**; //大数的n次方运算

**int operator%**(**const int** &)**const**; //大数对一个int类型的变量进行取模运算

**bool operator>**(**const** BigNum &T)**const**; //大数和另一个大数的大小比较

**bool operator>**(**const int** &t)**const**; //大数和一个int类型的变量的大小比较

**void print**(); //输出大数

};

**BigNum::BigNum**(**const int** b) //将一个int类型的变量转化为大数

{

**int** c,d=b; len=0;

**memset**(a,0,**sizeof**(a)); **while**(d>MAXN)

{

c=d-(d/(MAXN+1))\*(MAXN+1); d=d/(MAXN+1);

a[len++]=c;

}

a[len++]=d;

}

**BigNum::BigNum**(**const char** \*s) //将一个字符串类型的变量转化为大数

{

**int** t,k,index,L,i; **memset**(a,0,**sizeof**(a)); L=**strlen**(s); len=L/DLEN; **if**(L%DLEN)len++; index=0;

**for**(i=L-1;i>=0;i-=DLEN)

{

t=0;

k=i-DLEN+1;

**if**(k<0)k=0;

**for**(**int** j=k;j<=i;j++) t=t\*10+s[j]-'0';

a[index++]=t;

}

}

**BigNum::BigNum**(**const** BigNum &T):len(T.len) //拷贝构造函数

{

**int** i; **memset**(a,0,**sizeof**(a)); **for**(i=0;i<len;i++)

a[i]=T.a[i];

}

BigNum & **BigNum::operator=**(**const** BigNum &n) //重载赋值运算符，大数之间赋值运算

{

**int** i; len=n.len;

**memset**(a,0,**sizeof**(a)); **for**(i=0;i<len;i++)

a[i]=n.a[i];

return \*this;

}

istream& **operator>>**(istream &in,BigNum &b)

{

**char** ch[MAXSIZE\*4];

**int** i=-1; in>>ch;

**int** L=**strlen**(ch); **int** count=0,sum=0; **for**(i=L-1;i>=0;)

{

sum=0;

**int** t=1;

**for**(**int** j=0;j<4&&i>=0;j++,i--,t\*=10)

{

sum+=(ch[i]-'0')\*t;

}

b.a[count]=sum; count++;

}

b.len=count++;

**return** in;

}

ostream& **operator<<**(ostream& out,BigNum& b) //重载输出运算符

{

**int** i; cout<<b.a[b.len-1];

**for**(i=b.len-2;i>=0;i--)

{

**printf**("%04d",b.a[i]);

}

**return** out;

}

BigNum **BigNum::operator+**(**const** BigNum &T)**const** //两个大数之间的相加运算

{

BigNum t(\***this**);

**int** i,big; big=T.len>len?T.len:len; **for**(i=0;i<big;i++)

{

t.a[i]+=T.a[i];

**if**(t.a[i]>MAXN)

{

t.a[i+1]++;

t.a[i]-=MAXN+1;

}

}

**if**(t.a[big]!=0) t.len=big+1;

**else** t.len=big;

**return** t;

}

BigNum **BigNum::operator-**(**const** BigNum &T)**const** //两个大数之间的相减运算

{

**int** i,j,big; **bool** flag; BigNum t1,t2; **if**(\***this**>T)

{

t1=\***this**; t2=T;

flag=0;

}

else

{

t1=T;

t2=\***this**; flag=1;

}

big=t1.len;

**for**(i=0;i<big;i++)

{

**if**(t1.a[i]<t2.a[i])

{

j=i+1;

**while**(t1.a[j]==0) j++;

t1.a[j--]--;

**while**(j>i)

t1.a[j--]+=MAXN;

t1.a[i]+=MAXN+1-t2.a[i];

}

**else** t1.a[i]-=t2.a[i];

}

t1.len=big;

**while**(t1.a[len-1]==0 && t1.len>1)

{

t1.len--;

big--;

}

**if**(flag)

t1.a[big-1]=0-t1.a[big-1];

**return** t1;

}

BigNum **BigNum::operator\***(**const** BigNum &T)**const** //两个大数之间的相乘

{

BigNum ret;

**int** i,j,up;

**int** temp,temp1;

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

{

up=0;

**for**(j=0;j<T.len;j++)

{

temp=a[i]\*T.a[j]+ret.a[i+j]+up;

**if**(temp>MAXN)

{

temp1=temp-temp/(MAXN+1)\*(MAXN+1); up=temp/(MAXN+1); ret.a[i+j]=temp1;

}

else

{

up=0; ret.a[i+j]=temp;

}

}

**if**(up!=0) ret.a[i+j]=up;

}

ret.len=i+j;

**while**(ret.a[ret.len-1]==0 && ret.len>1)ret.len--;

**return** ret;

}

BigNum **BigNum::operator/**(**const int** &b)**const** //大数对一个整数进行相除运算

{

BigNum ret;

**int** i,down=0;

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

{

ret.a[i]=(a[i]+down\*(MAXN+1))/b; down=a[i]+down\*(MAXN+1)-ret.a[i]\*b;

}

ret.len=len;

**while**(ret.a[ret.len-1]==0 && ret.len>1) ret.len--;

**return** ret;

}

**int BigNum::operator%**(**const int** &b)**const** //大数对一个 int类型的变量进行取模

{

**int** i,d=0;

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

d=((d\*(MAXN+1))%b+a[i])%b;

**return** d;

}

BigNum **BigNum::operator^**(**const int** &n)**const** //大数的n次方运算

{

BigNum t,ret(1); **int** i; **if**(n<0)**exit**(-1); **if**(n==0)**return** 1;

**if**(n==1)**return** \***this**;

**int** m=n;

**while**(m>1)

{

t=\***this**;

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

t=t\*t; m-=i; ret=ret\*t;

**if**(m==1)ret=ret\*(\***this**);

}

**return** ret;

}

**bool BigNum::operator>**(**const** BigNum &T)**const** //大数和另一个大数的大小比较

{

**int** ln; **if**(len>T.len)**return true**; **else if**(len==T.len)

{

ln=len-1;

**while**(a[ln]==T.a[ln]&&ln>=0) ln--;

**if**(ln>=0 && a[ln]>T.a[ln])

return true; else

return false;

}

else

return false;

}

**bool BigNum::operator>**(**const int** &t)**const** //大数和一个int类型的变量的大小比较

{

BigNum b(t);

**return** \***this**>b;

}

**void BigNum::print**() //输出大数

{

**int** i;

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

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

**printf**("%04d",a[i]); **printf**("\n");

}

BigNum f[110];//卡特兰数

int main()

{

f[0]=1;

**for**(**int** i=1;i<=100;i++)

f[i]=f[i-1]\*(4\*i-2)/(i+1);//卡特兰数递推式

**int** n;

**while**(**scanf**("%d",&n)==1)

{

**if**(n==-1)**break**; f[n].print();

}

**return** 0;

}

3、strtok 和 sscanf 结合输入

空格作为分隔输入，读取一行的整数：

**gets**(buf); **int** v;

**char** \*p = **strtok**(buf," ");

**while**(p)

{

**sscanf**(p,"%d",&v);

p = **strtok**(NULL," ");

}

4、解决爆栈，手动加栈

**#pragma** comment(linker, "/STACK:1024000000,1024000000")

5、STL

* 1. 优先队列 priority\_queue

#### empty() 如果队列为空返回真

#### pop() 删除对顶元素

#### push() 加入一个元素

#### size() 返回优先队列中拥有的元素个数

#### top() 返回优先队列队顶元素 在默认的优先队列中，优先级高的先出队。在默认的 int 型中先出队的为较大的 数。

priority\_queue<**int**>q1;//大的先出对 priority\_queue<**int**,vector<**int**>,greater<**int**> >q2; //小的先出队

自定义比较函数：

**struct** cmp

{

bool operator ()(int x, int y)

{

**return** x > y; // x小的优先级高

//也可以写成其他方式，如： return p[x] > p[y];表示p[i]小的优先级高

}

};

priority\_queue<**int**, vector<**int**>, cmp>q;//定义方法

//其中，第二个参数为容器类型。第三个参数为比较函数。

结构体排序：

**struct** node

{

**int** x, y;

**friend bool operator <** (node a, node b)

{

**return** a.x > b.x; //结构体中，x小的优先级高

}

};

priority\_queue<node>q;//定义方法

//在该结构中，y为值, x为优先级。

//通过自定义operator<操作符来比较元素中的优先级。

//在重载”<”时，最好不要重载”>”，可能会发生编译错误

* 1. set 和 multiset

set 和 multiset 用法一样，就是 multiset 允许重复元素。 元素放入容器时，会按照一定的排序法则自动排序，默认是按照 less<>排序规则来排序。不 能修改容器里面的元素值，只能插入和删除。

自定义 int 排序函数：（默认的是从小到大的，下面这个从大到小）

**struct** classcomp {

**bool operator()** (**const int**& lhs, **const int**& rhs) **const**

{**return** lhs>rhs;}

};//这里有个逗号的，注意

multiset<**int**,classcomp> fifth; // class as Compare

#### 上面这样就定义成了从大到小排列了。

结构体自定义排序函数：

（定义 set 或者 multiset 的时候定义了排序函数，定义迭代器时一样带上排序函数）

**struct** Node

{

**int** x,y;

};

**struct** classcomp//先按照 x 从小到大排序，x相同则按照y从大到小排序

{

**bool operator()**(**const** Node &a,**const** Node &b)**const**

{

**if**(a.x!=b.x)**return** a.x<b.x;

**else return** a.y>b.y;

}

}; //注意这里有个逗号

multiset<Node,classcomp>mt; multiset<Node,classcomp>::iterator it;

**主要函数：**

###### begin() 返回指向第一个元素的迭代器

###### clear() 清除所有元素

###### count() 返回某个值元素的个数 empty() 如果集合为空，返回 true end() 返回指向最后一个元素的迭代器

###### erase() 删除集合中的元素 (参数是一个元素值，或者迭代器) find() 返回一个指向被查找到元素的迭代器

###### insert() 在集合中插入元素

###### size() 集合中元素的数目

###### lower\_bound() 返回指向大于（或等于）某值的第一个元素的迭代器

###### upper\_bound() 返回大于某个值元素的迭代器

###### equal\_range() 返回集合中与给定值相等的上下限的两个迭代器

(注意对于 multiset 删除操作之间删除值会把所以这个值的都删掉，删除一个要用迭代器)

6、输入输出外挂

//适用于正负整数

template <class T>

**inline bool scan\_d**(**T** &ret) {

**char** c; **int** sgn;

**if**(c=getchar(),c==EOF) **return** 0; //EOF **while**(c!='-'&&(c<'0'||c>'9')) c=getchar(); sgn=(c=='-')?-1:1;

ret=(c=='-')?0:(c-'0');

**while**(c=getchar(),c>='0'&&c<='9') ret=ret\*10+(c-'0'); ret\*=sgn;

**return** 1;

}

**inline void out**(**int** x) { **if**(x>9) out(x/10); putchar(x%10+'0');

}

7、莫队算法

莫队算法，可以解决一类静态，离线区间查询问题。

###### BZOJ 2038: [2009 国家集训队]小 Z 的袜子(hose)

###### Description

作为一个生活散漫的人，小 Z 每天早上都要耗费很久从一堆五颜六色的袜子中找出一双来穿。 终于有一天，小 Z 再也无法忍受这恼人的找袜子过程，于是他决定听天由命…… 具体来说， 小 Z 把这 N 只袜子从 1 到 N 编号，然后从编号 L 到 R(L

###### Input

输入文件第一行包含两个正整数 N 和 M。N 为袜子的数量，M 为小 Z 所提的询问的数量。 接下来一行包含 N 个正整数 Ci，其中 Ci 表示第 i 只袜子的颜色，相同的颜色用相同的数字 表示。再接下来 M 行，每行两个正整数 L，R 表示一个询问。

###### Output

包含 M 行，对于每个询问在一行中输出分数 A/B 表示从该询问的区间[L,R]中随机抽出两只 袜子颜色相同的概率。若该概率为 0 则输出 0/1，否则输出的 A/B 必须为最简分数。（详见 样例）

###### Sample Input

6 4

1 2 3 3 3 2

2 6

1 3

3 5

1 6

###### Sample Output

2/5

0/1

1/1

4/15

题解： *P*

  *i*   *i i*

  *i*  *i*

*C* 2 **

\* (**

####  1)/ 2

** 2  **

2

*C*

*R*  *L* 1

(*R*  *L*

####  1) \* (*R*

 *L*)/ 2

(*R*  *L*

####  1) \* (*R*

* *L*)

只需要统计区间内各个数出现次数的平方和

莫队算法，两种方法，一种是直接分成 sqrt(n)块，分块排序。 另外一种是求得曼哈顿距离最小生成树，根据 manhattan MST 的 dfs 序求解。

7.1 分块

**const int** MAXN = 50010; **const int** MAXM = 50010; **struct** Query

{

**int** L,R,id;

}node[MAXM];

long long gcd(long long a,long long b)

{

**if**(b == 0)**return** a;

**return** gcd(b,a%b);

}

**struct** Ans

{

**long long** a,b;//分数a/b

**void reduce**()//分数化简

{

**long long** d = gcd(a,b); a /= d; b /= d;

}

}ans[MAXM];

**int** a[MAXN]; **int** num[MAXN]; **int** n,m,unit;

**bool cmp**(Query a,Query b)

{

**if**(a.L/unit != b.L/unit)**return** a.L/unit < b.L/unit;

**else return** a.R < b.R;

}

void work()

{

**long long** temp = 0; **memset**(num,0,**sizeof**(num)); **int** L = 1;

**int** R = 0;

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

{

**while**(R < node[i].R)

{

R++;

temp -= (**long long**)num[a[R]]\*num[a[R]]; num[a[R]]++;

temp += (**long long**)num[a[R]]\*num[a[R]];

}

**while**(R > node[i].R)

{

temp -= (**long long**)num[a[R]]\*num[a[R]]; num[a[R]]--;

temp += (**long long**)num[a[R]]\*num[a[R]]; R--;

}

**while**(L < node[i].L)

{

temp -= (**long long**)num[a[L]]\*num[a[L]]; num[a[L]]--;

temp += (**long long**)num[a[L]]\*num[a[L]]; L++;

}

**while**(L > node[i].L)

{

L--;

temp -= (**long long**)num[a[L]]\*num[a[L]]; num[a[L]]++;

temp += (**long long**)num[a[L]]\*num[a[L]];

}

ans[node[i].id].a = temp - (R-L+1);

ans[node[i].id].b = (**long long**)(R-L+1)\*(R-L);

ans[node[i].id].reduce();

}

}

int main()

{

**while**(**scanf**("%d%d",&n,&m) == 2)

{

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

**scanf**("%d",&a[i]); **for**(**int** i = 0;i < m;i++)

{

node[i].id = i;

**scanf**("%d%d",&node[i].L,&node[i].R);

}

unit = (**int**)**sqrt**(n); sort(node,node+m,cmp); work();

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

**printf**("%lld/%lld\n",ans[i].a,ans[i].b);

}

**return** 0;

}

7.2 Manhattan MST 的 dfs 顺序求解

**const int** MAXN = 50010; **const int** MAXM = 50010; **const int** INF = 0x3f3f3f3f; **struct** Point

{

**int** x,y,id;

}p[MAXN],pp[MAXN];

**bool cmp**(Point a,Point b)

{

**if**(a.x != b.x) **return** a.x < b.x;

**else return** a.y < b.y;

}

//树状数组，找y-x大于当前的，但是y+x最小的

**struct** BIT

{

**int** min\_val,pos;

void init()

{

min\_val = INF; pos = -1;

}

}bit[MAXN];

**struct** Edge

{

**int** u,v,d;

}edge[MAXN<<2];

**bool cmpedge**(Edge a,Edge b)

{

**return** a.d < b.d;

}

**int** tot;

**int** n;

**int** F[MAXN];

int find(int x)

{

**if**(F[x] == -1) **return** x;

**else return** F[x] = find(F[x]);

}

**void addedge**(**int** u,**int** v,**int** d)

{

edge[tot].u = u; edge[tot].v = v; edge[tot++].d = d;

}

**struct** Graph

{

**int** to,next;

}e[MAXN<<1];

**int** total,head[MAXN];

void \_addedge(int u,int v)

{

e[total].to = v; e[total].next = head[u]; head[u] = total++;

}

int lowbit(int x)

{

**return** x&(-x);

}

**void update**(**int** i,**int** val,**int** pos)

{

**while**(i > 0)

{

**if**(val < bit[i].min\_val)

{

bit[i].min\_val = val; bit[i].pos = pos;

}

i -= lowbit(i);

}

}

**int ask**(**int** i,**int** m)

{

**int** min\_val = INF,pos = -1;

**while**(i <= m)

{

**if**(bit[i].min\_val < min\_val)

{

min\_val = bit[i].min\_val; pos = bit[i].pos;

}

i += lowbit(i);

}

**return** pos;

}

**int dist**(Point a,Point b)

{

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

}

**void Manhattan\_minimum\_spanning\_tree**(**int** n,Point p[])

{

**int** a[MAXN],b[MAXN];

tot = 0;

**for**(**int** dir = 0;dir < 4;dir++)

{

**if**(dir == 1 || dir == 3)

{

**for**(**int** i = 0;i < n;i++) swap(p[i].x,p[i].y);

}

**else if**(dir == 2)

{

**for**(**int** i = 0;i < n;i++) p[i].x = -p[i].x;

}

sort(p,p+n,cmp);

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

a[i] = b[i] = p[i].y - p[i].x; sort(b,b+n);

**int** m = unique(b,b+n) - b;

**for**(**int** i = 1;i <= m;i++) bit[i].init();

**for**(**int** i = n-1;i >= 0;i--)

{

**int** pos = lower\_bound(b,b+m,a[i]) - b + 1;

**int** ans = ask(pos,m);

**if**(ans != -1) addedge(p[i].id,p[ans].id,dist(p[i],p[ans]));

update(pos,p[i].x+p[i].y,i);

}

}

**memset**(F,-1,**sizeof**(F)); sort(edge,edge+tot,cmpedge); total = 0;

**memset**(head,-1,**sizeof**(head)); **for**(**int** i = 0;i < tot;i++)

{

**int** u = edge[i].u, v = edge[i].v; **int** t1 = find(u), t2 = find(v); **if**(t1 != t2)

{

F[t1] = t2;

\_addedge(u,v);

\_addedge(v,u);

}

}

}

**int** m;

**int** a[MAXN];

**struct** Ans

{

**long long** a,b;

}ans[MAXM];

**long long** temp ;

**int** num[MAXN];

**void add**(**int** l,**int** r)

{

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

{

temp -= (**long long**)num[a[i]]\*num[a[i]];

num[a[i]]++;

temp += (**long long**)num[a[i]]\*num[a[i]];

}

}

**void del**(**int** l,**int** r)

{

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

{

temp -= (**long long**)num[a[i]]\*num[a[i]]; num[a[i]]--;

temp += (**long long**)num[a[i]]\*num[a[i]];

}

}

**void dfs**(**int** l1,**int** r1,**int** l2,**int** r2,**int** idx,**int** pre)

{

**if**(l2 < l1) add(l2,l1-1); **if**(r2 > r1) add(r1+1,r2); **if**(l2 > l1) del(l1,l2-1); **if**(r2 < r1) del(r2+1,r1);

ans[pp[idx].id].a = temp - (r2-l2+1);

ans[pp[idx].id].b = (**long long**)(r2-l2+1)\*(r2-l2); **for**(**int** i = head[idx];i != -1;i = e[i].next)

{

**int** v = e[i].to;

**if**(v == pre) **continue**; dfs(l2,r2,pp[v].x,pp[v].y,v,idx);

}

**if**(l2 < l1)del(l2,l1-1); **if**(r2 > r1)del(r1+1,r2); **if**(l2 > l1)add(l1,l2-1); **if**(r2 < r1)add(r2+1,r1);

}

long long gcd(long long a,long long b)

{

**if**(b == 0) **return** a;

**else return** gcd(b,a%b);

}

int main()

{

**while**(**scanf**("%d%d",&n,&m) == 2)

{

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

**scanf**("%d",&a[i]); **for**(**int** i = 0;i < m;i++)

{

**scanf**("%d%d",&p[i].x,&p[i].y); p[i].id = i;

pp[i] = p[i];

}

Manhattan\_minimum\_spanning\_tree(m,p);

**memset**(num,0,**sizeof**(num)); temp = 0; dfs(1,0,pp[0].x,pp[0].y,0,-1); **for**(**int** i = 0;i < m;i++)

{

**long long** d = gcd(ans[i].a,ans[i].b);

**printf**("%lld/%lld\n",ans[i].a/d,ans[i].b/d);

}

}

**return** 0;

}

8、VIM 配置

set nu

set history=1000000

set tabstop=4 set shiftwidth=4 set smarttab

set cindent colo evening

set nobackup set noswapfile

set mouse=a

map <F6> :call CR()<CR> func! CR()

exec "w"

exec "!g++ % -o %<" exec "! ./%<" endfunc

imap <c-]> {<cr>}<c-o>O<left><right> map <F2> :call SetTitle()<CR>

func SetTitle()

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| let | l | = | 0 |  | | | | | |
| let | l | = | l | + | 1 | | | call | setline(l,'#include | <stdio.h>') |
| let | l | = | l | + | 1 | | | call | setline(l,'#include | <string.h>') |
| let | l | = | l | + | 1 | | | call | setline(l,'#include | <iostream>') |
| let | l | = | l | + | 1 | | | call | setline(l,'#include | <algorithm>') |
| let | l | = | l | + | 1 | | | call | setline(l,'#include | <vector>') |
| let | l | = | l | + | 1 | | | call | setline(l,'#include | <queue>') |
| let | l | = | l | + | 1 | | | call | setline(l,'#include | <set>') |
| let | l | = | l | + | 1 | | | call | setline(l,'#include | <map>') |
| let | l | = | l | + | 1 | | | call | setline(l,'#include | <string>') |
| let | l | = | l | + | 1 | | | call | setline(l,'#include | <math.h>') |
| let | l | = | l | + | 1 | | | call | setline(l,'#include | <stdlib.h>') |
| let | l | = | l | + | 1 | | | call | setline(l,'#include | <time.h>') |
| let | l | = | l | + | 1 | | | call | setline(l,'using namespace std;') | |
| let | l | = | l | + | 1 | | | call | setline(l,'') | |
| let | l | = | l | + | 1 | | | call | setline(l,'int main()') | |
| let | l | = | l | + | 1 | | | call | setline(l,'{') | |
| let | l | = | l | + | 1 | | | call | setline(l,' //freopen("in.txt","r",stdin);') | |
| let | l | = | l | + | 1 | | | call | setline(l,' //freopen("out.txt","w",stdout);') | |
| let | l | = | l | + | 1 | | | call | setline(l,' ') | |
| let | l | = | l | + | 1 | | | call | setline(l,' return 0;') | |
| let | l | = | l | + | 1 | | | call | setline(l,'}') | |

endfunc