By

io07-陈思嘉

r\_clover-任琪宇

smilewsw-王思薇 2015.11.5

Acm Code Library

BUPT\_CaiCai Musketeer

目录

[头文件 3](#_Toc434519552)

[输入输出挂 3](#_Toc434519553)

[bitset 4](#_Toc434519554)

[离散化 5](#_Toc434519555)

[多重背包 6](#_Toc434519556)

[数位DP 7](#_Toc434519557)

[数位DP(考虑前导零) 8](#_Toc434519558)

[快速幂运算 9](#_Toc434519559)

[KMP 10](#_Toc434519560)

[后缀数组 11](#_Toc434519561)

[后缀自动机 14](#_Toc434519562)

[字符串Hash 16](#_Toc434519563)

[AC自动机 和 Trie树 20](#_Toc434519564)

[AC自动机求模式串出现次数和 22](#_Toc434519565)

[ST表 25](#_Toc434519566)

[静态主席树 26](#_Toc434519567)

[动态主席树 28](#_Toc434519568)

[CDQ分治 33](#_Toc434519569)

[整体二分 46](#_Toc434519570)

[叉姐线段树 50](#_Toc434519571)

[插头dp 52](#_Toc434519572)

[树分治 54](#_Toc434519573)

[斜率优化DP 56](#_Toc434519574)

[最小表示法 58](#_Toc434519575)

[JAVA 59](#_Toc434519576)

[对拍脚本 62](#_Toc434519577)

[BigLL 62](#_Toc434519578)

[命令行指令 63](#_Toc434519579)

[Splay 63](#_Toc434519580)

[LCT维护最大生成树 67](#_Toc434519581)

[动态树LCT 73](#_Toc434519582)

[Dancing-Links 精确覆盖（数独） 78](#_Toc434519583)

[Dancing-Links 重复覆盖 81](#_Toc434519584)

[康拓展开 85](#_Toc434519585)

[2-SAT 86](#_Toc434519586)

[ISAP 网络流 87](#_Toc434519587)

[最小费用最大流 89](#_Toc434519588)

[尺取法 91](#_Toc434519589)

[倍增法求LCA 93](#_Toc434519590)

[边双连通分量 95](#_Toc434519591)

[点双连通分量 96](#_Toc434519592)

[LCT动态树 98](#_Toc434519593)

[强连通分量缩环 102](#_Toc434519594)

[二分图匹配 103](#_Toc434519595)

[O(n)预处理C(n,m) 104](#_Toc434519596)

[快速幂求矩阵前缀和 105](#_Toc434519597)

[莫比乌斯反演 105](#_Toc434519598)

[原根与离散对数 107](#_Toc434519599)

[.1 原根 107](#_Toc434519600)

[.2 离散对数 108](#_Toc434519601)

[高斯消元模板 110](#_Toc434519602)

[开关类模2同余方程求解 114](#_Toc434519603)

[FFT求大整数乘法 114](#_Toc434519604)

[计算几何 117](#_Toc434519605)

[点类 118](#_Toc434519606)

[直线类 118](#_Toc434519607)

[点p到线段st的距离 119](#_Toc434519608)

[点p到线段st的垂足，保存在cp里 119](#_Toc434519609)

[判断p点是否在线段st上（<=包括端点，<不包括端点） 119](#_Toc434519610)

[求点(x,y)关于直线AX+BY+C=0的对称点(X,Y) 119](#_Toc434519611)

[线段相交判定（规范相交） 120](#_Toc434519612)

[不规范相交 120](#_Toc434519613)

[判平行 120](#_Toc434519614)

[求两直线交点，存在res里 120](#_Toc434519615)

[将直线a沿法向量方向平移距离len得到的直线 121](#_Toc434519616)

[多边形类 121](#_Toc434519617)

[判断点是否在多边形内 121](#_Toc434519618)

[多边形的重心 122](#_Toc434519619)

[多边形内格点数 122](#_Toc434519620)

[判断是否为凸多边形 122](#_Toc434519621)

[给定N个圆，求一条直线最多能交几个圆O（n^2logn） 123](#_Toc434519622)

[求N个圆的最近距离O( n (logn)^2 ) 126](#_Toc434519623)

# 头文件

#include <cstdio>

#include <cstring>

#include <iostream>

#include <algorithm>

#include <cmath>

#include <vector>

#include <queue>

#include <stack>

#include <set>

#include <map>

//#include <tr1/unordered\_set>

//#include <tr1/unordered\_map>

#include <bitset>

//#pragma comment(linker, "/STACK:1024000000,1024000000")

#define lson l, m, rt<<1

#define rson m+1, r, rt<<1|1

#define inf 1e9

#define debug(a) cout << #a" = " << (a) << endl;

#define debugarry(a, n) for (int i = 0; i < (n); i++) { cout << #a"[" << i << "] = " << (a)[i] << endl; }

#define clr(x, y) memset(x, y, sizeof x)

#define LL long long

#define uLL unsigned LL

using namespace std;

# 输入输出挂

template<class T>

inline bool read(T &n)

{

T x = 0, tmp = 1;

char c = getchar();

while ((c < '0' || c > '9') && c != '-' && c != EOF) c = getchar();

if (c == EOF) return false;

if (c == '-') c = getchar(), tmp = -1;

while (c >= '0' && c <= '9') x \*= 10, x += (c - '0'), c = getchar();

n = x\*tmp;

return true;

}

template <class T>

inline void write(T n)

{

if (n < 0)

{

putchar('-');

n = -n;

}

int len = 0, data[20];

while (n)

{

data[len++] = n % 10;

n /= 10;

}

if (!len) data[len++] = 0;

while (len--) putchar(data[len] + 48);

}

# bitset

#include <bitset>//正常情况 加速32倍 可能还要加速更多

bitset<n>p;

b.any() b中是否存在置为1的二进制位？

b.none() b中不存在置为1的二进制位吗？

b.count() b中置为1的二进制位的个数

b.size() b中二进制位的个数

b[pos] 访问b中在pos处的二进制位

b.test(pos) b中在pos处的二进制位是否为1？

b.set() 把b中所有二进制位都置为1

b.set(pos) 把b中在pos处的二进制位置为1

b.reset() 把b中所有二进制位都置为0

b.reset(pos) 把b中在pos处的二进制位置为0

b.flip() 把b中所有二进制位逐位取反

b.flip(pos) 把b中在pos处的二进制位取反

b.to\_ulong() 用b中同样的二进制位返回一个unsigned long值

os << b 把b中的位集输出到os流

以及所有位操作。

＃include <bitset>

using std::bitset;

bitset<32> bitvec; //32位，全为0。

bitset<n> b; b有n位，每位都为0

bitset<n> b(u); b是unsigned long型u的一个副本

bitset<n> b(s); b是string对象s中含有的位串的副本

bitset<n> b(s, pos, n); b是s中从位置pos开始的n个位的副本

hdu 5413 n 2e4

普通 n^2 111.363s

bitset 2.385s

# 离散化

const int maxn = 1e5;

int A[maxn];

int sub[maxn],len;

int n;

int main()

{

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

{

len=0;

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

{

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

sub[len++]=A[i];

}

sort(sub,sub+len);

len = uniqe(sub,sub+len) - sub;

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

{

A[i] = lower\_bound(sub,sub+len,A[i])-sub+1;

}

}

return 0;

}}

# 多重背包

int dp[maxn];

int w[maxn],v[maxn],c[maxn];

int n,C;

void f()

{

clr(dp,0);

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

{

int cnt = c[i];

int k = 1;

while(cnt>=k)

{

for(int j=C;j>=w[i]\*k;j--)

{

dp[j] = max(dp[j],dp[j-w[i]\*k]+v[i]\*k);

}

cnt-=k;

k<<=1;

}

if(cnt)

{

k=cnt;

for(int j=C;j>=w[i]\*k;j--)

{

dp[j] = max(dp[j],dp[j-w[i]\*k]+v[i]\*k);

}

}

}

}

# 数位DP

#include<cstdio>

#include<cstdlib>

#include<algorithm>

#include<cmath>

#include<cstring>

#include<iostream>

using namespace std;

typedef long long LL;

const int MAX\_N = 1010;

const int MAX\_K = 10;

int num[9];

int dp[9][2];

/\*len：表示长度

s：当前的状态（对于此题就是上一位是否是6）

fp：表示之前的状态是否充满的（如果说充满的此处的放置就是有限制的 否则就可以随便放）

\*/

int dfs(int len, int s, bool fp)

{

if (len == 0) return 1;

if (!fp&&dp[len][s] != -1) return dp[len][s];

int res = 0;

int fmax = fp ? num[len] : 9; //根据充满状态选择限制条件

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

{

if (i == 4 || s&&i == 2) continue; //跳过62和4的情况

res += dfs(len - 1, i == 6, fp&&i == fmax);

}

return fp ? res : dp[len][s] = res;

}

int solve(int n)

{

int len = 0;

while (n != 0)

{

num[++len] = n % 10;

n /= 10;

}

return dfs(len, 0, 1);

}

int main()

{

int n, m;

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

while (scanf("%d%d", &n, &m), n | m){

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

}

return 0;

}

# 数位DP(考虑前导零)

int dfs(int len, int s, bool fp)

{

if (len == 0) return 1;

if (!fp && s >= 0 && dp[len][s] != -1) return dp[len][s];

int res = 0;

int fmax = fp? num[len] : 9;

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

{

if (s == -1 && i == 0)

res += dfs(len-1, -1, fp && i == fmax);

else if (s != i)

res += dfs(len-1, i, fp&& i == fmax);

}

return fp || s < 0 ? res : dp[len][s] = res;

}

int solve(int n)

{

int len = 0;

while(n != 0)

{

num[++len] = n%10;

n /= 10;

}

return dfs(len, -1, 1);

}

# 快速幂运算

struct mat

{

int m[100][100];

};

int n;

mat mul(mat a,mat b)

{

mat c;

clr(c.m,0);

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

for(int i=1;i<=n;i++) if(a.m[i][k])

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

c.m[i][j] += a.m[i][k] \* b.m[k][j];

}

mat add(mat a,mat b)

{

mat c;

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

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

c.m[i][j] = a.m[i][j] + b.m[i][j];

}

mat re;

mat sum\_pow\_mat(mat a,int n)

{

if(n%2)

{

if(n==1) return re=a;

mat p = sum\_pow\_mat(a,n-1);

re = mul(re,a);

return add(a,mul(a,p));

}

else

{

mat p = sum\_pow\_mat(a,n/2);

mat ans = add(p,mul(p,re));

re = mul(re,re);

return ans;

}

}

# KMP

vector<int>ans;

void getFail(char \*P,int \*f)

{

int m = strlen(P);

f[0]=0;

f[1]=0;

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

{

int j = f[i];

while(j&&P[i]!=P[j]) j=f[j];

f[i+1]=P[i]==P[j]?j+1:0；

}

}

void find(char \*T,char \*P,int \*f)

{

int n = strlen(T),m= strlen(P);

getFail(P,f);

int j =0;

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

{

while(j&&P[j]!=T[i]) j=f[j];

if(P[j]==T[i]) j++;

if(j==m) ans.push\_back(i-m+1);

}

}

# 后缀数组

#include<string.h>

const int maxn =1e5;

#define FOR(i,a,b) \

for(i=a; (a<b)?(i<=b):(i>=b) ; (a<b)?(i++):(i--) )

struct suffix\_array

{

char s[maxn];

int sa[maxn],t[maxn],t2[maxn],c[maxn];

int m,n;

///构造sa数组

void build\_sa()

{

int i,\*x=t,\*y=t2;

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

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

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

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

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

{

int p=0;

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

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

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

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

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

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

swap(x,y);

p=1;

x[sa[0]]=0;

FOR(i,1,n-1)

x[sa[i]] = y[sa[i-1]]==y[sa[i]] &&

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

if(p>=n) break;

m=p;

}

}

///匹配模式串

int len;

int cmp(char \*pa,int p)

{

return strncmp(pa,s+sa[p],len);

}

int find\_first(char \*P)

{

len = strlen(P);

if(cmp(P,0)<0) return -1;

if(cmp(P,n-1)>0) return -1;

int L=0,R=n-1,ans=n;

while(R>=L)

{

int M=L+(R-L)/2;

int res = cmp(P,M);

if(res<=0)

{

R=M-1;

if( res == 0 ) ans = min(ans,M);

}

else L=M+1;

}

if(ans==n) return -1;

else return ans;

}

int find\_last(char \*P)

{

len = strlen(P);

if(cmp(P,0)<0) return -1;

if(cmp(P,n-1)>0) return -1;

int L=0,R=n-1,ans=-1;

while(R>=L)

{

int M=L+(R-L)/2;

int res = cmp(P,M);

if(res>=0)

{

L=M+1;

if(res==0) ans = max(ans,M);

}

}

return ans;

}

///构造rank，height数组

int rank[maxn],height[maxn];

void getHeight()

{

int i,j,k=0;

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

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

{

if(k) k--;

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

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

height[rank[i]]=k;

}

}

int d[maxn][30],flog[maxn];

void RMQ\_init()

{

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

flog[0]=-1;

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

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

for(int i=0;i+(1<<j)<=n;i++)

d[i][j] = min(d[i][j-1],d[i+(1<<(j-1))][j-1]);

}

int RMQ(int L,int R)

{

int k = flog[R-L+1];

return min(d[L][k],d[R-(1<<k)+1][k]);

}

int query(int j,int k)

{

if(j==k) return n-k;

if(rank[j]>rank[k]) swap(j,k);

return RMQ(rank[j]+1,rank[k]);

}

void init(char \*ss,int mm=200)

{

strcpy(s,ss);

n=strlen(s)+1;

m=mm;

}

int all()

{

build\_sa();

getHeight();

RMQ\_init();

}

}sp;

# 后缀自动机

const int maxn = 9e5 \* 5;

///后缀自动机最少要开两倍空间

struct suffix\_automaton

{

int son[maxn][26],pre[maxn],step[maxn];

int last,si;

inline int tonum(char c)

{

return c-'a';

}

inline char toa(int x)

{

return x+'a';

}

inline void push\_back(int v)

{

step[++si]=v;

}

void Extend(int ch)

{

push\_back(step[last]+1);

int p = last , np = si;

while( p!=-1 && son[p][ch]==-1 )

son[p][ch]=np,

p=pre[p];

if( p==-1 ) pre[np]=0;

else{

int q = son[p][ch];

if( step[q] == step[p]+1 ) pre[np]=q;

else{

push\_back(step[p]+1);

int nq = si;

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

pre[nq] = pre[q];

pre[q] = pre[np] = nq;

while( p!=-1 && son[p][ch]==q )

son[p][ch]=nq,

p=pre[p];

}

}

last = np;

}

void init()

{

si=last=0;

clr(son,-1);

clr(pre,-1);

clr(step,0);

}

void add(char c){

Extend( tonum(c) );

}

void add(char \*s){

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

add(s[i]);

}

}suf;

void resrve(char \*s)

{

int n= strlen(s);

for(int i=0;i<(n+1)/2;i++)

swap(s[i],s[n-i-1]);

}

# 字符串Hash

/\*题意：

给出两个串A，B，让你找出B串在A串中匹配的第一个位置，匹配要求可以有最多两个位置不一样。

解法：

首先将A，B串的hash值求出来，然后就可以O(1)来求出每个子串的hash值了，判位置最裸的方法就是枚举每一个位置然后O(n)的判是否匹配，复杂度是O(n^2)，然后我们可以通过二分来加速判匹配的过程将其将成O(logn)的复杂度，具体操作如下：

二分一个求一个最长的匹配长度，然后跳过一个不匹配的位置然后再用一次二分求一个最长匹配长度，如此循环两次，如果求出来的最长匹配的长度+不匹配位置数（0 or 1 or 2）等于B串长的话就表明找到了一个符合条件的位置。\*/

#include <iostream>

#include <cstdio>

#include <cstdlib>

#include <algorithm>

#include <cmath>

#include <cstring>

#include <vector>

#include <queue>

#include <stack>

#include <set>

#include <vector>

//#pragma comment(linker, "/STACK:1024000000,1024000000")

#define lson l, m, rt<<1

#define rson m+1, r, rt<<1|1

#define inf 1e9

#define debug(a) cout << #a" = " << (a) << endl;

#define debugarry(a, n) for (int i = 0; i < (n); i++) { cout << #a"[" << i << "] = " << (a)[i] << endl; }

#define clr(x, y) memset(x, y, sizeof x)

using namespace std;

typedef unsigned long long uLL;

const int maxn = 100005;

const uLL magic = 7;

uLL base[maxn];

uLL hash\_a[maxn], hash\_b[maxn];

char s1[maxn], s2[maxn];

void init\_hash(int len, char cc[], uLL ha[])

{

ha[0] = 0;

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

ha[i] = ha[i - 1] \* magic + cc[i - 1];

base[0] = 1;

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

base[i] = base[i - 1] \* magic;

}

uLL sub\_hash(uLL ha[], int L, int R)

{

return ha[R] - ha[L] \* base[R - L];

}

bool cmp(int La, int Ra, int Lb, int Rb)

{

uLL ua = sub\_hash(hash\_a, La, Ra);

uLL ub = sub\_hash(hash\_b, Lb, Rb);

if (ua == ub) return true;

else return false;

}

int get\_max(int La, int Lb, int len)

{

int st = -1, ed = len + 1;

while (ed - st > 1)

{

int m = (st + ed) >> 1;

if (cmp(La, La + m, Lb, Lb + m) == true) st = m;

else ed = m;

}

return st;

}

int main()

{

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

int T;

scanf("%d", &T);

for (int kk = 1; kk <= T; kk++)

{

scanf("%s%s", s1, s2);

int len\_a = strlen(s1);

int len\_b = strlen(s2);

init\_hash(len\_a, s1, hash\_a);

init\_hash(len\_b, s2, hash\_b);

//debugarry(hash\_a, len\_a+1); debugarry(hash\_b, len\_b+1);

int ans = -1;

for (int i = 0; i + len\_b - 1 < len\_a; i++)

{

int ptr\_a = i, ptr\_b = 0;

int sum = 0, cnt = 0;

while (cnt <= 2)

{

int maxlen = get\_max(ptr\_a, ptr\_b, len\_b - ptr\_b);

sum += maxlen;

if (sum + cnt == len\_b)

{

ans = i;

break;

}

else

{

cnt++;

ptr\_a += maxlen + 1;

ptr\_b += maxlen + 1;

}

}

//debug(sum);

if (ans != -1) break;

}

printf("Case #%d: %d\n", kk, ans);

}

return 0;

}

**Hash another：**

#define ULL long long

cons ULL x = 233;

ULL pow\_x[maxn];

void Hash\_init(char \*T, ULL \*H)

{

int len = strlen(T);

pow\_x[0] = 1ull;

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

pow\_x[i] = pow\_x[i - 1] \* x;

H[len] = 0;

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

H[i] = H[i + 1] \* x + (ULL)T[i];

}

void Hash(char \*H, int i, int len)

{

return H[i] - H[i + len] \* pow\_x[len];

}

# AC自动机 和 Trie树

#define cls(p) clr(p,0)

const int maxn = 1e5;

const int maxsize = 30;

struct Trie

{

int ch[maxn][maxnsize];

int val[maxn];

vector<int>vv[maxn];

int sz;

void init()

{

sz = 1;

cls(ch[0]);

vv[0].clear();

}

int idx(int c)

{

return c - 'a';

}

void insert(char \*s,int v)

{

int u = 0 , n = strlen(s);

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

{

int c = idx(s[i]);

if(!ch[u][c])

{

cls(ch[sz]);

val[sz]=0;

vv[sz].clear();

ch[u][c] = sz++;

}

u=ch[u][c];

}

val[u]=v;

vv.push\_back(v);

}

///AC

vector<pair<int,int> >ans;

int last[maxn],f[maxn];

void print(int i,int j)

{

if(j)

{

ans.push\_back(make\_pair(i,j));

print(i,last[j]);

}

}

void find(char \*T)

{

ans.clear();

int n = strlen(T);

int j = 0;

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

{

int c = idx(T[i]);

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

j=ch[j][c];

if(val[j]) print(i,j);

else if (last[j]) print(i,last[j]);

}

}

int getFail()

{

queue<int>q;

f[0]=0;

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

{

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

int u = ch[0][c];

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

if(u)

{

f[u]=0;

q.push(u);

last[u]=0;

}

}

while(!q.empty())

{

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

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

{

int u=ch[r][c];

if(!u) continue;

q.push(u);

int v = f[r];

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

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

last[u] = val[f[u]]?f[u]:last[f[u]];

}

}

}

}

# AC自动机求模式串出现次数和

///普通AC自动机在求文本串T中模式串a【i】出现次数时\

时间复杂度为 O(len(T)+sumlen(a)+k) k为答案\

下面的自动机复杂度为 O(len(T)+sumlen(a)) 但只能求次数

#define cls(p) clr(p,0)

const int maxn = 1e5;

const int maxsize = 30;

struct Trie

{

int ch[maxn][maxnsize];

int val[maxn];

vector<int>vv[maxn];

int sz;

void init()

{

sz = 1;

cls(ch[0]);

vv[0].clear();

}

int idx(int c)

{

return c - 'a';

}

void insert(char \*s,int v)

{

int u = 0 , n = strlen(s);

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

{

int c = idx(s[i]);

if(!ch[u][c])

{

cls(ch[sz]);

val[sz]=0;

vv[sz].clear();

ch[u][c] = sz++;

}

u=ch[u][c];

}

val[u]=v;

vv.push\_back(v);

}

///AC

LL ans;

LL sum[maxn];

int last[maxn];

void print(int i,int j)

{

if(j)

{

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

ans+=sum[j];

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

}

}

void find(char \*T)

{

ans.clear();

int n = strlen(T);

int j = 0;

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

{

int c = idx(T[i]);

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

j=ch[j][c];

if(val[j]) print(i,j);

else if (last[j]) print(i,last[j]);

}

}

int getFail()

{

queue<int>q;

f[0]=0;

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

{

int u = ch[0][c];

if(u)

{

f[u]=0;

q.push(u);

last[u]=0;

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

sum[u] = (val[u]?1:0) + sum[last[u]];

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

}

}

while(!q.empty())

{

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

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

{

int u=ch[r][c];

if(!u) continue;

q.push(u);

int v = f[r];

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

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

last[u] = val[f[u]]?f[u]:last[f[u]];

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

sum[u] = (val[u]?1:0) + sum[last[u]];

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

}

}

}

}

# ST表

const int maxn= 100000;

//maxn 即数组大小

int flog[ ( maxn << 1 ) + 10 ] ;

int A[ maxn ];

int dmax[ maxn ][ 30 ];

int n;

void RMQ\_init(int \*A) //RMQ 初始化

{

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

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

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

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

flog[ 0 ] = -1;

for( int i = 1; i < 2 \* maxn ; i ++) flog[ i ] = flog[ i >> 1 ] + 1;

}

int RMQ(int L,int R) //RMQ 查询

{

int k = flog[ R - L + 1 ];

return max( dmax[L][k] , dmax[R-(1<<k)+1][k] );

}

int dsum[maxn][30];

void st\_sum\_init(int \*A)

{

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

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

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

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

flog[ 0 ] = -1;

for( int i = 1; i < 2 \* maxn ; i ++) flog[ i ] = flog[ i >> 1 ] + 1;

}

int st\_sum(int L,int R)

{

if(L>R) return 0;

int k = flog[R-L+1];

return dmax[L][k] + st\_sum(L+(1<<k),R);

}

# 静态主席树

#include <iostream>

#include <cstdio>

#include <cstdlib>

#include <algorithm>

#include <cmath>

#include <cstring>

#include <vector>

#include <queue>

#include <stack>

#include <set>

#include <vector>

#include <deque>

#include <set>

//#pragma comment(linker, "/STACK:1024000000,1024000000")

#define lson l, (l+r>>1), ls[rt]

#define rson (l+r>>1)+1, r, rs[rt]

#define inf 1e9

#define debug(a) cout << #a" = " << (a) << endl;

#define debugarry(a, n) for (int i = 0; i < (n); i++) { cout << #a"[" << i << "] = " << (a)[i] << endl; }

#define clr(x, y) memset(x, y, sizeof x)

#define LL long long

using namespace std;

const int maxn = 1e5 + 20, maxs = maxn \* 20;

#define head(p) ( p >= 0 ? h[p] : 0 )

struct \_\_sad

{

int ls[maxs], rs[maxs];

int sum[maxs];

int h[maxn];

int si, len;

void pushup(int rt)

{

sum[rt] = sum[ls[rt]] + sum[rs[rt]];

}

void build(int p, int add, int l, int r, int &rt, int rt2)

{

if (!rt)

{

if (p<l || p>r)

{

rt = rt2;

return;

}

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

rt = si++;

}

if (l == r)

{

sum[rt] = sum[rt2] + add;

return;

}

build(p, add, lson, ls[rt2]);

build(p, add, rson, rs[rt2]);

pushup(rt);

}

void init(int \*A, int n)

{

clr(h, 0);

si = 1;

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

len = n;

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

{

build(A[i], 1, 0, n, head(i - 1));

}

}

int query(int k, int l, int r, int rt, int rt2)

{

if (l == r) return l;

int nk = sum[ls[rt]] - sum[ls[rt2]];

if (nk >= k) return query(k, lson, ls[rt2]);

return query(k - nk, rson, rs[rt2]);

}

int query(int L, int R, int k)

{

return query(k, 0, len, head(R), head(L - 1));

}

};

# 动态主席树

用于解决动态修改某一个数，动态查询区间第k大

空间复杂度为 nlgnlgn

时间复杂度为 nlgnlgn

zoj 2112 Dynamic Rankings

裸动态主席树（其实这道题用整体二分更好）

N 50000 M 10000

输入格式

N M

a1 ... an

(M){

Q l r k

C pos t

}

#include <iostream>

#include <cstdio>

#include <cstdlib>

#include <algorithm>

#include <cmath>

#include <cstring>

#include <vector>

#include <queue>

#include <stack>

#include <set>

#include <vector>

#include <map>

#include <tr1/unordered\_set>

#include <tr1/unordered\_map>

//#pragma comment(linker, "/STACK:1024000000,1024000000")

#define inf 1e9

#define debug(a) cout << #a" = " << (a) << endl;

#define debugarry(a, n) for (int i = 0; i < (n); i++) { cout << #a"[" << i << "] = " << (a)[i] << endl; }

#define clr(x, y) memset(x, y, sizeof x)

#define LL long long

using namespace std;

const int maxn = 100010; /// N×2

const int M = 8000030; /// （M+N)\*800

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[100010];

void Init\_hash(int k)

{

sort(t, t + k);

m = unique(t, t + k) - t;

}

int get\_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)>>1;

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("input.txt","r",stdin);

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

{

tot = 0;

m = 0;

q = maxn;

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

{

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

t[m++] = a[i];

}

scanf("%d",&q);

int op;

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

{

scanf("%d",&op);

if(op==2)

{

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] , get\_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,get\_hash(a[query[i].l]),-1);

Modify(query[i].l,get\_hash(query[i].r),1);

a[query[i].l] = query[i].r;

}

}

}

return 0;

}

# CDQ分治

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////

CDQ分治，即中序遍历

可解决降维问题，和动态转静态问题

顺序一般为 (l,m) -> (l,r) -> (m+1,r)

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////

HDU 5432 Boring Class

序列 L0～Ln，R0～Rn

选取一些位置，要求L递增，R递减。

输出字典序最小的答案。

解法，因为需要输出字典序最小，需要从后向前进行CDQ

#include<iostream>

#include<cstdio>

#include<algorithm>

#include<cmath>

#include<cstring>

#include <stdio.h>

#include <string>

#define clr(x, y) memset(x, y, sizeof x)

#define inf 1e9

using namespace std;

template<class T>

inline bool read(T &n)

{

T x = 0, tmp = 1;

char c = getchar();

while((c < '0' || c > '9') && c != '-' && c != EOF) c = getchar();

if(c == EOF) return false;

if(c == '-') c = getchar(), tmp = -1;

while(c >= '0' && c <= '9') x \*= 10, x += (c - '0'),c = getchar();

n = x\*tmp;

return true;

}

template <class T>

inline void write(T n)

{

if(n < 0)

{

putchar('-');

n = -n;

}

int len = 0,data[20];

while(n)

{

data[len++] = n%10;

n /= 10;

}

if(!len) data[len++] = 0;

while(len--) putchar(data[len]+48);

}

const int maxn = 55005;

struct node

{

int Li, Ri, id;

bool operator < (const node & A) const

{

return Ri > A.Ri;

}

} q[maxn], tmp[maxn];

int Li[maxn], Ri[maxn], su[maxn];

int dp[maxn], rmax[maxn];

int n;

void upp(int x, int a)

{

while(x <= n)

{

rmax[x] = max(rmax[x], a);

x += x & -x;

}

}

void cl(int x) /// CDQ分治中用到树状数组要这样清空

{

while(x <= n)

{

rmax[x] = 0;

x += x & -x;

}

}

int get\_max(int x)

{

int ans = 0;

while(x > 0)

{

ans = max(ans, rmax[x]);

x -= x & -x;

}

return ans;

}

void CDQ(int L, int R)

{

if (L == R)

{

int &ret = dp[q[L].id];

ret = max(ret, 1);

return ;

}

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

CDQ(mid+1, R);

int L1 = L, L2 = mid+1;

int ptr = mid+1;

sort(q + L, q + mid + 1);

sort(q + mid + 1, q + R+1);

for (int i = L; i <= mid; i++)

{

while(ptr <= R && q[i].Ri <= q[ptr].Ri)

{

upp(q[ptr].Li, dp[q[ptr].id]);

ptr++;

}

int &ret = dp[q[i].id];

ret = max(ret, get\_max(q[i].Li) + 1);

}

for(int i=mid+1;i<=R;i++) ///\*\*特殊清空

cl(q[i].Li);

for (int i = L; i <= mid; i++)

{

while(q[i].id!=i)

swap(q[i], q[q[i].id]);

}

CDQ(L, mid);

}

int main()

{

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

//freopen("output.txt", "w", stdout);

int len;

string ans;

int maxlen, rr;

while(read(n))

{

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

{

read(q[i].Li);

su[i] = q[i].Li;

q[i].id = i;

Li[i] = q[i].Li;

}

sort(su, su + n);

len = unique(su, su + n) - su;

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

q[i].Li = lower\_bound(su, su + len,q[i].Li) - su + 1;

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

{

read(q[i].Ri);

su[i] = q[i].Ri;

Ri[i] = q[i].Ri;

}

clr(dp, 0);

clr(rmax,0);

CDQ(0, n-1);

//for (int i = 0; i < n; i++) printf("Ri=%d i=%d dp=%d\n", q[i].Ri, q[i].id, dp[q[i].id]);

maxlen = 0; rr = -1; ans = "";

for (int i = 0; i < n; i++) maxlen = max(maxlen, dp[i]);

write(maxlen); putchar('\n');

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

{

if (dp[i] == maxlen)

{

if (rr == -1 || (Li[rr] >= Li[i] && Ri[rr] <= Ri[i]))

{

maxlen--;

if(rr!=-1) putchar(' ');

write(i+1);

rr = i;

}

}

}

putchar('\n');

///cout<<"maxlen:"<<maxlen<<endl;

}

return 0;

}

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////

hdu 5354 Bipartite Graph

给一张图 ， 要求找到一个点，删除后成为二分图

解法 ： CDQ + 可撤销(种族)并查集

保证在进入（l,r）前，（l，r）内的边没有加入，之外的边全部加入

再递归处理

#include <iostream>

#include <cstdio>

#include <cstdlib>

#include <algorithm>

#include <cmath>

#include <cstring>

#include <vector>

#include <queue>

#include <stack>

#include <set>

#pragma comment(linker, "/STACK:1024000000,1024000000")

#define lson l, m, rt<<1

#define rson m+1, r, rt<<1|1

#define inf 1e9

#define clr(x, y) memset(x, y, sizeof x)

using namespace std;

const int maxn = 200005;

struct edge

{

int to, next;

}G[maxn<<1];

struct node

{

int u, v, hu, hv, fau, fav,colu,colv;

node(){}

node(int a, int b, int c, int d, int e, int f,int g,int h)

{

u = a; v = b; hu = c; hv = d; fau = e; fav = f;

colu=g;

colv=h;

}

};

stack<node> stk;

int head[maxn], si;

int h[maxn], fa[maxn];

int ans[maxn], n, m;

int col[maxn]; //0 代表相同 1 代表不同

void init(int \_n)

{

for (int i = 0; i <= \_n; i++)

{

fa[i] = i;

h[i] = 1;

///

col[i]=0;

}

}

void add(int u, int v)

{

G[si].to = v;

G[si].next = head[u];

head[u] = si++;

}

int find\_fa(int x)

{

int o = x;

while(fa[o] != o) o = fa[o];

return o;

}

int find\_col(int x)

{

if (fa[x] == x) return 1;

///return 1^find\_col(fa[x]);

return col[x]^find\_col(fa[x]);

}

bool Merge(int u, int v)

{

int a = find\_fa(u), b = find\_fa(v);

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

if (a == b)

{

if (x == y)

{

//printf("~~%d %d~fu = %d fv = %d\n", u, v, a, b);

return false;

}

return true;

}

stk.push(node(a, b, h[a], h[b], fa[a], fa[b],col[a],col[b]));

if (h[a] > h[b])

{

fa[b] = a, h[a] += h[b];

///

col[b] = x ^ y ^ 1;

}

else

{

fa[a] = b, h[b] += h[a];

///

col[a] = x ^ y ^ 1;

}

return true;

}

bool unite(int L, int R, int a, int b)

{

for (int u = L; u <= R; u++)

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

{

int v = G[i].to;

if (a <= v && v <= b) continue;

if (!Merge(u, v)) return false;

}

return true;

}

void get\_del(int x)

{

node tmp;

while(stk.size() > x)

{

tmp = stk.top(); stk.pop();

int u = tmp.u, v = tmp.v;

h[u] = tmp.hu; h[v] = tmp.hv;

fa[u] = tmp.fau; fa[v] = tmp.fav;

/\*\*/col[u] = tmp.colu; col[v] = tmp.colv;

}

}

void cdq(int l,int r)

{

if(l == r)

{

ans[l] = 1;

return ;

}

int pre = stk.size();

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

if(unite(m+1, r, l, m)) /// 加入右面的边

cdq(l,m);

else{

for (int i = l; i <= m; i++) ans[i] = 0;

}

get\_del(pre); /// 删去右面的边

if(unite(l, m, m+1, r)) /// 加入左面的边

cdq(m+1,r);

else{

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

}

get\_del(pre); /// 删去左面的边

return ;

}

int main()

{

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

int T;

scanf("%d", &T);

while(T--)

{

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

init(n);

clr(head, -1); si = 0;

while(stk.size()) stk.pop();

for (int i = 0, st, ed; i < m; i++)

{

scanf("%d%d", &st, &ed);

add(st, ed); add(ed, st);

}

cdq(1, n);

for (int i = 1; i <= n; i++) printf("%d", ans[i]);

printf("\n");

}

return 0;

}

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////

bnu 12753 Arnooks's Defensive Line

插入一些区间，并查询有多少区间包含它

解法：裸CDQ动态转静态

（对于某些CDQ问题，可能可以使用先序遍历或后序遍历，但最好用中序遍历）

#include <iostream>

#include <cstdio>

#include <cstdlib>

#include <algorithm>

#include <cmath>

#include <cstring>

#include <vector>

#include <queue>

#include <stack>

#include <set>

#include <vector>

//#pragma comment(linker, "/STACK:1024000000,1024000000")

#define lson l, m, rt<<1

#define rson m+1, r, rt<<1|1

#define inf 1e9

#define debug(a) cout << #a" = " << (a) << endl;

#define debugarry(a, n) for (int i = 0; i < (n); i++) { cout << #a"[" << i << "] = " << (a)[i] << endl; }

#define clr(x, y) memset(x, y, sizeof x)

using namespace std;

typedef unsigned long long uLL;

const int maxn = 500000 + 30;

int ans[maxn];

struct \_\_sad

{

int l,r;

char c;

int id;

bool operator <(const \_\_sad &a) const{

if(l!=a.l) return l<a.l;

else return r < a.r ;

}

}A[maxn],B[maxn],C[maxn];

int rsum[maxn\*4];

void upp(int x,int add)

{

while( x < maxn \* 4 )

{

rsum[x] += add;

x += x&-x;

}

}

void clear(int x)

{

while( x < maxn \* 4 )

{

rsum[x] = 0;

x += x&-x;

}

}

int get(int x)

{

int ret=0;

while(x)

{

ret += rsum[x];

x -= x&-x;

}

return ret;

}

void CDQ(int l,int r)

{

if(l == r) return ;

int mid = l+r >>1;

int lb = 0 ,lc = 0;

for(int i=l;i<=mid;i++)

{

if(A[i].c=='+') B[ lb++ ] = A[i];

}

for(int i=mid+1;i<=r;i++)

{

if(A[i].c=='?') C[ lc++ ] = A[i];

}

sort( B , B + lb );

sort( C , C + lc );

int sum=0;

int cn=0;

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

{

while(cn<lb&&B[cn].l<=C[i].l)

{

upp( B[cn].r , 1 );

sum++;

cn++;

}

int ret = sum - get( C[i].r-1 );

ans[ C[i].id ] += ret;

}

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

clear(B[i].r);

CDQ( l , mid );

CDQ( mid+1 , r );

}

int sub[maxn\*4],len;

int main()

{

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

int n;

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

{

char s[30];

len = 0;

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

{

scanf("%s%d%d",s,&A[i].l,&A[i].r);

A[i].c=\*s;

A[i].id=i;

sub[len++] = A[i].l;

sub[len++] = A[i].r;

}

sort(sub,sub+len);

len = unique(sub,sub+len) - sub;

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

{

A[i].l = lower\_bound(sub,sub+len,A[i].l) - sub + 1;

A[i].r = lower\_bound(sub,sub+len,A[i].r) - sub + 1;

}

clr(ans,0);

clr(rsum,0);

CDQ(1,n);

for(int i=1;i<=n;i++) if(A[i].c=='?')

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

}

return 0;

}

# 整体二分

#include <iostream>

#include <cstdio>

#include <cstdlib>

#include <algorithm>

#include <cmath>

#include <cstring>

#include <vector>

#include <queue>

#include <stack>

#include <set>

#include <vector>

#include <deque>

#include <set>

//#pragma comment(linker, "/STACK:1024000000,1024000000")

#define lson l, m, ls[rt]

#define rson m+1, r, rs[rt]

#define inf 1e9

#define debug(a) cout << #a" = " << (a) << endl;

#define debugarry(a, n) for (int i = 0; i < (n); i++) { cout << #a"[" << i << "] = " << (a)[i] << endl; }

#define clr(x, y) memset(x, y, sizeof x)

#define LL long long

using namespace std;

const int maxn = 2e6;

struct \_\_sad

{

int l , r , id;

int st , k , v , add;

}p[maxn],p1[maxn],p2[maxn];

int ans[maxn];

int n;

int rsum[maxn];

void upp(int x,int add)

{

while(x<=n)

{

rsum[x] += add;

x += x&-x;

}

}

int get(int x)

{

int ret=0;

while(x)

{

ret += rsum[x];

x -= x&-x;

}

return ret;

}

void Bin(int st,int ed,int l,int r)

{

if(st>ed) return ;

if(l==r)

{

for(int i=st;i<=ed;i++) if( p[i].st==2 )

ans[ p[i].id ] = l;

return ;

}

int mid = l+(r-l)/2;

int ta1=0,ta2=0;

for(int i=st;i<=ed;i++)

{

if(p[i].st==1)

{

if( p[i].v<=mid )

{

p1[ta1++] = p[i];

upp( p[i].l , p[i].add );

}

else p2[ta2++] = p[i];

}

else

{

int t = get( p[i].r ) - get( p[i].l - 1 );

if( t >= p[i].k ) p1[ta1++]=p[i];

else p[i].k -= t , p2[ta2++]=p[i];

}

}

for(int i=st;i<=ed;i++)

{

if(p[i].st==1)

if( p[i].v<=mid )

upp( p[i].l , p[i].add\*(-1) );

}

for(int i=0;i<ta1;i++)

p[ i+st ] = p1[ i ];

for(int i=0;i<ta2;i++)

p[ i+st+ta1 ] = p2[ i ];

Bin( st , st + ta1 - 1 , l , mid );

Bin( st + ta1 , ed , mid+1 , r );

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

整体二分如果用于处理边查询边修改的问题，下标本身就是一维，不能在Bin子区间之前加入东西

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

}

int A[maxn];

int sub[maxn],len;

int main()

{

//freopen("1007.in", "r", stdin);

//freopen("output.txt","w",stdout);

int q;

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

{

int cnt=0,acnt=0;

int a,st,l,r,v,k;

len=0;

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

{

scanf("%d",&a);

A[i]=a;

sub[len++]=a;

p[cnt].l=i; p[cnt].st=1;

p[cnt].v=a; p[cnt].add=1;

cnt++;

}

scanf("%d",&q);

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

{

scanf("%d",&st);

if(st==1)

{

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

sub[len++]=v;

p[cnt].l=l; p[cnt].st=1;

p[cnt].v=A[l]; p[cnt].add=-1;

cnt++;

p[cnt].l=l; p[cnt].st=1;

p[cnt].v=A[l]=v; p[cnt].add=1;

cnt++;

}

else{

scanf("%d%d%d",&l,&r,&k);

p[cnt].l=l; p[cnt].r=r;

p[cnt].st=2; p[cnt].k=k;

p[cnt].id=acnt++;

cnt++;

}

}

sort(sub,sub+len);

len = unique( sub , sub + len ) - sub;

for(int i=0;i<cnt;i++) if(p[i].st==1)

{

p[i].v = lower\_bound( sub , sub+len , p[i].v ) - sub + 1;

}

Bin(0,cnt-1,0,len+7);

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

printf( "%d\n" , sub[ ans[i] - 1 ] );

}

return 0;

}

# 叉姐线段树

const int maxn = 2e5;

int getid(x,y)

{

return x+y|y!=x;

}

#define ls getid(l,l+r>>1)

#define rs getid((l+r>>1)+1,r)

#define lson l,r,ls

#define rson m+1,r,rs

int rm[maxn<<1];

void pushup(int l,int r,int rt)

{

rm[rt]=rm[ls]+rm[rs];

}

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

{

if( l == r )

{

scanf("%d",&rm[rt]);

return ;

}

int m=l+r>>1;

build(lson);

build(rson);

pushup(l,r,rt);

}

void update(int p,int c,int l,int r,int rt){

if(l==r){

rm[rt]=c;

return ;

}

int m=l+r>>1;

if( p <= m ) update(p,c,lson);

else update(p,c,rson);

pushup(rt);

}

int query(int L,int R,int l,int r,int rt){

if( L<=l && r<=R ){

return rm[rt];

}

int m=l+r>>1;

int ret = 0;

if( L<=m ) ret += query(L,R,lson);

if( m<R ) ret += query(L,R,rson);

return ret;

}

# 插头dp

在n\*m的矩阵中，有些格子有树，没有树的格子不能到达，找一条或多条回路，吃完所有的树，求有多少中方法。

HDU 1693

#include <cstdio>

#include <cstring>

#include <iostream>

#include <algorithm>

#include <cmath>

#include <vector>

#include <queue>

#include <stack>

#include <set>

#include <map>

#include <tr1/unordered\_set>

#include <tr1/unordered\_map>

#include <bitset>

#include <cassert>

//#pragma comment(linker, "/STACK:1024000000,1024000000")

#define lson l, m, rt<<1

#define rson m+1, r, rt<<1|1

#define inf 1e9

#define debug(a) cout << #a" = " << (a) << endl;

#define debugarry(a, n) for (int i = 0; i < (n); i++) { cout << #a"[" << i << "] = " << (a)[i] << endl; }

#define clr(x, y) memset(x, y, sizeof x)

#define LL long long

#define uLL unsigned LL

using namespace std;

const int maxn = 20;

int G[maxn][maxn];

int n,m;

tr1::unordered\_map<int,LL>Q[2][2];

tr1::unordered\_map<int,LL>::iterator ite;

void print(int x)

{

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

if( x & (1<<i) ) printf("1");

else printf("0");

}

LL solve()

{

int i=0,j=0,r=0;

Q[r][0].clear();

Q[r][1].clear();

Q[r][0][0] = 1ll;

while(1)

{

Q[r^1][0].clear();

Q[r^1][1].clear();

if( j == m )

{

for( ite=Q[r][0].begin() ; ite != Q[r][0].end() ; ite++)

Q[r^1][0][ ite->first ] += ite->second;

}else if( G[i][j] == 0 )

{

for( ite=Q[r][0].begin() ; ite != Q[r][0].end() ; ite++)

if( (ite->first) & (1<<j) )

;

else

Q[r^1][0][ ite->first ] += ite->second;

}else

{

for( ite=Q[r][0].begin() ; ite != Q[r][0].end() ; ite++)

{

int s = ite->first;

if( s & (1<<j) )

{

Q[r^1][0][s] += ite->second;

Q[r^1][1][s^(1<<j)] += ite->second;

}

else

{

Q[r^1][1][s|(1<<j)] += ite->second;

}

}

for( ite=Q[r][1].begin() ; ite != Q[r][1].end() ; ite++)

{

int s = ite->first;

if( s & (1<<j) )

{

Q[r^1][0][s^(1<<j)] += ite->second;

}

else

{

Q[r^1][1][s] += ite->second;

Q[r^1][0][s|(1<<j)] += ite->second;

}

}

}

r^=1;

if( j == m ) i++,j=0;

else j++;

if( i == n ) break;

}

return Q[r][0][0];

}

int main()

{

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

int T,CASE=0;

scanf("%d",&T);

while( T-- )

{

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

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

for(int j=0;j<m;j++)

scanf("%d",&G[i][j]);

printf("Case %d: There are %I64d ways to eat the trees.\n",++CASE,solve());

}

return 0;

}

# 树分治

const int maxn = 1e5;

struct edge{

int to,next;

}G[maxn<<2];

int h[maxn],si;

void add(int u,int v)

{

G[si].to=v;

G[si].next=h[u];

h[u]=si++;

}

int done[maxn];

int sum[maxn];

int fN;

int dfs1(int u,int pa)

{

sum[u]=1;

for(int i=h[u];~i;i=G[i].next){

int v = G[i].to;

if( done[v] || pa == v ) continue;

sum[u] += dfs1( v , u );

}

return sum[u];

}

int dfs2(int u,int pa)

{

for(int i=h[u];~i;i=G[i].next){

int v = G[i].to;

if( done[v] || pa == v ) continue;

if( sum[v] \* 2 >= fN ) return dfs2(v,u);

}

return u;

}

void dfs(int u)

{

fN = dfs1(u,-1);

int r = dfs2(u,-1);

sol(r);

done[r]=1;

for(int i=h[r];~i;i=G[i].next)

{

int v = G[i].to;

if( done[v] == 0 )

dfs(v);

}

}

void solve()

{

clr(done,0);

dfs(1);

}

# 斜率优化DP

#include <cstdio>

#include <cstring>

#include <iostream>

#include <algorithm>

#include <cmath>

#include <vector>

#include <queue>

#include <stack>

#include <set>

#include <map>

#include <tr1/unordered\_set>

#include <tr1/unordered\_map>

#include <bitset>

#include <cassert>

//#pragma comment(linker, "/STACK:1024000000,1024000000")

#define lson l, m, rt<<1

#define rson m+1, r, rt<<1|1

#define inf 1e9

#define debug(a) cout << #a" = " << (a) << endl;

#define debugarry(a, n) for (int i = 0; i < (n); i++) { cout << #a"[" << i << "] = " << (a)[i] << endl; }

#define clr(x, y) memset(x, y, sizeof x)

#define LL long long

#define uLL unsigned LL

using namespace std;

const int maxn = 500000+30;

LL X[maxn] , Y[maxn];

int he,ta;

void push(LL x,LL y)

{

while( he>ta+1 && (Y[he-1]-Y[he-2])\*(x-X[he-2]) >= (y-Y[he-2])\*(X[he-1]-X[he-2]) )

he--;

X[he] = x;

Y[he] = y;

he++;

}

LL cal(int i,LL k){

return Y[i] - k\*X[i];

}

LL Find(LL k)

{

while( he>ta+1 && cal(ta+1,k) <= cal(ta,k) )

ta++;

return cal(ta,k);

}

int main()

{

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

int n,m;

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

{

int a;

LL sum=0;

he=ta=0;

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

{

scanf("%d",&a);

sum+=a;

LL t = sum\*sum+m;

if(he>ta)

t = min( t , Find(2\*sum) + sum\*sum + m );

push( sum , t + sum\*sum );

if( i == n-1 )

printf("%lld\n",t);

}

}

return 0;

}

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

公式为0

dp[i] = dp[j] + sum[i]\*sum[i] - 2\*sum[i]\*sum[j] + sum[j]\*sum[j] + m;

dp[j] + sum[j] \* sum[j] + m = 2 \* sum[i] \* sum[j] + dp[i] - sum[i] \* sum[i];

y = dp[j] + sum[j] \* sum[j] + m

k = 2 \* sum[i]

x = sum[j]

d = dp[i] - sum[i] \* sum[i]

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

# 最小表示法

///只能寻找距离开头最近的字典序最小

int find(char \*s)

{

int n = strlen(s),p=0,q=1;

while(p<n&&q<n)

{

int i;

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

if( s[(p+i)%n] != s[(q+i)%n])

break;

if( s[(p+i)%n] > s[(q+i)%n] ) /// 字典序最小

///if( s[(p+i)%n] < s[(q+i)%n] ) /// 字典序最大

p = p + i + 1;

else

q = q + i + 1;

if(p==q) q++;

}

return min(p,q); ///返回的下标从0开始

}

# JAVA

框架：

import java.io.\*;

import java.util.\*;

import java.math.\*;

import java.text.\*;

public class Main

public static void main(String args[]) throws Exception

{

Scanner cin = new Scanner(new BufferedInputStream(System.in));

//Scanner cin=new Scanner(new File("text.txt"));

PrintWriter cout = new PrintWriter(new BufferedOutputStream(System.out));

//PrintWriter cout = new PrintWriter(new File("output.txt"));

/\*\*

主体在这里

\*\*/

/\*\*

int T=cin.nextInt();

while(T-->0)

{

}

\*\*/

/\*\*

while(cin.hasNext())

{

}

\*\*/

cout.flush();

}

}

注意：

交题时千万不要交成c++！！！ T\_T;

1.建议java全部使用显示类型转换，不支持其他类型到boolean的转换。

2. PrintWriter cout = new PrintWriter(new BufferedOutputStreamSystem.out));

使用了输出缓冲区，当没有cout.flush();时不会输出（道理同cout的 cout<<endl;），

调试时要注意。

3.java 字符串数组非常难用，建议一律使用String。

4.对于输出浮点数保留几位小数的问题，可以使用DecimalFormat类，

import java.text.\*;

DecimalFormat f = new DecimalFormat("#.00#");

DecimalFormat g = new DecimalFormat("0.000");

double a = 123.45678, b = 0.12;

System.out.println(f.format(a));

System.out.println(f.format(b));

System.out.println(g.format(b));

这里0指一位数字，#指除0以外的数字。

（这种方式对BigDecimal无效）

BigDecimal ：四舍五入

d = d . setScale( scale , BigDecimal.ROUND\_HALF\_UP ) ;

5.下面在java.util包里Arrays类的几个方法可替代C/C++里的memset、 qsort/sort 和 bsearch:

Arrays.fill()

Arrays.sort()

Arrays.binarySearch()

6.所有函数，全局变量一律加上static，省事。

使用BigInteger，BigDecimal：

1.建议写上

static BigInteger bi(int n)

{

return BigInteger.valueOf((long)n);

}

static BigInteger bi(String s)

{

return new BigInteger(s);

}

static BigDecimal bd(String s)

{

return new BigDecimal(s);

}

static BigDecimal bd(double n)

{

return BigDecimal.valueOf(n);

}

不然你会被累死的。

2.比较时

用这个 p.equals(bi(0L));

不能用p.intValue()==0;

3.BigDecimal输出时

c.stripTrailingZeros().toPlainString()

stripTrailingZeros()去末位零。

toPlainString() 防止输出诸如1e-9

4.诸如add（）一类函数，不对原值进行改变，是返回值进行改变。

要写成p=p.add(q);

# 对拍脚本

echo begin

#!/bin/becho beginash

while true; do

./r > input

./a < input > outputa

./b < input > outputb

date

#echo finish one cmpare

diff outputa outputb

if [ $? -ne 0 ] ; then break; fi

done

# BigLL

#define LL long long

const LL BIGINF = 1e18;

struct bigLL

{

LL p , big;

bigLL(){};

bigLL(LL x,LL b=0):p{x},big(b){};

bigLL operator + ( const bigLL & b ) const{

bigLL c( p+b.p , big+b.big );

c.big += c.p / BIGINF;

c.p %= BIGINF;

}

void print()

{

if( big )

{

printf("%lld",big);

LL t = BIGINF;

while( t>1 )

{

printf("%lld",(p%t)/(t/10));

t/=10;

}

}

else printf("%lld",p);

}

}

# 命令行指令

命令行指令

g++ -o r r.cpp

./r

chmod +x test.sh

./test.sh

javac Main.java

java Main

# Splay

/\*

~splay~

Key points:

(1) 结点的连接一定要用link\_child

(2) 每个结点在数列上的位置： 将这个结点旋转到根然后看它左边结点的个数

(3) 删除和添加结点之后一定要及时push\_up

\*/

const int max\_node = 2e5 + 10;

const int maxn = 2e5 + 10;

struct node

{

node\* ch[2];

node\* p;

LL sum, add, val, num;

bool rev;

node (): sum(0), add(0), val(0), num(0), rev(0){}

bool d()

{

return p->ch[1] == this;

}

void link\_child(node\* o, int d)

{

ch[d] = o;

o->p = this;

}

void up\_add(int x)

{

add += x;

sum += num\*x;

val += x;

}

void node\_rev()

{

rev ^= 1;

}

void push\_up()

{

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

sum = ch[0]->sum + ch[1]->sum + val;

}

void push\_down();

} Tnull, \*null = &Tnull;

void node :: push\_down()

{

if (add)

{

if (ch[0] != null) ch[0]->up\_add(add);

if (ch[1] != null) ch[1]->up\_add(add);

add = 0;

}

if (rev)

{

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

if (ch[0] != null) ch[0]->node\_rev();

if (ch[1] != null) ch[1]->node\_rev();

node\_rev();

}

}

node \* root, mem[max\_node], \*tot, \*arrow;

LL a[maxn];

int n, q, k1, k2;

void rot(node \* now)

{

node\* p = now->p;

p->push\_down();

now->push\_down();

int d = now->d();

p->link\_child(now->ch[!d], d);

now->p = p->p;

if (now->p != null)

p->p->link\_child(now, p->d());

now->link\_child(p, !d);

p->push\_up();

if (now->p == null)

root = now;

}

void splay(node\* now, node\* goal = null)

{

while(now->p != goal)

{

if (now->p->p == goal)

rot(now);

else

{

now->d() == now->p->d() ? rot(now->p) : rot(now);

rot(now);

}

}

now->push\_up();

if (now->p == null)

root = now;

}

node\* Find(int rk)

{

for (node\* now = root;;)

{

now->push\_down();

int sz = now->ch[0]->num;

if (sz == rk)

return now;

else if (rk > sz)

rk -= sz+1, now = now->ch[1];

else

now = now->ch[0];

}

}

node\* &get(int l, int r)

{

node\* \_l = Find(l-1);

node\* \_r = Find(r+1);

splay(\_l);

splay(\_r, \_l);

return \_r->ch[0];

}

node\* make(LL val)

{

tot->p = null;

tot->val = tot->sum = val;

tot->rev = false;

tot->add = 0;

tot->ch[0] = tot->ch[1] = null;

tot->num = 1;

return tot++;

}

node\* build(int l, int r)

{

if (l > r)

return null;

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

node\* now = make(a[m]);

now->link\_child(build(l, m-1), 0);

now->link\_child(build(m+1, r), 1);

now->push\_up();

return now;

}

void init(int \_n)

{

tot = mem;

root = make(0);

root->link\_child(make(0), 1);

root->ch[1]->link\_child(build(1,\_n), 0);

root->ch[1]->push\_up();

root->push\_up();

}

//-------

//void Treavel(node\* x);

node\* del(int pp)

{

node\* pred = Find(pp-1);

node\* succ = Find(pp+1);

splay(pred);

splay(succ, pred);

node\* tmp = succ->ch[0];

succ->ch[0] = null;

succ->push\_up();

pred->push\_up();

return tmp;

}

void Insert(int pp, int val)

{

node\* pred = Find(pp);

node\* succ = Find(pp+1);

splay(pred);

splay(succ, pred);

succ->link\_child(make(val), 0);

succ->push\_up();

pred->push\_up();

}

# LCT维护最大生成树

/\*

LCT维护边可以参考此份代码

\*/

#include <stdio.h>

#include <string.h>

#include <map>

#include <algorithm>

#include <iostream>

using namespace std ;

typedef long long LL ;

#define clr( a , x ) memset ( a , x , sizeof a )

#define ls ( o << 1 )

#define rs ( o << 1 | 1 )

#define lson ls , l , m

#define rson rs , m + 1 , r

#define root 1 , 1 , n

#define mid ( ( l + r ) >> 1 )

const int MAXN = 100005 ;

const int MAXE = 2000005 ;

const int INF = 0x3f3f3f3f ;

struct Edge {

int to, from;

Edge (){}

Edge ( int to , int from) : to (to) , from (from){}

} ;

struct Node\* null ;

struct Node {

Node\* c[2] ;

Node\* f ;

bool flip ;

int minv , val ;

int eidx , idx ;

void newnode ( int v , int i ) {

c[0] = c[1] = f = null ;

minv = val = v ;

eidx = idx = i ;

flip = 0 ;

}

void rev () {

if ( this == null ) return ;

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

flip ^= 1 ;

}

void up () {

if ( this == null ) return ;

if ( val <= c[0]->minv && val <= c[1]->minv ) {

minv = val ;

eidx = idx ;

} else if ( c[0]->minv <= c[1]->minv && c[0]->minv <= val ) {

minv = c[0]->minv ;

eidx = c[0]->eidx ;

} else {

minv = c[1]->minv ;

eidx = c[1]->eidx ;

}

}

void down () {

if ( this == null ) return ;

if ( flip ) {

c[0]->rev () ;

c[1]->rev () ;

flip = 0 ;

}

}

bool is\_root () {

return f == null || f->c[0] != this && f->c[1] != this ;

}

void sign\_down () {

if ( !is\_root () ) f->sign\_down () ;

down () ;

}

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

c[d] = o ;

o->f = this ;

}

void rot ( int d ) {

Node\* p = f ;

Node\* g = f->f ;

p->setc ( c[d] , !d ) ;

if ( !p->is\_root () ) g->setc ( this , f == g->c[1] ) ;

else f = g ;

setc ( p , d ) ;

p->up () ;

}

void splay () {

sign\_down () ;

while ( !is\_root () ) {

if ( f->is\_root () ) rot ( this == f->c[0] ) ;

else {

if ( f == f->f->c[0] ) {

if ( this == f->c[0] ) f->rot ( 1 ) , rot ( 1 ) ;

else rot ( 0 ) , rot ( 1 ) ;

} else {

if ( this == f->c[1] ) f->rot ( 0 ) , rot ( 0 ) ;

else rot ( 1 ) , rot ( 0 ) ;

}

}

}

up () ;

}

void access () {

Node\* o = this ;

for ( Node\* x = null ; o != null ; x = o , o = o->f ) {

o->splay () ;

o->setc ( x , 1 ) ;

o->up () ;

}

splay () ;

}

void make\_root () {

access () ;

rev () ;

}

void link ( Node\* o ) {

make\_root () ;

f = o ;

}

void cut () {

access () ;

c[0] = c[0]->f = null ;

up () ;

}

void cut ( Node\* o ) {

make\_root () ;

o->cut () ;

}

int get\_min ( Node\* o ) {

make\_root () ;

o->access () ;

return o->eidx ;

}

} ;

Node pool[MAXN + MAXE] ;

Node\* cur ;

Node\* node[MAXN] ;

Node\* edge[MAXE] ;

Edge E[MAXE + MAXN] ;

int U[MAXE] , V[MAXE] ;

int n,idx;

LL ans;

LL dp[MAXN];

bool used[MAXE];

void init (int n) {

idx = 0, ans=0;

cur = pool;

cur->newnode ( INF , -1 ) ;

null = cur ++ ;

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

cur->newnode(INF, -1) ;

node[i] = cur ++ ;

}

}

void add\_edge(int u,int v,bool flag)

{

// if(v==15)printf("add%d\_\_\_%d:\n",u,v);

if(flag){

U[idx] = u ;

V[idx] = v ;

cur->newnode(u, idx);

edge[idx]=cur;

E[idx]=Edge(v,u);

edge[idx]->link(node[u]);

edge[idx]->link(node[v]);

used[idx]=true;

ans+=u;

idx++;

cur++;

// if(v==15)printf("successful\n");

}

else{

int eidx=node[u]->get\_min(node[v]);

/\* if(v==15){

printf("should dele E[%d]\_\_\_\_%d->%d\n",eidx,E[eidx].from,E[eidx].to);

}\*/

if(~eidx&&E[eidx].from<u){

U[idx] = u;

V[idx] = v;

cur->newnode(u, idx);

edge[idx]=cur;

E[idx]=Edge(v,u);

// if(v==15)printf("successful\n");

// if(v==15)printf("dele %d\_\_\_%d\n",U[eidx],V[eidx]);

edge[eidx]->cut (node[U[eidx]]) ;

edge[eidx]->cut (node[V[eidx]]) ;

edge[idx]->link (node[u]);

edge[idx]->link (node[v]);

used[eidx]=false;

used[idx]=true;

ans+=u-E[eidx].from;

idx++;

cur++;

}

}

}

void solve () {

n=100000;

init(n);

vector<int> vs;

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

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

if(i%j==0){

vs.push\_back(j);

if(i/j!=j&&j!=1){

vs.push\_back(i/j);

}

}

}

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

for(int j=vs.size()-1;j>=0;j--){

add\_edge(vs[j],i,j==vs.size()-1);

}

// if(i%1000==0)printf("\_\_%d\n",i);

vs.clear();

dp[i]=ans;

}

}

int main()

{

solve();

int m;

while(cin>>m)printf("%I64d\n",dp[m]);

}

# 动态树LCT

/\*

HDU 4010

包含如下几个操作：

(1) 将x 和 y 之间加一条边，将这两个结点所在的树合并

(2) 将x变成树的根，删除y与其父亲的边 (一定要注意这个关系)

(3) 将x ~ y 的路径上每个结点的值增加w

(4) 查询x ~ y路径上结点值的最大值

\*/

const int maxn = 300000 + 20;

struct Node\* null;

struct Node

{

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

LL val, rmax, add;

bool rev;

inline void make(LL \_val)

{

add = 0;

val = rmax = \_val;

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

}

inline void add\_val(LL \_val)

{

if (this == null) return ;

rmax += \_val;

add += \_val;

val += \_val;

}

inline void push\_up()

{

rmax = val;

rmax = max(rmax, ch[0]->rmax);

rmax = max(rmax, ch[1]->rmax);

}

inline void setc(Node\* p, int d)

{

ch[d] = p;

p->fa = this;

}

inline bool d()

{

return fa->ch[1] == this;

}

inline bool isroot()

{

return fa == null || fa->ch[0] != this && fa->ch[1] != this;

}

inline void flip()

{

if (this == null) return;

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

rev ^= 1;

}

inline void push\_down()

{

if (add)

{

ch[0]->add\_val(add);

ch[1]->add\_val(add);

add = 0;

}

if (rev)

{

ch[0]->flip();

ch[1]->flip();

rev = 0;

}

}

inline void go()

{

if (!isroot()) fa->go();

push\_down();

}

inline void rot()

{

Node\* f = fa, \*ff = fa->fa;

int c = d(), cc = fa->d();

f->setc(ch[!c], c);

this->setc(f, !c);

if (ff->ch[cc] == f) ff->setc(this, cc);

else this->fa = ff;

f->push\_up();

}

inline Node\* splay()

{

go();

while(!isroot())

{

if (!fa->isroot())

d() == fa->d() ? fa->rot() : rot();

rot();

}

push\_up();

return this;

}

inline Node\* access()

{

for (Node\* p = this, \*q = null; p != null; q = p, p = p->fa)

{

p->splay()->setc(q, 1);

p->push\_up();

}

return splay();

}

inline Node\* find\_root()

{

Node\* x;

for (x = access(); x->push\_down(), x->ch[0] != null; x = x->ch[0]);

return x;

}

void make\_root()

{

access()->flip();

}

void cut()

{

access();

ch[0]->fa = null;

ch[0] = null;

push\_up();

}

void cut(Node\* x)

{

if (this == x || find\_root() != x->find\_root())

{printf("-1\n");return ;}

else

{

make\_root();

x->cut();

}

}

void link(Node\* x)

{

if (find\_root() == x->find\_root())

{printf("-1\n");return ;}

else

{

make\_root();

fa = x;

}

}

}pool[maxn], \*tail;

Node\* node[maxn];

int n, a[maxn], q;

pair<int,int> E[maxn];

void solve\_add(int u, int v, LL cc)

{

if (node[u]->find\_root() != node[v]->find\_root())

{printf("-1\n");return ;}

node[u]->make\_root();

node[v]->access();

node[v]->add\_val(cc);

}

LL solve\_query(int u, int v)

{

if (node[u]->find\_root() != node[v]->find\_root())

return -1;

node[u]->make\_root();

node[v]->access();

return node[v]->rmax;

}

int main()

{

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

int st, ed, kk, cc;

LL tmp;

while(scanf("%d", &n) == 1)

{

tail = pool;

null = tail++;

null->make(-inf);

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

scanf("%d%d", &E[i].first, &E[i].second);

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

scanf("%I64d", &tmp),node[i] = tail++, node[i]->make(tmp);

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

node[E[i].first]->link(node[E[i].second]);

scanf("%d", &q);

while(q--)

{

scanf("%d", &kk);

if (kk == 1)

{

scanf("%d%d", &st, &ed);

node[st]->link(node[ed]);

}

else if (kk == 2)

{

scanf("%d%d", &st, &ed);

node[st]->cut(node[ed]);

}

else if (kk == 3)

{

scanf("%I64d%d%d", &cc, &st, &ed);

solve\_add(st, ed, cc);

}

else if (kk == 4)

{

scanf("%d%d", &st, &ed);

printf("%I64d\n", solve\_query(st, ed));

}

}

printf("\n");

}

return 0;

}

# Dancing-Links 精确覆盖（数独）

/\*

注意行和列的下标都一定要从1开始！！

\*/

const int N = 9;

const int maxn = N\*N\*N + 10;

const int maxm = N\*N\*4 + 10;

const int maxnode = maxn\*maxm;

int ANS[9][9];

struct DLX

{

int n,m,sz;

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;

sz = m;

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

}

void Link(int r,int c)

{

++S[Col[++sz]=c];

Row[sz] = r;

D[sz] = D[c];

U[D[c]] = sz;

U[sz] = c;

D[c] = sz;

if(H[r] < 0)H[r] = L[sz] = R[sz] = sz;

else

{

R[sz] = R[H[r]];

L[R[H[r]]] = sz;

L[sz] = H[r];

R[H[r]] = sz;

}

}

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++)

{

ans[i]--;

int k = ans[i]%9; ans[i] /= 9;

int y = ans[i]%9; ans[i] /= 9;

int x = ans[i]%9; ans[i] /= 9;

ANS[x][y] = k+1;

}

for (int i = 0; i < 9; i++)

{

for (int j = 0; j < 9; j++)

printf("%d", ANS[i][j]);

}

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;

}

} dlx;

int n, m;

char cc[100];

void add(int i, int j, int idx)

{

int idy = (i/3)\*3 + j/3;

int r = i\*81 + j\*9 + idx;

int a = i\*9 + j + 1, b = i\*9 + idx + 81;

int c = j\*9 + idx + 81\*2, d = idy\*9 + idx + 81\*3;

dlx.Link(r, a);

dlx.Link(r, b);

dlx.Link(r, c);

dlx.Link(r, d);

}

int main()

{

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

//freopen("output.txt", "w", stdout);

int n = N\*N\*N, m = N\*N\*4;

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

{

if (strcmp(cc, "end") == 0) break;

dlx.init(n, m);

for (int i = 0; i < 9; i++)

for (int j = 0; j < 9; j++)

{

if (cc[i\*N + j] == '.')

{

for (int idx = 1; idx <= 9; idx++)

add(i, j, idx);

}

else

{

int idx = cc[i\*N + j] - '0';

add(i, j, idx);

}

}

dlx.Dance(0);

}

return 0;

}

# Dancing-Links 重复覆盖

/\*

HDU5046

题意：

给出n个城市，两两城市之间有一个距离，从中选择K个城市建成机场，让最大的距离最小(n<= 60),时限是1500ms

解法：

二分答案，然后用DLX重复覆盖来判

\*/

#include <stdio.h>

#include <string.h>

#include <iostream>

#include <algorithm>

#include <vector>

#include <queue>

#include <set>

#include <map>

#include <string>

#include <math.h>

#include <stdlib.h>

#include <time.h>

#include <assert.h>

using namespace std;

const int maxnode = 4000;

const int MaxM = 70;

const int MaxN = 70;

int K;

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 ands,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)

{

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[maxnode];

// A\* 剪枝

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;

}

bool Dance(int d)

{

if(d + f() > K)return false;

if(R[0] == 0)return d <= K;

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);

if(Dance(d+1))return true;

for(int j = L[i];j != i;j = L[j])resume(j);

resume(i);

}

return false;

}

};

DLX g;

struct Point

{

int x,y;

void input()

{

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

}

}city[MaxM];

long long dis(Point a,Point b)

{

return (long long)abs(a.x-b.x)+(long long)abs(a.y-b.y);

}

int main()

{

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

//freopen("E.out","w",stdout);

int T;

int n;

scanf("%d",&T);

int iCase = 0;

while(T--)

{

iCase++;

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

assert(n >= 1 && n <= 60 && K >= 1 && K <= n);

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

city[i].input();

assert(abs(city[i].x) <= 1000000000);

assert(abs(city[i].y) <= 1000000000);

}

long long l = 0, r = 100000000000LL;

long long ans = 0;

while(l <= r)

{

long long mid = (l+r)/2;

g.init(n,n);

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

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

if(dis(city[i],city[j]) <= mid)

g.Link(i+1,j+1);

if(g.Dance(0)){r = mid-1;ans = mid;}

else l = mid+1;

}

printf("Case #%d: %I64d\n",iCase,ans);

}

return 0;

}

# 康拓展开

/\*

将3\*3的矩阵hash到0~9!

\*/

const int base[9] = {1,1,2,6,24,120,720,5040,40320};

struct sad

{

int f[3][3], h, hash\_id;

int x, y;

int get\_hash()

{

int a[9],i,j,ii,jj,k=0,ans=0;

for(i=0; i<3; i++)

{

for(j=0; j<3; j++)

a[k++] = f[i][j];

}

for(i=0; i<9; i++)

{

k=0;

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

if(a[j]>a[i])k++;

ans += base[i]\*k;

}

return ans;

}

};

# 2-SAT

/\*

复杂度O(n)

\*/

struct two\_sat

{

int n;

vector<int> G[maxn << 1];

bool mark[maxn << 1];

int S[maxn << 1], c;

bool dfs(int x)

{

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

if (mark[x]) return true;

mark[x] = true;

S[c++] = x;

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

{

if (!dfs(G[x][i])) return false;

}

return true;

}

void init(int n)

{

this->n = n;

for (int i = 0; i < (n << 1); i++) G[i].clear();

clr(mark, 0);

}

// x = xval or y = yval

//2i + 1 means true 2i means false

void add\_clause(int x, int xval, int y, int yval)

{

x = x \* 2 + xval;

y = y \* 2 + yval;

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

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

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

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

}

bool solve()

{

for (int i = 0; i < (n << 1); i += 2)

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

{

c = 0;

if (!dfs(i))

{

while (c > 0) mark[S[--c]] = false;

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

}

}

return true;

}

} sat;

# ISAP 网络流

const int maxn = 1050;

const int maxm = 100005;

struct edge

{

int to, next;

int cap;

} G[maxm];

int head[maxn], si;

void add(int st, int ed, int val)

{

G[si].to = ed;

G[si].cap = val;

G[si].next = head[st];

head[st] = si++;

G[si].to = st;

G[si].cap = 0;

G[si].next = head[ed];

head[ed] = si++;

}

int nn, n, m;

int h[maxn], gap[maxn];

int source, sink;

int dfs(int u, int cost)

{

if (u == sink) return cost;

int minh = nn - 1, lv = cost, d;

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

{

int v = G[i].to;

int val = G[i].cap;

if (val > 0)

{

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

{

d = min(val, lv);

d = dfs(v, d);

G[i].cap -= d;

G[i ^ 1].cap += d;

lv -= d;

if (h[source] >= nn) return cost - lv;

if (lv == 0) break;

}

if (h[v] < minh) minh = h[v];

}

}

if (lv == cost)

{

--gap[h[u]];

if (gap[h[u]] == 0) h[source] = nn;

h[u] = minh + 1;

++gap[h[u]];

}

return cost - lv;

}

int sap(int st, int ed)

{

source = st;

sink = ed;

int ret = 0;

clr(gap, 0); clr(h, 0);

gap[st] = nn;

while (h[st] < nn)

{

ret += dfs(st, inf);

}

return ret;

}

# 最小费用最大流

const int maxn = 2000;

const int maxm = 2500000;

struct edge

{

int to, next;

int cap, flow, cc;

} G[maxm];

int head[maxn], si;

int pre[maxn], dis[maxn];

bool vis[maxn];

int nn, n, m, k;

void add(int st, int ed, int val, int cost)

{

G[si].to = ed;

G[si].cap = val;

G[si].cc = cost;

G[si].flow = 0;

G[si].next = head[st];

head[st] = si++;

G[si].to = st;

G[si].cap = 0;

G[si].cc = -cost;

G[si].flow = 0;

G[si].next = head[ed];

head[ed] = si++;

}

bool spfa(int s, int t)

{

queue<int> que;

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

{

dis[i] = inf;

vis[i] = false;

pre[i] = -1;

}

dis[s] = 0;

vis[s] = true;

que.push(s);

while (que.size())

{

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

vis[u] = false;

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

{

int v = G[i].to;

if (G[i].cap > G[i].flow &&

dis[v] > dis[u] + G[i].cc)

{

dis[v] = dis[u] + G[i].cc;

pre[v] = i;

if (!vis[v])

{

vis[v] = true;

que.push(v);

}

}

}

}

// if (dis[t] > 0) return false; 加上这句话可以保证只求出最小费用，不要求最大流

if (pre[t] == -1) return false;

return true;

}

int min\_cost\_maxflow(int s, int t, int &cost)

{

int flow = 0;

cost = 0;

while (spfa(s, t))

{

int rmin = inf;

for (int i = pre[t]; i != -1; i = pre[G[i ^ 1].to])

rmin = min(rmin, G[i].cap - G[i].flow);

for (int i = pre[t]; i != -1; i = pre[G[i ^ 1].to])

{

G[i].flow += rmin;

G[i ^ 1].flow -= rmin;

cost += G[i].cc \* rmin;

}

flow += rmin;

}

return flow;

}

# 尺取法

/\*

HDU 5289

求一个序列连续区间最大值-最小值不超过K的个数

\*/

#include <cstdio>

#include <iostream>

#include <algorithm>

#include <cstring>

#include <string>

#include <cmath>

#include <queue>

#include <bitset>

#define clr(x, y) memset(x, y, sizeof x)

#define inf 1000000000

using namespace std;

typedef long long LL;

const int maxn = 200010;

int A[maxn];

int dmax[maxn][30];

int dmin[maxn][30];

int flog[(maxn << 1) + 10];

int n, K;

int rmax\_init(int A[])

{

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

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

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

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

}

int rmax\_find(int L, int R)

{

int k = flog[R - L + 1];

return max(dmax[L][k], dmax[R - (1 << k) + 1][k]);

}

int rmin\_init(int A[])

{

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

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

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

dmin[i][j] = min(dmin[i][j - 1], dmin[i + (1 << (j - 1))][j - 1]);

}

int rmin\_find(int L, int R)

{

int k = flog[R - L + 1];

return min(dmin[L][k], dmin[R - (1 << k) + 1][k]);

}

int main()

{

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

flog[0] = -1;

for (int i = 1; i < 2 \* maxn; i++) flog[i] = flog[i >> 1] + 1;

int T, he, ta;

LL ans;

scanf("%d", &T);

while (T--)

{

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

for (int i = 0; i < n; i++) scanf("%d", &A[i]);

rmax\_init(A); rmin\_init(A);

he = ta = 0;

ans = 0;

while (he < n)

{

while (ta < n && rmax\_find(he, ta) - rmin\_find(he, ta) < K) { ta++; }

ans += ta - he;

he++;

}

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

}

return 0;

}

# 倍增法求LCA

/\*

倍增法求LCA，复杂度O(nlogn)

\*/

const int maxn = 100000;

const int maxk = 30;

struct edge

{

int to, next;

} G[maxn << 1];

int head[maxn], si;

int parent[maxk][maxn]; //注意第一维为小的

int depth[maxn];

void dfs(int u, int p, int d)

{

parent[0][u] = p;

depth[u] = d;

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

{

int v = G[i].to;

if (v != p) dfs(v, u, d + 1);

}

}

void init\_lca(int \_n)

{

dfs(1, -1, 0);

for (int k = 0; k + 1 < maxk; k++)

{

for (int u = 1; u <= \_n; u++) //注意下标是0~n-1 还是1~n

{

if (parent[k][u] < 0) parent[k + 1][u] = -1;

else parent[k + 1][u] = parent[k][parent[k][u]];

}

}

}

int get\_lca(int u, int v)

{

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

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

{

if ((depth[v] - depth[u]) >> k & 1)

{

v = parent[k][v];

}

}

if (u == v) return u;

for (int k = maxk - 1; k >= 0; k--)

{

if (parent[k][u] != parent[k][v])

{

u = parent[k][u];

v = parent[k][v];

}

}

return parent[0][u];

}

# 边双连通分量

const int maxn = 2000;

const int maxm = 550;

struct edge

{

int to, next;

}G[maxn << 1];

int head[maxn], si;

int pre[maxn], dfs\_clock, bridge;

int par[maxn], n;

void init(int \_n)

{

for (int i = 0; i <= \_n; i++)

par[i] = i;

}

int Find(int x)

{

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

return par[x] = Find(par[x]);

}

void unite(int x, int y)

{

x = Find(x);

y = Find(y);

if (x == y) return;

par[x] = y;

}

int Tarjan(int u, int fa)

{

int lowu = pre[u] = ++dfs\_clock;

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

{

int v = G[i].to;

if (v == fa) continue;

if (pre[v] == 0)

{

int lowv = Tarjan(v, u);

lowu = min(lowu, lowv);

if (lowv <= pre[u]) unite(u, v);

// 表示是桥

else if (lowv > pre[u]) bridge++;

}

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

}

return lowu;

}

void find\_bridge(int \_n)

{

clr(pre, 0);

dfs\_clock = 0;

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

if (!pre[i]) Tarjan(i, -1);

}

# 点双连通分量

#include <cstdio>

#include <iostream>

#include <algorithm>

#include <cstring>

#include <vector>

#include <queue>

#define clr(x, y) memset(x, y, sizeof x)

#define inf 1000000000

using namespace std;

typedef long long LL;

const int maxn = 1505;

int pre[maxn], iscut[maxn], bccno[maxn], dfs\_clock, bcc\_cnt;

vector<int> G[maxn], bcc[maxn];

int n, m;

struct edge

{

int st, ed;

};

stack<edge> S;

int dfs(int u, int fa)

{

int lowu = pre[u] = ++dfs\_clock;

int child = 0;

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

{

int v = G[u][i];

Edge e = (Edge){ u, v };

if (!pre[v])

{

S.push(e);

child++;

int lowv = dfs(v, u);

lowu = min(lowu, lowv);

if (lowv >= pre[u])

{

iscut[u] = true;

bcc\_cnt++; bcc[bcc\_cnt].clear();

for (;;)

{

edge x = S.top(); S.pop();

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

{

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

bcc\_cnt[x.u] = bcc\_cnt;

}

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

{

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

bcc\_cnt[x.v] = bcc\_cnt;

}

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

}

}

}

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

{

S.push(e);

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

}

}

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

return lowu;

}

void find\_bcc(int n)

{

clr(pre, 0); clr(iscut, 0); clr(bccno, 0);

dfs\_clock = bcc\_cnt = 0;

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

{

if (!pre[i]) dfs(i, -1);

}

}

# LCT动态树

/\*

包含最基本的加入操作，删除操作，询问两个点是否联通

\*/

#include <cstdio>

#include <cstring>

#include <algorithm>

using namespace std;

const int MAXN = 10005;

struct Node\* null;

struct Node {

Node\* c[2];

Node\* f;

int flip;

void newnode() {

c[0] = c[1] = f = null;

flip = 0;

}

void reverse() {

if (this == null) return;

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

flip ^= 1;

}

void link\_child(Node\* o, int d) {

c[d] = o;

o->f = this;

}

int is\_root() {

return f == null || f->c[0] != this && f->c[1] != this;

}

void push\_down() {

if (flip) {

c[0]->reverse();

c[1]->reverse();

flip = 0;

}

}

void sign\_down() {

if (!is\_root()) f->sign\_down();

push\_down();

}

void rotate(int d) {

Node\* p = f;

Node\* g = p->f;

p->link\_child(c[d], !d);

if (!p->is\_root()) {

if (p == g->c[0]) g->link\_child(this, 0);

else g->link\_child(this, 1);

}

else f = g;

this->link\_child(p, d);

}

void splay() {

sign\_down();

while (!is\_root()) {

if (f->is\_root()) rotate(this == f->c[0]);

else {

if (f == f->f->c[0]) {

if (this == f->c[0]) f->rotate(1), rotate(1);

else rotate(0), rotate(1);

}

else {

if (this == f->c[1]) f->rotate(0), rotate(0);

else rotate(1), rotate(0);

}

}

}

}

void access() {

Node\* o = this;

Node\* x = null;

while (o != null) {

o->splay();

o->link\_child(x, 1);

x = o;

o = o->f;

}

splay();

}

Node\* find\_root() {

access();

Node\* o = this;

while (o->c[0] != null) o = o->c[0];

return o;

}

void make\_root() {

access();

reverse();

}

void cut() {

access();

c[0]->f = null;

c[0] = null;

}

void cut(Node\* o) {

if (o->find\_root() != find\_root()) return;

make\_root();

o->cut();

}

void link(Node\* o) {

if (o == this || o->find\_root() == find\_root()) return;

make\_root();

f = o;

}

void query(Node\* o) {

if (o->find\_root() == find\_root()) printf("Yes\n");

else printf("No\n");

}

};

Node pool[MAXN];

Node\* node[MAXN];

Node\* cur;

int n, m;

void clear() {

cur = pool;

null = cur++;

null->newnode();

}

void solve() {

char s[20];

int x, y;

clear();

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

node[i] = cur++;

node[i]->newnode();

}

while (m--) {

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

if (s[0] == 'C') node[x]->link(node[y]);

if (s[0] == 'D') node[x]->cut(node[y]);

if (s[0] == 'Q') node[x]->query(node[y]);

}

}

int main() {

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

return 0;

}

# 强连通分量缩环

const int maxn = 1000;

struct edge

{

int to, next;

}G[maxn << 1];

int head[maxn], si;

int pre[maxn], lowlink[maxn], sccno[maxn], dfs\_clock, scc\_cnt;

stack<int> S;

void Tarjan(int u)

{

pre[u] = lowlink[u] = ++dfs\_clock;

S.push(u);

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

{

int v = G[i].to;

if (!pre[v])

{

Tarjan(v);

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

}

else if (!sccno[v])

{

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

}

}

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

{

scc\_cnt++;

while (1)

{

int x = S.top(); S.pop();

sccno[x] = scc\_cnt;

if (x == u) break;

}

}

}

void find\_scc(int \_n)

{

dfs\_clock = scc\_cnt = 0;

clr(sccno, 0); clr(pre, 0);

for (int i = 0; i < \_n; i++)

if (!pre[i]) Tarjan(i);

}

# 二分图匹配

/\*

二分图匹配模板

跑点数较少的一侧点， 复杂度为O(VE)

对于一般图匹配，将可以将点数乘以2建成二分图，通过二分图匹配可以求出来正确的答案(ans/2)

\*/

const int maxn = 3000;

vector<int> G[maxn];

int match[maxn], used[maxn];

bool dfs(int u, int mark)

{

used[u] = mark;

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

{

int v = G[u][i];

int w = match[v];

if (w < 0 || used[w] != mark && dfs(w, mark))

{

match[v] = u;

match[u] = v;

return true;

}

}

return false;

}

int bipartite\_matching(int \_n)

{

int res = 0;

clr(match, -1);

for (int u = 0; u < \_n; u++)

{

if (match[u] < 0)

{

if (dfs(u, u)) res++;

}

}

return res;

}

# [O(n)预处理C(n,m)](http://blog.csdn.net/smilewsw/article/details/47701785)

const int maxn=1000100;

const int mod=1000000007;

typedef long long LL;

LL fac[maxn],f[maxn],inv[maxn];

void init()

{

fac[0]=fac[1]=f[0]=f[1]=inv[0]=inv[1]=1;

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

{

fac[i]=fac[i-1]\*i%mod; // n！

LL t=mod/i,k=mod%i;

f[i]=(mod-t)\*f[k]%mod; // n的逆元

inv[i]=inv[i-1]\*f[i]%mod; // n！的逆元

}

}

LL c(LL n,LL m)

{

if(n<m) return 0;

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

}

# 快速幂求矩阵前缀和

const int mod=2015;

struct matrix

{

int m[60][60];

};

matrix mul(matrix a,matrix b)

matrix add(matrix a,matrix b)

matrix ma;

matrix sum\_pow(matrix a,int x)

{

matrix ans;

if(x%2==0)

{

matrix p = sum\_pow(a,x/2);

ans = add( p , mul( p , ma ) );

ma = mul( ma , ma );

}

else

{

if(x==1) return ma = a;

ans = add( a , mul( sum\_pow( a , x-1 ) , a ) );

ma = mul( ma , a );

}

return ans;

}

# 莫比乌斯反演

莫比乌斯反演的两种形式：

http://img.blog.csdn.net/20140417200055140

http://img.blog.csdn.net/20140417200609375

**定理：**http://img.blog.csdn.net/20140416155848046和http://img.blog.csdn.net/20140416155900312是定义在非负整数集合上的两个函数，并且满足条件http://img.blog.csdn.net/20140416155727296，那么我们得到结论 http://img.blog.csdn.net/20140416160124562

例如：F[x]表示gcd为x的倍数的区间的个数；

f[x]表示gcd为x的区间的个数；

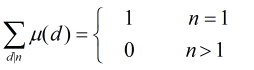
在上面的公式中有一个http://img.blog.csdn.net/20140416160256500函数，它的定义如下：

（1）若http://img.blog.csdn.net/20140416160429140，那么http://img.blog.csdn.net/20140416160522250

（2）若http://img.blog.csdn.net/20140416160632156，http://img.blog.csdn.net/20140416160716765均为互异素数，那么http://img.blog.csdn.net/20140416160909015

（3）其它情况下http://img.blog.csdn.net/20140416160949437

对于http://img.blog.csdn.net/20140416160256500函数，它有如下的常见性质：

（1）对任意正整数http://img.blog.csdn.net/20140416161324484有  

（2）对任意正整数http://img.blog.csdn.net/20140416161324484有 http://img.blog.csdn.net/20140416162025500

//预处理1~100000的莫比乌斯系数

int mu[MAXN];

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

{

int target= i==1 ? 1:0;

int delta=target-mu[i];

mu[i]=delta;

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

mu[j]+=delta;

}

//得到<=r的质数的各种组合的乘积

int prime[30]={-2,-3,-5,-7,-11,-13,-17,-19,-23,-29,-31,-37,-41,-43,-47,-53,-59};

//用负数，便于容斥时判加减

vector <int> m;

void init()

{

m.clear();

for(int i=0;abs(prime[i])<=r;i++)

{

int si=m.size();

for(int j=0;j<si;j++)

{

if(abs(m[j]\*prime[i])<=63)

m.push\_back(m[j]\*prime[i]);

}

m.push\_back(prime[i]);

}

}

# 原根与离散对数

## .1 原根

指数的定义：若m>1,(a,m)=1,则使得同余式a^r=1(mod m)成立的最小正整数r叫做a对于模m的指数(阶),记r为ordm(a).

原根的定义：若(a,m)=1,m>0,且ordm(a)=φ(m),则a称为模m的一个原根.

注：并不是所有的模m都存在原根. 原根存在的条件：模m为1, 2, 4, p^t, 2p^t，其中p为奇质数，t为正整数时，此模m才有原根。

原根的性质:若ordm(a)=t, u是一个正整数，则ordm(a^u)=t/(t,u).  
推论: 若a是模m的一个原根, m为整数且m>1, 则a^u是模m的一个原根的充要条件是(u,φ(m))=1.

问题：求一个mod p意义下的原根（p为素数）

原理：由于最小原根通常较小, 可以通过从小到大枚举正整数来快速寻找一个原根.对p-1的每个素因子a，检查g^((p-1)/a)=1(mod p)是否成立，如果成立则说明不是原根

//求原根

vector<LL>a;

bool g\_test(LL g,LL p)

{

for(LL i=0;i<a.size();i++)

{

if(pow\_mod(g,(p-1)/a[i],p)==1)

return 0;

}

return 1;

}

LL primitive\_root(LL p)

{

LL tmp=p-1;

a.clear();

for(LL i=2;i<=tmp/i;i++)

{

if(tmp%i==0)

{

a.push\_back(i);

while(tmp%i==0)

tmp/=i;

}

}

if(tmp!=1)

a.push\_back(tmp);

LL g=1;

while(true)

{

if(g\_test(g,p))

return g;

g++;

}

}

## .2 离散对数

**当模**http://img.blog.csdn.net/20140212125227906**有原根时,设**http://img.blog.csdn.net/20140212125339046**为模**http://img.blog.csdn.net/20140212125227906**的一个原根，则当**http://img.blog.csdn.net/20140212125608562**时，**http://img.blog.csdn.net/20140212125742156**。**

**问题１：**给定A,B,C,求同余方程http://img.blog.csdn.net/20140212130455484的解，其中http://img.blog.csdn.net/20140212130549828是素数。

**问题２：**给定Ｘ,B,C,求同余方程http://img.blog.csdn.net/20140212130455484的解，其中http://img.blog.csdn.net/20140212130549828是素数。

**分析：**利用离散对数的知识，先求模http://img.blog.csdn.net/20140212130549828的一个原根http://img.blog.csdn.net/20140212125339046，那么就有http://img.blog.csdn.net/20140212131305875。对于http://img.blog.csdn.net/20140212131437531，用Baby Step Giant Step能很好地解决.那么这样我们再用扩展欧几里得算法可以计算出A或者http://img.blog.csdn.net/20140212131607546，快速幂再进一步求http://img.blog.csdn.net/20140212125958234，所以就可以完美解决。

**那么，如果**http://img.blog.csdn.net/20140212130549828**为合数呢？**

**其实，如果**http://img.blog.csdn.net/20140212130549828**为合数，我们要做的第一件事就是把**http://img.blog.csdn.net/20140212130549828**素因子分解，即**http://img.blog.csdn.net/20140212132208250**，**

**那么我们分别计算**http://img.blog.csdn.net/20140212132433703**，然后用CRT(中国剩余定理)合并即可。**

**那么对于**http://img.blog.csdn.net/20140212132433703**，有两种情况：**

**（1）**http://img.blog.csdn.net/20140212133815218

**（2）**http://img.blog.csdn.net/20140212133932437

**对于情况（1），我们就直接先求原根，然后利用离散对数来解决。**

**而对于情况（2），我们有**http://img.blog.csdn.net/20140212134210421**，这样就转化为情况（1）了。假设对于每个**http://img.blog.csdn.net/20140212132433703**的解的个数为**http://img.blog.csdn.net/20140212134532593**，那么方程**http://img.blog.csdn.net/20140212130455484**的解的个数为**http://img.blog.csdn.net/20140212134633296

//baby\_step

LL pow\_mod(LL a,LL p,LL n)

{

LL ans=1;

while(p)

{

if(p&1)

ans=(ans\*a)%n;

p>>=1;

a=(a\*a)%n;

}

return ans;

}

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

{

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

else

{

gcd(b,a%b,d,y,x);

y-=x\*(a/b);

}

}

LL inv(LL a,LL n)

{

LL d,x,y;

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

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

}

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

{

int m,v,e=1,i;

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

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

map<int,int> x;

x[1]=0;

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

{

e=e\*a%n;

if(!x.count(e))

x[e]=i;

}

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

{

if(x.count(b))

return i\*m+x[b];

b=b\*v%n;

}

return -1;

}

# 高斯消元模板

高斯消元法的原理是：

若用初等行变换将增广矩阵 化为 ，则AX = B与CX = D是同解方程组。  
所以我们可以用初等行变换把增广矩阵转换为行阶梯阵，然后回代求出方程的解。

以上是线性代数课的回顾，下面来说说高斯消元法在编程中的应用。

首先，先介绍程序中高斯消元法的步骤：  
(我们设方程组中方程的个数为equ，变元的个数为var，注意：一般情况下是n个方程，n个变元，但是有些题目就故意让方程数与变元数不同)

1. 把方程组转换成增广矩阵。

2. 利用初等行变换来把增广矩阵转换成行阶梯阵。  
枚举k从0到equ – 1，当前处理的列为col(初始为0) ，每次找第k行以下(包括第k行)，col列中元素绝对值最大的列与第k行交换。如果col列中的元素全为0，那么则处理col + 1列，k不变。

3. 转换为行阶梯阵，判断解的情况。

① 无解  
当方程中出现(0, 0, …, 0, a)的形式，且a != 0时，说明是无解的。

② 唯一解  
条件是k = equ，即行阶梯阵形成了严格的上三角阵。利用回代逐一求出解集。

③ 无穷解。  
条件是k < equ，即不能形成严格的上三角形，自由变元的个数即为equ – k，但有些题目要求判断哪些变元是不缺定的。  
    这里单独介绍下这种解法：  
首先，自由变元有var - k个，即不确定的变元至少有var - k个。我们先把所有的变元视为不确定的。在每个方程中判断不确定变元的个数，如果大于1个，则该方程无法求解。如果只有1个变元，那么该变元即可求出，即为确定变元。

以上介绍的是求解整数线性方程组的求法，复杂度是O(n3)。

浮点数线性方程组的求法类似，但是要在判断是否为0时，加入EPS，以消除精度问题。

求解同余方程组问题。与一般求解线性方程组的问题类似，只要在求解过程中加入取余即可。

**————————————————————————————**

// 高斯消元法解方程组(Gauss-Jordan elimination).

参数含义：有equ个方程，var个变元。

增广矩阵行数为equ,分别为0到equ-1,列数为var+1,分别为0到var.

返回值：-2表示有浮点数解，但无整数解，

-1表示无解，0表示唯一解，

大于0表示无穷解，并返回自由变元的个数

调用函数：gdc(a,b), lcm(a,b)

const int MAXN=50;

int a[MAXN][MAXN];//增广矩阵

int x[MAXN];//解集

bool free\_x[MAXN];//标记是否是不确定的变元

int Gauss(int equ,int var)

{

int i,j,k;

int max\_r;// 当前这列绝对值最大的行.

int col;//当前处理的列

int ta,tb;

int LCM;

int temp;

int free\_x\_num;

int free\_index;

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

{

x[i]=0;

free\_x[i]=true;

}

//转换为阶梯阵.

col=0; // 当前处理的列

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

{// 枚举当前处理的行.

// 找到该col列元素绝对值最大的那行与第k行交换.(为了在除法时减小误差)

max\_r=k;

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

{

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

}

if(max\_r!=k)

{// 与第k行交换.

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

}

if(a[k][col]==0)

{// 说明该col列第k行以下全是0了，则处理当前行的下一列.

k--;

continue;

}

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

{// 枚举要删去的行.

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

{

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

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

tb = LCM/abs(a[k][col]);

if(a[i][col]\*a[k][col]<0)tb=-tb;//异号的情况是相加

for(j=col;j<var+1;j++)

{

a[i][j] = a[i][j]\*ta-a[k][j]\*tb;

}

}

}

}

**// 1. 无解的情况:** 化简的增广阵中存在(0, 0, ..., a)这样的行(a != 0).

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

{

if (a[i][col] != 0) return -1;

}

**// 2. 无穷解的情况:** 在var \* (var + 1)的增广阵中出现(0, 0, ..., 0)这样的行，即说明没有形成严格的上三角阵, 且出现的行数即为自由变元的个数.

if (k < var)

{

// 首先，自由变元有var - k个，即不确定的变元至少有var - k个.

for (i = k - 1; i >= 0; i--)

{

free\_x\_num = 0;

// 用于判断该行中的不确定的变元的个数，如果超过1个，则无法求解，它们仍然为不确定的变元.

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

{

if (a[i][j] != 0 && free\_x[j]) free\_x\_num++, free\_index = j;

}

if (free\_x\_num > 1) continue; // 无法求解出确定的变元.

temp = a[i][var];

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

{

if (a[i][j] != 0 && j != free\_index) temp -= a[i][j] \* x[j];

}

x[free\_index] = temp / a[i][free\_index]; // 求出该变元.

free\_x[free\_index] = 0; // 该变元是确定的.

}

return var - k; // 自由变元有var - k个.

}

// **3. 唯一解的情况:** 在var \* (var + 1)的增广阵中形成严格的上三角阵.

// 计算出Xn-1, Xn-2 ... X0.

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

{

temp = a[i][var];

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

{

if (a[i][j] != 0) temp -= a[i][j] \* x[j];

}

if (temp % a[i][i] != 0) return -2; // 说明有浮点数解，但无整数解.

x[i] = temp / a[i][i];

}

return 0;

}

# 开关类模2同余方程求解

49 if(max\_r != k)

50 {

51 for(int j = col; j < var+1; j++)

52 swap(a[k][j],a[max\_r][j]);

53 }

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

55 {

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

57 {

58 for(int j = col;j < var+1;j++)

59 a[i][j] ^= a[k][j];

60 }

61 }

62 }

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

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

65 return -1;//无解

66 if(k < var) return var-k;//自由变元个数

67 //唯一解，回代

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

69 {

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

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

72 x[i] ^= (a[i][j] && x[j]);

73 }

74 return 0;

75 }

# FFT求大整数乘法

使用FFT一定要注意控制好长度，长度要为2^k.而且大于等于len1+len2-1

#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 r,i;

complex(double \_r = 0.0,double \_i = 0.0)

{

r = \_r; i = \_i;

}

complex operator +(const complex &b)

{

return complex(r+b.r,i+b.i);

}

complex operator -(const complex &b)

{

return complex(r-b.r,i-b.i);

}

complex operator \*(const complex &b)

{

return complex(r\*b.r-i\*b.i,r\*b.i+i\*b.r);

}

};

/\*

\* 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].r /= 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].r+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;

}

计算几何

误差修正

**点类：**

<complex>，叉积，点积，单位向量，旋转，判等，abs计算距离

**线段类：**

求点到线段的距离

求点到线段的垂足

求两直线交点

求点(x,y)关于直线AX+BY+C=0的对称点(X,Y)

判断两点是否在直线的同侧

判断点是否在线段上

判断线段是否相交

判断直线平行

将直线沿法向量平移距离len

**多边形类：**

面积

判断点是否在多边形内

判断是否为凸多边形

多边形重心

多边形格点数

**圆类：**

N个圆与一条直线的最多交点

求N个圆的最近距离O( n (logn)^2 )

#include <cmath>

using namespace std;

const double eps=1e-8;

const double pi=acos(-1.0);

const int maxn=1000;

int cmp(double x) { return x<-eps?-1:x>eps;

# 点类

#include <complex>

typedef complex<double> point;

//conj(b)返回b的复共轭

inline double dot(const point&a,const point&b)

{return (a\*conj(b)).real();}

inline double det(const point&a,const point&b)

{return (conj(a)\*b).imag();}

//单位向量

point unit(const point&v) {return v/abs(v);}

//逆时针旋转90度后的向量

point ortho(const point&v) {return v\*point(0,1);}

//逆时针旋转angle后的向量

point rotate\_point(const point&v,const double&angle)

{return v\*point(cos(angle),sin(angle));}

bool equal(const double a,const double b) {return abs(a-b)<eps;}

bool equal(const point&a,const point&b) {return abs(a-b)<eps;}

# 直线类

struct line

{

point a,b;

line(){}

line(point x,point y):a(x),b(y){}

};

///判断两点是否在线段的同侧（异侧<eps）

int same\_side(point p1,point p2,line l)

{

return det(l.a,p1,l.b)\*det(l.a,p2,l.b)>eps

}

## 点p到线段st的距离

double dis\_point\_segment(point p,point s,point t)

{

if(cmp(dot(p-s,t-s)<0))

return abs(p-s);

if(cmp(dot(p-t,s-t)<0))

return abs(p-t);

//点p到直线st的距离

return fabs(det(s-p,t-p)/abs(s-t));

}

## 点p到线段st的垂足，保存在cp里

void point\_proj\_line(point p,point s,point t,point &cp)

{

double r=dot(t-s,p-s)/dot(t-s,t-s);

cp=s+r\*(t-s);

}

## 判断p点是否在线段st上（<=包括端点，<不包括端点）

bool point\_on\_segment(point p,point s,point t)

{

return cmp(det(p-s,t-s))==0&&cmp(dot(p-s,p-t))<=0;

}

## 求点(x,y)关于直线AX+BY+C=0的对称点(X,Y)

void symmetric\_point()

{

X=x-2\*A\*(A\*x+B\*y+C)/(A\*A+B\*B);

Y=y-2\*B\*(A\*x+B\*y+C)/(A\*A+B\*B);

}

## 线段相交判定（规范相交）

bool segment\_proper\_intersection1(point a1,point a2,point b1,point b2)

{

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

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

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

}

## 不规范相交

bool segment\_proper\_intersection2(point a1,point a2,point b1,point b2)

{

if(point\_on\_segment(a1,b1,b2)||point\_on\_segment(a2,b1,b2))

return true;

if(point\_on\_segment(b1,a1,a2)||point\_on\_segment(b2,a1,a2))

return true;

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

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

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

}

## 判平行

bool parallel(line a,line b)

{

return !cmp(det(a.a-a.b,b.a-b.b));

}

## 求两直线交点，存在res里

bool line\_make\_point(line a,line b,point &res)

{

if(parallel(a,b)) return false;

double s1=det(a.a-b.a,b.b-b.a);

double s2=det(a.b-b.a,b.b-b.a);

res=(s1\*a.b-s2\*a.a)/(s1-s2);

return true;

}

## 将直线a沿法向量方向平移距离len得到的直线

line move\_d(line a,double len)

{

point d=a.b-a.a;

d=d/abs(d);

d=rotate\_point(d,pi/2);

return line(a.a+d\*len,a.b+d\*len);

}

# 多边形类

struct polygon

{

int n;

point a[maxn];

polygon(){}

double area()

{

double sum=0;

a[n]=a[0];

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

sum+=det(a[i+1],a[i]);

return fabs(sum/2);

}

## 判断点是否在多边形内

///返回0——外，1——内，2——上

int point\_in(point t)

{

int num=0,d1,d2,k;

a[n]=a[0];

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

{

if(point\_on\_segment(t,a[i],a[i+1]))

return 2;

k=cmp(det(a[i+1]-a[i],t-a[i]));

d1=cmp(a[i].imag()-t.imag());

d2=cmp(a[i+1].imag()-t.imag());

if(k>0&&d1<=0&&d2>0) num++;

if(k<0&&d2<=0&&d1>0) num--;

}

return num!=0;

}

int segment\_in(point l1,point l2)

{

}

## 多边形的重心

point mass\_center()

{

point ans=point(0,0);

if(cmp(area())==0) return ans;

a[n]=a[0];

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

ans=ans+(a[i]+a[i+1])\*det(a[i+1],a[i]);

return ans/area()/6.0;

}

## 多边形内格点数

int boder\_int\_point\_num()

{

int num=0;

a[n]=a[0];

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

num+=gcd(abs(int(a[i+1].real()-a[i].real())),

abs(int(a[i+1].imag()-a[i].imag())));

return num;

}

int inside\_int\_point\_num()

{

return int(area())+1-boder\_int\_point\_num()/2;

}

## 判断是否为凸多边形

///允许邻边共线

int is\_convex()

{

int s[3]={1,1,1};

for(int i=0;i<n&&s[0]|s[2];i++)

s[ 1 + cmp( det( a[(i+1)%n]-a[i],a[(i+2)%n]-a[i] ) )]=0;

return s[0]|s[2];

}

///不允许邻边共线

int is\_convex()

{

int s[3]={1,1,1};

for(int i=0;i<n&&s[1]&&s[0]|s[2];i++)

s[ 1 + cmp( det( a[(i+1)%n]-a[i],a[(i+2)%n]-a[i] ) )]=0;

return s[1]&&s[0]|s[2];

}

}

# 给定N个圆，求一条直线最多能交几个圆O（n^2logn）

const int maxn=2000;

const double pi=acos(-1.0);

const double eps=1e-10;

int cmp(double x)

{

return x<-eps?-1:x>eps;

}

inline double fix(double arg) {

while(arg > pi) arg -= 2\*pi;

while(arg <= -pi) arg += 2\*pi;

return arg;

}

typedef complex<double > point;

typedef point vec;

inline double dot(const point &a,const point &b)

{

return (a\*conj(b)).real();

}

inline double det(const point &a,const point &b)

{

return (conj(a)\*b).imag();

}

point rotate\_point(const point &v,const double &angle)

{

return v\*point(cos(angle),sin(angle));

}

struct circle

{

point o;

double r;

}c[maxn];

struct line

{

double angle;

int id,flag;

line(double aa,int c1,int c2):angle(aa),id(c1),flag(c2){}

bool operator <(const line&A) const

{

if(cmp(angle-A.angle)!=0)

return angle<A.angle;

return flag>flag;

}

};

vector <line> s;

void getTangents(const circle &A,const circle &B,int &id,int &sum)

{

double d=abs(A.o-B.o);

//A内含B

if(cmp(A.r-B.r-d)>0) return;

//B内含或内切A

if(cmp(B.r-A.r-d)>=0)

{

sum++;

return;

}

vec tmp=B.o-A.o;

double base=atan2(tmp.imag(),tmp.real());

//A内切B

if(cmp(A.r-B.r-d)==0)

{

s.push\_back(line(fix(base),id,1));

s.push\_back(line(fix(base),id,-1));

return;

}

double ang1=acos((A.r-B.r)/d);

double ang2=acos((A.r+B.r)/d);

s.push\_back(line(fix(base-ang1),id,1));

s.push\_back(line(fix(base+ang1),id,-1));

if(cmp(d-A.r-B.r)>0)

{

s.push\_back(line(fix(base+ang2),id,1));

s.push\_back(line(fix(base-ang2),id,-1));

}

}

bool vis[maxn];

int solve()

{

int ans=0,sum=0;

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

int si=s.size();

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

{

if(s[i%si].flag==1)

{

vis[s[i%si].id]=1;

sum++;

}

else

{

if(vis[s[i%si].id]==1)

{

vis[s[i%si].id]=0;

sum--;

}

}

ans=max(ans,sum);

}

return ans;

}

int main()

{

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

int T,n;

double x,y;

scanf("%d",&T);

for(int t=1;t<=T;t++)

{

scanf("%d",&n);

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

{

scanf("%lf%lf%lf",&x,&y,&c[i].r);

c[i].o=point(x,y);

}

int ans=0;

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

{

s.clear();

int sum=1;

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

{

if(i==j) continue;

getTangents(c[i],c[j],j,sum);

}

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

sum+=solve();

ans=max(ans,sum);

}

printf("Case #%d: %d\n",t,ans);

}

return 0;

}

# 求N个圆的最近距离O( n (logn)^2 )

题目描述：给你n（50000）个不相交的圆，告诉你各个圆的xy坐标和半径，问你最近的两个圆之间的距离（两圆心之间的距离减去各自的半径）

#include<stdio.h>  
#include<iostream>  
#include<set>  
#include<algorithm>  
#include<cmath>  
using namespace std;  
#define N 50010  
#define eps 1e-8  
double x[N],y[N],r[N];  
int Left[N], Right[N], up[N], up\_rank[N];  
int n;  
double mid;  
set<int> se;  
typedef set<int>::iterator it;  
double Sqr(double a)  
{  
    return a\*a;  
}  
bool cmp\_left(int a,int b)  
{  
    return x[a]-r[a]<x[b]-r[b];  
}  
bool cmp\_right(int a,int b)  
{  
    return x[a]+r[a]<x[b]+r[b];  
}  
bool cmp\_up(int a,int b)  
{  
    if(y[a]==y[b])  
        return x[a]<x[b];  
    return y[a]<y[b];  
}  
bool Count(int a,int b)  
{  
    a=up[a];  
    b=up[b];  
    double s1=Sqr(x[a]-x[b])+Sqr(y[a]-y[b]);  
    double s2=Sqr(r[a]+r[b]+mid+mid);  
    if(s1<=s2)  
        return true;  
    return false;  
}  
bool Insert(int a)  
{  
    it i=se.insert(a).first;  
    if(i!=se.begin())  
    {  
        if(Count(a,\*--i))  
            return true;  
        i++;  
    }  
    if(++i!=se.end())  
    {  
        if(Count(a,\*i))  
            return true;  
    }  
    return false;  
}  
bool Judge()  
{  
    se.clear();  
    int i=0,j=0;  
    while(i<n || j<n)  
    {  
        if(j==n ||(i!=n && x[Left[i]]-r[Left[i]]-mid<x[Right[j]]+r[Right[j]]+mid))  
        {  
            if(Insert(up\_rank[Left[i++]]))  
                return true;  
        }  
        else  
            se.erase(up\_rank[Right[j++]]);  
    }  
    return false;  
}  
double Solve()  
{  
    double ll,rr;  
    ll=0;  
    rr=sqrt(Sqr(x[0]-x[1])+Sqr(y[0]-y[1]))-r[0]-r[1];  
    while(ll+eps<rr)  
    {  
        mid=(ll+rr)\*0.5;  
        if(Judge())  
            rr=mid;  
        else  
            ll=mid+eps;  
    }  
    return ll+rr;  
}  
int main()  
{  
    int t,i;  
    scanf("%d",&t);  
    while(t--)  
    {  
        scanf("%d",&n);  
        for(i=0;i<n;i++)  
            scanf("%lf%lf%lf",&x[i],&y[i],&r[i]);  
        for(i=0;i<n;i++)  
            Left[i]=Right[i]=up[i]=i;  
        sort(Left,Left+n,cmp\_left);  
  
        sort(Right,Right+n,cmp\_right);  
  
        sort(up,up+n,cmp\_up);  
        for(i=0;i<n;i++)  
            up\_rank[up[i]]=i;  
        printf("%.6f\n",Solve());  
    }  
    return 0;  
}