Olympiad in Informatics 模板复习手册

Kvar_ispw17

November 9, 2017

Contents

1 数学

1.1 $O(1)$ 最大公约数

```
#include <cstdio>
#include <algorithm>
#include <cstring>
#include <cmath>
using namespace std;

typedef long long ll;
const int N = 10000000 + 5;
const int sqrtN = sqrt(N);
int st[N], tot, d[N][5];
int _gcd[sqrtN + 5][sqrtN + 5];
bool notp[N];

void sieve() {
    notp[1] = d[1][0] = d[1][1] = d[1][2] = 1;
    for (ll i = 2; i <= N; i++) {
        if (!notp[i]) {
            d[i][0] = d[i][1] = 1;
            d[i][2] = i;
            st[++tot] = i;
        }
        for (ll j = 1; j <= tot && i * st[j] <= N;
           j++) {
            ll k = i * st[j];
            notp[k] = true;
            ll p = d[i][0] * st[j];
            if (p < d[i][1]) {
                d[k][0] = p;
                d[k][1] = d[i][1];
                d[k][2] = d[i][2];
            } else if (p < d[i][2]) {
                d[k][0] = d[i][1];
                d[k][1] = p;
                d[k][2] = d[i][2];
            } else {
                d[k][0] = d[i][1];
                d[k][1] = d[i][2];
```

```
                d[k][2] = p;
            }
            if (i % st[j]); else break;
        }
    }
}

void calc() {
    for (ll i = 0; i <= sqrtN; i++)
        _gcd[i][0] = _gcd[0][i] = i;
    for (ll i = 1; i <= sqrtN; i++) {
        for (ll j = 1; j <= i; j++)
            _gcd[i][j] = _gcd[j][i] = _gcd[i - j][j];
    }
}

ll gcd(ll a, ll b) {
    int *x = d[a], g = 1;
    for (ll i = 0; i < 3; i++) {
        ll d;
        if (x[i] <= sqrtN)
            d = _gcd[x[i]][b % x[i]];
        else if (b % x[i])
            d = 1;
        else d = x[i];
        g *= d; b /= d;
    }
    return g;
}

ll A, B;

int main() {
    sieve();
    calc();
    return 0;
}
```

1.2 拓展欧几里得算法

```
int exgcd(int a, int b, int &x, int &y) {
    if (b == 0) { x = 1; y = 0; return a; }
    int ret = exgcd(b, a % b, y, x);
    y -= a / b * x;
    return ret;
}
/**
*  other solutions:
*  {x} = x + b / gcd(x, y) * t
*  {y} = y - a / gcd(x, y) * t
*  where t in Z+
*/
```

1.3 长整型快速乘

```
#define EPS 1e-8
long long mul(long long a, long long b, long long p) {
    long long ret = a * b - (long long)((long
       double)a / p * b + EPS) * p;
    return ret < 0 ? ret + p : ret;
}

long long mul(long long a, long long b, long long p) {
    long long ret = 0;
    while (b) {
        if (b & 1) ret = (ret + a) % p;
        a = (a + a) % p;
        a >>= 1;
    }
    return ret;
}
```

1.4 欧拉筛

```
#pragma GCC optimize("O3")
#include <algorithm>
#include <cstdio>
#include <cstring>
#include <cmath>
#define N int(1e7 + 10)
using namespace std;

int num_mindiv[N];
int min_div[N];
int sum_div[N], t1[N], t2[N];
int num_div[N];
int phi[N];
int miu[N];
bool is_prime[N];
int prime[N], tot;

void sieve(int n) {
    miu[1] = 1;
    memset(is_prime, true, sizeof is_prime);
    for (int i = 2; i <= n; i++) {
        if (is_prime[i]) {
            min_div[i] = prime[++tot] = i;
            phi[i] = i - 1;
            num_mindiv[i] = 1;
            num_div[i] = 2;
            miu[i] = -1;
            sum_div[i] = 1 + i;
            t1[i] = 1 + i;
            t2[i] = i;
        }
        for (int j = 1; j <= tot && prime[j] * i <=
           n; j++) {
            int k = prime[j] * i;
            is_prime[k] = false;
            if (i % prime[j] == 0) {
            phi[i * prime[j]] = phi[i] * prime[j];
            num_mindiv[k] = num_mindiv[i] + 1;
```

```
            num_div[k] = num_div[i] / (num_mindiv[i]
               + 1) * (num_mindiv[k] + 1);
            miu[i] = 0;
            t2[k] = t2[i] * prime[j];
            t1[k] = t1[i] + t2[k];
            sum_div[k] = sum_div[i] / t1[i] * t1[k];
            break;
        }
        phi[k] = phi[i] * phi[prime[j]];
        num_mindiv[k] = 1;
        num_div[k] = num_div[i] * 2;
        miu[k] = -miu[i];
        sum_div[k] = sum_div[i] * sum_div[prime[j]];
        t1[k] = 1 + prime[j];
        t2[k] = prime[j];
    }
}
}

int main() {
    sieve(10000000);
    return 0;
}
```

1.5 BSGS 算法

```
# include <stdio.h>
# include <math.h>
# include <map>
using namespace std;
typedef long long LL;
LL bsgs(LL A, LL B, LL C) {
    LL m, v, e = 1, i;
    m = ceil(sqrt(C));
    v = pow(A, C - 1 - m, C);
    map<LL, LL> hash;
    hash[1] = m;
    for (i = 1; i < m; ++i) {
        e = (e * A) % C;
        if (!hash[e]) hash[e] = i;
    }
    for (i = 0; i < m; ++i) {
        if (hash[B]) {
            LL ret = hash[B];
            hash.clear();
            return i * m + (ret == m ? 0 : ret);
        }
        B = (B * v) % C;
    }
    return -1;
}
int main() {
    LL A, B, C;
    // A ^ x = B mod C
    while (scanf("%lld%lld%lld", &C, &A, &B) != EOF) {
        LL ans = bsgs(A, B, C);
        if (ans == -1) printf("no␣solution\n");
        else printf("%lld\n", ans);
    }
    return 0;
}
```

1.6 中国剩余定理

```
#define LL long long
LL china(int n, LL *m, LL *a) {
    LL M = 1, d, y, x = 0;
    for (int i = 1; i <= n; i++) M *= m[i];
    for (int i = 1; i <= n; i++) {
        LL w = M / m[i];
        exgcd(m[i], w, d, d, y);
        x = (x + y * w * a[i]) % M;
    }
    return (x + M) % M;
}
LL m[15], a[15];
int main() {
    int n;
    scanf("%d", &n);
    for (int i = 1; i <= n; i++)
    scanf("%lld%lld", &m[i], &a[i]);
    printf("%lld", china(n, m, a));
}
```

1.7 $O(n)$ 求逆元

```
void get_inv() {
    inv[1] = 1;
    for (int i = 2; i <= n; i++)
    inv[i] = (n − n / i) * inv[n % i] % n;
}
```

1.8 高精度整数

```
/**
*   This is a implement for Big Integer in C++.
*   @author Kvar_ispw17
*   @email enkerewpo@gmail.com
*/

#include <bits/stdc++.h>
using namespace std;

typedef long long LL;

const int N = 20005;
const LL zip = 1e8;

LL Pow(LL a, LL b) { LL ret = 1; while (b) { if (b &
   1) {ret *= a;} a *= a; b >>= 1; } return ret; }

class BigInt {
    public:
    LL a[N];
    bool minus;
    BigInt() {
        minus = false;
        memset(a, 0, sizeof a);
    }
    template <typename _Tp>
    BigInt(_Tp x) {
        int p = 1, tmp = 0, y;
        a[0] = 1;
        while (x) {
            y = x % 10;
            tmp += y * Pow(10, p - 1);
            p %= 8;
            x /= 10;
            if (p == 0) {
                a[a[0]] = tmp;
                tmp = 0;
                a[0]++;
            }
```

```
            p++;
        }
        if (tmp) a[a[0]] = tmp;
        else a[0]--;
    }
    void read() {
        char s[N];
        scanf("%s", s + 1);
        int len = strlen(s + 1), st = 1;
        if (s[1] == '-') st = 2, minus = true;
        int p = 1, tmp = 0, y;
        a[0] = 1;
        for (int i = len; i >= st; i--) {
            y = (s[i] - '0') % 10;
            tmp += y * Pow(10, p - 1), p %= 8;
            if (p == 0) {
                a[a[0]] = tmp;
                tmp = 0;
                a[0]++;
            }
            p++;
        }
        if (tmp) a[a[0]] = tmp;
        else a[0]--;
    }
    void print() {
        if ((a[0] == 0) ||
           ((a[0] == 1) &&
            (a[1] == 0))) {
            printf("0");
            return;
        }
        if (minus) {
            printf("-");
            if (a[a[0]]) printf("%lld", a[a[0]]);
            for (int i = a[0] - 1; i; i--)
                printf("%08lld", a[i]);
            return;
        }
        if (a[a[0]]) {
            printf("%lld", a[a[0]]);
```

```
    }
    for (int i = a[0] - 1; i; i--)
        printf("%08lld", a[i]);
}
BigInt abs() {
    BigInt ret = *this;
    ret.minus = 0;
    return ret;
}
LL operator [] (const unsigned int idx) const {
    return a[idx];
}
LL& operator [] (const unsigned int idx) {
    return a[idx];
}
friend bool operator ==
(const BigInt lhs, const BigInt rhs) {
    if (lhs.minus ^ rhs.minus) return false;
    if (lhs[0] == rhs[0]) {
        for (int i = 1; i <= lhs[0]; i++)
        if (lhs[i] != rhs[i])
            return false;
        return true;
    } else return false;
}
friend bool operator >
(const BigInt lhs, const BigInt rhs) {
    if (lhs[0] == rhs[0]) {
        for (int i = 1; i <= lhs[0]; i++) {
            if (lhs[i] > rhs[i])
                return true;
            else return false;
        }
        return false;
    } else return lhs[0] > rhs[0];
}
friend BigInt operator + (BigInt a, BigInt b) {
    BigInt ret;
    bool f = a.minus ^ b.minus, g = false;
    if (f) {
        if (a.minus) g = true;
```

```
            a = a.abs(), b = b.abs();
            ret = a - b;
            if (g) ret.minus ^= 1;
            return ret;
        }
        for (int i = 1; i <= max(a[0], b[0]); i++) {
            ret[i] += a[i] + b[i];
            ret[i + 1] = ret[i] / zip;
            ret[i] %= zip;
        }
        ret[0] = max(a[0], b[0]);
        if (ret[ret[0] + 1]) {
            ret[0]++;
        }
        if (a.minus) ret.minus = true;
        return ret;
    }
    BigInt operator * (int rhs) {
        bool f = (rhs < 0) ^ minus;
        BigInt c;
        rhs = std::abs(rhs);
        for (int i = 1; i <= a[0]; i++) {
            c.a[i] += a[i] * rhs;
            if (c.a[i] >= zip) {
                c.a[i + 1] += c.a[i] / zip;
                c.a[i] %= zip;
            }
        }
        c.a[0] = a[0];
        if (c.a[c.a[0] + 1] > 0) c.a[0]++;
        if (f) c.minus = true;
        return c;
    }
    BigInt operator * (const BigInt rhs) const {
        BigInt ret;
        bool f = false;
        if (minus ^ rhs.minus) f = true;
        for (int i = 1; i <= a[0]; i++)
        for (int j = 1; j <= rhs[0]; j++) {
            ret[i + j - 1] += a[i] * rhs[j];
            ret[i + j] += ret[i + j - 1] / zip;
```

```
            ret[i + j - 1] %= zip;
        }
        ret[0] = max(0LL, a[0] + rhs[0] - 1);
        if (ret[a[0] + rhs[0]]) {
            ret[0]++;
        }
        if (f) ret.minus = true;
        return ret;
    }
    friend BigInt operator -
    (BigInt lhs, BigInt rhs) {
        if (lhs == rhs) {
            return BigInt(0);
        }
        if (lhs.minus ^ rhs.minus) {
            if (rhs.minus) return lhs + rhs.abs();
            else {
                BigInt ret = lhs.abs() + rhs.abs();
                ret.minus = true;
                return ret;
            }
        }
        bool right = lhs > rhs;
        lhs = lhs.abs(), rhs = rhs.abs();
        if (rhs > lhs) swap(lhs, rhs);
        BigInt ret;
        memcpy(ret.a, lhs.a, sizeof lhs.a);
        for (int i = 1; i <= lhs.a[0]; i++) {
            ret[i] -= rhs[i];
            if (ret[i] < 0)
                ret[i + 1]--, ret[i] += zip;
        }
        while (ret[0] >= 1 && !ret[ret[0]])
            ret[0]--;
        if (!right) ret.minus = true;
        return ret;
    }
    BigInt operator / (int rhs) {
        BigInt c;
        LL x = 0;
        for (int i = a[0]; i; i--) {
```

```
            x = x * zip + a[i];
            c.a[i] = x / rhs, x = x % rhs;
        }
        c.a[0] = a[0];
        if (c.a[0] && !c.a[c.a[0]])
            c.a[0]--;
        return c;
    }
    LL operator % (int rhs) {
    BigInt c;
    LL x = 0;
    for (int i = a[0]; i; i--) {
        x = x * zip + a[i];
        c.a[i] = x / rhs;
        x = x % rhs;
    }
    c.a[0] = a[0];
    if (c.a[0] && !c.a[c.a[0]])
        c.a[0]--;
    return x;
}
};

int main() {
    BigInt a, b;
    a.read(), b.read();
    BigInt c = a + b;
    c.print(); puts("");
    c = a - b;
    c.print(); puts("");
    c = a * b;
    c.print(); puts("");
    return 0;
}
```

1.9 快速傅里叶变换

```
#include <bitset>
#include <cstdio>
#include <cstring>
#include <cmath>
#include <algorithm>
#define N 400005
#define pi acos (-1.0) // PI value
using namespace std;
struct complex {
    double r, i;
    complex (double real = 0.0, double image = 0.0) {
        r = real; i = image;
    }
    // The following is the definition of three kinds
       of imaginary arithmetic
    complex operator + (const complex o) {
        return complex (r + o.r, i + o.i);
    }
    complex operator - (const complex o) {
        return complex (r - o.r, i - o.i);
    }
    complex operator * (const complex o) {
        return complex (r * o.r - i * o.i, r * o.i +
           i * o.r);
    }
} x1 [N], x2 [N];
char a [N / 2], b [N / 2];
int sum [N]; // The result exists in sum
int vis [N];
void brc (complex * a, int l) {
    // original bingliang binary parallel reversal
       replacement Taifeng do not understand, wrote
       an O (n) ......
    memset (vis, 0, sizeof (vis)); // O (logn) is
       followed
    for (int i = 1; i < l - 1; i ++) {
        int x = i, y = 0;
        int m = (int) log2 (l) + 0.1;
        if (vis [x]) continue;
```

```
        while (m--) {
            y <<= 1;
            y |= (x & 1);
            x >>= 1;
        }
        vis [i] = vis [y] = 1;
        swap (a [i], a [y]);
    }
}
void fft (complex * y, int l, double on) {// FFT O
   (nlogn)
    // where DFT is on== 1, on== - 1 is IDFT
    register int h, i, j, k;
    complex u, t;
    brc (y, l); // Call reversal permutation
    for (h = 2; h <= l; h <<= 1) { // Control the
       number of layers
        // initial unit complex root
        complex wn (cos (on * 2 * pi / h), sin (on *
           2 * pi / h));
        for (j = 0; j < l; j += h) { // Controls the
           start subscript
            complex w (1, 0); // Initialize the
               spiral factor
            for (k = j; k < j + h / 2; k ++) { //
               paired
                u = y [k];
                t = w * y [k + h / 2];
                y [k] = u + t;
                y [k + h / 2] = u - t;
                w = w * wn; // Update the spiral
                   factor
            } // It is said that the operation above
               is called butterfly operation ...
        }
    }
    if (on == -1) for (i = 0; i < l; i ++) y [i] .r
       /= l; // IDFT
}
int main () {
    int l1, l2, l;
```

```
    register int i;
    while (scanf("%s%s", a, b) != EOF) {
        l1 = strlen (a);
        l2 = strlen (b);
        l = 1;
        while (l < l1 * 2 || l < l2 * 2) l <<= 1;
           //The number of boundaries into 2 ^ n
        // with the dichotomy and reverse replacement
        for (i = 0; i < l1; i ++) { // Inverted
            x1 [i] .r = a [l1 - i - 1] - '0';
            x1 [i] .i = 0.0;
        }
        for (; i < l; i ++) x1 [i] .r = x1 [i] .i =
           0.0;
        // Initialize the extra count to zero
        for (i = 0; i < l2; i ++) {
            x2 [i] .r = b [l2 - i - 1] - '0';
            x2 [i] .i = 0.0;
        }
        for (; i < l; i ++) x2 [i] .r = x2 [i] .i =
           0.0;
        fft (x1, l, 1); // DFT (a)
        fft (x2, l, 1); // DFT (b)
        for (i = 0; i < l; i ++) x1 [i] = x1 [i] * x2
           [i];
        fft (x1, l, -1); // IDFT (a * b)
        for (i = 0; i < l; i ++) sum [i] = x1 [i] .r
           + 0.5; // round off
        for (i = 0; i < l; i ++) { // Carry
            sum [i + 1] += sum [i] / 10;
            sum [i] %= 10;
        }
        l = l1 + l2 - 1;
        while (sum [l] <= 0 && l > 0) l--; //
           retrieve the most significant bit
        for (i = l; i >= 0; i--) putchar (sum [i] +
           '0'); // output in reverse
        putchar ('\n');
    }
    return 0;
}
```

1.10 Miller Rabin 素数探测

```
#include <bits/stdc++.h>

/* QUICK POWER */
template <typename T>
T power(T A, T B, T P) {
    T ret = 1;
    while (B) {
        if (B & 1) ret = ret * A % P;
        A = A * A % P; B >>= 1;
    } return ret;
}

/* MILLER RABIN PRIME */
const int u[15] = {2, 3, 5, 7, 11, 13, 17, 19, 23,
   29, 31, 37}, cas = 12;

template <typename T>
bool miller_check(T a, T p) {
    T x = p - 1;
    if (power<T>(a, p - 1, p) != 1) return false;
    while (~x & 1) x >>= 1; /* x is an even number */
    a = power<T>(a, x, p);
    while (x < p - 1) {
        if (a != 1 && a != p - 1) {
            a = a * a % p;
            if (a == 1) return false;
        } else a = a * a % p;
        x <<= 1;
    }
    return true;
}
template <typename T>
bool miller(T p) {
    if (p == 1 || p == 0) return 0;
    for (int i = 0; i < cas; i++) {
        //printf("P: %d NOW: %d\n", p, u[i]);
        if (p == u[i]) return true;
        if (!miller_check<T>(u[i], p)) return false;
    }
```

```
    return true;
}

int main() {
    return 0;
}
```

1.11 Pollard Rho 因数分解

```
#include <bits/stdc++.h>
/**
* This is a Template of Pollard Rho and .Miller Rabin
   Prime Test
* email: enkerewpo@gmail.com
* blog: enkerewpo.github.io
*/
ll mult_mod(ll a, ll b, ll c) {
    a %= c;
    b %= c;
    ll ret = 0;
    while (b) {
        if (b & 1) {
            ret += a;
            ret %= c;
        }
        a <<= 1;
        if (a >= c)a %= c;
        b >>= 1;
    }
    return ret;
}

ll q_pow(ll x, ll n, ll mod) {
    if (n == 1) return x % mod;
    x %= mod;
    ll tmp = x;
    ll ret = 1;
    while (n) {
        if (n & 1) ret = mult_mod(ret, tmp, mod);
        tmp = mult_mod(tmp, tmp, mod);
```

```
        n >>= 1;
    }
    return ret;
}

bool check(ll a, ll n, ll x, ll t) {
    ll ret = q_pow(a, x, n);
    ll last = ret;
    for (int i = 1; i <= t; i++) {
        ret = mult_mod(ret, ret, n);
        if (ret == 1 && last != 1 && last != n - 1)
           return true;
        last = ret;
    }
    if (ret != 1) return true;
    return false;
}

bool Miller_Rabin(ll n) {
    if (n < 2) return false;
    if (n == 2) return true;
    if ((n & 1) == 0) return false;
    ll x = n - 1;
    ll t = 0;
    while ((x & 1) == 0) {
        x >>= 1;
        t++;
    }
    for (int i = 0; i < S; i++) {
        ll a = rand() % (n - 1) + 1;
        if (check(a, n, x, t))
        return false;
    }
    return true;
}

ll fac[100];
int tot;

ll gcd(ll a, ll b) {
    if (a == 0) return 1;
```

```
    if (a < 0) return gcd(-a, b);
    while (b) {
        ll t = a % b;
        a = b;
        b = t;
    }
    return a;
}

ll Pollard_rho(ll x, ll c) {
    ll i = 1, k = 2;
    ll x0 = rand() % x;
    ll y = x0;
    while (1) {
        i++;
        x0 = (mult_mod(x0, x0, x) + c) % x;
        ll d = gcd(y - x0, x);
        if (d != 1 && d != x) return d;
        if (y == x0) return x;
        if (i == k) {
            y = x0;
            k += k;
        }
    }
}

void findfac(ll n) {
    if (Miller_Rabin(n)) {
        fac[tot++] = n;
        return;
    }
    ll p = n;
    while (p >= n)
    p = Pollard_rho(p, rand() % (n - 1) + 1);
    findfac(p);
    findfac(n / p);
}
```

2 字符串

2.1 AC 自动机

```
#include <cstdio>
#include <string>
#include <cstring>
#include <algorithm>
using namespace std;
const int maxn = 100000;
struct point {
    int nxt[26];
    int fail, cnt;
} a[maxn];

int cnt, root;
int que[maxn], head, tail;
char A[maxn];

inline void insert(char* s) {
    int now = root;
    for (int i = 0; s[i] != '\0'; ++i) {
        int x = s[i] - 'a';
        if (a[now].nxt[x] == 0) a[now].nxt[x] = cnt++;
        now = a[now].nxt[x];
    }
    return;
}

inline void build() {
    for (int i = 0; i < 26; ++i)  {
        if (a[root].nxt[i] != 0) {
            int p = a[root].nxt[i];
            a[p].fail = root;
            que[tail++] = p;
        }
    }
    while (head < tail) {
        int p = que[head++];
```

```
        for (int i = 0; i < 26; ++i) if (a[p].nxt[i])
           {
            int x = a[p].nxt[i];
            int j = a[p].fail;
            while (j != root && a[j].nxt[i] == 0) j =
               a[j].fail;
            if (a[j].nxt[i]) j = a[j].nxt[i];
            a[x].fail = j;
            que[tail++] = x;
        }
    }
    return;
}

inline void solve(char* s) {
    int now = root;
    for (int i = 0; s[i] != '\0'; ++i) {
        int x = s[i] - 'a';
        while (now != root && a[now].nxt[x] == 0) now
           = a[now].fail;
        if (a[now].nxt[x]) now = a[now].nxt[x];
        ++a[now].cnt;
    }
    for (int i = tail - 1; i >= 0; --i) {
        int p = a[que[i]].fail;
        a[p].cnt += a[que[i]].cnt;
    }
    return;
}
int main() {
    int n;
    cnt = 2;
    root = 1;
    scanf("%d", &n);
    while (n--) scanf("%s", A), insert(A);
    build();
    scanf("%s", A);
    solve(A);
    printf("%d", a[2].cnt);
    return 0;
}
```

2.2 KMP 算法

```
#include <bits/stdc++.h>
#define N 10000000 + 5
using namespace std;

typedef long long ll;
ll ans;
char str[N], t[N];

ll nxt[N];
int main() {
    scanf("%s", t + 1);
    scanf("%s", str + 1);
    int lent = strlen(t + 1);
    int lenstr = strlen(str + 1);
    int j = 0;
    nxt[1] = 0;
    for (int i = 2; i <= lent; i++) {
        while (j > 0 && t[i] != t[j + 1]) j = nxt[j];
        if (t[j + 1] == t[i]) j++;
        nxt[i] = j;
    }
    j = 0;
    for (int i = 1; i <= lenstr; i++) {
        while (j > 0 && str[i] != t[j + 1]) j =
           nxt[j];
        if (t[j + 1] == str[i]) j++;
        if (j == lent) {

            ans++;
            j = nxt[j];
        }
    }
    printf("%lld\n", ans);
    return 0;
}
```

2.3 Manacher 算法

```
#include<cstdio>
#include<cstring>
#include<iostream>
#include<algorithm>

using namespace std;
const int maxn = 1000010;

int n, len[maxn * 2];
char S[maxn * 2];

inline void solve() {
    n = strlen(S + 1);
    n = n * 2 - 1;
    for (int i = n; i > 0; --i) if (i & 1) S[i] =
       S[(i + 1) / 2];
    else S[i] = '#';
    int mx = 0, id = 0;
    for (int i = 1; i <= n; ++i) {
        if (mx <= i) len[i] = 1;
        else len[i] = min(mx - i + 1, len[2 * id -
           i]);
        while (i - len[i] > 0 && i + len[i] <= n &&
           S[i + len[i]] == S[i - len[i]]) ++len[i];
        if (i + len[i] - 1 > mx) {
            mx = i + len[i] - 1;
            id = i;
        }
    }
}

int main() {
    scanf("%s", S + 1);
    solve();
    printf("S=␣%s\n", S + 1);
    for (int i = 1; i <= n; ++i)
       printf("len[%d]=%d\n", i, len[i]);
    return 0;
}
```

3 图论

3.1 树链剖分

```
#include <bits/stdc++.h>
using namespace std;

#define N 30000 + 5
#define inf 0x7f7f7f7f

int n, q, hd[N], nxt[N], to[N], w[N], edge, tot;
int siz[N], dep[N], u_son[N], fa[N], top[N], tr[N],
   dtr[N];

namespace segtree {

    int TR[4 * N], MAX[4 * N];

    void up(int p) {
        TR[p] = TR[p << 1] + TR[p << 1 | 1];
        MAX[p] = max(MAX[p << 1], MAX[p << 1 | 1]);
    }

    void build(int l, int r, int p) {
        if (l == r) {
            TR[p] = w[dtr[l]];
            MAX[p] = TR[p];
            return;
        }
        int mid = (l + r) / 2;
        build(l, mid , p << 1);
        build(mid + 1, r, p << 1 | 1);
        up(p);
    }

    void change(int l, int r, int P, int val, int p) {
        if (l == r && l == P) {
            TR[p] = val;
            MAX[p] = TR[p];
            return;
```

```
        }
        int mid = (l + r) / 2;
        if (P <= mid) change(l, mid, P, val, p << 1);
        else change(mid + 1, r, P, val, p << 1 | 1);
        up(p);
    }

    int ask_max(int l, int r, int L, int R, int p) {
        if (l > R || r < L) return -inf;
        if (L <= l && r <= R) return MAX[p];
        int mid = (l + r) / 2;
        return max(ask_max(l, mid, L, R, p << 1),
           ask_max(mid + 1, r, L, R, p << 1 | 1));
    }

    int ask_sum(int l, int r, int L, int R, int p) {
        if (l > R || r < L) return 0;
        if (L <= l && r <= R) return TR[p];
        int mid = (l + r) / 2;
        return ask_sum(l, mid, L, R, p << 1) +
           ask_sum(mid + 1, r, L, R, p << 1 | 1);
    }

}

namespace HLD {

    void add(int a, int b) {
        nxt[++edge] = hd[a], to[edge] = b, hd[a] =
           edge;
        nxt[++edge] = hd[b], to[edge] = a, hd[b] =
           edge;
    }

    void dfs1(int u, int __fa, int __dep) {
        fa[u] = __fa;
        dep[u] = __dep;
        siz[u] = 1;
        for (int c = hd[u]; c; c = nxt[c]) {
            int v = to[c];
            if (v != __fa) {
```

```
            dfs1(v, u, __dep + 1);
            siz[u] += siz[v];
            if (!u_son[u] || siz[v] >
               siz[u_son[u]]) u_son[u] = v;
        }
    }
}

void dfs2(int u, int id) {
    top[u] = id;
    tr[u] = ++tot;
    dtr[tot] = u;
    if (!u_son[u]) return;
    dfs2(u_son[u], id);
    for (int c = hd[u]; c; c = nxt[c]) {
        int v = to[c];
        if (v != u_son[u] && v != fa[u]) dfs2(v,
           v);
    }
}

int max(int a, int b) {
    int f1, f2, ret = -inf;
    f1 = top[a];
    f2 = top[b];
    while (f1 != f2) {
        if (dep[f1] < dep[f2]) swap(a, b),
           swap(f1, f2);
        ret = std::max(ret, segtree::ask_max(1,
           n, tr[f1], tr[a], 1));
        a = fa[f1];
        f1 = top[a];
    }
    if (dep[a] < dep[b]) swap(a, b);
    ret = std::max(ret, segtree::ask_max(1, n,
       tr[b], tr[a], 1));
    return ret;
}

int sum(int a, int b) {
    int f1, f2, ret = 0;
```

```
        f1 = top[a];
        f2 = top[b];
        while (f1 != f2) {
            if (dep[f1] < dep[f2]) swap(a, b),
               swap(f1, f2);
            ret += segtree::ask_sum(1, n, tr[f1],
               tr[a], 1);
            a = fa[f1];
            f1 = top[a];
        }
        if (dep[a] < dep[b]) swap(a, b);
        ret += segtree::ask_sum(1, n, tr[b], tr[a],
           1);
        return ret;
    }

}

int main() {
    scanf("%d", &n);
    for (int i = 1; i < n; i++) {
        int a, b;
        scanf("%d%d", &a, &b);
        HLD::add(a, b);
    }
    HLD::dfs1(1, 0, 1);
    HLD::dfs2(1, 1);
    for (int i = 1; i <= n; i++) scanf("%d", &w[i]);
    segtree::build(1, n, 1);
    scanf("%d", &q);
    while (q--) {
        string op;
        cin >> op;
        int a, b;
        scanf("%d%d", &a, &b);
        if      (op == "QMAX")     printf("%d\n",
           HLD::max(a, b));
        else if (op == "QSUM")     printf("%d\n",
           HLD::sum(a, b));
        else                       segtree::change(1,
           n, tr[a], b, 1);
```

```
    }
    return 0;
}
```

3.2 Floyd 传递闭包

```
void floyd() {
    int i, j, k;
    int n;
    bool* con = new bool[n + 1][n + 1];
    for (int k = 1; k <= n; k++)
    for (int i = 1; i <= n; i++)
    for (int j = 1; j <= n; j++)
    con[i][j] |= con[i][k] & con[k][j];
}
```

3.3 点双连通分量

```
#include <iostream>
#include <cstdio>
#include <algorithm>
#include <cstring>
#include <string>
#include <cmath>
#include <vector>
#include <stack>
using namespace std;

const int maxn = 1000;

struct Edge {// structure of the side of the stack
    int u, v;
    Edge(int uu, int vv) {
        u = uu;
        v = vv;
    }
};
stack <Edge> s;
```

```
struct edge {// The edge structure of the chained
   forward starburst
    int v, next;
} edges[maxn];

int n, m; // The number of nodes, the number of
   undirected edges
int e, head[maxn];
int pre[maxn]; // The timestamp of the first visit
int dfs_clock; // timestamp
int iscut[maxn]; // mark whether the node is cut top
int bcc_cnt; // dot_ The number of double connected
   components
int bccno[maxn]; // The point to which the node
   belongs _ The number of the dual connected
   component
vector <int> bcc[maxn]; // dot_double connected
   component

void addedges(int u, int v) {// Add edge
    edges[e].v = v;
    edges[e].next = head[u];
    head[u] = e ++;
    edges[e].v = u;
    edges[e].next = head[v];
    head[v] = e ++;
}

int dfs(int u, int fa) {
    int lowu = pre[u] = ++ dfs_clock;
    int child = 0;
    for (int i = head[u]; i != -1; i = edges[i].next)
       {
        int v = edges[i].v;
        Edge e = (Edge) {u, v};
        if (! pre[v]) {
            s.push(e);
            child ++;
            int lowv = dfs(v, u);
            lowu = min(lowu, lowv); // Update lowu
               with descendants
```

```
            if (lowv >= pre[u]) { // Find a subtree
               that satisfies the condition of the cut
                iscut[u] = 1;
                bcc_cnt ++;
                bcc[bcc_cnt].clear();
                for (;;) { // save the bcc information
                    Edge x = s.top(); s.pop();
                    if (bccno[x.u] != bcc_cnt)
                       {bcc[bcc_cnt].push_back(x.u);
                       bccno[x.u] = bcc_cnt;}
                    if (bccno[x.v] != bcc_cnt)
                       {bcc[bcc_cnt].push_back(x.v);
                       bccno[x.v] = bcc_cnt;}
                    if (x.u == u && x.v == v) break;

                }
            }
        } else if (pre[v] < pre[u] && v != fa) { //
           Update lowu
            s.push(e);
            lowu = min(lowu, pre[v]);
        }
    }
    if (fa < 0 && child == 1) iscut[u] = 0; // If the
       root node has only one subtree then it is not
       a cut
    return lowu;
}

void init() {
    memset(pre, 0, sizeof(pre));
    memset(iscut, 0, sizeof(iscut));
    memset(head, -1, sizeof(head));
    memset(bccno, 0, sizeof(bccno));
    e = 0; dfs_clock = 0; bcc_cnt = 0;
}

int main() {
    int u, v;
    while (scanf("%d%d", &n, &m) != EOF) {
        init();
```

```
        for (int i = 0; i < m; i ++) {
            scanf("%d%d", &u, &v);
            addedges(u, v);
        }
        dfs(1, -1);
        for (int i = 1; i <= bcc_cnt; i ++) {
            for (int j = 0; j < (int)bcc[i].size(); j
               ++)
                cout << bcc[i][j] << "";
            cout << endl;
        }
    }
    return 0;
}
```

3.4 边双连通分量

```
#include <iostream>
#include <cstdio>
#include <algorithm>
#include <cstring>
#include <string>
#include <cmath>
#include <vector>
using namespace std;

const int maxn = 1000;
struct Edge {
    int no, v, next; // no: Number of the edge
} edges[maxn];

int n, m, ebcnum; // number of nodes, number of
   undirected edges, number of edges_double connected
   components
int e, head[maxn];
int pre[maxn]; // The timestamp of the first visit
int dfs_clock; // timestamp
int isbridge[maxn]; // mark if the edge is a bridge
vector <int> ebc[maxn]; // Edge_double connected
   component
```

```
void addedges(int num, int u, int v) {// Add edge
    edges[e].no = num;
    edges[e].v = v;
    edges[e].next = head[u];
    head[u] = e++;
    edges[e].no = num++;
    edges[e].v = u;
    edges[e].next = head[v];
    head[v] = e++;
}

int dfs_findbridge(int u, int fa) {// Find all the
   bridges
    int lowu = pre[u] = ++dfs_clock;
    for (int i = head[u]; i != -1; i = edges[i].next)
       {
        int v = edges[i].v;
        if (! pre[v]) {
            int lowv = dfs_findbridge(v, u);
            lowu = min(lowu, lowv);
            if (lowv > pre[u]) {
                isbridge[edges[i].no] = 1; // bridge
            }
        } else if (pre[v] < pre[u] && v != fa) {
            lowu = min(lowu, pre[v]);
        }
    }
    return lowu;
}

void dfs_coutbridge(int u, int fa) {// Saves the
   information of the edge_double connected component
    ebc[ebcnum].push_back(u);
    pre[u] = ++dfs_clock;
    for (int i = head[u]; i != -1; i = edges[i].next)
       {
        int v = edges[i].v;
        if (! isbridge[edges[i].no] && ! pre[v])
           dfs_coutbridge(v, u);
    }
```

```
}

void init() {
    memset(pre, 0, sizeof(pre));
    memset(isbridge, 0, sizeof(isbridge));
    memset(head, -1, sizeof(head));
    e = 0; ebcnum = 0;
}

int main() {
    int u, v;
    while (scanf("%d%d", &n, &m) != EOF) {
        init();
        for (int i = 0; i < m; i++) {
            scanf("%d%d", &u, &v);
            addedges(i, u, v);
        }
        dfs_findbridge(1, -1);
        memset(pre, 0, sizeof(pre));
        for (int i = 1; i <= n; i++) {
            if (! pre[i]) {
                ebc[ebcnum].clear();
                dfs_coutbridge(i, -1);
                ebcnum++;
            }
        }
        for (int i = 0; i < ebcnum; i++) {
            for (int j = 0; j < (int)ebc[i].size();
               j++)
            cout << ebc[i][j] << "␣";
            cout << endl;
        }
    }
    return 0;
}
```

3.5 最近公共祖先

```
#include <bits/stdc++.h>
using namespace std;
```

```
int dep[10], fa[10][10];
int lca(int x, int y) {
    if (dep[x] < dep[y]) swap(x, y);
    int d = dep[x] - dep[y];
    for (int i = 0; i <= 18; i++) {
        if (d & (1 << i)) x = fa[x][i];
    }
    for (int i = 18; i >= 0; i--) {
        if (fa[x][i] != fa[y][i]) {
            x = fa[x][i], y = fa[y][i];
        }
    }
    if (x == y) return x;
    else return fa[x][0];
}
```

3.6 差分约束系统

```
#include <algorithm>
#include <cstring>
#include <cstdio>
#include <queue>
#include <cassert>

#define N 200005 + 5
#define inf 1e9
using namespace std;
#define int long long

int hd[2 * N], nxt[2 * N], to[2 * N], w[2 * N], tot;
int n, k;
void add(int a, int b, int c) {
    nxt[++tot] = hd[a], to[hd[a] = tot] = b, w[tot] =
       c;
}

int dis[4 * N];
int cnt[4 * N];
bool vis[4 * N];
```

```
int spfa() {
    memset(dis, -inf, sizeof dis);
    queue<int> q;
    q.push(0);
    vis[0] = true;
    dis[0] = 0;
    cnt[0]++;
    while (!q.empty()) {
        int u = q.front();
        q.pop();
        if (cnt[u] == n + 1) return false;
        vis[u] = false;
        for (int e = hd[u]; e != -1; e = nxt[e]) {
            int v = to[e];
            if (dis[u] + w[e] > dis[v]) {
                dis[v] = dis[u] + w[e];
                if (!vis[v]) {
                    q.push(v);
                    cnt[v]++;
                    vis[v] = true;
                }
            }
        }
    }
    return true;
}

signed main() {
    memset(hd, -1, sizeof hd);
    scanf("%lld%lld", &n, &k);
    while (k--) {
        int x, a, b;
        scanf("%lld%lld%lld", &x, &a, &b);
        switch (x) {
            case 1 : { add(a, b, 0), add(b, a, 0);
               break;}                                  //
               A   =   B
            case 2 : { if (a == b) return puts("-1"),
               0; else add(a, b, 1); break;}  //  A
               <   B
```

```
                case 3 : { add(b, a, 0); break;}
                                                    //  A
                    >=  B
                case 4 : { if (a == b) return puts("-1"),
                   0; else add(b, a, 1); break;}  //  A
                   >   B
                case 5 : { add(a, b, 0); break;}
                                                    //  A
                    <=  B
            }
        }
        for (int i = n; i >= 1; i--) add(0, i, 1);
        if (spfa()) {
            int ans = 0;
            for (int i = 1; i <= n; i++) ans += dis[i];
            printf("%lld\n", ans);
        } else puts("-1");
        return 0;
}
```

3.7 虚树

```
#include <algorithm>
#include <cstring>
#include <cstdio>
/**
*  This is a Hollow Tree implement by Kvar_ispw17
*  @author Kvar_ispw17
*  @email enkerewpo@gmail.com
*/
using namespace std;

const int N = 10;
const int M = 10;

int n, m, key[5];
int dep[N], fa[N][10], dfn[N], tim;

/**
```

```
*  Sample input : |V| = 6, |E| = 5 E = { (1, 2) (2,
   5) (2, 3) (3, 4) (3, 6) }
*  Sample key-node input : V' = { 3, 6, 5 }
*
*  Defination : node '0' as the super root.
*            node '1' as the tree root.
*/

int hd[N], nxt[2 * M], to[2 * M], tot;
/**
* add edge to the original tree
* @param a from
* @param b to
*/
void add(int a, int b) {
    nxt[++tot] = hd[a], to[hd[a] = tot] = b;
}

int hd2[N], nxt2[2 * M], to2[2 * M], tot2, from2[2 *
   M];
/**
* add edge to the hollow tree
* @param a from
* @param b to
*/
void add2(int a, int b) {
    nxt2[++tot2] = hd2[a], to2[hd2[a] = tot2] = b,
       from2[tot2] = a;
}

int lca(int a, int b) {
    if (dep[a] < dep[b]) swap(a, b);
    int d = dep[a] - dep[b];
    for (int j = 0; j <= 5; j++) {
        if (d & (1 << j)) a = fa[a][j];
    }
    if (a == b) return a;
    for (int j = 5; j >= 0; j--) {
        if (fa[a][j] != fa[b][j]) {
            a = fa[a][j], b = fa[b][j];
        }
```

```
    }
    return fa[a][0];
}

bool Cmp_Fn(const int &a, const int &b) {
    return dfn[a] < dfn[b];
}

/**
* main algorithm for building the hollow tree.
*/
void build() {
    sort(key + 1, key + m + 1, Cmp_Fn);
    int s[N], top = -1;
    memset(s, 0, sizeof s);
    s[++top] = 0;
    int cnt = m;
    for (int i = 1; i <= m; i++) {
        int u = key[i];
        int lca = ::lca(u, s[top]);
        if (lca == s[top]) s[++top] = u;
        else {
            while (top - 1 >= 0 && dep[s[top - 1]] >=
               dep[lca]) {
                add2(s[top - 1], s[top]);
                add2(s[top], s[top - 1]);
                top--;
            }
            if (s[top] != lca) {
                add2(lca, s[top]);
                add2(s[top], lca);
                s[top] = lca;
                key[++cnt] = lca;
            }
            s[++top] = u;
        }
    }
    for (int i = 0; i < top; i++) {
        add2(s[i], s[i + 1]);
        add2(s[i + 1], s[i]);
    }
```

```
}

/**
* print the hollow tree int depth by first-order dfs
* @param u current node
* @param d current depth
*/
void print(int u, int _fa, int d) {
    for (int i = 1; i <= d; i++) printf("␣␣␣␣");
    printf("%d\n", u);
    for (int e = hd2[u]; e; e = nxt2[e]) {
        int v = to2[e];
        if (v != _fa) {
            print(v, u, d + 1);
        }
    }
}

void dfs(int u, int _fa, int d) {
    dep[u] = d;
    dfn[u] = ++tim;
    fa[u][0] = _fa;
    for (int e = hd[u]; e; e = nxt[e]) {
        int v = to[e];
        if (v != _fa) {
            dfs(v, u, d + 1);
        }
    }
}

/**
* program main function
* @return program stantard return code
*/
int main() {
    scanf("%d", &n);
    for (int i = 1; i < n; i++) {
        int a, b;
        scanf("%d%d", &a, &b);
        add(a, b);
        add(b, a);
```

```
    }
    dfs(1, 0, 1);
    for (int j = 1; j <= 5; j++) {
        for (int i = 1; i <= n; i++) {
            fa[i][j] = fa[fa[i][j - 1]][j - 1];
        }
    }
    scanf("%d", &m);
    for (int i = 1; i <= m; i++) scanf("%d", &key[i]);
    build();
    puts("printing␣the␣hollow␣tree␣of␣input␣by␣
       depth:");
    print(0, -1, 0);
    return 0;
}
/*
Sample input:
6
1 2
2 3
2 5
3 4
3 6
3
3 6 4
*/
```

3.8 Dinic 网络流

```
#include <bits/stdc++.h>
using namespace std;
#define N 1000 + 5
#define INF 0x7f7f7f7f
#define M 100000 + 5
int hd[N], nxt[2 * M], f[2 * M], to[2 * M], tot = 1;

int S, T;

void add(int a, int b, int c) {
```

```
    nxt[++tot] = hd[a], to[hd[a] = tot] = b, f[tot] =
       c;
    nxt[++tot] = hd[b], to[hd[b] = tot] = a, f[tot] =
       0;
}

int dis[N];
int que[N];

bool check() {
    memset(dis, -1, sizeof dis);
    int head = 0, tail = -1;
    dis[que[++tail] = S] = 0;
    while (head <= tail) {
        int u = que[head++];
        for (int e = hd[u]; e; e = nxt[e]) {
            int v = to[e];
            if (dis[v] == -1 && f[e])
            dis[que[++tail] = v] = dis[u] + 1;
        }
    }
    return dis[T] != -1;
}

int argument(int u, int rem) {
    if (u == T) return rem;
    int cnt = 0;
    for (int e = hd[u]; e && rem > cnt; e = nxt[e]) {
        int v = to[e];
        if (dis[u] + 1 == dis[v] && f[e]) {
            int ret = argument(v, min(rem - cnt,
               f[e]));
            f[e] -= ret;
            f[e ^ 1] += ret;
            cnt += ret;
        }
    }
    if (!cnt) dis[u] = -1;
    return cnt;
}
```

```
int dinic() {
    int cnt = 0, tmp;
    while (check())
    while (tmp = argument(S, INF))
    cnt += tmp;
    return cnt;
}
```

3.9 SPFA 最小费用流

```
#include <bits/stdc++.h>
using namespace std;

const int N = 1100, INF = 0x3f3f3f3f;
const int M = N * N;

int pre[N], d[N], p[N], ans, cnt, hd[N], q[M], l, r;

struct edge {
    int u, v, w, c, nxt;
} e[M];

void init() {
    memset(hd, -1, sizeof(hd));
    ans = cnt = 0;
}

void add(int u, int v, int w, int c) {
    e[cnt].u = u, e[cnt].v = v, e[cnt].w =  w,
       e[cnt].c = c, e[cnt].nxt = hd[u], hd[u] =
       cnt++;
    e[cnt].u = v, e[cnt].v = u, e[cnt].w = -w,
       e[cnt].c = 0, e[cnt].nxt = hd[v], hd[v] =
       cnt++;
}

void update(int s, int t) {
    int i, f = INF;
    for (i = t; i != s; i = e[pre[i]].u)
    f = min(f, e[pre[i]].c);
```

```
    for (i = t; i != s; i = e[pre[i]].u) {
        e[pre[i]].c -= f;
        e[pre[i] ^ 1].c += f;
        ans += f * e[pre[i]].w;
    }
}

int spfa(int s, int t) {
    int i, u, v, w;
    memset(p, 0, sizeof(p));
    memset(pre, -1, sizeof(pre));
    memset(d, 0x3f, sizeof(d));
    l = r = 0;
    q[++r] = s, p[s] = 1, d[s] = 0;
    while (l < r) {
        p[u = q[++l]] = 0;
        for (i = hd[u]; i != -1; i = e[i].nxt) {
            v = e[i].v, w = e[i].w;
            if (e[i].c && d[v] > d[u] + w) {
                d[v] = d[u] + w;
                pre[v] = i;
                if (!p[v]) {
                    p[v] = 1;
                    q[++r] = v;
                }
            }
        }
    }
    if (pre[t] == -1)
    return 0;
    return 1;
}
void solve(int s, int t) {
    ans = 0;
    while (spfa(s, t)) update(s, t);
}
```

3.10 匈牙利算法

```
bool hungary(int u) {
```

```
    for (it = _v[u].begin(); it != _v[u].end(); it++)
       {
        int v = *it;
        if (!vis[v]) {
            vis[v] = true;
            if (!fa[v] || hungary(fa[v])) {
                fa[v] = u;
                return true;
            }
        }
    }
    return false;
}

int main() {
    for (int i = 1; i <= n; i++) {
        memset(vis, 0, sizeof vis);
        hungary(i);
    }
    return 0;
}
```

3.11 2-SAT

3.11.1 判定

```
#include <cstdio>
#include <algorithm>
#include <cstring>
#include <vector>
#include <stack>
#define N 10000
#define M 10000 + 5
using namespace std;

int T, n, m, x;
vector<int> V[N];
int main() {
    scanf("%d", &T);
    while (T--) {
        init();
        scanf("%d%d", &n, &m);
        for (int i = 1; i <= m; i++) {
            int a, b;
            char op1, op2;
            scanf("%*c%c%d␣%c%d", &op1, &a, &op2, &b);
            a <<= 1; if (op1 == 'h') a |= 1;
            b <<= 1; if (op2 == 'h') b |= 1;
            V[a ^ 1].push_back(b);
            V[b ^ 1].push_back(a);
        }
        for (int i = 2; i <= 2 * n + 1; i++) if
           (!dfn[i]) tarjan(i);
        bool f = true;
        for (int i = 1; i <= n; i++) {
            if (scc[i << 1] == scc[i << 1 | 1]) {
                f = false;
                break;
            }
        }
        if (f) puts("GOOD");
        else puts("BAD");
    }
```

```
    return 0;
}
```

3.11.2 输出方案

```
while(!q.empty()) {
    int u = q.front();
    q.pop();
    if(choose[u] != -1) continue;
    choose[u] = 0;
    choose[u^1] = 1;
    for(int i = 0; i < V[u].size(); i++) {
        --in[G2[u][i]];
        if(!in[V[u][i]]) q.push(V[u][i]);
    }
}
```

输出的时候，按照原来缩点之后的图的反向建图，然后进行拓扑排序，对于拓扑排序出来的每一个点，判断这个点（这个点其实是缩点之后的圈）是否被标记颜色，如果没有被标记过，则认为是颜色 1，并且将它的对应点标记为-1。

3.12 拓扑排序

```
#include <algorithm>
#include <cstring>
#include <cstdio>
#include <iostream>
#include <bitset>
#include <stack>
#define N (2000 + 5)
#define M (N * N >> 1) + 5
using namespace std;

int hd[N], nxt[M], to[M], tot;
int hd2[N], nxt2[M], to2[M], tot2;
int in[N];
void add(int a, int b) { nxt[++tot] = hd[a], to[hd[a]
   = tot] = b; }
void add2(int a, int b) { nxt2[++tot2] = hd2[a],
   to2[hd2[a] = tot2] = b, in[b]++; }
```

```
bitset<N> dp[N];
int n, _n, scc[N], siz[N], ans;

void topological_sort() {
    int q[N], head = 0, tail = -1;
    memset(q, 0, sizeof q);
    for (int i = 1; i <= _n; i++) {
        if (in[i] == 0) {
            q[++tail] = i;
            ans += siz[i] * dp[i].count();
        }
    }
    while (head <= tail) {
        int u = q[head++];
        for (int e = hd2[u]; e; e = nxt2[e]) {
            int v = to2[e];
            dp[v] |= dp[u];
            if (--in[v] == 0) {
                q[++tail] = v;
                ans += siz[v] * dp[v].count();
            }
        }
    }

}

bool vis[N];
int low[N], dfn[N], tim, x;
stack<int> s;
void tarjan(int u) {
    low[u] = dfn[u] = ++tim;
    s.push(u);
    vis[u] = true;
    for (int e = hd[u]; e; e = nxt[e]) {
        int v = to[e];
        if (!dfn[v]) {
            tarjan(v);
            low[u] = min(low[u], low[v]);
        } else if (vis[v]) low[u] = min(low[u],
           dfn[v]);
```

```
    }
    if (low[u] == dfn[u]) {
        _n++; do {
            x = s.top();
            s.pop();
            vis[x] = false;
            scc[x] = _n;
            siz[_n]++;
            dp[_n][x] = 1;
        } while (x != u);
    }
}

void prerequites() {
    for (int i = 1; i <= n; i++) if (!dfn[i])
       tarjan(i);
    for (int u = 1; u <= n; u++) {
        for (int e = hd[u]; e; e = nxt[e]) {
            int v = to[e];
            if (scc[v] != scc[u]) {
                add2(scc[u], scc[v]);
            }
        }
    }
}

int main() {
    scanf("%d", &n);
    for (int i = 1; i <= n; i++) {
        char s[N];
        scanf("%s", s + 1);
        for (int j = 1; j <= n; j++) {
            if (s[j] - '0') add(j, i);
        }
    }
    prerequites();
    topological_sort();
    printf("%d\n", ans);
    return 0;
}
```

3.13 树的直径

```
#include <cstdio>
#include <cstring>
#include <algorithm>
#include <cctype>
#include <cmath>
#define N 500000 + 5
#define inf 0x7fffffff
using namespace std;
#define mem(x) memset(x, 0, sizeof (x))

int fa[N], vis[N], dis[N], maxdis, maxi;
int hd[N], nxt[N], to[N], w[N], tot;
int n, s, diameter, ans;
int dia[N], dtot;
bool f;

void add(int a, int b, int c) {
    nxt[++tot] = hd[a], to[hd[a] = tot] = b, w[tot] =
       c;
    nxt[++tot] = hd[b], to[hd[b] = tot] = a, w[tot] =
       c;
}
void push(int u) { f = 1; while (u) { dia[++dtot] =
   u; u = fa[u]; } }
void dfs(int u, int Fa, int final) {
    if (final == 1 && dis[u] == maxdis && !f) {
       push(u); return; } fa[u] = Fa;
    for (int e = hd[u]; e; e = nxt[e]) {
        int v = to[e]; if (v != Fa) {
            if (final == 2 && vis[v]) continue;
            dis[v] = dis[u] + w[e];
            if (dis[v] > maxdis) maxdis = dis[v],
               maxi = v;
            dfs(v, u, final);
        }
    }
}
int main() {
    read(n), read(s); ans = inf;
```

```
    for (int i = 1; i < n; i++) { int a, b, c;
       read(a), read(b), read(c); add(a, b, c); }
    int tmp; dfs(1, 0, 0), maxdis = 0, mem(dis),
       mem(fa), dfs(tmp = maxi, 0, 0), dfs(tmp, 0, 1);
    diameter = maxdis;
    return 0;
}
```

3.14 树分块

```
#include <cstdio>
#include <algorithm>
#include <cstring>
#include <vector>
#define N 1000 + 5
using namespace std;
vector<int> _v[N];
int stack[N], top = -1;
int n, b;
int color[N], capital[N], cnt;

void dfs(int u, int fa) {
    int limit = top;
    vector<int>::iterator i;
    for (i = _v[u].begin(); i != _v[u].end(); i++) {
        int v = *i;
        if (v != fa) {
            dfs(v, u);
            if (top - limit >= b) {
                capital[++cnt] = u;
                while (top != limit)
                   color[stack[top--]] = cnt;
            }
        }
    }
    stack[++top] = u;
}

int main() {
    scanf("%d%d", &n, &b);
```

```
    for (int i = 1; i < n; i++) {
        int a, b;
        scanf("%d%d", &a, &b);
        _v[a].push_back(b);
        _v[b].push_back(a);
    }
    dfs(1, 0);
    while (top >= 0) color[stack[top--]] = cnt;
    printf("%d\n", cnt);
    for (int i = 1; i <= n; i++) printf("%d␣",
       color[i]);
    puts("");
    for (int i = 1; i <= cnt; i++) printf("%d␣",
       capital[i]);
    puts("");
    return 0;
}
```

3.15 静态点分治

```
#include<cstdio>
#include<cstring>
#include<iostream>
#include<algorithm>
#define M 20200
using namespace std;
struct node {
    int to, f, nxt;
    bool ban;
} e[M << 1];
int head[M], tot = 1;
int n, ans;
int Siz[M];
void add(int x, int y, int z) {
    e[++tot].to = y;
    e[tot].f = z;
    e[tot].nxt = head[x];
    head[x] = tot;
}
void get_center(int x, int from, int Size, int &cg) {
```

```
    int i; Siz[x] = 1;
    bool flag = 1;
    for (i = head[x]; i; i = e[i].nxt) {
        if (e[i].ban || e[i].to == from) continue;
        get_center(e[i].to, x, Size, cg);
        if (Siz[e[i].to] > Size >> 1)
        flag = 0;
        Siz[x] += Siz[e[i].to];
    }
    if (Size - Siz[x] > Size >> 1) flag = 0;
    if (flag) cg = x;
}
void dfs(int x, int from, int dis, int cnt[]) {
    int i;
    Siz[x] = 1;
    cnt[dis]++;
    for (i = head[x]; i; i = e[i].nxt) {
        if (e[i].ban || e[i].to == from)
        continue;
        dfs(e[i].to, x, (dis + e[i].f) % 3, cnt);
        Siz[x] += Siz[e[i].to];
    }
}
void clac(int x) {
    int i;
    int cnt[3] = {1, 0, 0};
    for (i = head[x]; i; i = e[i].nxt) {
        if (e[i].ban) continue;
        int _cnt[3] = {0};
        dfs(e[i].to, x, e[i].f, _cnt);
        ans += cnt[0] * _cnt[0];
        ans += cnt[1] * _cnt[2];
        ans += cnt[2] * _cnt[1];
        cnt[0] += _cnt[0];
        cnt[1] += _cnt[1];
        cnt[2] += _cnt[2];
    }
}
void divide_and_conquer(int root, int Size) {
    int i, cg;
    if (Size == 1) return;
```

```
    get_center(root, 0, Size, cg);
    clac(cg);
    for (i = head[cg]; i; i = e[i].nxt) {
        if (e[i].ban)
        continue;
        e[i].ban = e[i ^ 1].ban = 1;
        divide_and_conquer(e[i].to, Siz[e[i].to]);
    }
}

int main() {
    int i, x, y, z;
    cin >> n;
    for (i = 1; i < n; i++) {
        scanf("%d%d%d", &x, &y, &z);
        add(x, y, z % 3);
        add(y, x, z % 3);
    }
    divide_and_conquer((1 + n) >> 1, n);
    ans = ans * 2 + n;
    int gcd = __gcd(ans, n * n);
    cout << ans / gcd << '/' << n*n / gcd << endl;
}
```

4 数据结构

4.1 二维树状数组

```
int lowbit(int x) {
    return x & -x;
}

void add(int x, int y, int d) {
    int i, j;
    for (i = x; i < N; i += lowbit(i))
    for (j = y; j < N; j += lowbit(j)) mat[i][j] += d;
}

int sum(int x, int y) {
    int ret = 0;
    int i, j;
    for (i = x; i > 0; i -= lowbit(i))
    for (j = y; j > 0; j -= lowbit(j)) ret +=
       mat[i][j];
    return ret;
}
```

4.2 二维线段树

```
#include <bits/stdc++.h>
using namespace std;

const int maxn = 1000 * 1000 * 1.5;
// 2D line tree
struct node {
    short x1, y1, x2, y2; // (a, b) - (c, d)
    float max;
    int * ch;
};
node tree [maxn]; // space: O (N * M * 2) // In fact,
   1.4 * N * M is sufficient
int tol;
void maketree (int x1, int y1, int x2, int y2) {//
   time: O (2 * N * M)
```

```
    tol ++;
    int k = tol;
    tree [k] .x1 = x1;
    tree [k] .y1 = y1;
    tree [k] .x2 = x2;
    tree [k] .y2 = y2;
    tree [k] .max = 0.0;
    if (x1 == x2 && y1 == y2) {// c == d
        tree [k] .ch = 0;
        return;

    }
    tree [k] .ch = new int [4];
    int midx = (x1 + x2) / 2;
    int midy = (y1 + y2) / 2;
    if (x1 <= midx && y1 <= midy) {// lower left
        tree [k] .ch [0] = tol + 1;
        maketree (x1, y1, midx, midy);

    } else
    tree [k] .ch [0] = 0;
    if (midx + 1 <= x2 && midy + 1 <= y2) {// top
       right
        tree [k] .ch [1] = tol + 1;
        maketree (midx + 1, midy + 1, x2, y2);

    } else
    tree [k] .ch [1] = 0;
    if (x1 <= midx && midy + 1 <= y2) {// top left
        tree [k] .ch [2] = tol + 1;
        maketree (x1, midy + 1, midx, y2);

    } else
    tree [k] .ch [2] = 0;
    if (midx + 1 <= x2 && y1 <= midy) {// lower right
        tree [k] .ch [3] = tol + 1;
        maketree (midx + 1, y1, x2, midy);

    } else
    tree [k] .ch [3] = 0;
}
```

```
// time: O (logN) // N * N rectangle
void insert (int x, int y, int L, int k) {// Here is
   a rectangle for the insertion [x, y] - [x, y] .. A
   unit square
    if (tree [k] .x1 == tree [k] .x2 && tree [k] .y1
       == tree [k] .y2) {// leaf node //
        if (L > tree [k] .max)
        tree [k] .max = L;
        return;

    }
    int midx = (tree [k] .x1 + tree [k] .x2) / 2;
    int midy = (tree [k] .y1 + tree [k] .y2) / 2;
    if (x <= midx) {
        if (y <= midy)
        insert (x, y, L, tree [k] .ch [0]);
        else
        insert (x, y, L, tree [k] .ch [2]);

    } else {// x> midx
        if (y <= midy)
        insert (x, y, L, tree [k] .ch [3]);
        else
        insert (x, y, L, tree [k] .ch [1]);

    }
    float m = tree [tree [k] .ch [0]]. max;
    for (int i = 1; i < 4; i ++)
    m = max (m, tree [tree [k] .ch [i]]. max);
    tree [k] .max = m;
}
inline bool corss (int x1, int y1, int x2, int y2,
   int k) {// Determine if the two rectangles
   intersect
    int x3 = tree [k] .x1;
    int y3 = tree [k] .y1;
    int x4 = tree [k] .x2;
    int y4 = tree [k] .y2;
    if (y2 < y3 || y4 < y1 || x4 < x1 || x2 < x3)
    return false;
    else
```

```
    return true;
}
// time: O (logN) // N * N rectangle
float Query (int x1, int y1, int x2, int y2, int k) {
    if (corss (x1, y1, x2, y2, k) == false || tree
       [k] .max == 0) // The rectangles do not
       intersect or the rectangle max == 0 returns 0
    return 0;
    if (x1 <= tree [k] .x1 && y1 <= tree [k] .y1 &&
       tree [k] .x2 <= x2 && tree [k] .y2 <= y2) //
       If you want to query the rectangular coverage
       The current rectangle .. Then returns the
       current rectangle of the max value
    return tree [k] .max;
    // int midx = (tree [k] .x1 + tree [k] .x2) / 2;
    // int midy = (tree [k] .y1 + tree [k] .y2) / 2;
    float m [4] = {0, 0, 0, 0};
    for (int i = 0; i < 4; i ++)
    m [i] = Query (x1, y1, x2, y2, tree [k] .ch [i]);
    float mm = m [0];
    for (int i = 1; i < 4; i ++)
    mm = max (mm, m [i]);
    return mm;
}
```

4.3 堆

```
#include <bits/stdc++.h>
/**
* This is a heap practice implement coded by
   Kvar_ispw17.
* email: enkerewpo@gmail.com
* blog: enkerewpo.github.io
*/
namespace __gnu_cxx_heap {

    int inline __left(int _x) {
        return _x * 2;
    }
```

```
int inline __right(int _x) {
    return _x * 2 + 1;
}

int inline __parent(int _x) {
    return floor(_x / 2);
}

template <typename _Tp> void inline
/**
* @brief This function will obtain the heap
   feature along in subheap rooted at @idx.
*        using the datas in array @A[] with size
   @heap_size.
* @param A         Input Array
* @param idx       Rooted being processed root
* @param heap_size the size of the full structure
*/
max_heapify(_Tp A[], int idx, size_t heap_size) {
    int l = __left(idx);
    int r = __right(idx);
    int max_e;
    if (l <= (int)heap_size && A[l] > A[idx])
       max_e = l;
    else max_e = idx;
    if (r <= (int)heap_size && A[r] > A[max_e])
       max_e = r;
    if (max_e != idx) {
        std::swap(A[max_e], A[idx]);
        max_heapify(A, max_e, heap_size);
    }
}

template <typename _Tp> void inline
/**
* @brief This is going to build a heap from
   pointer @begin to @end.
* @param begin pointer
* @param end   pointer
*/
build_max_heap(_Tp* begin, _Tp* end) {
```

```
    size_t size = end - begin - 1;
    for (int iter = size / 2; iter >= 0; iter--) {
        max_heapify(begin, iter, size);
    }
}

template <typename _Tp> inline
/**
* [__swap description]
* @param lhs [description]
* @param rhs [description]
*/
void __swap(_Tp &lhs, _Tp &rhs) {
    _Tp tmp = rhs;
    rhs = lhs;
    lhs = tmp;
}

template <typename _Tp> inline
/**
* sort on heap function
* @param begin pointer
* @param end   pointer
*/
void sort_heap(_Tp* begin, _Tp* end) {
    build_max_heap(begin, end);
    size_t size = end - begin - 1;
    for (int iter = size; iter >= 1; iter--) {
        __swap(begin[0], begin[iter]);
        max_heapify(begin, 0, --size);
    }
}

template <typename _Tp> inline
_Tp maximum_heap(_Tp* begin) {
    return begin[0];
}

template <typename _Tp> inline
/**
* pop heap top element to positoin @end
```

```
* @param begin pointer
* @param end   pointer
*/
void pop(_Tp* begin, _Tp* end) {
    size_t size = end - begin - 1;
    if (end - begin < 1) {
        fprintf(stderr, "heap␣size␣is␣zero␣when␣
           it␣was␣instruct␣to␣pop()\n");
    }
    __swap(begin[0], begin[size]);
    size--;
    max_heapify(begin, 0, size);
}

template <typename _Tp> inline
/**
* increase key in heap
* @param A   array store _Tp vars
* @param idx index
* @param key keyword
*/
void heap_increase(_Tp A[], int idx, _Tp key) {
    A[idx] = key;
    while (idx > 1 and A[__parent(idx)] < A[idx])
       {
        __swap(A[idx], A[__parent(idx)]);
        idx = __parent(idx);
    }
}

template <typename _Tp> inline
/**
* push to heap function
* @param begin pointer
* @param end   pointer
* @param _x    input _Tp type varibles
*/
void push(_Tp* begin, _Tp* end, _Tp _x) {
    size_t size = end - begin;
    heap_increase(begin, size, _x);
}
```

```
}

int main() {
    int a[20] = {0, 4, 1, 3, 2, 16, 9, 10, 14, 8, 7};
    puts("original␣sequences:");
    for (int i = 1; i <= 10; i++) printf("%d␣", a[i]);
    __gnu_cxx_heap::build_max_heap(a + 1, a + 10 + 1);
    printf("\n\n");

    puts("after␣build_max_heap:");
    for (int i = 1; i <= 10; i++) printf("%d␣", a[i]);
    printf("\n\n");

    puts("after␣sort_heap:");
    __gnu_cxx_heap::sort_heap(a + 1, a + 10 + 1);
    for (int i = 1; i <= 10; i++) printf("%d␣", a[i]);
    printf("\n\n");

    __gnu_cxx_heap::build_max_heap(a + 1, a + 10 + 1);
    puts("got␣maximum␣element:");
    printf("%d\n", __gnu_cxx_heap::maximum_heap(a +
       1));
    printf("\n");

    puts("after␣pop␣element:");
    __gnu_cxx_heap::pop(a + 1, a + 10 + 1);
    for (int i = 1; i <= 10; i++) printf("%d␣", a[i]);
    printf("\n\n");

    __gnu_cxx_heap::build_max_heap(a + 1, a + 10 + 1);
    puts("after␣push␣15:");
    __gnu_cxx_heap::push(a + 1, a + 11, 15);
    for (int i = 1; i <= 11; i++) printf("%d␣", a[i]);
    printf("\n\n");

    return 0;
}
```

4.4 左偏树

```
#include <bits/stdc++.h>
using namespace std;
/**
* This is a leftist heap practice implement coded by
   Kvar_ispw17.
* email: enkerewpo@gmail.com
* blog: enkerewpo.github.io
*/
class node {
    public:
    node* ls;
    node* rs;
    int x, npl;
};

typedef node heap;
typedef node* node_addr;

heap* mesh(heap* &lhs, heap* &rhs) {
    if (lhs == NULL) return rhs;
    if (rhs->x > lhs->x) swap(lhs, rhs);
    lhs->rs = mesh(lhs->rs, rhs);
    int l, r;
    if (lhs->ls == NULL) l = 0;
    else l = lhs->ls->npl;
    r = lhs->rs->npl;
    if (l < r) swap(lhs->ls, lhs->rs);
    else lhs->npl = lhs->rs->npl + 1;
    return lhs;
}

heap* meld(heap* &lhs, heap* &rhs) {
    if (lhs == NULL)
    return rhs;
    if (rhs == NULL)
    return lhs;
    return mesh(lhs, rhs);
}
```

```
void insert(int _x, heap* &h) {
    node* x = new node;
    x->x = _x;
    x->ls = NULL;
    x->rs = NULL;
    x->npl = 0;
    h = meld(x, h);
}

int pop(heap* &h) {
    int ret = h->x;
    h = meld(h->ls, h->rs);
    return ret;
}

int main() {
    heap* h = NULL;
    while (1) {
        int t;
        cin >> t;
        insert(t, h);
    }
    return 0;
}
```

4.5 pb_ds 可并堆

```
#include <ext/pb_ds/priority_queue.hpp>

using namespace __gnu_pbds;

typedef priority_queue<int, std::greater<int>,
   pairing_heap_tag, std::allocator<char> > heap;
heap p;
heap::iterator it;

int main() {
    p.push(5);
    p.push(2);
    p.push(10);
```

```
    p.push(1);
    for (int x : p) printf("%d␣", x); // iterating by
       position
    printf("\n");
    for (it = p.begin(); it != p.end(); it++) {
        if (*it == 10) {
            p.erase(it);
        }
        if (*it == 2) {
            p.modify(it, 2333);
        }
    }
    for (int x : p) printf("%d␣", x);
    printf("\n");
    heap tmp;
    tmp.push(142857);
    p.join(tmp);
    for (int x : p) printf("%d␣", x);
    printf("\n");
    for (int x : tmp) printf("%d␣", x);
    printf("\n");
    return 0;
}
```

4.6 伸展树

```
#include <bits/stdc++.h>
#define MAXN 1000 + 5

int cnt, rt;
int Add[MAXN];

struct Tree {
    int key, num, size, fa, son[2];
} T[MAXN];

inline void PushUp (int x) {
    T[x].size = T[T[x].son[0]].size +
       T[T[x].son[1]].size + T[x].num;
}
```

```
inline void PushDown (int x) {
    if (Add[x]) {
        if (T[x].son[0]) {
            T[T[x].son[0]].key += Add[x];
            Add[T[x].son[0]] += Add[x];

        }
        if (T[x].son[1]) {
            T[T[x].son[1]].key += Add[x];
            Add[T[x].son[1]] += Add[x];

        }
        Add[x] = 0;

    }
}

inline int Newnode (int key, int fa) {// Create a new
   node and return
    ++ cnt;
    T[cnt].key = key;
    T[cnt].num = T[cnt].size = 1;
    T[cnt].fa = fa;
    T[cnt].son[0] = T[cnt].son[1] = 0;
    return cnt;
}

inline void Rotate (int x, int p) {// 0 L-1
   dextrorotation
    int y = T[x].fa;
    PushDown (y);
    PushDown (x);
    T[y].son[!p] = T[x].son[p];
    T[T[x].son[p]].fa = y;
    T[x].fa = T[y].fa;
    if (T[x].fa)
    T[T[x].fa].son[T[T[x].fa].son[1] == y] = x;
    T[x].son[p] = y;
    T[y].fa = x;
    PushUp (y);
```

```
    PushUp (x);
}

void Splay (int x, int To) {// Move the x node to the
   child of To
    while (T[x].fa != To) {
        if (T[T[x].fa].fa == To)
        Rotate (x, T[T[x].fa].son[0] == x);
        else {
            int y = T[x].fa, z = T[y].fa;
            int p = (T[z].son[0] == y);
            if (T[y].son[p] == x) Rotate (x, ! p),
               Rotate (x, p);
            else Rotate (y, p), Rotate (x, p);// a
               word spin

        }

    }
    if (To == 0) rt = x;
}

int GetPth (int p, int To) {// Returns the pth
   smallest node and moves to the child of To
    if (! rt || p > T[rt].size) return 0;
    int x = rt;
    while (x) {
        PushDown (x);
        if (p >= T[T[x].son[0]].size + 1 && p <=
           T[T[x].son[0]].size + T[x].num)
        break;
        if (p > T[T[x].son[0]].size + T[x].num) {
            p -= T[T[x].son[0]].size + T[x].num;
            x = T[x].son[1];

        } else
        x = T[x].son[0];

    }
    Splay (x, 0);
    return x;
```

```
}

int Find (int key) {
    // Returns the key value of the node if not
       returned 0 If it is transferred to the root
    if (! rt) return 0;
    int x = rt;
    while (x) {
        PushDown (x);
        if (T[x].key == key) break;
        x = T[x].son[key > T[x].key];

    }
    if (x) Splay (x, 0);
    return x;
}

int Prev () {
    // return to the root node of the non - focus
    if (! rt || ! T[rt].son[0]) return 0;
    int x = T[rt].son[0];
    while (T[x].son[1]) {
        PushDown (x);
        x = T[x].son[1];

    }
    Splay (x, 0);
    return x;
}

int Succ () {// Returns the successor to the root node
    if (! rt || ! T[rt].son[1]) return 0;
    int x = T[rt].son[1];
    while (T[x].son[0]) {
        PushDown (x);
        x = T[x].son[0];

    }
    Splay (x, 0);
    return x;
}
```

```
void Insert (int key) {// Insert the key value
    if (! rt)
    rt = Newnode (key, 0);
    else {
        int x = rt, y = 0;
        while (x) {
            PushDown (x);
            y = x;
            if (T[x].key == key) {
                T[x].num ++;
                T[x].size ++;
                break;

            }
            T[x].size ++;
            x = T[x].son[key > T[x].key];

        }
        if (! x)
        x = T[y].son[key > T[y].key] = Newnode (key,
           y);
        Splay (x, 0);

    }
}

void Delete (int key) {// Delete 1 node whose value
   is key
    int x = Find (key);
    if (! x) return;
    if (T[x].num > 1) {
        T[x].num--;
        PushUp (x);
        return;

    }
    int y = T[x].son[0];
    while (T[y].son[1])
    y = T[y].son[1];
    int z = T[x].son[1];
```

```
    while (T[z].son[0])
    z = T[z].son[0];
    if (! y && ! z) {
        rt = 0;
        return;

    }
    if (! y) {
        Splay (z, 0);
        T[z].son[0] = 0;
        PushUp (z);
        return;

    }
    if (! z) {
        Splay (y, 0);
        T[y].son[1] = 0;
        PushUp (y);
        return;

    }
    Splay (y, 0);
    Splay (z, y);
    T[z].son[0] = 0;
    PushUp (z);
    PushUp (y);
}

int GetRank (int key) {// Get the number of nodes
   with value <= key
    if (! Find (key)) {
        Insert (key);
        int tmp = T[T[rt].son[0]].size;
        Delete (key);
        return tmp;

    } else
    return T[T[rt].son[0]].size + T[rt].num;
}
```

```
void Delete (int l, int r) {// Delete all nodes with
   values   in[l, r] l!= r
    if (! Find (l)) Insert (l);
    int p = Prev ();
    if (! Find (r)) Insert (r);
    int q = Succ ();
    if (! p && ! q) {
        rt = 0;
        return;

    }
    if (! p) {
        T[rt].son[0] = 0;
        PushUp (rt);
        return;

    }
    if (! q) {
        Splay (p, 0);
        T[rt].son[1] = 0;
        PushUp (rt);
        return;

    }
    Splay (p, q);
    T[p].son[1] = 0;
    PushUp (p);
    PushUp (q);
}
```

4.7 pb_ds 伸展树

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
#include <bits/stdc++.h>
using namespace __gnu_pbds;
using namespace std;
typedef tree<int, null_type, less<int>,
   splay_tree_tag, tree_order_statistics_node_update>
   splay;
```

```
splay s;
splay :: iterator it;
int main() {

    for (int i = 20; i >= 15; i--) s.insert(i);  //15
       16 17 18 19 20
    for (it = s.begin(); it != s.end(); it++)
       printf("%d␣", *it);
    printf("\n");
    it = s.find_by_order(3);
    printf("NO.4␣=␣%d\n", *it);     // return NO.k+1
       if you input find_by_order(k)
    int rnk = s.order_of_key(16);
    printf("RANK␣OF␣16␣is␣%d\n", rnk + 1);  // return
       how many items are "less" than your input
    splay t; t.clear();
    t.join(s); // copy 's' to 't'
    printf("␣␣␣␣t␣=␣");
    for (it = t.begin(); it != t.end(); it++)
       printf("%d␣", *it);
    printf("\n");
    splay u; u.clear();
    t.split(17, u); // clear 'u' and move items
       "greater" then 17 to 'u'
    printf("␣␣␣␣u␣=␣");
    for (it = u.begin(); it != u.end(); it++)
       printf("%d␣", *it);
    printf("\n");
    return 0;
}
```

4.8 bitset

```
#include <cstdio>
#include <iostream>
#include <bitset>
using namespace std;
#define MAXN 20

int main() {
```

```
    bitset<5> a("10010");
    bitset<5> b("01110");
    cout << (a ^ b) << endl; // 11100
    cout << (a & b) << endl; // 00010
    a.flip();
    cout << a << endl; // 01101
    return 0;
}
```

4.9 可持久化线段树

```
#include<cstdio>
#include<cstring>
#include<iostream>
#include<algorithm>
#include<vector>

using namespace std;
const int MAXN = 1e5 + 5;
int n, m, cnt, root[MAXN], a[MAXN], x, y, k;

struct node {
    int l, r, sum;
} T[MAXN * 30];
vector<int> v;

int getid(int x) {
    return lower_bound(v.begin(), v.end(), x) -
       v.begin() + 1;
}

void update(int l, int r, int &x, int y, int pos) {
    T[++cnt] = T[y], T[cnt].sum++, x = cnt;
    if (l == r) return;
    int mid = (l + r) >> 1;
    if (mid >= pos) update(l, mid, T[x].l, T[y].l,
       pos);
    else update(mid + 1, r, T[x].r, T[y].r, pos);
}
```

```
int query(int l, int r, int x, int y, int k) {
    if (l == r) return l;
    int mid = (l + r) >> 1;
    int sum = T[T[y].l].sum - T[T[x].l].sum;
    if (sum >= k) return query(l, mid, T[x].l,
       T[y].l, k);
    else return query(mid + 1, r, T[x].r, T[y].r, k -
       sum);
}

int main() {
    while (scanf("%d%d", &n, &m) != EOF) {
    v.clear();
    memset(root, 0, sizeof(root));
    for (int i = 1; i <= n; i++) {
        scanf("%d", &a[i]);
        v.push_back(a[i]);
    }
    sort(v.begin(), v.end());
    v.erase(unique(v.begin(), v.end()), v.end());
    for (int i = 1; i <= n; i++) {
        update(1, n, root[i], root[i - 1],
           getid(a[i]));
    }
    while (m--) {
            scanf("%d%d%d", &x, &y, &k);
            printf("%d\n", v[query(1, n, root[x - 1],
               root[y], k) - 1]);
        }
    }
    return 0;
}
```

4.10 跳跃表

```
#include <cstdio>

#define MAX_LEVEL 10 // The maximum number of layers

//node
typedef struct nodeStructure {
    int key;
    int value;
    struct nodeStructure *forward[1];
} nodeStructure;

// jump table
typedef struct skiplist {
    int level;
    nodeStructure *header;
} skiplist;

// Create the node
nodeStructure *createNode(int level, int key, int
   value)
{
    nodeStructure *ns =(nodeStructure *)
       malloc(sizeof(nodeStructure) + level
       *sizeof(nodeStructure *));
    ns->key = key;
    ns->value = value;
    return ns;
}

// Initialize the jump table
skiplist *createSkiplist()
{
    skiplist *sl =(skiplist *)
       malloc(sizeof(skiplist));
    sl->level = 0;
    sl->header = createNode(MAX_LEVEL-1,0,0);
    for(int i = 0; i <MAX_LEVEL; i++) {
        sl->header->forward[i] = NULL;
    }
```

```
    return sl;
}

// Randomly generated layers
int randomLevel()
{
    int k = 1;
    while(rand()% 2)
    k++;
    k =(k <MAX_LEVEL)? k: MAX_LEVEL;
    return k;
}

// insert node
bool insert(skiplist *sl, int key, int value)
{
    nodeStructure *update[MAX_LEVEL];
    nodeStructure *p, *q = NULL;
    p = sl->header;
    int k = sl->level;
    // Find the location you want to insert from the
       top
    // fill update
    for(int i = k-1; i >= 0; i--) {
        while((q = p->forward[i]) &&(q->key <key)) {
            p = q;
        }
        update[i] = p;
    }
    // can not insert the same key
    if(q && q->key == key) {
        return false;
    }

    // Generate a random number of layers K
    // Create a new node to insert q
    // Insert layer by layer
    k = randomLevel();
    // update the level of the jump table
    if(k>(sl->level)) {
        for(int i = sl->level; i <k; i++) {
```

```
            update[i] = sl->header;
        }
        sl->level = k;
    }

    q = createNode(k, key, value);
    // Update node pointer layer by layer, same as
       normal list insertion
    for(int i = 0; i <k; i++) {
        q->forward[i] = update[i]->forward[i];
        update[i]->forward[i] = q;
    }
    return true;
}

// Search the value of the specified key
int search(skiplist *sl, int key)
{
    nodeStructure *p, *q = NULL;
    p = sl->header;
    // Search from the highest level
    int k = sl->level;
    for(int i = k-1; i >= 0; i--) {
        while((q = p->forward[i]) &&(q->key <= key)) {
            if(q->key == key) {
                return q->value;
            }
            p = q;
        }
    }
    return NULL;
}

// delete the specified key
bool deleteSL(skiplist *sl, int key)
{
    nodeStructure *update[MAX_LEVEL];
    nodeStructure *p, *q = NULL;
    p = sl->header;
    // Search from the highest level
    int k = sl->level;
```

```
    for(int i = k-1; i >= 0; i--) {
        while((q = p->forward[i]) &&(q->key <key)) {
            p = q;
        }
        update[i] = p;
    }
    if(q && q->key == key) {
        / / Delete layer by layer, and delete the
           same general list
        for(int i = 0; i <sl->level; i++) {
            if(update[i]->forward[i] == q) {
                update[i]->forward[i] = q->forward[i];
            }
        }
        free(q);
        // If you delete the node is the largest
           layer, you need to re-maintain the jump
           table
        for(int i = sl->level - 1; i >= 0; i--) {
            if(sl->header->forward[i] == NULL) {
                sl->level--;
            }
        }
        return true;
    } else
    return false;
}

void printSL(skiplist *sl)
{
    // print from the highest level
    nodeStructure *p, *q = NULL;

    // Search from the highest level
    int k = sl->level;
    for(int i = k-1; i >= 0; i--) {
        p = sl->header;
        while(q = p->forward[i]) {
            printf("%␣d->", p->value);
            p = q;
        }
```

```
        printf("\n");
    }
    printf("\n");
}
int main()
{
    skiplist *sl = createSkiplist();
    for(int i = 1; i <= 19; i++) {
        insert(sl, i, i *2);
    }
    printSL(sl);
    //search for
    int i = search(sl, 4);
    printf("i␣=%␣d␣\n", i);
    //delete
    bool b = deleteSL(sl, 4);
    if(b) printf("delete␣successfully␣\n");
    printSL(sl);
    return 0;
}
```

5 动态规划

5.1 斜率优化

```
// BZOJ 1991
#include <cstdio>
#include <cstring>
#include <cctype>
#include <algorithm>
#include <cmath>
#define slope_optimize
using namespace std;
#define N 1000000 + 5
typedef long long ll;
#define int ll
int n;
ll a, b, c;
ll A[N], S[N], dp[N];
struct point {
    point() {}
    point(int _, int __) : x(_), y(__) {}
    point(int _, int __, int ___) : x(_), y(__),
       id(___) {}
    int x, y, id;
};
template <typename _Tp>
void read(_Tp &a, char c = 0, int op = 1) {
    for (c = getchar(); !isdigit(c); c = getchar()) c
       == '-' ? op = -1 : op = 1;
    for (a = 0; isdigit(c); a = a * 10 + c - '0', c =
       getchar());
    a *= op;
}
double slp(point a, point b) { return double(b.y -
   a.y) / double(b.x - a.x); }
ll w(int x, int y) { ll s = S[y] - S[x - 1]; return a
   * s * s + b * s + c; }
point q[N];
int l, r;
signed main() {
```

```
    read(n), read(a), read(b), read(c);
    for (int i = 1; i <= n; i++) read(A[i]), S[i] =
       S[i - 1] + A[i];
    slope_optimize {
        for (int i = 1; i <= n; i++) {
            double k = S[i];
            while (l < r && slp(q[l], q[l + 1]) < k)
               l++;
            point bst = q[l];
            dp[i] = dp[bst.id] + w(bst.id + 1, i);
            point now = point(2 * a * S[i], dp[i] + a
               * S[i] * S[i] - b * S[i], i);
            while (l < r && slp(q[r], now) < slp(q[r
               - 1], q[r])) r--;
            q[++r] = now;
        }
    }
    printf("%lld\n", dp[n]);
    return 0;

}
```

5.2 有依赖的背包问题

```
#include <cstdio>
#include <algorithm>
#include <stack>
#include <cstring>

#define N 100 + 5
#define M 500 + 5
using namespace std;

int dp[N][M];

int hd[N], nxt[N], to[N], tot;
int hd2[N], nxt2[N], to2[N], in2[N], tot2;
void add (int a, int b) { nxt [++tot ] = hd [a], to
   [hd [a] = tot ] = b; }
```

```
void add2(int a, int b) { nxt2[++tot2] = hd2[a],
   to2[hd2[a] = tot2] = b, in2[b]++; }

int n, m, _n, color[N], low[N], dfn[N], tim, sv[N],
   sw[N];
int v[N], w[N];
bool vis[N];
stack<int> s;

void tarjan(int u) {
    low[u] = dfn[u] = ++tim;
    s.push(u);
    vis[u] = true;
    for (int e = hd[u]; e; e = nxt[e]) {
        int v = to[e];
        if (!dfn[v]) {
            tarjan(v);
            low[u] = min(low[u], low[v]);
        } else if (vis[v])low[u] = min(low[u],
           dfn[v]);
    }
    if (low[u] == dfn[u]) {
        _n++; int x; do {
            x = s.top();
            s.pop();
            vis[x] = false;
            color[x] = _n;
            sv[_n] += v[x];
            sw[_n] += w[x];
        } while (x != u);
    }
}

void dfs(int i) {
    for (int e = hd2[i]; e; e = nxt2[e]) {
        int v = to2[e];
        dfs(v);
        for (int j = m - sw[i]; j >= 0; j--) {
            for (int k = 0; k <= j; k++) {
                dp[i][j] = max(dp[i][j], dp[i][k] +
                   dp[v][j - k]);
```

```
            }
        }
    }
    for (int j = m; j >= 0; j--) {
        if (j >= sw[i]) dp[i][j] = dp[i][j - sw[i]] +
           sv[i];
        else dp[i][j] = 0;
    }
}

int main() {
    scanf("%d%d", &n, &m);
    for (int i = 1; i <= n; i++) scanf("%d", &w[i]);
    for (int i = 1; i <= n; i++) scanf("%d", &v[i]);
    for (int i = 1; i <= n; i++) {
        int a;
        scanf("%d", &a);
        if (a) add(a, i);
    }
    for (int i = 1; i <= n; i++) if (!dfn[i])
       tarjan(i);
    for (int i = 1; i <= n; i++) {
        for (int e = hd[i]; e; e = nxt[e]) {
            int v = to[e];
            if (color[v] != color[i]) add2(color[i],
               color[v]);
        }
    }
    for (int i = 1; i <= _n; i++) if (in2[i] == 0)
       add2(_n + 1, i);
    dfs(_n + 1);
    printf("%d\n", dp[_n + 1][m]);
    return 0;
}
```

5.3 最长公共上升子序列

```
#include <bits/stdc++.h>

using namespace std;
```

```
const int N = 5005;

int a[N], b[N], f[N][N], g[N][N];

void work(int x, int y, int z) {
    if (!z) return;
    while (a[x] != b[y]) --x;
    work(x, g[x][y], z - 1);
    printf("%d␣", b[y]);
}
int main() {
    int n, m, i, tma, pre, j, ans = 0, t;
    scanf("%d", &n);
    for (i = 1; i <= n; ++i) scanf("%d", &a[i]);
    scanf("%d", &m);
    for (i = 1; i <= m; ++i) scanf("%d", &b[i]);
    for (i = 1; i <= n; ++i)
    for (tma = 0, j = 1; j <= m; ++j) {
        if (b[j] < a[i] && f[i - 1][j] > tma) tma =
           f[i - 1][pre = j];
        if (a[i] == b[j]) {f[i][j] = tma + 1;
           g[i][j]=pre;} else f[i][j] = f[i - 1][j];
        if (f[i][j] > ans) ans = f[i][t = j];
    }
    printf("%d\n", ans);
    if (ans) {
        work(n, t, ans);
        printf("\n");
    }
    return 0;
}
```

6 其他

6.1 带修改莫队算法

```
#include <cstdio>
#include <algorithm>
#include <cstring>
#include <cmath>
#define N 1000000 + 5
#define M 1000000 + 5
using namespace std;
#define int long long
int n, m, block;
int a[N], tmp[N], ans[M];

int qcnt;
struct query {
    query() {}
    query(int _l, int _r, int _d, int _i) : l(_l),
       r(_r), dfn(_d), id(_i) {}
    int l, r, dfn, id;
    bool operator < (const query &rhs) const {
        return (l / block) == (rhs.l / block) ? (r ==
           rhs.r ? dfn < rhs.dfn : r < rhs.r) : l /
           block < rhs.l / block;
    }

} q[M];


int ccnt;
struct change {
    change() {}
    change(int _p, int _v, int _pr) : pos(_p),
       val(_v), pre(_pr) {}
    int pos, val, pre;

} c[M];

int cnt[N], tot;
```

```
int L, R, DFN;

void add(int p) { tot += (++cnt[p]) == 1; }
void del(int p) { tot -= (--cnt[p]) == 0; }
void rev(int p, int v) {
    if(L <= p && p <= R) add(v), del(a[p]);
    a[p] = v;
}

void mo() {
    for(int i = 1; i <= qcnt; i++) {
        while(DFN < q[i].dfn) DFN++, rev(c[DFN].pos,
           c[DFN].val);
        while(DFN > q[i].dfn) rev(c[DFN].pos,
           c[DFN].pre), DFN--;
        while(R < q[i].r) R++, add(a[R]);
        while(R > q[i].r) del(a[R]), R--;
        while(L < q[i].l) del(a[L]), L++;
        while(L > q[i].l) L--, add(a[L]);
        ans[q[i].id] = tot;
    }
}

signed main() {
    scanf("%lld%lld", &n, &m);
    block = sqrt(n);
    for(int i = 1; i <= n; i++) {
        scanf("%lld", &a[i]);
        tmp[i] = a[i];
    }
    int a, b;
    for(int i = 1; i <= m; i++) {
        char op[3];
        scanf("%s%lld%lld", op, &a, &b);
        if(op[0] == 'Q') qcnt++, q[qcnt] = query(a,
           b, ccnt, qcnt);
        else c[++ccnt] = change(a, b, tmp[a]), tmp[a]
           = b;
    }
    sort(q + 1,q + qcnt + 1);
    mo();
```

```
    for(int i = 1; i <= qcnt; i++) printf("%lld\n",
       ans[i]);
    return 0;
}
```