By

io07

r\_clover 2015.9.10

Acm Code Library

目录

[头文件 3](#_Toc432627263)

[输入输出挂 5](#_Toc432627264)

[bitset 6](#_Toc432627265)

[离散化 8](#_Toc432627266)

[多重背包 9](#_Toc432627267)

[数位DP 11](#_Toc432627268)

[快速傅里叶变换 13](#_Toc432627269)

[数论变化 14](#_Toc432627270)

[素数测试 16](#_Toc432627271)

[莫比乌斯反演 16](#_Toc432627272)

[快速幂运算 18](#_Toc432627273)

[KMP 20](#_Toc432627274)

[后缀数组 21](#_Toc432627275)

[字符串Hash 27](#_Toc432627276)

[后缀数组 34](#_Toc432627277)

[后缀自动机 35](#_Toc432627278)

[AC自动机 和 Trie树 37](#_Toc432627279)

[回文自动机 42](#_Toc432627280)

[ST表 43](#_Toc432627281)

[静态主席树 45](#_Toc432627282)

[动态主席树 49](#_Toc432627283)

[CDQ分治 59](#_Toc432627284)

[莫队算法 84](#_Toc432627285)

[整体二分 86](#_Toc432627286)

[2-SAT字典序最小的解 93](#_Toc432627287)

[2-SAT 99](#_Toc432627288)

[Dinic网络流 101](#_Toc432627289)

[ISAP 网络流 104](#_Toc432627290)

[SW全局最小割 107](#_Toc432627291)

[最小费用最大流 108](#_Toc432627292)

[最大权闭合子图 113](#_Toc432627293)

[最大密度子图 114](#_Toc432627294)

[二分图最小点权覆盖 115](#_Toc432627295)

[二分图最大点权独立集 115](#_Toc432627296)

[尺取法 115](#_Toc432627297)

[树链剖分 119](#_Toc432627298)

[倍增法求LCA 121](#_Toc432627299)

[边双连通分量 124](#_Toc432627300)

[点双连通分量 127](#_Toc432627301)

[LCT动态树 130](#_Toc432627302)

[强连通分量缩环 138](#_Toc432627303)

[二分图匹配 140](#_Toc432627304)

# 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

int sub[maxn], len, n, A[maxn];

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;

}

# 快速傅里叶变换

P operator\*(P a,P b)

{

return P(a.first\*b.first-a.second\*b.second,a.first\*b.second+a.second\*b.first);

}

P operator+(P a,P b)

{

return P(a.first+b.first,a.second+b.second);

}

P operator-(P a,P b)

{

return P(a.first-b.first,a.second-b.second);

}

void FFT ( P y[] , int n , int rev ) {

for ( int i = 1 , j , k , t ; i < n ; ++ i ) {

for ( j = 0 , k = n >> 1 , t = i ; k ; k >>= 1 , t >>= 1 ) j = j << 1 | t & 1 ;

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

}

for ( int s = 2 , ds = 1 ; s <= n ; ds = s , s <<= 1 ) {

P wn = P ( cos ( rev \* 2 \* pi / s ) , sin ( rev \* 2 \* pi / s ) ) , w = P ( 1 , 0 ) , t ;

for ( int k = 0 ; k < ds ; ++ k , w = w \* wn ) {

for ( int i = k ; i < n ; i += s ) {

y[i + ds] = y[i] - ( t = w \* y[i + ds] ) ;

y[i] = y[i] + t ;

}

}

}

if ( rev == -1 ) for ( int i = 0 ; i < n ; ++ i ) y[i].first /= n ;

}

# 数论变化

const LL mod = 786433;

LL wn[40];

int g,prime[1400],L=0;

LL pow(LL x,LL n,LL mod)

{

LL res=1;

while(n>0){

if(n&1)res=res\*x%mod;

x=x\*x%mod;

n>>=1;

}

return res;

}

bool check(int x)

{

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

if(pow(x,(mod-1)/prime[i],mod)==1)return false;

}

return true;

}

void init()

{

LL x=mod-1;

L=0;

for(int i=2;i<=sqrt(x)+1;i++){

if(x%i!=0)continue;

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

prime[L++]=i;

}

if(x!=1)prime[L++]=x;

for(g=2;;g++){

if(check(g))break;

}

for(int i=0;i<20;i++){

LL x=1<<i;

wn[i]=pow(g,(mod-1)/x,mod);

}

}

void NTT(LL a[], int len, int on)

{

int j = len >> 1;

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

{

if(i < j) swap(a[i], a[j]);

int k = len >> 1;

while(j >= k)

{

j -= k;

k >>= 1;

}

if(j < k) j += k;

}

int id = 0;

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

{

id++;

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

{

LL w = 1;

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

{

LL u = a[k] % mod;

LL t = w \* (a[k + h / 2] % mod) % mod;

a[k] = (u + t) % mod;

a[k + h / 2] = ((u - t) % mod + mod) % mod;

w = w \* wn[id] % mod;

}

}

}

if(on == -1)

{

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

swap(a[i], a[len - i]);

LL Inv = pow(len, mod - 2, mod);

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

a[i] = a[i] % mod \* Inv % mod;

}

}

# 素数测试

/\*Miller-Rabin算法：

根据费马小定理： 对于任何与素数p互质的a，有a^(p-1)%p=1;

即，如果我们能找到一个a使得 a^(p-1)%p=1不成立，则p不为素数

令p-1=(2^t)\*x;

则有 a^(p-1)%p= (a^x%p)^(2^t)%p，即：可先计算出2^x，然后通过不断地平方来计算a^(p-1)，只要过程中某一步出现了结果为1，则根据模运算的规则

最后的结果也必然为1，则可直接返回“合数”。

\*/

# 莫比乌斯反演

void init()

{

CLR(vis,0);

tot=0;

mu[1]=1;

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

{

if(!vis[i])

{

prime[tot++]=i;

mu[i]=-1;

}

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

{

if(i\*prime[j]>=MAXN)

break;

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

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

{

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

break;

}

else

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

}

}

inv[0]=inv[1]=1;

fac[0]=fac[1]=1;

fac\_inv[0]=fac\_inv[1]=1;

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

{

inv[i]=(MOD-MOD/i)\*inv[MOD%i]%MOD;

fac[i]=fac[i-1]\*i%MOD;

fac\_inv[i]=fac\_inv[i-1]\*inv[i]%MOD;

}

CLR(sum,0);

sum[1]=1;

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

{

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

if(i%j==0)

{

sum[i]=(sum[i]+j\*mu[i/j])%MOD;

if(j\*j!=i)

sum[i]=(sum[i]+i/j\*mu[j])%MOD;

}

}

pow2[0]=1;

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

pow2[i]=pow2[i-1]\*2%MOD;

}

# 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);

m = mm;

}

int all()

{

build\_sa();

getHeight();

RMQ\_init();

}

}sp;

# 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];

}

# 后缀数组

int rk[MAX\_N],sa[MAX\_N],height[MAX\_N],w[MAX\_N],wa[MAX\_N],ret[MAX\_N];

void getSa (int len,int up)

{

int \*k = rk;

int \*id = height;

int \*r = ret;

int \*cnt = wa;

for(int i=0;i<up;i++) cnt[i] = 0;

for(int i=0;i<len;i++) cnt[k[i] = w[i]]++;

for(int i=0;i<up;i++) cnt[i+1] += cnt[i];

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

sa[--cnt[k[i]]] = i;

}

for(int d = 1,p = 0; p < len;){

for(int i = len - d; i < len; i++) id[p++] = i;

for(int i=0;i<len;i++)if(sa[i] >= d) id[p++] = sa[i] - d;

for(int i=0;i<len;i++) r[i] = k[id[i]];

for(int i=0;i<up;i++) cnt[i] = 0;

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

for(int i=0;i<up;i++) cnt[i+1] += cnt[i];

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

sa[--cnt[r[i]]] = id[i];

}

swap(k,r);

p = 0;

k[sa[0]] = p++;

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

if(sa[i]+d < len && sa[i+1]+d <len &&r[sa[i]] == r[sa[i+1]]&& r[sa[i]+d] == r[sa[i+1]+d])

k[sa[i+1]] = p - 1;

else k[sa[i+1]] = p++;

}

if(p >= len) return ;

d \*= 2,up = p, p = 0;

}

}

void getHeight(int len) {

for(int i=0;i<len;i++) rk[sa[i]] = i;

height[0] = 0;

for(int i = 0,p = 0; i < len - 1; i++) {

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

while(i+p < len&& j+p < len&& w[i+p] == w[j+p]) {

p++;

}

height[rk[i]] = p;

p = max(0,p - 1);

}

}

int getSuffix(char s[]) {

int len = strlen(s),up = 0;

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

w[i] = s[i];

up = max(up,w[i]);

}

w[len++] = 0;

getSa(len,up+1);

getHeight(len);

return len;

}

# 后缀自动机

max：即代码中 len 数组，它表示该状态能够接受的最长的字符串长度。

min：表示该状态能够接受的最短的字符串长度。实际上等于该状态的 fail 指针指向的结点的 len + 1。

max-min+1：表示该状态能够接受的不同的字符串数。

right：即 end-set 的个数，表示这个状态在字符串中出现了多少次，该状态能够表示的所有字符串均出现过 right 次。

fail：fail 指向了一个能够表示当前状态表示的所有字符串的最长公共后缀的结点。所有的状态的 fail 指针构成了一个 fail 树，恰好是字符串的逆序的后缀树。

fail 树的拓扑序：序列中第i个状态的子结点必定在它之后，父结点必定在它之前。

\*

struct sam{

int len[MAXN],next[MAXN][26],fail[MAXN],L,last,dp[MAXN],right[MAXN],deg[MAXN],vs[MAXN];

sam(){

init();

}

void init(){

L=0;

last=newnode(0,-1);

}

int newnode(int l,int pre)

{

fail[L]=pre;

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

len[L]=l;

right[L]=0;

L++;

return L-1;

}

void add(int x,int l)

{

int pre=last,now=newnode(l+1,-1);

last=now;

while(~pre&&next[pre][x]==-1){

next[pre][x]=now;

pre=fail[pre];

}

if(pre==-1)fail[now]=0;

else{

int bro=next[pre][x];

if(len[bro]==len[pre]+1){

fail[now]=bro;

}

else{

int fa=newnode(len[pre]+1,fail[bro]);

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

next[fa][i]=next[bro][i];

}

fail[bro]=fa,fail[now]=fa;

while(~pre&&next[pre][x]==bro){

next[pre][x]=fa;

pre=fail[pre];

}

}

}

}

void build(std::string s)

{

int n=s.length();

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

add(s[i]-'a',i);

}

memset(deg,0,sizeof deg);

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

dp[i]=len[i];

if(~fail[i])deg[fail[i]]++;

}

int h=0,t=0;

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

if(!deg[i])vs[t++]=i;

}

while(h<t){

int v=vs[h++];

if(~fail[v]){

deg[fail[v]]--;

if(!deg[fail[v]]){

vs[t++]=fail[v];

}

}

}

}

}SAM;

# 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;

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];

void print(int i, int j)

{

if (j)

{

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

printf(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 = u = 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;

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]];

}

}

}

}

# 回文自动机

const int MAXN = 200105 ;

const int L = 27;

const unsigned long long mod = 1125899839733759LL;

const unsigned long long base = 9987;

struct PalindLomic\_Tree {

int next[MAXN][L] ;//next指针，next指针和字典树类似，指向的串为当前串两端加上同一个字符构成

int fail[MAXN] ;//fail指针，失配后跳转到fail指针指向的节点

int cnt[MAXN] ;

int num[MAXN] ;

int len[MAXN] ;//len[i]表示节点i表示的回文串的长度

int S[MAXN] ;//存放添加的字符

int last ;//指向上一个字符所在的节点，方便下一次add

int n ;//字符数组指针

int p ;

unsigned long long HASH[MAXN];

int newnode ( int l ) {

for ( int i = 0 ; i < L ; ++ i ) next[p][i] = 0 ;

cnt[p] = 0 ;

num[p] = 0 ;

len[p] = l ;

p++;

return p-1;

}

void init () {//初始化

p = 0 ;

newnode ( 0 ) ;

newnode ( -1 ) ;

last = 0 ;

n = 0 ;

S[n] = -1 ;

fail[0] = 1 ;

HASH[0]=L+1;

HASH[1]=L+2;

}

int get\_fail ( int x ) {

while ( S[n - len[x] - 1] != S[n] ) x = fail[x] ;

return x ;

}

void add ( int c ) {

c -= 'a';

S[++ n] = c ;

int cur = get\_fail ( last ) ;

if ( !next[cur][c] ) {

int now = newnode ( len[cur] + 2 ) ;

fail[now] = next[get\_fail ( fail[cur] )][c] ;

next[cur][c] = now ;

num[now] = num[fail[now]] + 1 ;

HASH[now]=(HASH[cur]\*base%mod+c)%mod;

}

last = next[cur][c] ;

cnt[last] ++ ;

}

void count () {

for ( int i = p - 1 ; i >= 0 ; -- i ){

cnt[fail[i]] += cnt[i] ;

}

}

};

# 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;

}

# 莫队算法

对于所有无修改的区间(二维)查询[l,r],如果答案ans[l,r]与ans[l,r+1]、ans[l-1,r]或者ans[l,r-1]、ans[l+1,r]存在直接的递推关系，

那么则可以使用莫队算法。

首先，离线读入所有的查询并且将查询分成sqrt(n)块，以左端点所在的块为第一关键字，右端点为第二关键字排序，然后按顺序依次计算所有的

查询。假设当前已经计算出了[l[i],r[i]]的答案，对于[ l[i+1],r[i+1] ]的答案，只需由前面查询i的答案暴力地加上或者减去区间l[i]+1到l[i+1]

以及r[i]+1到r[i+1]的的影响即可。总复杂度不超过O(n\*sqrt(n))；

bool cmp\_l(node a,node b)

{

if(a.l/blocks==b.l/blocks)return a.r<b.r;

return a.l/blocks<b.l/blocks;

}

void solve()

{

memset(numa,0,sizeof numa);

memset(numb,0,sizeof numb);

memset(ans,0,sizeof ans);

int L=0,R=0;

LL temp=0;

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

// printf("[%d,%d]->[%d,%d]\n",L,R,E[i].l,E[i].r);

while(R<E[i].r){

++R;

if(A[R]<=K&&K-A[R]<=N)temp-=numa[K-A[R]]\*numb[A[R]];

numb[A[R]]++;

if(A[R]<=K&&K-A[R]<=N)temp+=numa[K-A[R]]\*numb[A[R]];

}

while(R>E[i].r){

if(A[R]<=K&&K-A[R]<=N)temp-=numa[K-A[R]]\*numb[A[R]];

numb[A[R]]--;

if(A[R]<=K&&K-A[R]<=N)temp+=numa[K-A[R]]\*numb[A[R]];

R--;

}

while(L<E[i].l){

L++;

if(K>=A[L]&&K-A[L]<=N)temp-=numa[A[L]]\*numb[K-A[L]];

numa[A[L]]++;

if(K>=A[L]&&K-A[L]<=N)temp+=numa[A[L]]\*numb[K-A[L]];

}

while(L>E[i].l){

if(K>=A[L]&&K-A[L]<=N)temp-=numa[A[L]]\*numb[K-A[L]];

numa[A[L]]--;

if(K>=A[L]&&K-A[L]<=N)temp+=numa[A[L]]\*numb[K-A[L]];

L--;

}

ans[E[i].id]=temp;

}

}

# 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);

}

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;

}

# 2-SAT字典序最小的解

HDU 1814

/\*

HDU 1814

求出字典序最小的解

C++ 2652ms 2316K

\*/

#include<stdio.h>

#include<iostream>

#include<algorithm>

#include<iostream>

using namespace std;

const int MAXN=16010;

const int MAXM=100000;

struct Node

{

int a,b,pre,next;

}E[MAXM],E2[MAXM];

int \_n,n,m;

int V[MAXN],ST[MAXN][2],Q[MAXN],Q2[MAXN],vst[MAXN];

bool res\_ex;

void init\_d()

{

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

E[i].a=E[i].pre=E[i].next=E2[i].a=E2[i].pre=E2[i].next=i;

m=n;

}

void add\_edge(int a,int b)

{

E[m].a=a;E[m].b=b;E[m].pre=E[a].pre;E[m].next=a;E[a].pre=m;E[E[m].pre].next=m;

E2[m].a=b;E2[m].b=a;E2[m].pre=E2[b].pre;E2[m].next=b;E2[b].pre=m;E2[E2[m].pre].next=m;

m++;

}

void solve()

{//1

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

{

V[i]=0;

vst[i]=0;

}

res\_ex=1;

int i,i1,i2,j,k,front,rear,front2,rear2;

int len;

bool ff;

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

{//2

if(V[\_i<<1]==1||V[(\_i<<1)+1]==1)continue;//找一对未确定的点

i=\_i<<1;len=0;

if(!V[i])

{//3

ST[len][0]=i;

ST[len++][1]=1;

if(!V[i^1])

{

ST[len][0]=i^1;

ST[len++][1]=2;

}

Q[front=rear=0]=i;

vst[i]=i1=n+i;

Q2[front2=rear2=0]=i^1;

vst[i^1]=i2=(n<<1)+i;

ff=1;

for(;front<=rear;front++)

{//4

j=Q[front];

for(int p=E[j].next;p!=j;p=E[p].next)

{//5

k=E[p].b;

if(V[k]==2||vst[k]==i2||V[k^1]==1||vst[k^1]==i1)

{ff=0;break;}

if(vst[k]!=i1)

{//6

Q[++rear]=k;vst[k]=i1;

if(!V[k])

{

ST[len][0]=k;

ST[len++][1]=1;

}

}//6

if(vst[k^1]!=i2)

{//6

Q2[++rear2]=k^1;vst[k^1]=i2;

if(!V[k])

{

ST[len][0]=k^1;

ST[len++][1]=2;

}

}//6

}//5

if(!ff)break;

}//4

if(ff)

{//4

for(;front2<=rear2;front2++)

{//5

j=Q2[front2];

for(int p=E2[j].next;p!=j;p=E2[p].next)

{//6

k=E2[p].b;

if(V[k]==1||vst[k]==i1)

{ff=0;break;}

if(vst[k]!=i2)

{

vst[k]=i2;Q2[++rear]=k;

if(!V[k])

{

ST[len][0]=k;

ST[len++][1]=2;

}

}

}//6

if(!ff)break;

}//5

if(ff)

{

for(int j=0;j<len;j++)V[ST[j][0]]=ST[j][1];

continue;

}

}//4

}//3

i=(\_i<<1)+1;len=0;

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

//下面这段和上面完全一样的，可以复制。但是要保证上面写对

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

if(!V[i])

{//3

ST[len][0]=i;

ST[len++][1]=1;

if(!V[i^1])

{

ST[len][0]=i^1;

ST[len++][1]=2;

}

Q[front=rear=0]=i;

vst[i]=i1=n+i;

Q2[front2=rear2=0]=i^1;

vst[i^1]=i2=(n<<1)+i;

ff=1;

for(;front<=rear;front++)

{//4

j=Q[front];

for(int p=E[j].next;p!=j;p=E[p].next)

{//5

k=E[p].b;

if(V[k]==2||vst[k]==i2||V[k^1]==1||vst[k^1]==i1)

{ff=0;break;}

if(vst[k]!=i1)

{//6

Q[++rear]=k;vst[k]=i1;

if(!V[k])

{

ST[len][0]=k;

ST[len++][1]=1;

}

}//6

if(vst[k^1]!=i2)

{//6

Q2[++rear2]=k^1;vst[k^1]=i2;

if(!V[k])

{

ST[len][0]=k^1;

ST[len++][1]=2;

}

}//6

}//5

if(!ff)break;

}//4

if(ff)

{//4

for(;front2<=rear2;front2++)

{//5

j=Q2[front2];

for(int p=E2[j].next;p!=j;p=E2[p].next)

{//6

k=E2[p].b;

if(V[k]==1||vst[k]==i1)

{ff=0;break;}

if(vst[k]!=i2)

{

vst[k]=i2;Q2[++rear]=k;

if(!V[k])

{

ST[len][0]=k;

ST[len++][1]=2;

}

}

}//6

if(!ff)break;

}//5

if(ff)

{

for(int j=0;j<len;j++)V[ST[j][0]]=ST[j][1];

continue;

}

}//4

}//3

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

if(V[\_i<<1]+V[(\_i<<1)+1]!=3){res\_ex=0;break;}

}//2

}//1

//点的编号必须从0开始，2\*i和2\*i+1是一对sat

int main()

{

int M;

int x,y;

while(scanf("%d%d",&\_n,&M)!=EOF)

{

n=\_n<<1;

init\_d();

while(M--)

{

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

x--;

y--;

if(x!=(y^1))

{

add\_edge(x,y^1);

add\_edge(y,x^1);

}

}

solve();

if(res\_ex)

{

for(int i=0;i<n;i++)//V为0为不确定，1为确定选择，2为确定不选择

if(V[i]==1)printf("%d\n",i+1);

}

else printf("NIE\n");

}

return 0;

}

# 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) //两个中至少有一个为真

{

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;

# Dinic网络流

struct Dinic

{

struct Edge

{

int to,next;

ll f;

} e[MAXM];

int n,src,sink;

int head[MAXN];

int tot;

void addedge(int u,int v,ll c)

{

e[++tot].to=v;

e[tot].f=c;

e[tot].next=head[u];

head[u]=tot;

e[++tot].to=u;

e[tot].f=0;

e[tot].next=head[v];

head[v]=tot;

}

void Resize(int n)

{

this->n=n;

}

void Clear()

{

CLR(head,-1);

tot=1;

}

bool vis[MAXN];

int dist[MAXN];

void bfs()

{

CLR(dist,0);

queue<int> que;

vis[src]=true;

que.push(src);

while(!que.empty())

{

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

for(int i=head[u]; ~i; i=e[i].next)

{

if(e[i].f>0 && !vis[e[i].to])

{

que.push(e[i].to);

dist[e[i].to]=dist[u]+1;

vis[e[i].to]=true;

}

}

}

}

ll dfs(int u,ll delta)

{

if(u==sink || delta==0)

return delta;

else

{

ll ret=0;

for(int i=head[u]; ~i; i=e[i].next)

if(e[i].f && dist[e[i].to]==dist[u]+1)

{

ll dd=dfs(e[i].to,min(e[i].f,delta));

e[i].f-=dd;

e[i^1].f+=dd;

delta-=dd;

ret+=dd;

}

return ret;

}

}

ll MaxFlow(int s,int t)

{

ll ret=0;

this->src=s;this->sink=t;

while(1)

{

CLR(vis,0);

bfs();

if(!vis[sink])

return ret;

ret+=dfs(src,LLINF);

}

}

void print()

{

puts("GRAPH:");

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

{

printf("%d ->",i);

for(int j=0;~j;j=e[j].next)

printf("%d(%d) ",e[j].to,e[i].f);

putchar('\n');

}

}

};

Dinic g;

# ISAP 网络流

struct ISAP

{

struct nedge

{

int u,v,next;

ll c;

} e[M\*2];

int head[N],tot,n;

int ss,tt;

void Resize(int n)

{

this->n=n;

}

void Clearall()

{

tot=1;

CLR(head,0);

}

void Addedge(int u,int v,ll w)

{

e[++tot].v=v;

e[tot].u=u;

e[tot].c=w;

e[tot].next=head[u];

head[u]=tot;

e[++tot].v=u;

e[tot].u=v;

e[tot].c=0;

e[tot].next=head[v];

head[v]=tot;

}

int dep[N],gap[N];

void bfs()

{

CLR(dep,-1);

CLR(gap,0);

queue<int> que;

gap[0]=1;

dep[tt]=0;

que.push(tt);

while(!que.empty())

{

int u=que.front();

que.pop();

for(int i=head[u]; i; i=e[i].next)

if(dep[e[i].v]==-1)

{

que.push(e[i].v);

dep[e[i].v]=dep[u]+1;

++gap[dep[e[i].v]];

}

}

}

int res,cur[N],num[N];

int top;

int MaxFlow(int s,int t)

{

this->ss=s;

this->tt=t;

bfs();

top=res=0;

memcpy(cur,head,sizeof(head));

int u=ss,i;

while(dep[ss]<n)

{

if(u==tt)

{

int temp=INF;

int inser;

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

if(temp>e[num[i]].c)

{

temp=e[num[i]].c;

inser=i;

}

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

{

e[num[i]].c-=temp;

e[num[i]^1].c+=temp;

}

res+=temp;

top=inser;

u=e[num[top]].u;

}

if(u!=tt && gap[dep[u]-1]==0)

break;

for(i=cur[u]; i; i=e[i].next)

if(e[i].c != 0 && dep[u]==dep[e[i].v]+1)

break;

if(i)

{

cur[u]=i;

num[top++]=i;

u=e[i].v;

}

else

{

int mi=n;

for(i=head[u]; i; i=e[i].next)

if(e[i].c>0 && mi>dep[e[i].v])

{

mi=dep[e[i].v];

cur[u]=i;

}

--gap[dep[u]];

dep[u]=mi+1;

++gap[dep[u]];

if(u!=ss)

u=e[num[--top]].u;

}

}

return res;

}

void print()

{

puts("GRAPH:");

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

{

if(!head[i])

continue;

printf("%d ->",i);

for(int j=head[i]; j; j=e[j].next)

printf("%d(%I64d) ",e[j].v,e[j].c);

putchar('\n');

}

}

};

# SW全局最小割

const int MAXN = 305;

int n,m,v[MAXN],mat[MAXN][MAXN],dis[MAXN];

bool vis[MAXN];

int res,s;

int Stoer\_Wagner(int n)

{

int i, j;

int res = INF;

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

v[i] = i+1;//初始化第i个结点就是i !!!!! 注意下标

while (n > 1)

{

int maxp = 1,prev = 0;

for (i = 1; i < n; i++) //初始化到已圈集合的割大小,并找出最大距离的顶点

{

dis[v[i]] = mat[v[0]][v[i]];

vis[v[i]] = 0;

if (dis[v[i]] > dis[v[maxp]])

maxp = i;

}

vis[v[0]] = true;

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

{

vis[v[maxp]] = true;

if (i == n - 1) //只剩最后一个没加入集合的点，更新最小割

{

res = min(res,dis[v[maxp]]);

for (j = 0; j < n; j++) //合并最后一个点以及推出它的集合中的点

{

mat[v[prev]][v[j]] += mat[v[j]][v[maxp]];

mat[v[j]][v[prev]] = mat[v[prev]][v[j]];

}

v[maxp] = v[--n];//第maxp个节点去掉，第n个节点变成第maxp个

}

else

{

prev = maxp;

maxp = -1;

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

if (!vis[v[j]]) //将上次求的maxp加入集合，合并与它相邻的边到割集

{

dis[v[j]] += mat[v[prev]][v[j]];

if (maxp == -1 || dis[v[maxp]] < dis[v[j]])

maxp = j;

}

}

}

}

return res;

}

int main()

{

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

while (read(n)&&read(m),read(s)&&(n+m+s))

{

CLR(mat,0);

int x,y,z;

while (m--)

{

read(x),read(y),read(z);

mat[x][y] += z;

mat[y][x] += z;

}

printf("%d\n",Stoer\_Wagner(n));

}

}

# 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;

}

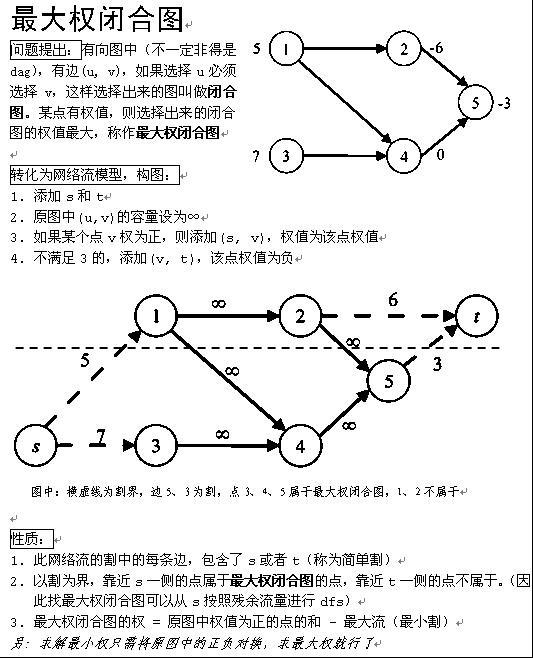
flow += rmin;

}

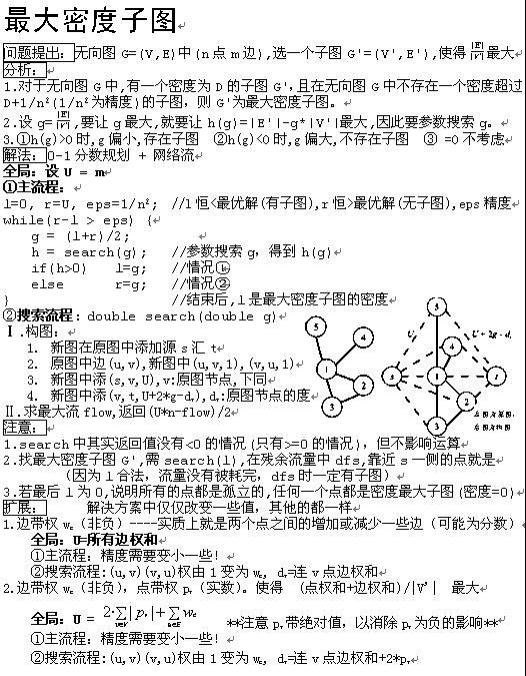
return flow;

}

# 最大权闭合子图



# 最大密度子图



**解法一：**

假设答案为k ，则要求解的问题是：选出一个合适的点集V和边集E，令(|E| - k \* |V|)  
取得最大值。所谓“合适”是指满足如下限制：若选择某条边，则必选择其两端点。

建图:以原图的边作为左侧顶点，权值为1；原图的点作为右侧顶点，权值为-k(相当于 支出k)。

若原图中存在边(u,v)，则新图中添加两条边([uv]-->u), ([uv]-->v)，转换为最大权闭合子图。

**解法二：**

把原图中的无向边转换成两条有向边，容量为1。

设一源点，连接所有点，容量为U（取m）。

设一汇点，所有点连接汇点，容量为 U+2g-dv 。

二分枚举最大密度g，其中dv为v的度。

判断(U\*n-MaxFlow)/2.0>=0。

最后跳出的L就是最大密度。

拿这个L再重新建图，求最大流。

然后从s出发bfs或者dfs，走残留容量大于0的边，所有能到达的点就是答案。

# 二分图最小点权覆盖

sum=val(点集X) + val(点集Y)

所有的v属于X连<s,v>点权为val(v)

所有的v属于Y连<v,t> 点权为val(v)

原图的<v,u>连<v,u>cap = INF

求解最小割 = 最小点权覆盖集

原理：网络流中的任意一个割都对应一个点权覆盖集，假设有没被覆盖的边e<u,v>，则点<s,u>与<v,t>没被割那么必有从s到t的流量不为0的增广路与割的定义矛盾

量化关系：最大流的最小割定理

# 二分图最大点权独立集

二分图点权，一边连向汇点，一边连向源点，将所有点权相加，然后连边的是在原图中不相连的点，然后用SUM-MAXFLOW就是答案。

# 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;

}

# 树链剖分

struct Edge

{

int u,v,w;

int next;

}e[MAXN<<1];

int head[MAXN],tot;

int siz[MAXN],p[MAXN],son[MAXN],dep[MAXN];

int dfn[MAXN],top[MAXN],tim;

int sum[MAXN<<1],la[MAXN<<1];

int bingchafa[MAXN];

int getfather(int x)

{

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

return bingchafa[x] = getfather(bingchafa[x]);

}

void init()

{

CLR(head,-1);CLR(sum,0);

tot=tim=0;

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

S.clear();

V.clear();

}

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

{

e[tot].u=u;e[tot].v=v;e[tot].w=w;

e[tot].next=head[u];head[u]=tot++;

}

void dfs(int u,int fa=0,int depth=0)

{

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

p[u]=fa;dep[u]=depth;

for(int i=head[u];~i;i=e[i].next)

{

int v=e[i].v;

if(v==p[u]) continue;

dfs(v,u,depth+1);

siz[u]+=siz[v];

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

son[u]=v;

}

}

void getid(int u,int fa)

{

dfn[u]=++tim;top[u]=fa;

if(son[u]!=-1)

getid(son[u],fa);

for(int i=head[u];~i;i=e[i].next)

{

int v=e[i].v;

if(v==p[u] || v==son[u]) continue;

getid(v,v);

}

}

void change(int u,int v)

{

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

{

if(dep[top[u]]>dep[top[v]])

swap(u,v);

update2(1,tim,dfn[top[v]],dfn[v]);

v=p[top[v]];

}

if(u==v) return;

if(dep[u]>dep[v])

swap(u,v);

update2(1,tim,dfn[son[u]],dfn[v]);

}

int query(int u,int v)

{

int ans=0;

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

{

if(dep[top[u]]>dep[top[v]])

swap(u,v);

ans+=query(1,tim,dfn[top[v]],dfn[v]);

v=p[top[v]];

}

if(u==v) return ans;

if(dep[u]>dep[v])

swap(u,v);

ans+=query(1,tim,dfn[u]+1,dfn[v]);

return ans;

}

# 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;

}